In presenting this dissertation in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my permission.

Signature _____

Edward Lum

Date 11/30/17

SIMULATING THE KATANA EFFECT
MONTE CARLO NEUTRON TRANSPORT COMBINED WITH FINITE ELEMENT ANALYSIS TO CALCULATE NEGATIVE REACTIVITY DUE TO DUCT-BOWING

By Edward Lum

A dissertation
Submitted in partial fulfillment
Of the requirements for the degree of
Doctor of Philosophy in Nuclear Science and Engineering
Idaho State University
December 2017

To the Graduate Faculty

The members of the committee appointed to examine the dissertation of EDWARD LUM find it satisfactory and recommend that it be accepted.

Dr. Chad Pope
Major Advisor

Dr. George Imel
Committee Member

Committee Member

Dr. Charles Solbrig
Committee Member

Dr. Hossein Mousavinezhad
Graduate Faculty Representative

Dedication

May this work glorify God above all things, for He is the provider of creation that we get to discover and explore. I would not be presenting this work without His guidance and blessings.

Colossians 3:23

> "Whatever you do, work heartily, as for the Lord and not for men…"

Biography

Edward was born January 10, 1987 in St. Mary's Hospital, London. He spent his early years in Dry Drayton, Cambridge until the age of 12 when he, along with his mother and sister, moved to Pittsburgh, PA in 2000 to be closer to his grandparents. He attended Winchester Thurston High School and graduated in 2005. During his senior year in high school, Edward enlisted in the United States Air Force Reserve as an integrated avionics technician. He spent 8 years working at the Pittsburgh Air Reserve station, where he deployed multiple times to Europe and the Middle East. After accomplishing two years in active duty training, he applied and was accepted to Geneva College in Beaver Falls, PA where he pursued a B.S. in physics and applied mathematics. In the summer of 2010, he attended an internship at Idaho State University in the nuclear engineering department and in August 2011, he began his master's in nuclear engineering. In December the following year, Edward completed his master's work and decided that he wanted to continue his pursuit of nuclear engineering education and began his doctoral research in January of 2013. After four years of research Edward passed his dissertation defense on November 17, 2017 leading to a December 2017 graduation from Idaho State University.

Acknowledgement

The work presented here is the culmination of nearly four years of effort. Like any large endeavor, it was not a singular effort. It required the support and encouragement of co-workers and family. I would like to acknowledge my dissertation committee for their encouragement in undertaking this challenging problem. Specifically, I would like to thank Dr. Chad Pope for his continued guidance both in science and in life. His support of this dissertation was instrumental to its completion. The reason that I have pursued nuclear engineering as a career is Dr. Jay Kunze, his excitement and dedication to the subject was and is an inspiration. The members of the New Geneva Orthodox Presbyterian Church have been in constant prayer for me and the dissertation since I arrived in Idaho. Their encouragement was greatly appreciated. My mother, Cynthia Lum, provided extensive editorial revisions and emotional support during my graduate school career, and I want to give a special acknowledgement for her assistance.

The final individual I would like to acknowledge is my wife, Breanna Lum. She has been there since the very beginning and has struggled just as much as I have through the process. Her support and encouragement cannot be quantified nor over-appreciated. Through many long nights where I was working and she was sleeping alone to her putting up with the constant hum and heat of computers. Thank you, Breanna, with all my love.

Table of Contents

List of Tables

List of Figures

Abstract

The research presented here focuses on simulating the duct-bowing reactivity coefficient for EBR-II run 138B. Quantification of duct-bowing has been a persistent problem since liquid metal fast reactors were first designed and operated. Simulation has been difficult because the level-of-detail required to fully simulate this effect has exceeded most computing capabilities. The new method shown here utilizes a finite element analysis code called ANSYS to analysis the thermal and structural components. The thermal expansion is calculated from temperature and power data. The displacement data are placed into an MCNP model of EBR-II run 138B where neutrons are passed through the simulated bowed ducts. Multiple MCNP files were created such that a set of changes in k_{eff} were compiled and a temperature coefficient was calculated. The method was validated using GODIVA-IV as a benchmark. The values calculated for the GODIVA-IV thermal expansion reactivity coefficient agreed with reported values.

1.0 INTRODUCTION

Experimental Breeder Reactor II (EBR-II) pioneered many advances in nuclear engineering, including on-site reprocessing of the reactor fuel. These advances were researched and characterized, but several phenomena proved extremely difficult to quantify. One such phenomena was the reactivity effect of thermal bowing of the reactor fuel assemblies (duct-bowing). Due to the high-power density and pool-type design, a large change in temperature (~350 ^{o}C) existed from the center of the core to the reactor boundary. This change in temperature led to duct deformation known as duct-bowing. Due to the extreme complexity duct-bowing is extremely difficult to calculate.

The duct-bowing phenomena has existed since Experimental Breeder Reactor I. During that time, efforts were made to minimize the duct-bowing because it was a positive reactivity coefficient and difficult to quantify. The duct-bowing phenomena was made more favorable in EBR-II where the positive reactivity was decreased to a small part of the power band, however, quantification was not fully achieved. Multiple efforts were made to calculate the movement of the ducts, but these efforts led to highly uncertain answers. The duct-bowing reactivity coefficient was accounted for by quantifying the other reactor coefficients and then attributing the reactivity that was left over to duct-bowing. While this method matched the experimental data, it also had high uncertainty due to the combined uncertainty of the other parameters.

Duct-bowing is extremely difficult to quantify because of the depth of complexity. Quantification of a single bowed duct is a complex effect due to the specificity of input required for proper deformation. Further complicating duct-bowing is that reactors normally contain hundreds of ducts, integrating these complex effects into one large effect. These deformations cannot have any real simplification applied since localized hot-spots can have significant effects on thermal expansion. The non-linear nature of the thermal expansion coefficient leads to the requirement of a finely discretized temperature function to account for local temperature hot-spots and subsequent thermal expansions. Accounting for one duct is a complex task, and fast reactor cores are usually comprised of hundreds of ducts. Even if the ducts could be treated in isolation, the complexity of accounting for hundreds of bowed ducts is significant. The final complex component is the total integration into a reactor effect. The ducts are not in isolation: each duct acts on its neighbors, meaning individual duct movement is contributed to by the region where the duct is located. The above reasons are why duct-bowing is difficult to quantify.

The following work presents a method for quantifying duct-bowing. The primary enhancement in this new method was the retention of the duct-level effects to the entire reactor compared to the old method where the reactor was smeared into simplified parts. Ducts can move and be moved by the surrounding ducts, even ducts that are not direct neighbors. Verification of the new method was done using the critical experiment GODIVA-IV.

1.1 History

EBR-II was a test reactor operated by Argonne National Laboratory. The reactor was operational between 1964 and 1994.[a] From 1964 to 1969, the original purpose of EBR-II was to show that a sodium-cooled fast reactor could be used as a power station which utilized an on-site reprocessing facility to recycle its own fuel. This mission was accomplished in September 1969 and would later be referred to as phase I. EBR-II had five of these phases; each phase was started at a different time but continued until the reactor

[a] John Sackett was consulted and provided many details as to the history and operation of EBR-II

was shut down in September of 1994. Figure 1 shows a chart of the phases and where they began. [1]

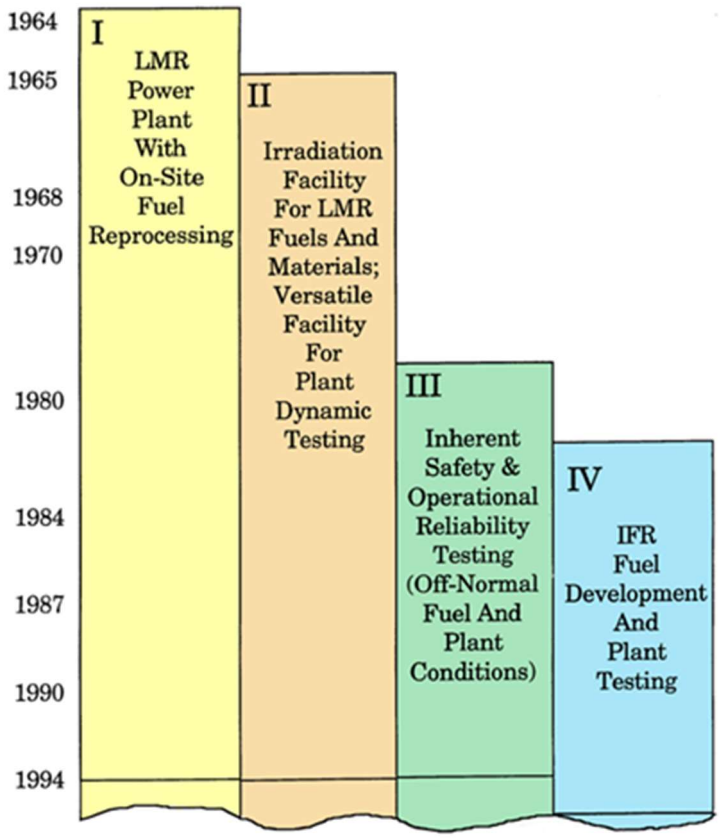

Figure 1. Phases of EBR-II.

EBR-II went dry critical on September 30, 1960. Tests were done in a dry critical condition to confirm calculations and to write procedures for both wet critical and the approach to power. Two years later, the sodium plant was completed, wet critical was established in November of 1962. In the coming years, the reactor slowly increased operating power until September of 1969 when 62.5 MWth was achieved. From that time, 20 MWe was sold to the Idaho Power Company.

During the early years of EBR-II, it was primarily used as a demonstration facility for a liquid metal fast reactor with on-site reprocessing. In May 1965, the reactor used recycled fuel for the first time. This lead to the demonstration of a closed fuel cycle in

September of 1969. After 1965, the facility initiated phase II where it was used as an irradiation test facility where experimental assemblies were irradiated. EBR-II continued in this role until early 1978 when it was deemed that continued experimental irradiations would decrease the utilization of the reactor. It was at this point that phase III was initiated and new programs were proposed to test design basis accident conditions for future Liquid Metal Reactors (LMR).

One of the proposed set of tests was the Safety Heat Removal Tests (SHRT). These experiments were to test an LMR during catastrophic failures of heat removal at full power. The purpose of these experiments was to prove that EBR-II had enough passive safety features to shut down the reactor and use natural convection for the decay heat removal. There were dozens of SHRT experiments, but the most severe of these was SHRT 45 during run 138B conducted on April 3, 1986. The conditions of the plant during SHRT 45 were 100% power, 100% initial primary flow, 104% secondary flow, both the secondary and primary pumps were coasted down to zero percent and the reactor SCRAM mechanism was disabled. Table 1 lists the plant conditions during the test. [2] The result of this experiment was the reactor shutdown to approximately zero percent power within fifteen minutes. Peak pin temperature remained within safe limits and post-analysis showed only minimal damage to the fuel. There was also no detectable fuel breach during and after the fuel was removed. That experiment not only proved the inherent safety of the reactor but also was proof that a commercial LMR power reactor could withstand catastrophic failure without significant damage.

EBR-II continued to operate and perform experiments until August 1994. EBR-II last operated on September 27, 1994.

Table 1. Plant Condition for SHRT 40, 41, and 45

SHRT No.	Initial power (% of rated)	Initial primary flow (% of rated)	Initial secondary flow (% of nominal)	Primary pump coast-down condition	Auxiliary pump condition	Secondary pump coast-down condition	Predicted peak cladding temperature of driver (ºC)	
							Nominal	w/uncertainties
40	50	100	68	Passively controlled, 95s	On battery	Trip of 2400-V breaker to M-G set	598	635
41	50	100	68	Actively controlled, 200 s	Off	Same as SHRT 40[a]	598	622
45	100	100	104	Same as SHRT 40	On battery	Same as SHRT 41	747	802

EBR-II had to be modified to conduct SHRT 45. The reason for the modification was to ensure there were still mechanisms in place to shut the reactor down in case of unexpected behavior. The goal was to preserve the original plant so the tests would be as analogous as possible to station blackout without SCRAM, but also to provide a means to shut down the reactor if it deviated from calculated predictions.

One of the systems modified was the primary coolant system, specifically the primary pump controller. The pump coast-down function is a critical characteristic when determining peak temperature. The shape and duration of the coast-down was configured using the pump controllers to simulate loss-of-power to the pumps. Preliminary investigation of SHRT showed that the coolant pump controllers were insufficient to provide a variety of coast-down function shapes. A passive method was required to simulate true blackout. This meant opening the 2400V circuit breaker to the pumps. The active method would simulate the coast-down by controlling the pump to reduce speed in

[a] The original published table had the condition “same as SHRT 41”, this is assumed to be an error because the condition is for SHRT 41. The correction was assumed to be SHRT 40.

a manner which simulated a coast-down while keeping power to the pump. Run 138B used the active method for pump control.

1.2 Basic Description

The following sections provide a basic description of EBR-II run 138B, necessary for duct-bowing analysis. A more detailed description can be found in appendix C.1.

1.2.1 Reactor

EBR-II was a sodium cooled fast reactor, which was operational from 1964 through 1994. It had a maximum heat output of 62.5 MW (about 20 MW electric). Although initially designed to breed more fuel than it consumed, it was later reconfigured to operate as an irradiation facility where a variety of fuels and structural materials were tested. Table 2 contains some reactor specifications. [3]

EBR-II consisted of 637 vertical, hexagonally shaped, removable assemblies. As shown in Figure 2, these assemblies were divided into three regions (moving outward from the middle): the core, an inner blanket, and an outer blanket. The number of assemblies within each region varied over the years with changing configurations due to the experimental nature of the reactor. In the core region were the driver assemblies containing 91 fuel elements each. The fuel was made of enriched uranium metal alloyed with a small percentage of other elements to improve fuel properties and clad within stainless-steel tubes. Also in the core region were 2 safety and 8 control type assemblies. Both safety and control assemblies contained 61 fuel elements; they were inserted to increase reactor power or lowered from the core to reduce the reactor power. The control rods were later upgraded to a high worth version, which included a B_4C poison region above the fuel to maximize the reactivity swing. Other assemblies in the driver included stainless-steel dummies, half worth drivers, and experimental/instrumentation assemblies. The inner blanket region initially consisted of assemblies which contained depleted uranium for breeding new fissile fuel and reflecting neutrons back toward the center of the core. After proving the breeding concept, these were replaced with stainless-steel reflectors more compatible with the goal

of an irradiation facility. The outer blanket region consisted almost entirely of depleted uranium assemblies, again for breeding and reflection.

Table 2. EBR-II Basic Specifications

Number of Assemblies	637
Number of Core Driver Sub-assemblies	127
Enrichment	67 %
Total ^{235}U mass	24,955 g
Operating Temperature	473 ºC
Coolant	Sodium
Coolant Flow Rate	34070 lpm
Assembly Pitch	Hexagonal 5.89 cm
Core Effective Height	34.29 cm
Core Effective Diameter	69.67 cm
Power (Electric)	62.5 MW (20 MW)

The assemblies were approximately 92 inches long, although only about 14 inches were uranium fuel. Above and beneath the fuel were neutron reflectors, which also began as depleted uranium but were replaced with stainless steel in run 138B. At the top of each assembly was an adapter for removal, and at the bottom was an adapter which fit into a grid plenum support structure. Orifices at the bottom of each assembly allowed sodium coolant to flow upward, with larger holes (and therefore greater flow) for those assemblies in the center region which produced the highest heat.

EBR II was a pool-type reactor rather than the more common loop-type reactor. The reactor core, two primary coolant centrifugal pumps, and intermediate heat exchanger were all contained within a large stainless-steel vessel along with 337,000 liters of primary sodium coolant. With the pool-type design, any leaks within the primary coolant piping would simply drain into the primary coolant pool. The design eliminates any loss-of-coolant accidents which are typically accounted for in light water reactors. While

such a leak would impact plant efficiency, no leakage of primary sodium coolant outside the vessel would occur. Heat from the primary coolant was transferred to a secondary sodium loop through a heat exchanger submerged within the primary pool. Thus, heat from the reactor was removed to the secondary sodium loop while minimizing neutron activation of the secondary sodium. Finally, the secondary sodium was used to generate superheated steam for electricity generation. Within the stainless-steel vessel, the EBR-II core was supported by a grid plenum structure and an upper part of the reactor vessel cover that served as a neutron shield. Surrounding the core, within the reactor vessel, were radial layers of graphite and borated graphite shielding.

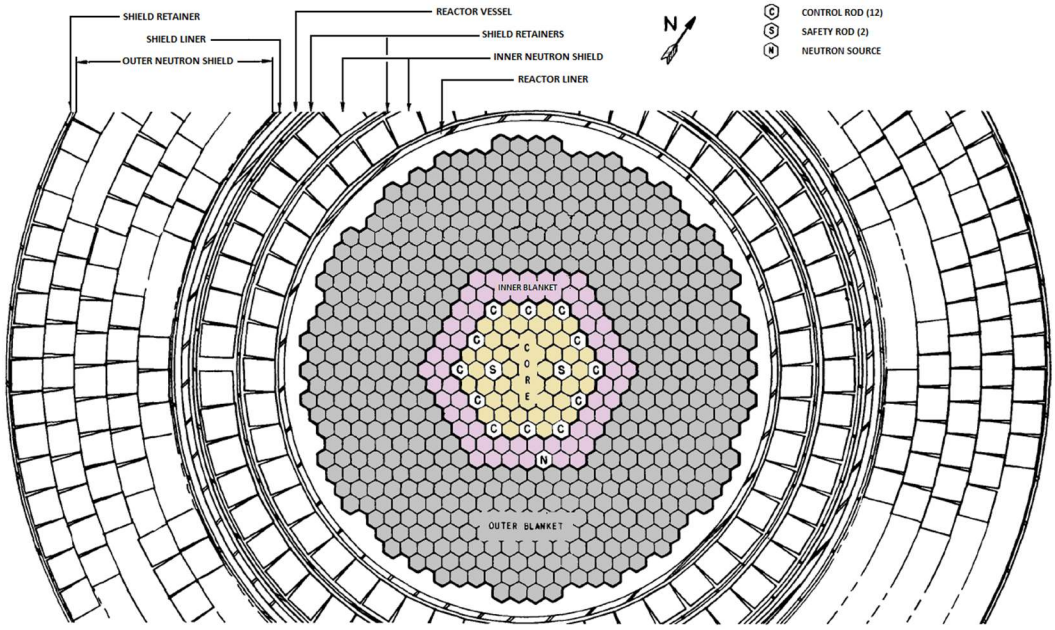

Figure 2. EBR-II reactor core.

The top cover could be removed when required to enable replacement and handling of assemblies, and contained penetrations to allow for the control rod drive mechanism. Surrounding the reactor vessel radially were additional layers of graphite and borated graphite shielding. The entire vessel was submerged in about 10 feet of liquid sodium in a large pool-type design, with the surrounding sodium being forced into the core and blanket regions using two main centrifugal pumps. Figure 3 shows a simple reactor plot of run 138B core configuration, the different colors represent different materials.

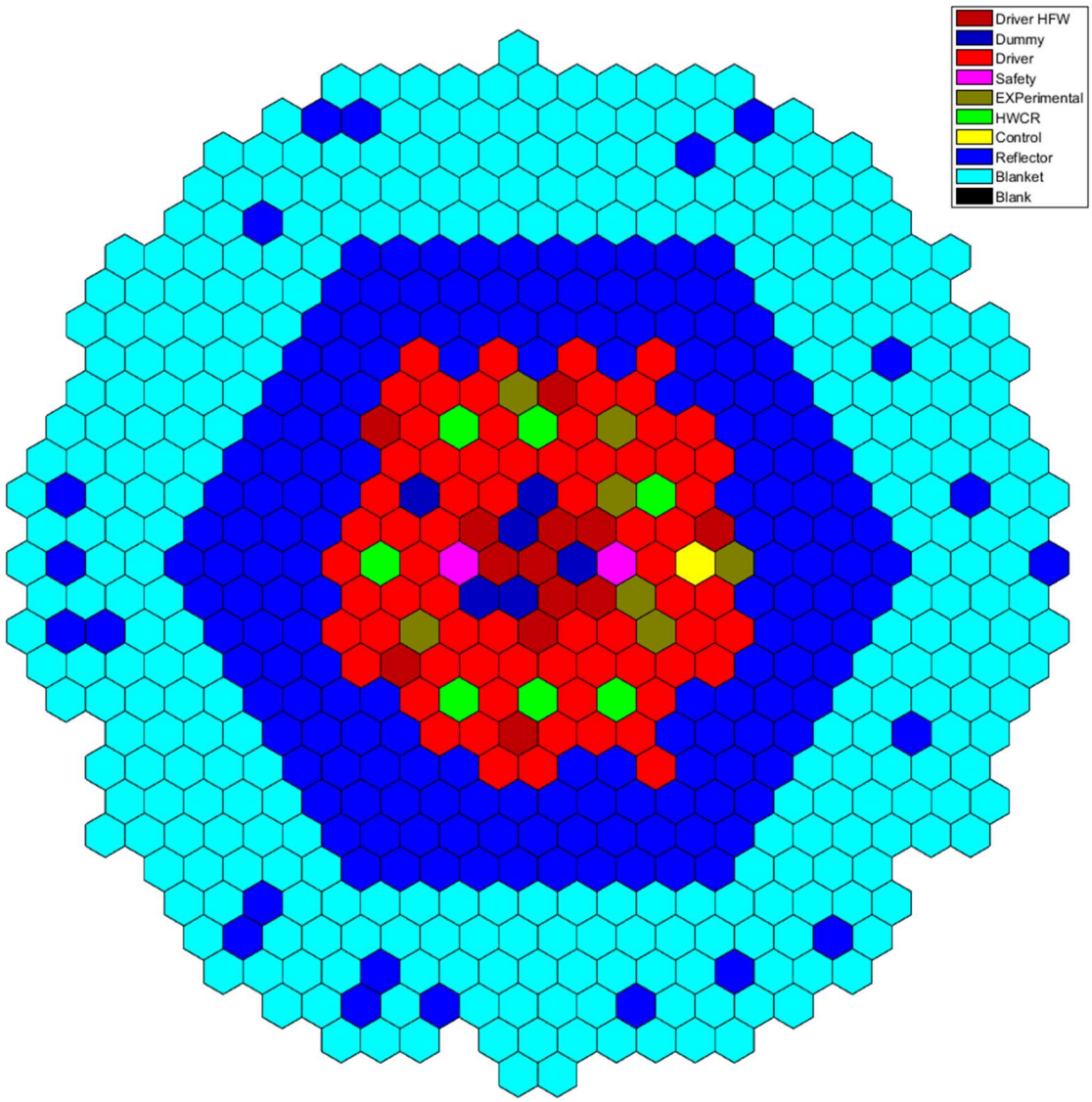

Figure 3. EBR-II run 138B core figure.

1.2.2 Subassembly

An EBR-II driver subassembly was broken down into four main sections. The upper extension contained a stainless-steel shield block. The purpose of which was to replace the depleted uranium blanket with a material that was not sodium. The upper extension was welded to the top of the duct. The following part descriptions can be referenced in Figure 4.

The core region contained the fuel elements. The number of elements varied depending on the subassembly, but a typical driver contained 91 fuel elements. The elements had a helical wire wrap which ran the length of the fuel element. The primary purpose of the wire wrap was to provide even spacing between all the elements and the duct wall.

The lower extension was like the upper extension, acting as a shield, but also contained the grid where the fuel elements were attached. The grid contained T-bars where the fuel elements would slide onto the grid. This prevented an axial movement of the fuel elements but did not restrain the elements where they became rigid on the bar. The lower extension was welded at the bottom of the duct.

The lower adapter was welded directly to the lower extension. The purpose of the adapter was to lock the subassembly to the reactor grid plate. When the subassembly was locked into the grid plate, sodium was forced through orifices in the lower adapter; flowed axially through the subassembly, and then out through the top of the upper extension. The sodium flowing through the duct did not interact with the pool sodium until after it left the intermediate heat exchanger.

Appendix C.2 shows more detailed photographs of an EBR-II subassembly, lower extension, and lower adapter.

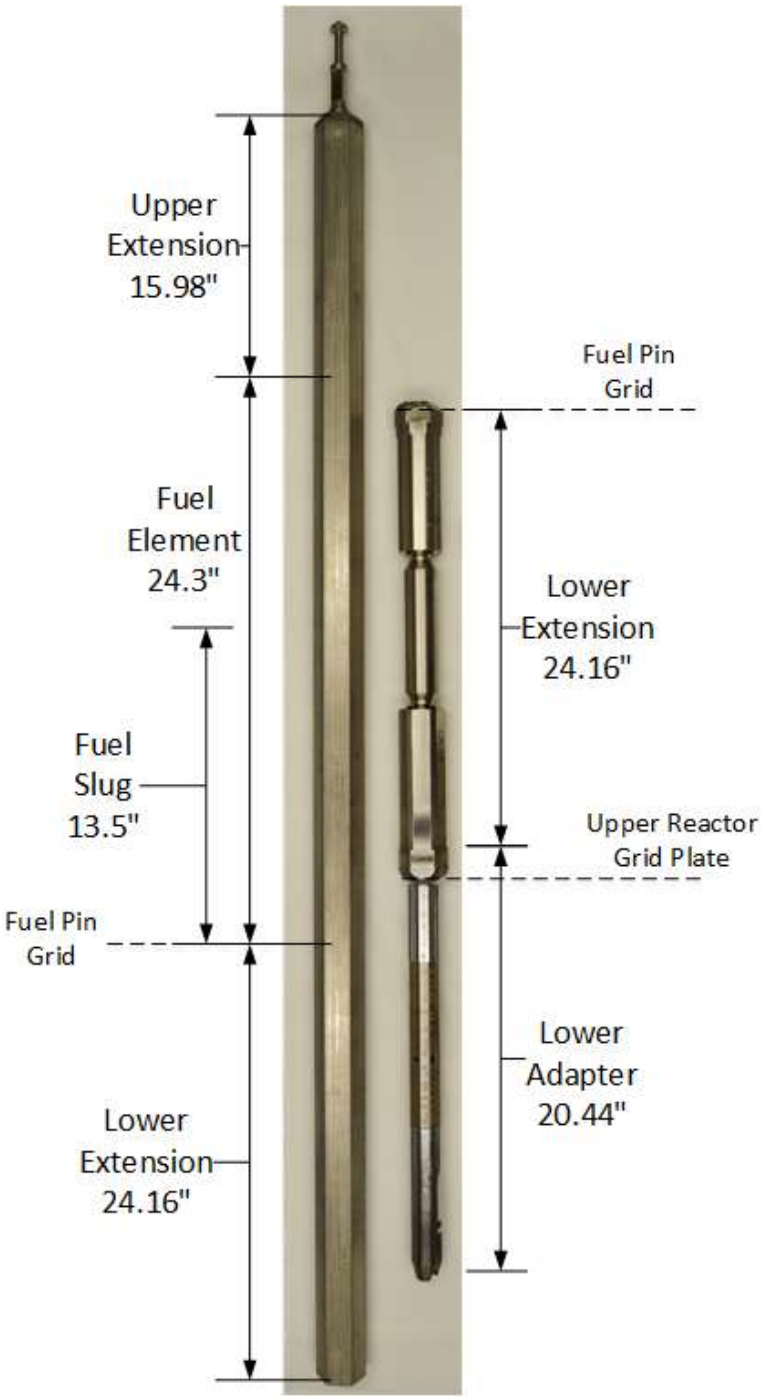

Figure 4. Annotated EBR-II driver subassembly.

2.0 DIFFICULTIES OF THERMAL EXPANSION MODELING

2.1 Reactivity Coefficients

EBR-II contained a multitude of reactivity effects. Table 3 shows a list of the reactivity coefficients calculated by Argonne National Laboratory run 138B. [2] They constitute the primary reactivity effects. Many of these effects occur in the first 6 rows of the EBR-II core. (The definition of a row is in appendix C.1.) These 6 rows are where the enriched uranium was contained and subsequently a majority of the power was generated.

Table 3. Calculated Reactivity Feedback Components and Uncertainties for Run 138B

Components	Nominal Reactivity Feedback Coefficient		Power Reactivity Decrement (PRD) ($)	Estimated Uncertainty (%)
	Contributing Region (rows)	Values used in the analysis (10^{-4} \$/^{O}C)		
Driver-Fuel Expansion	1-6	-4.91	-0.0687	11
Driver Sodium and Steel Expansion				
Sodium	1-6	-11.9	-0.0812	7
Steel	1-6	-2.2	-0.0155	10
Total	1-6	-14.1	-0.0967	7
Axial Upper Reflector Sodium Expansion	1-6	-6.32	-0.0667	7
Radial Reflector Sodium Expansion	7-10	-2.2	-0.0104	-
Doppler Effect	1-6	-0.671	-0.0109	10
Control-rod Bank Extension	5	-7.16	-0.0672	5
Upper Grid Plate Expansion	1-6	-14.5	0	20
Thermal Bowing	2-10	+9.8	+0.025	25
Axial Lower Reflector Sodium Expansion	1-6	-6.32	0	7

2.1.1 Sodium Coolant

EBR-II was a fast reactor that used liquid metal sodium as its primary coolant. Sodium has a high heat capacity, high thermal conduction, and low melting point. It was

for these reasons that it was chosen as the primary coolant. Sodium is not entirely invisible to fast neutrons and has a slight moderating effect. [3] The consequence of this is the sodium density has a significant impact on the reactor physics of EBR-II. More specifically, sodium-thermal-expansion is the second highest negative reactivity effect.

Sodium reactivity coefficients are broken down by general parts within the core. Driver sodium was located around the fuel elements. For a typical driver this region was ~63 cm in height, fully encapsulating the fuel elements.

Axial upper reflector sodium was within the upper extension of the fueled subassemblies. Radial reflector sodium was located inside of the core region of the reflector subassemblies within rows 7 through 10. Axial lower reflector sodium was in the lower extension of the fueled subassemblies.

The axial reflector sodium and driver sodium constituted all the sodium within the core region that had a significant impact on reactivity. Figure 5 shows the general regions of the reactor core.

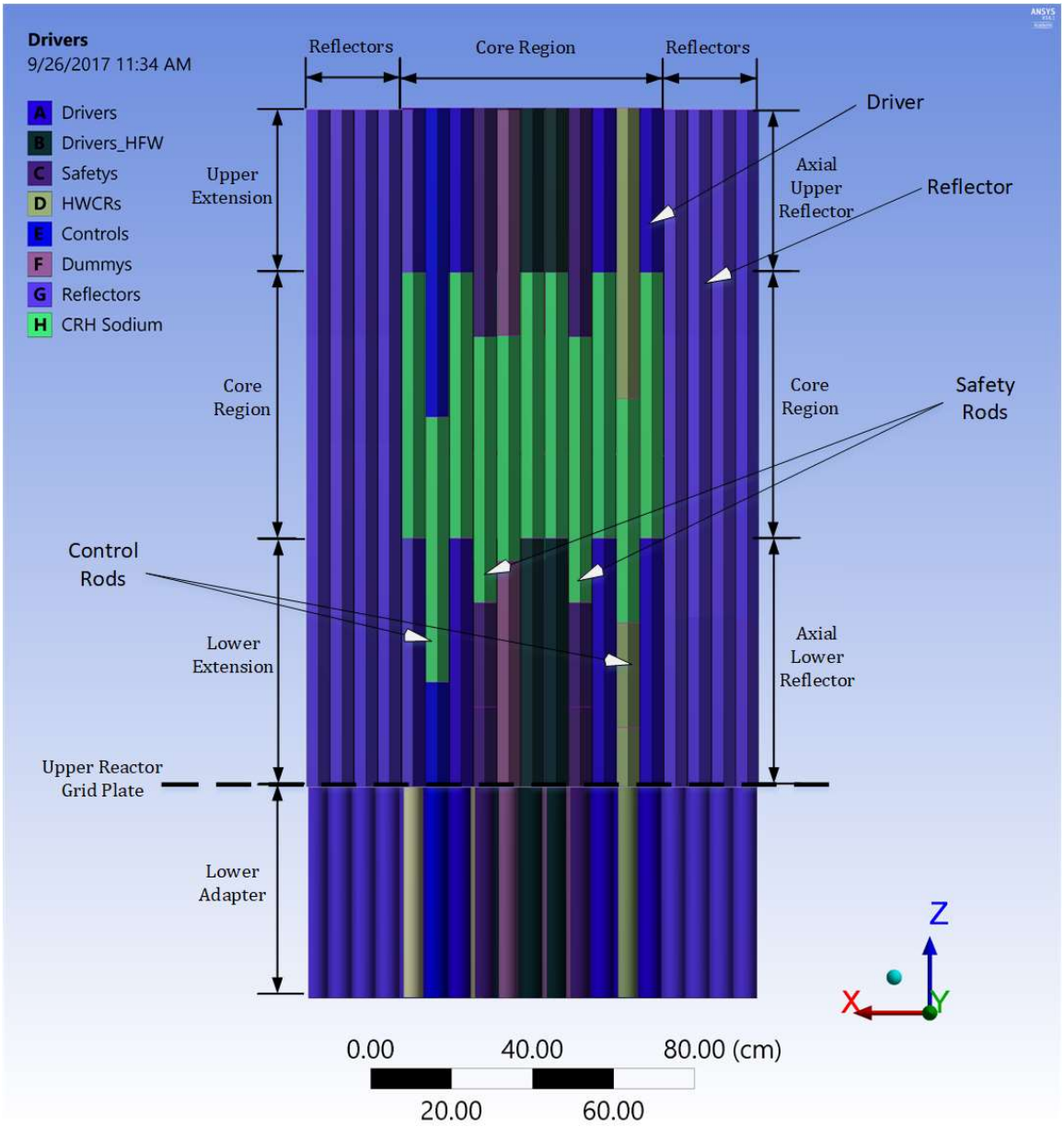

Figure 5. CAD figure of EBR-II core (elevation view).

2.1.2 Internal structures

The most significant reactivity effect was the thermal expansion of the structures of EBR-II. These structures were made of SS304L, while some of the subassemblies were constructed of SS316. The difference in thermal expansion is negligible considering how few subassemblies are SS316.

The steel expansion within the driver subassemblies was a small but contributing reactivity coefficient. It consisted of the steel within the subassembly, primarily the small pitch change that occurred when the duct and fuel element grid expanded. The entire subassembly also changed length, but this was an insignificant effect because everything below the core region had an even temperature. The even temperature was due to the sodium inlet temperature keeping the structures at the bulk temperature. Expansion did occur during the initial heating from room temperature to zero power, but this expansion was even across the entire core.

Control-rod bank extension was the thermal expansion of the control rod drive mechanism. The high worth control and control subassemblies had a drive mechanism above the core with a rod that attached to the top of the assembly. This mechanism would lift the core region of the subassembly into the core. As the rod thermally expanded, it would push the subassembly fuel regions out of the core. In the case of high worth control subassemblies, the expansion would insert poison from the top.

Upper grid plate expansion was the most significant reactivity coefficient. The upper part of the reactor grid plate was the stainless-steel grid where the subassemblies were attached. The main reason it had such a significant impact on reactivity was that it effectively changed the pitch of the subassemblies. Subassemblies were only mechanically restrained at the lower adapter which locked into the grid. The subassemblies rested against the side of the reactor liner and were free at the upper pole piece.

2.1.3 Other Reactivity Effects

Doppler effect is the change in cross sections due to temperature. While it was quantifiable, it was insignificant compared to the other reactivity effects in the fast spectrum.

Driver fuel expansion was the thermal expansion of the fuel slugs inside of the fuel elements. This effect was also minor because most of the fuel was burned and had undergone flux-induced swelling.

2.2 Katana Effect History

The katana effect is more commonly known as thermal-bowing. Bowing, however, is an inaccurate description of the geometry change. Bowing would be accurate assuming no axial change in temperature. Any reactor that utilizes axial coolant flow will have a change in temperature in the axial direction. This change will also be positive in the direction of flow. Figure 6 is a representation of a katana compared to a duct that has undergone deformation. The shape is suggestive of a katana and not a bow. Bowing implies axial symmetry which in most instances is not indicative of the thermally deformed duct.

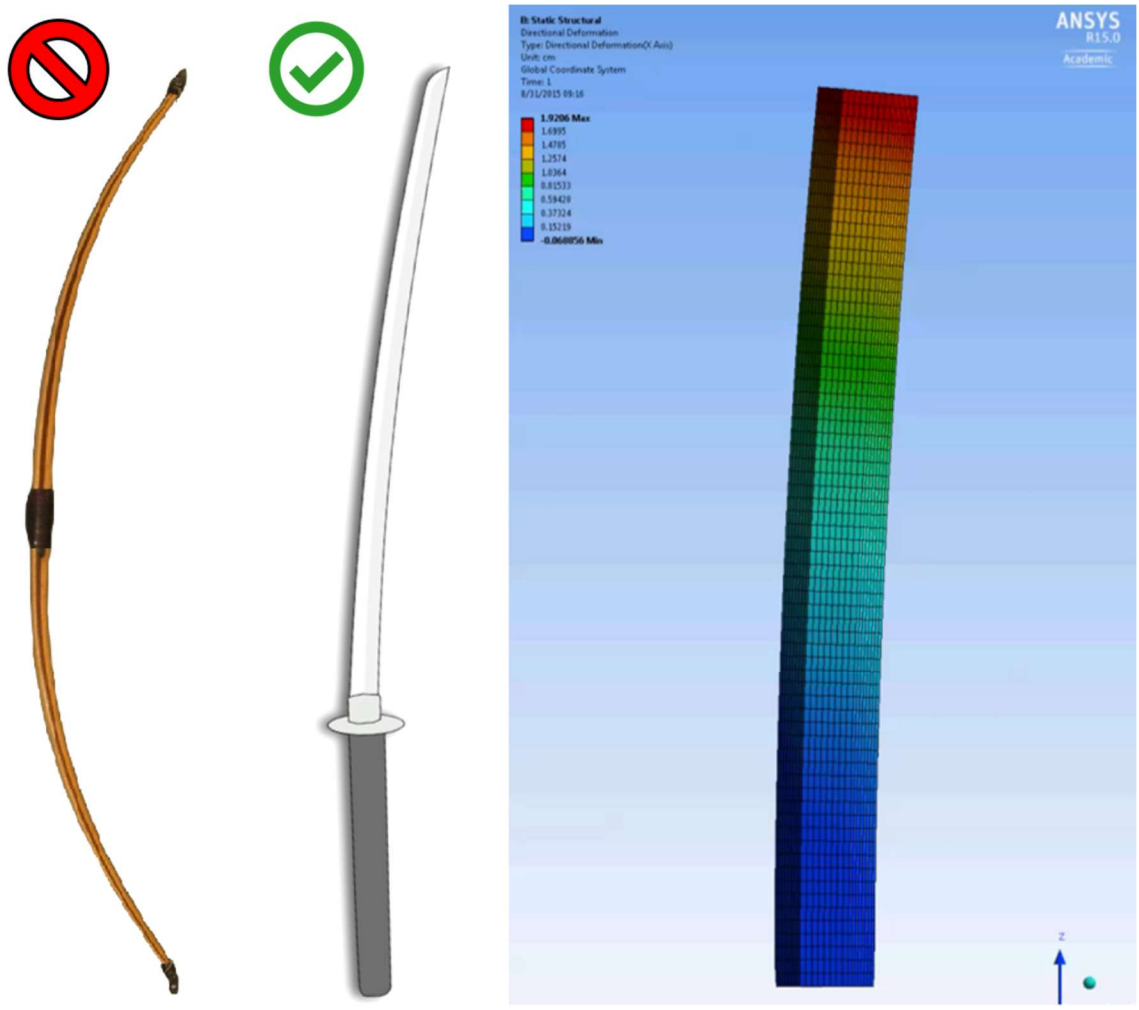

Figure 6. Katana vs. bowed duct.

EBR-II had axial cooling flowing from the bottom of the core to the top, leading to cold inlet temperatures at the bottom and hot outlet temperatures at the top. Quantifying the reactivity effect of this phenomena is the focus of this dissertation.

The katana effect has existed since the original construction of EBR-I. Experience from EBR-I "demonstrated that mechanical bowing of fuel elements toward the center of the reactor caused by the power gradient across the reactor produced a positive power coefficient." [4] The reason for the inward bowing of the elements was due to the upper assembly restraint. The bowing was not significant enough to cause major problems from a safety perspective, but it led to a concern for the research and design of EBR-II. The power produced from EBR-II was larger than EBR-I, leading to concerns that bowing would have a much greater significance as a positive reactivity coefficient. These concerns were primarily due to how the subassemblies would be restrained. No mechanical restraint could be used above the core midplane due to thermal expansion effects which led to the top of the core expanding more than the bottom. The difference in expansion is called core flowering. Core flowering would lead to a core configuration that would pinch the fueled subassemblies using the liner wall as a cantilever. The katana effect is further complicated by the grid plate which has a different thermal expansion regime that is independent of reactor power and the subassemblies. The grid plate would expand due to the difference in room temperature versus sodium coolant temperature. This complicates the katana effect because the grid plate determines the pitch of the subassemblies and remains at bulk sodium temperature for power-up. The mechanical restraints and subassembly geometry had to accommodate grid plate expansion, core flowering, and removeable subassemblies.

The primary solution was to introduce a clearance between the subassemblies using small buttons. These buttons produced a radial restraint right above the core midplane. This meant the subassemblies were restrained at the grid plate and the core midplane. The bow cantilever would only allow the top of the core to expand outward leading to a mostly negative reactivity. This solution did not completely mitigate the positive reactivity due to bowing, but it did reduce the powerband where bowing caused a positive reactivity. This reduction lead to an overall negative reactivity effect.

During the initial physics designs of EBR-II, bowing was included in the physics designs but was not fully quantified. “The influence of fuel element and subassembly bowing is not included in the [table 4] result.” [5]

Table 4. Components of Temperature and Power Coefficient of Reactivity for EBR-II

Core	Predicted* $((\Delta k/k)/\ ^oC)$	Inferred from ZPR-II Measurements $((\Delta k/k)/\ ^oC)$
Axial growth of fuel	$-0.39x10^{-5}$	$-0.34 \pm 0.02x10^{-5}$
Radial growth of fuel (displacement of Na)	$-0.09x10^{-5}$	$-0.057x10^{-5}$
Axial growth of structure (density change)	$-0.039x10^{-5}$	$-0.033x10^{-5}$
Density change of coolant	$-0.87x10^{-5}$	$-0.98x10^{-5}$
Radial growth of support structure	$-0.97x10^{-5}$	$-0.92x10^{-5}$
Doppler Effect	$+0.04x10^{-5}$average	--
Bowing	--	--
Gaps		
Density change of coolant	$-0.38x10^{-5}$	$-0.33x10^{-5}$
Density change of structure	$-0.036x10^{-5}$	$-0.04x10^{-5}$
Upper and lower blanket		
Density change of coolant	$-0.21x10^{-5}$	$-0.21x10^{-5}$
Radial growth of uranium jacket	$-0.016x10^{-5}$	
Axial growth of blanket	$-0.024x10^{-5}$	
Axial growth of jacket	$-0.021x10^{-5}$	$-0.0064x10^{-5}$
Inner blanket		
Density change of coolant	$-0.2x10^{-5}$	$-0.30x10^{-5}$
Axial growth of blanket uranium	$-0.066x10^{-5}$	--
Axial growth of jacket	$-0.022x10^{-5}$	$-0.054x10^{-5}$
Radial growth of uranium and jacket	$-0.07x10^{-5}$	
Radial growth of supporting structure	$-0.17x10^{-5}$	
Bowing	0	
Outer blanket		
Density change of coolant	$-0.017x10^{-5}$	$-0.011x10^{-5}$
Axial growth of blanket uranium	$-0.014x10^{-5}$	
Axial growth of jacket	$-0.003x10^{-5}$	$-0.0012x10^{-5}$
Radial growth of supporting structure	$-0.034x10^{-5}$	

During these initial design stages, the katana effect calculation methods had high uncertainty. It was accounted for in the safety analysis by taking the "most pessimistic

course" [5]. These analyses were taken from experience with EBR-I and were utilized for the initial startup.

Analysis of runs 1 through 24 showed that the power coefficient data were "reasonably linear and strongly negative." [3] This matched calculated predictions and showed that the "pessimistic" approximation could reasonably bound the uncertainty due to the katana effect. The general core configuration for runs 1 through 24 consisted of five rows of drivers and then eleven rows of blankets. For run 25, the configuration was changed by replacing rows seven and eight with stainless-steel reflector subassemblies. Row six contained a mix of drivers and reflectors. Changing these rows to reflectors caused a significant change in the power reactivity decrement. The reduction in power reactivity decrement was half of what was predicted for run 24. The change was observed to start around 10 MWth and continue all the way to full power. Kinetics experiments revealed that the difference in reactivity was not caused by any prompt effects and that it was a delayed effect that was driving the change.

The delayed effects were radial core expansion, subassembly bowing, and control rod extension thermal expansion. A series of experiments followed that made attempts at isolating each delayed effect to discover if it was the cause of the positive reactivity seen in run 24. While each had some small contribution, it was shown through extensive analysis that bowing was the cause of the large positive reactivity.

A series of heat transfer calculations lead to the conclusion that an inverted temperature gradient existed from the boundary between the blankets and reflectors. [3] Power was being generated in the first row of blankets. This lead to the cold side of the

duct to be facing the reflector and subsequently causing a bow towards the reflectors. Figure 7 shows inward blanket bowing.

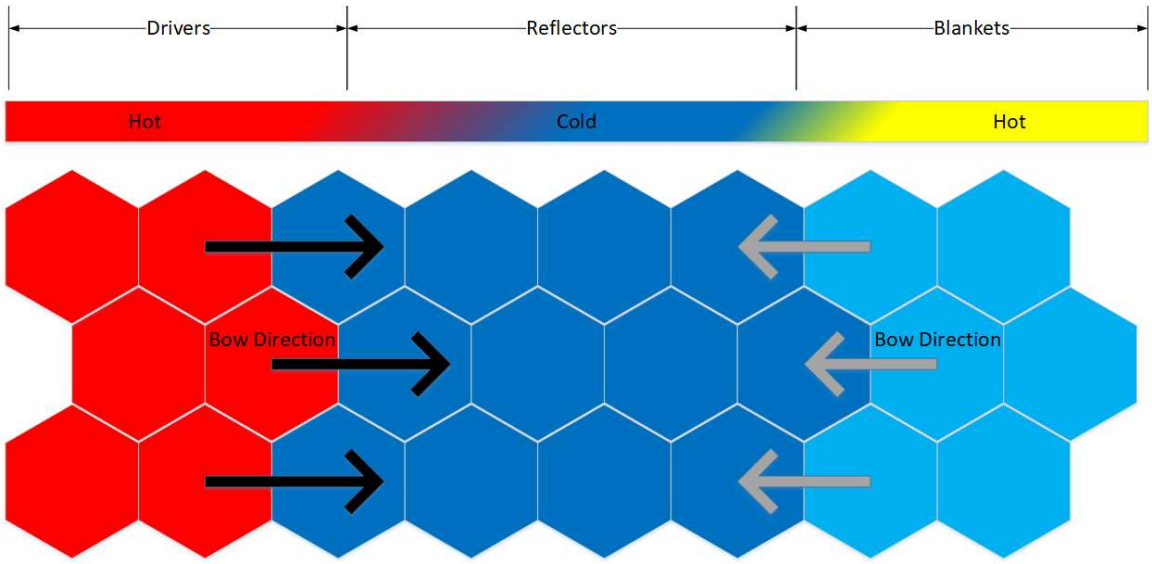

Figure 7. Inverted bow.

2.3 Old Methods of Katana Effect Value Determination

Multiple codes were created to model the katana effect during the time of EBR-II. No matter the code, the same underlying problems existed. These problems were related to the quality of the input data for calculating the coefficient; specifically, the mechanical models used to calculate the displacement data. The following quote is from work done on a code called RHOBOW about the current level of accuracy when calculating duct-bowing.

"Assembly displacements: a rather complex chain of thermo-hydraulic and mechanical computations is used for estimating assembly displacements during a transient. These phenomena are extremely hard to compute accurately, due to the structural complexity of large modern [liquid metal fast breeder reactors]. The final accuracy of these computations is probably no better than 30%." [6]

A further quote from that same report in the conclusions section:

"It seems now acceptable to state that the accuracy of assembly bowing feedback coefficients is not limited by the scheme which is used to compute them, but rather by the quality of the information provided to that scheme." [6]

What the author was stating in the previous quote was that the nuclear calculation methods were not the primary cause of the uncertainty, but that the mechanical interactions used as an input were the problem. The primary code used to calculate both the reactivity change and the displacement data was a code called BOW-V. [7]

BOW-V used a radially symmetric model where subassemblies were assigned rows. The net displacement was calculated for each row and then was applied as a force to the other rows. The main advantage of this program was that it could calculate multiple row interactions, not just the adjacent rows. For example, the movement of row 8 could affect the displacement of row 5. The other main advantage was the ability to model multiple axial nodes. This was important because within one subassembly existed multiple pivot points which affect the movement of the subassembly. A representation of the BOW-V model of EBR-II is shown in figure 8 and figure 9.

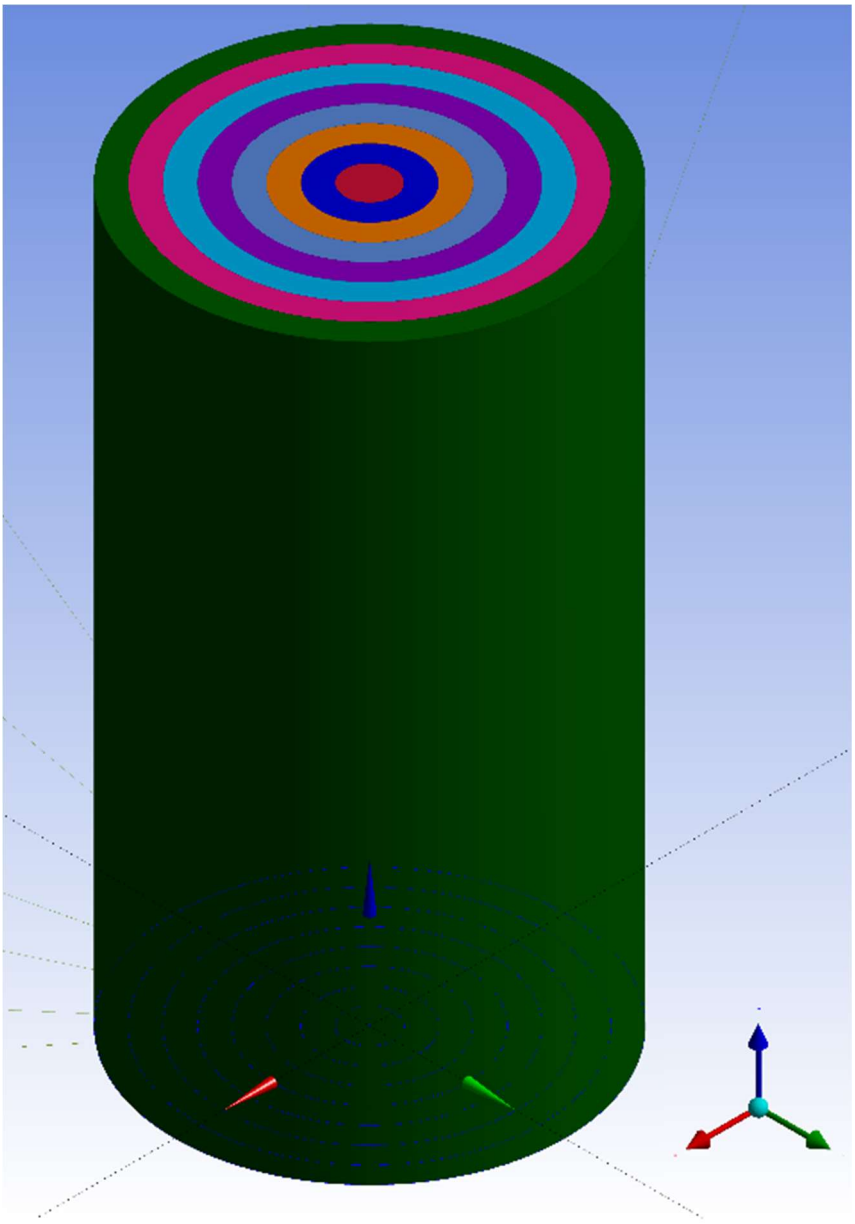

Figure 8. BOW-V model of EBR-II core with colors defining rows.

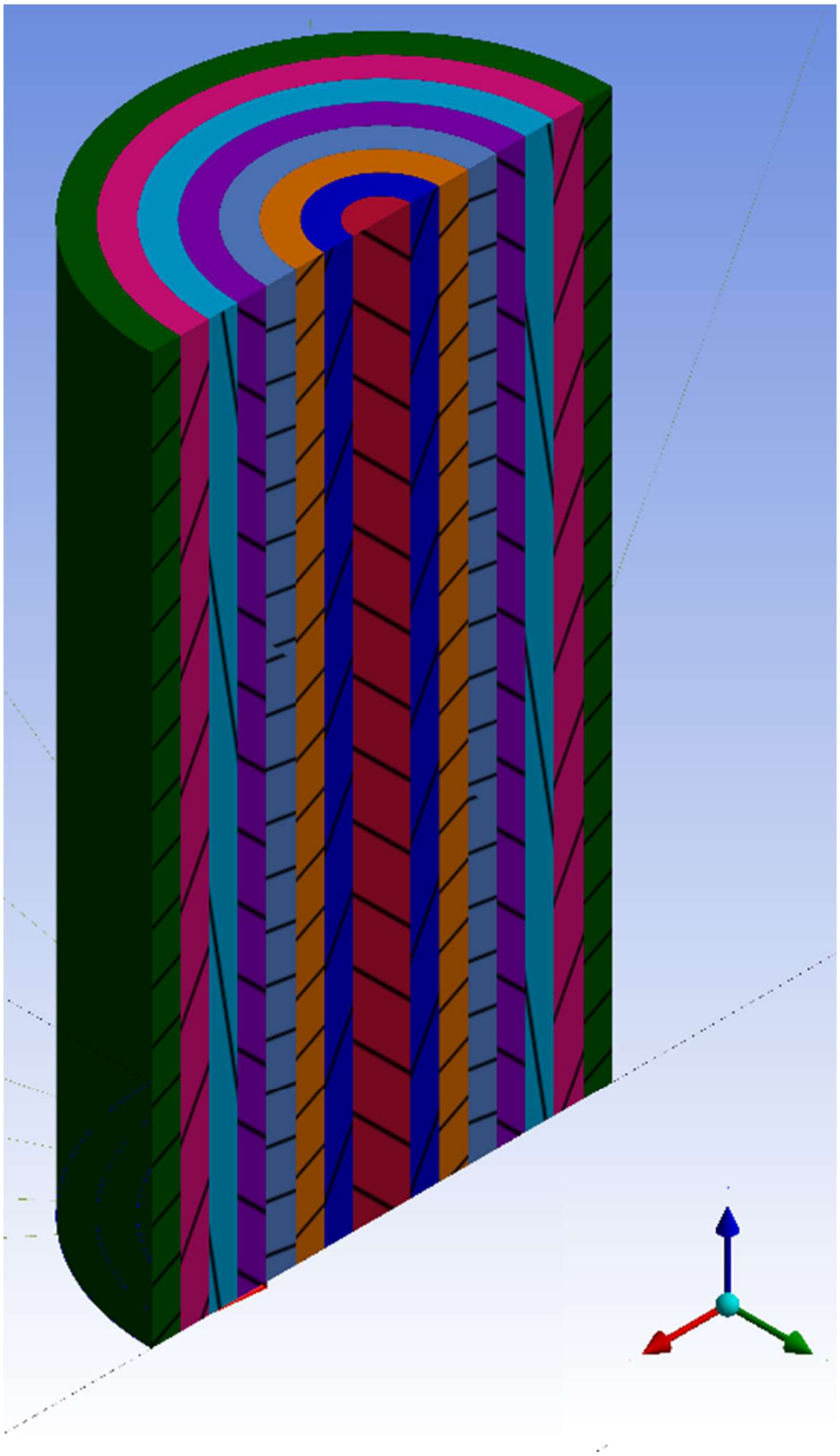

Figure 9. BOW-V model of EBR-II core cutaway with colors defining rows.

The primary disadvantage of BOW-V was the smearing of multiple subassemblies into one row that acted on the adjacent row. The smearing technique is only accurate when the core is radially symmetric. Run 138B core was highly non-symmetric. Dummy subassemblies were used in multiple locations leading to a temperature drop at those locations. The temperature drops caused a deformation different than a power producing subassembly. The non-typical bowing had a greater impact on the net displacement than a typical deformed duct because it was not moving radially out from the center of the core. BOW-V averaged all the deformations losing the greater impact of the non-typical deformations. BOW-V was updated multiple times to include more axial nodes, but the underlying problem was never resolved. Figure 10 shows the BOW-V method rows smearing compared to the heterogeneity of EBR-II run 138B configuration.

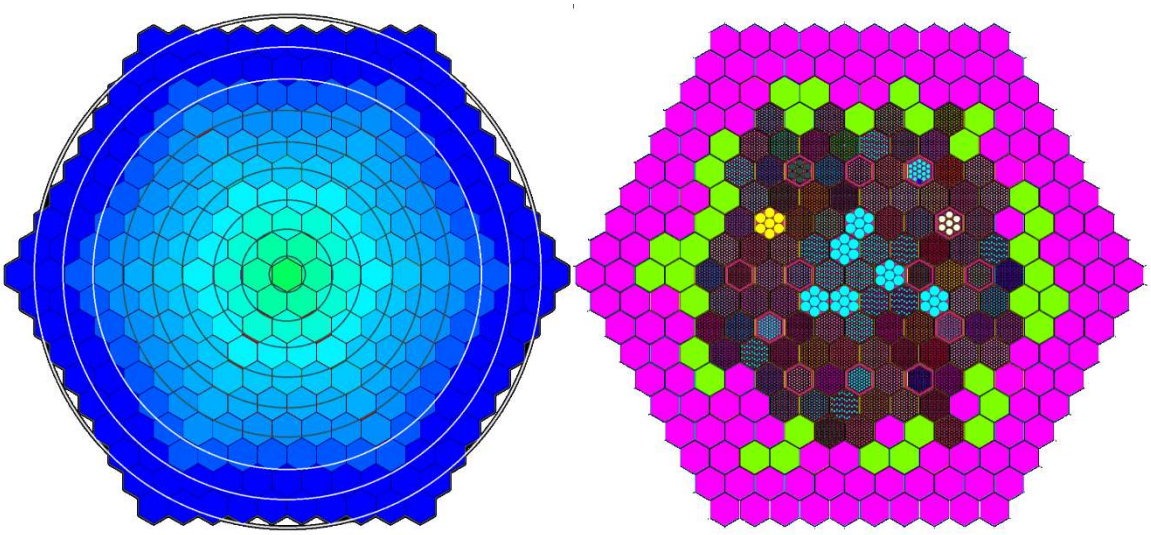

Figure 10. BOW-V model versus detailed model.

2.4 Katana Effect Complexity

The katana effect is a complex effect to model. Generally, it is a problem of specificity in temperature and displacement. The requirements are precise temperature profile, highly detailed individual bowed duct, and integrated effect of duct-to-duct mechanical interactions. These requirements lead to a very large model that also has very fine details. Without modeling each of the individual effects, the integrated effect will

result in a large-scale propagation of errors. This is one of the reasons for the high uncertainty estimate.

2.4.1 Well-known Temperature Profile

The katana effect is driven by thermal expansion of the ducts, specifically the thermal expansion due to the temperature gradients across the ducts. The gradients are not radially symmetric, nor are they axially symmetric across the duct. The integrated effect of the temperature gradients leads to an overall bowed duct. A simple linear or polynomial ΔT is insufficient to properly model the temperature profile. This complicates the modeling of the katana effect because a precise temperature profile of the core is required. A precise temperature model is difficult to acquire since in most cases it requires a full heat transfer and thermal hydraulic model. These temperature models would have to output a nearly continuous function of temperature across an entire core which even for simple systems can be difficult.

2.4.2 Individual Duct Movement

The main complexity from the katana effect is the individual movements of the ducts. Each duct bows in response to the thermal gradient across the duct and the movements of the surrounding ducts. Due to EBR-II's primary mission being a research reactor, the core was designed to have each subassembly be easily removeable and replaced with a different subassembly. This led to a highly heterogeneous core in burnup, type, and power. Efforts were made to keep the flux profile flat and symmetric but power generation for each subassembly was different. No one subassembly was surrounded by the same configuration as another, leading to different temperature gradients across every duct.

Mechanically isolating the ducts would still lead to independent movement due to the specific temperature gradients. The independent movement requires modeling the ducts individually and not homogenizing sets of ducts. For EBR-II, this requires modeling at minimum the core region of 169 subassemblies, dramatically increasing model complexity.

2.4.3 Duct-to-duct Interaction

EBR-II's subassemblies were designed with a 0.002 in gap between the buttons of neighboring ducts and the buttons were 0.014 in tall, leading to a total distance between the ducts of 0.03 in. This meant that some small amount of bowing could occur without any significant interaction between the ducts. The core at low-power does not exhibit a significant flowering effect, meaning on average the overall position of the fuel regions of the ducts has only moved inward toward the center of the core. This is the primary reason that at low-power there is a positive reactivity coefficient.

As the reactor power increases, the clearances between the ducts disappears leading to duct-to-duct interaction. The individual duct movement is a result of thermal expansion and net stress produced from the six surrounding ducts. The core does have an overall "flowered" effect, but because of the duct-to-duct interactions, the duct movement is not symmetric or consistent. Treating the ducts in mechanical isolation is not correct given the duct-to-duct interactions. Addressing these interactions requires a model that can resolve net stress and displacement between the ducts. Section 2.2 demonstrated the problem that can be caused by an inverted temperature gradient. Row nine bowing in the opposite direction changed the bowing behavior of the entire inner core proving the need for modeling duct-to-duct mechanical interactions.

2.4.4 Large and Small Details

EBR-II subassemblies contain large structures and small details. The katana effect requires modeling a significant portion of the core leading to a very large model. The complexity is found in the interaction areas between the ducts. The buttons between the subassemblies have a very small interaction area which is required to be modeled with high fidelity to resolve the forces. As bowing increases, the top edge of the duct encounters a neighboring duct; the interaction area of the edge is even smaller compared to the button area. This leads to the modeling requirement that sufficient detail be retained to model the duct-to-duct interactions and that the model must be large enough to encompass the ducts.

3.0 FINITE ELEMENT ANALYSIS COUPLED WITH MONTE CARLO NEUTRON TRANSPORT

3.1 Monte Carlo, Finite Element Analysis Coupling

Sections 2.4.1, 2.4.2, 2.4.3, and 2.4.4 detailed modeling requirements to capture information necessary to model the katana effect. It has also been shown that the mechanical analysis and the neutron transport will both need methods that allow for very complex geometry.

Finite element analysis (FEA) is a method sufficient to retain the details and perform the necessary geometry deformations. The primary problem with using FEA is the scale of the problem can grow to a point where reasonable computing power will be insufficient to retain the detail. Careful iteration on the mesh specifications is required to balance mesh size against mesh quality. FEA is used to model the thermo-mechanical interactions. It is a two-part process where the heat transfer model generates the time dependent temperature everywhere in the geometry. These data are then transferred to the structural analysis using the same timescale where the temperature is used as an input and the output is structural displacement.

The code chosen for modeling the thermal and structural components of the katana effect was ANSYS. ANSYS is a suite of engineering simulation tools used to analyze multiphysics effects on structures, fluids, semiconductors, etc. The simulation tools utilize finite element analysis to solve large- and small-scale mechanical engineering problems. ANSYS is one of a handful of publicly available codes that has the capabilities to model the katana effect and this is the reason that it was selected as the mechanical analysis code.

Monte Carlo based neutron transport currently is a method that has the geometry capabilities commensurate with FEA. The primary problem with the neutron transport simulation is not the scale but the geometry limitations that exist in current codes. FEA has a more mature geometry capability due to the geometry being defined as an unstructured mesh. There simply have been more applications, and hence more users, of mechanical engineering tools than nuclear engineering. Nuclear engineering tools in general do not

have the same level of sophistication in complex geometry. While Monte Carlo methods have begun to adopt unstructured mesh-type geometries, they are still simple in comparison to FEA unstructured mesh capabilities. In place of unstructured mesh, the EBR-II neutron transport model receives the averaged displacement data calculated from FEA. Stochastic transport is performed through the deformed geometry and effective multiplication is calculated.

Monte Carlo N Particle (MCNP) was chosen as the neutron transport code. MCNP has an ability to model complex geometries like what is required to model the katana effect. MCNP also has an extensive history of verification and has been used to model small geometry changes.

Monte Carlo neutron transport and FEA are the best tools to model the complex geometry of the katana effect, however, some simplifications will still be required. Specific details on the analysis flow are available in section 5.2.

3.2 Finite Element Analysis General Theory

3.2.1 An Element

FEA is an analytical technique where a computer aided design (CAD) based geometry is meshed. The meshing process, which is not described, divides the geometry into small parts called elements. These elements are standard shapes like tetrahedrons or cuboids that are combined to create complex geometries. An element's purpose is to be a building block for a geometry and a boundary for a function describing a phenomenon. For example, an element can describe the change in temperature from one side of the element to another. Elements can be 1-dimensional, 2-dimensional, or 3-dimensional depending on the user's needs.

3.2.2 A Node

An element is comprised of nodes. For simple elements, the nodes are only located at the vertices of the element. For complex elements, nodes appear on the vertices and at the midpoint between the nodes. Figure 11 is an example of a 10-node tetrahedron element. Nodes R, P, Q, O, M, and N are the midpoint nodes called mid-side nodes. L, I, J, and K are vertex nodes.

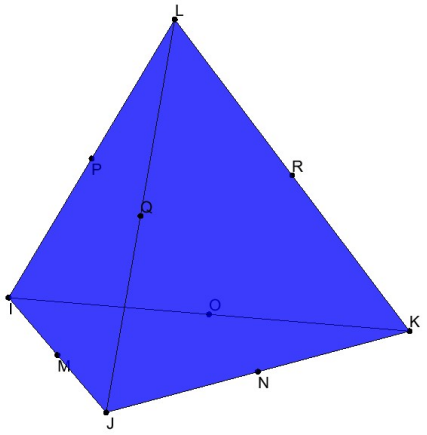

Figure 11. 10-node tetrahedron element.

Figure 11 is drawn with the lengths between the nodes being the same and with the mid-sided nodes in-line with the vertex nodes. The lengths of the sides can vary depending on where the element is located. Variable lengths between nodes and mid-side node position change to accommodate the geometry. A high-quality element is an element where the lengths between nodes are approximately the same and where the mid-side node positions are roughly in-line with the vertex nodes. High quality means that when the element is used to simulate the phenomena, the solution can be easily found. A node also defines a point where a phenomenon is quantified. For example, node J from figure 11 could have a solution of T_1 and node N could be T_2. Quantified solution values only exist on nodes.

3.2.3 Shape Function

The lengths between the nodes serve two purposes. The first purpose is to describe the physical distance between the nodes, and when meshed, describes the node position relative to the entire geometry. The node that is positioned at the origin matches the origin of the CAD geometry. The second purpose is to describe the function between the nodes that determines how a phenomenon changes. The function is called the shape function. The shape function is then operated on by the unknown variable, yielding a shape function that describes how the variable changes over the element. Continuing with the example, nodes

J and N have a shape function that describes how the temperature changed between the nodes, leading to the solutions of T_1 and T_2, respectively. Equation 1 is an example of a complete element shape function for temperature of figure 11. L_1 through L_4 represent the length between nodes and T_I through T_R represent the node temperatures.

$$\begin{aligned} T = {} & T_I(2L_1 - 1)L_1 + T_J(2L_2 - 1)L_2 + T_k(2L_3 - 1)L_3 \\ & + T_L(2L_4 - 1)L_4 + 4T_M L_1 L_2 + T_N L_2 L_3 + T_O L_1 L_3 \\ & + T_P L_1 L_4 + T_Q L_2 L_4 + T_R L_3 L_4 \end{aligned} \quad (1)$$

3.2.4 Global Matrices

A global matrix is constructed using the shape functions. Each element of the matrix represents a node. Each column and row represent how one node is connected to the other nodes in the same element. Consideration is taken for nodes that are shared between elements. One node might be shared between two elements, which means that node has connections to all nodes in both elements. Further details about the mathematics of FEA can be found in Moaveni. [8]

The global matrix is then used to obtain the global matrices of physics effects. For thermal analysis, these effects are element specific heat, element mass transport, element diffusion conductivity, element convection surface conductivity, element mass flux, element convection surface heat flow, and element heat generation. Further detail on the derivations of these matrices is available in appendix E.1. Equation 2 is the general heat flow equation for an element. The variables are defined in appendix E.2.2.

$$[C_e^t]\{\dot{T}_e\} + ([K_e^{tm}] + [K_e^{tb}] + [K_e^{tc}])\{T_e\} = \{Q_e^f\} + \{Q_e^c\} + \{Q_e^g\} \quad (2)$$

The global heat flow equation is created by a summation of all the element effects. For example, global conductance matrix K is the summation of all element conductivity. Equation 3 is the global heat flow equation.

$$[C]\{\dot{T}\} + [K]\{T\} = \{Q^a\} \quad (3)$$

A similar process is followed for the structural element equation and global structural equation. The variables are defined in appendix E.4. Equation 4 is the structural element matrix and equation 5 is the global structural matrix. Equation 5 is the static formulation because all thermal expansion is assumed to be instantaneous and gravity is a constant. There are no time-dependent variables.

$$\left([K_e] + \left[K_e^f\right]\right)\{u\} - \{F_e^{th}\} = [M_e]\{\ddot{u}\} + \left\{F_e^{pr}\right\} + \{F_e^{nd}\} \tag{4}$$

$$[K]\{u\} = \{F^a\} \tag{5}$$

3.2.5 Solution Methods

Solving equations 3 and 5 requires performing an iterative method call Newton-Raphson method. An initial guess for the unknown vector of degree(s) of freedom (DOF) values is created. The vector would consist of the unknown quantities. For thermal analysis, the unknown is temperature (T). In structural analysis, the unknowns are 3-dimensional displacement (u,w,v). Using the assumed unknown vector u, K and the restoring load F^{nr} are calculated. The restoring load changes depending on the analysis type. Thermal analysis restoring load is the sum of inter-element heat flows. Structural analysis restoring load is the restoring force calculated from element stresses. K is the same as described in section 3.2.4. The applied load is also calculated from heat generation rates for thermal analysis or node force vectors for structural. The latter is a combination of thermal and mechanical stresses. Using the previous quantities, a formulation of the change in solution can be made. Equation 6 is the definition of solution change between iterations where i is the iteration number.

$$[K_i]\{\Delta u_i\} = \{F^a\} - \{F_i^{nr}\} \tag{6}$$

The right-hand side is the out-of-balance condition that exists between each iteration. When the difference between F^{nr} and F^a reaches some tolerance value, the solution u_i is considered converged and the next substep is calculated.

3.3 Monte Carlo Neutron Transport General Theory

Monte Carlo neutron transport is a method by which a stochastic approach is taken to simulating particle transport. This is also known as the "random walk" approach to particle tracking. Figure 12 is a representation of Monte Carlo particle transport in a critical system.

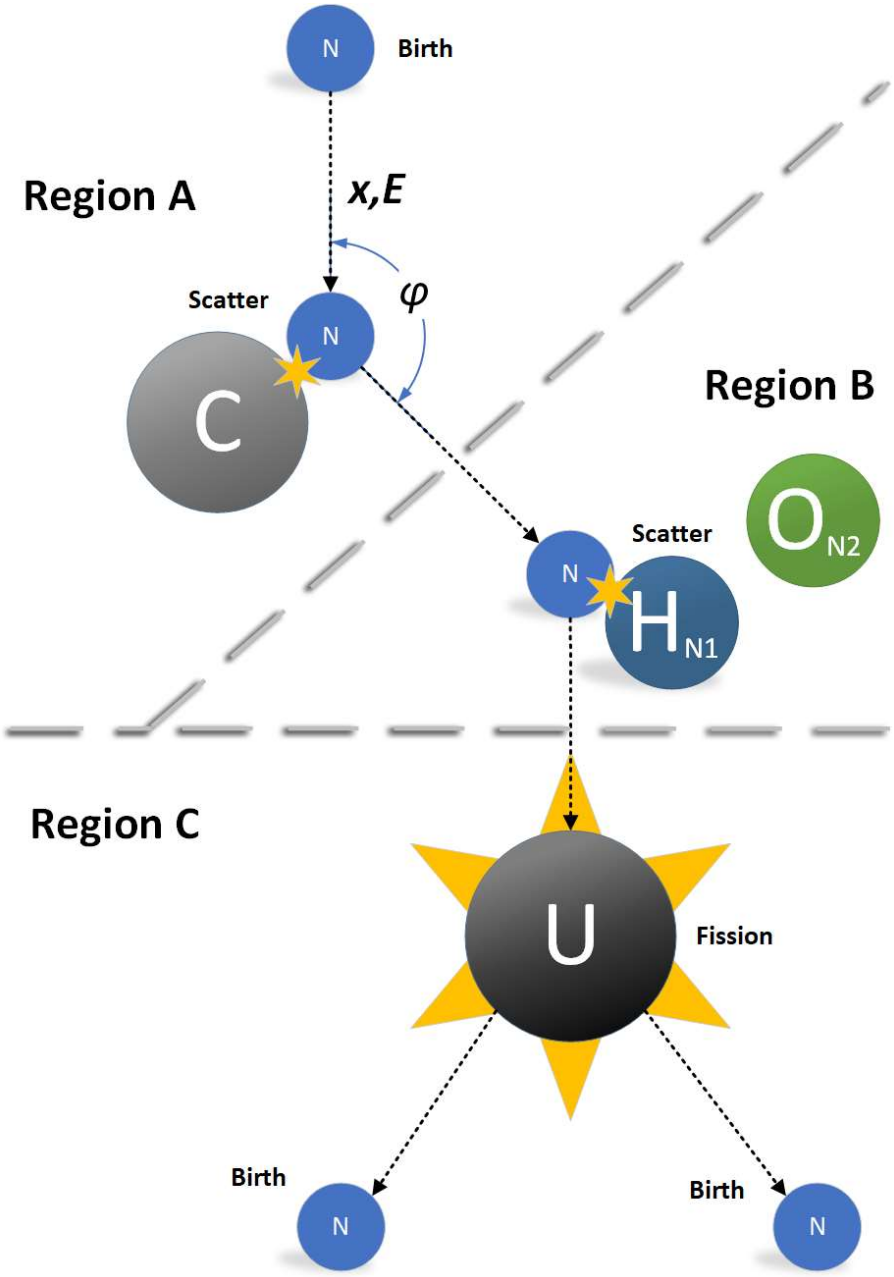

Figure 12. Random walk method of neutron tracking.

Particle tracking using a stochastic method requires numerous random numbers to be generated since all aspects of particle tracking are randomly sampled. Generating random numbers is impossible for a computer because an algorithm must be used to yield the quantity of random numbers required. The solution is to use a pseudo random number generator. A pseudo random number generator is an equation that can generate a string of random numbers to a specific amount, then repeat. The length of the string of random numbers without repetition is called the period. A good random number generator will have a period that is large enough to not repeat during the same problem. Most Monte Carlo codes have multiple types of generators, but the most common type is a linear congruential generator. [9] The random numbers are usually defined between 0 and 1. Using the random numbers, particle tracking can begin. The following generalized Monte Carlo particle transport theory uses figure 12 as a reference.

The starting location for the first generation of particles is usually supplied by the user. The particle is given a random direction and energy. The random components are defined by a limiting function. For example, the direction is an angle from 0 to 2π and the energy is a random selection from an energy distribution. For fission, the energy distribution is the Watt-fission spectrum. The distance a particle travels is determined by the total cross section for the material and the random sample of possible distance in that material. The next event to occur is to determine with what isotope the particle has collided. A normalized distribution of the isotopic cross sections for a material is created and a random number is used to determine which isotope in the distribution is selected.

Referring to the example in figure 12, after the birth of the neutron, the neutron collides with carbon because carbon is the only element that exists in region A. Another normalized distribution of cross sections using a random number for selection is performed to determine what happens at the collision. For carbon, it is either scatter or absorption. In the example, scatter occurs. A random direction is selected and a new distance is calculated.

Region B of the figure has two materials possible for neutron collision. Another normalized distribution of cross sections is done using random numbers and the neutron scatters from hydrogen. Region C contains uranium where a fission event occurs. The

fission event generates two new neutrons. If Monte Carlo transport is used to determine k-effective, the starting locations for the new neutrons are stored for the next generation. After a set number of particles have been birthed and tracked, the generation is considered complete. The next generation uses the starting locations stored from the previous generation and performs more tracking. Each generation determines the particle density and using the previous generation particle density, a new k-effective is calculated. This is known as the power iteration procedure. [10]

$$(L + T - S)\varphi^{(n+1)} = \frac{1}{K_{eff}^{n}} M\varphi^{n} \tag{7}$$

Where:

$L =$ leakage operator

$T =$ collision operator

$S =$ scatter-in operator

$M =$ fission multiplication operator

$\varphi^{n} =$ particle density of n generation

The generations continue until both k_{eff} and the particle density converge. Successive generations after convergence reduce the statistical uncertainty on k-effective and the particle density.

4.0 GODIVA-IV VERIFICATION

A verification was required to show that a FEA Monte Carlo coupling could yield satisfactory results. GODIVA-IV was chosen for the verification because it is a simple system where negative reactivity consists mostly of thermal expansion. The cylindrical configuration also is geometrically like the hexagonal lattice core of EBR-II.

GODIVA-IV was a critical neutron burst apparatus used to research fundamental nuclear physics. The system is designed to create super-critical bursts of neutrons that last from seconds to microseconds. Figure 13 is the ANSYS model representation of GODIVA-IV. The model has been extensively reviewed and verified to be as accurate as possible. This model was created from an MCNP geometry input file that was used for a criticality safety benchmark in the International Criticality Safety Benchmark Evaluation Project (ICSBEP) handbook. [11] The following steps will be nearly identical to what will be performed on the EBR-II model. The only difference is that GODIVA-IV's main negative reactivity component was the thermal expansion of the radius. The following demonstrates that the new method can be used to calculate a measured value.

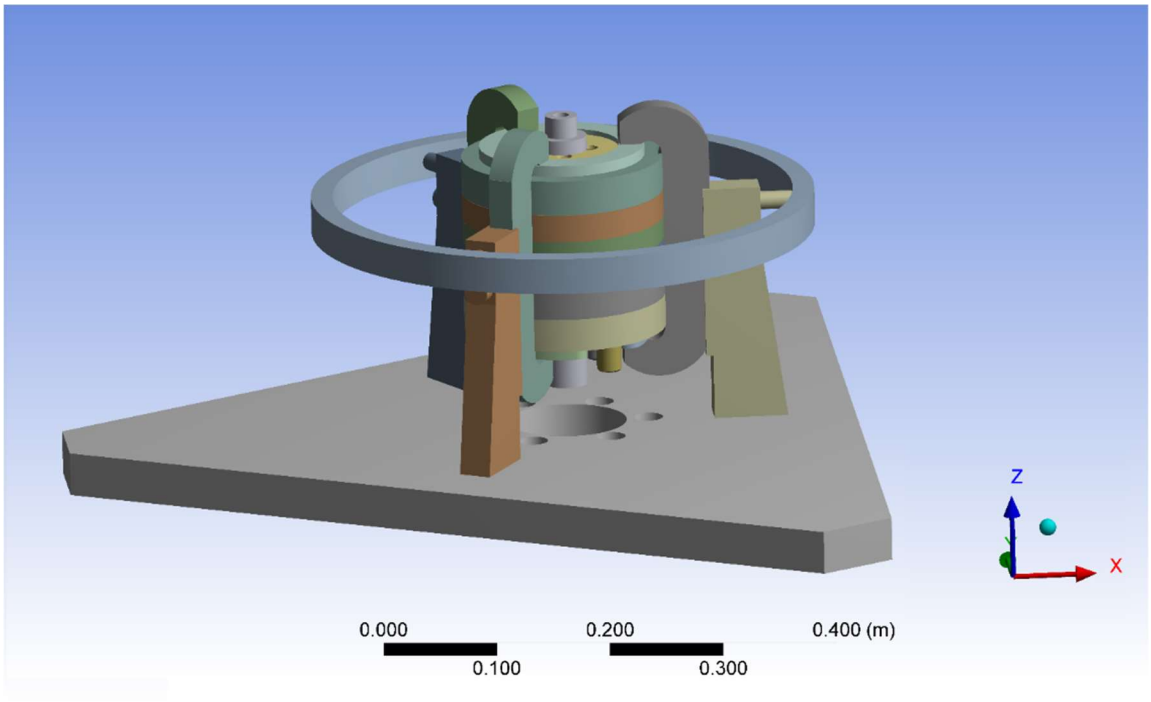

Figure 13. GODIVA-IV ANSYS model.

The procedure was the insertion of the safety block and the control rods until delayed critical was achieved. The safety block was then retracted and the burst rod was armed. The safety block was inserted back into GODIVA-IV and a pneumatic mechanism connected to the burst rod rapidly inserted it into the assembly causing a super-critical prompt burst to occur. [12] Figure 14 shows the two rod types used in GODIVA-IV. This burst led to a rapid increase in temperature and structural expansion. The burst that was modeled for the verification was a 1.029$ reactivity insertion that lead to a 68°C-average change in temperature of GODIVA-IV. The temperature was then measured by temperature probes located in fuel ring 104 and the safety block.

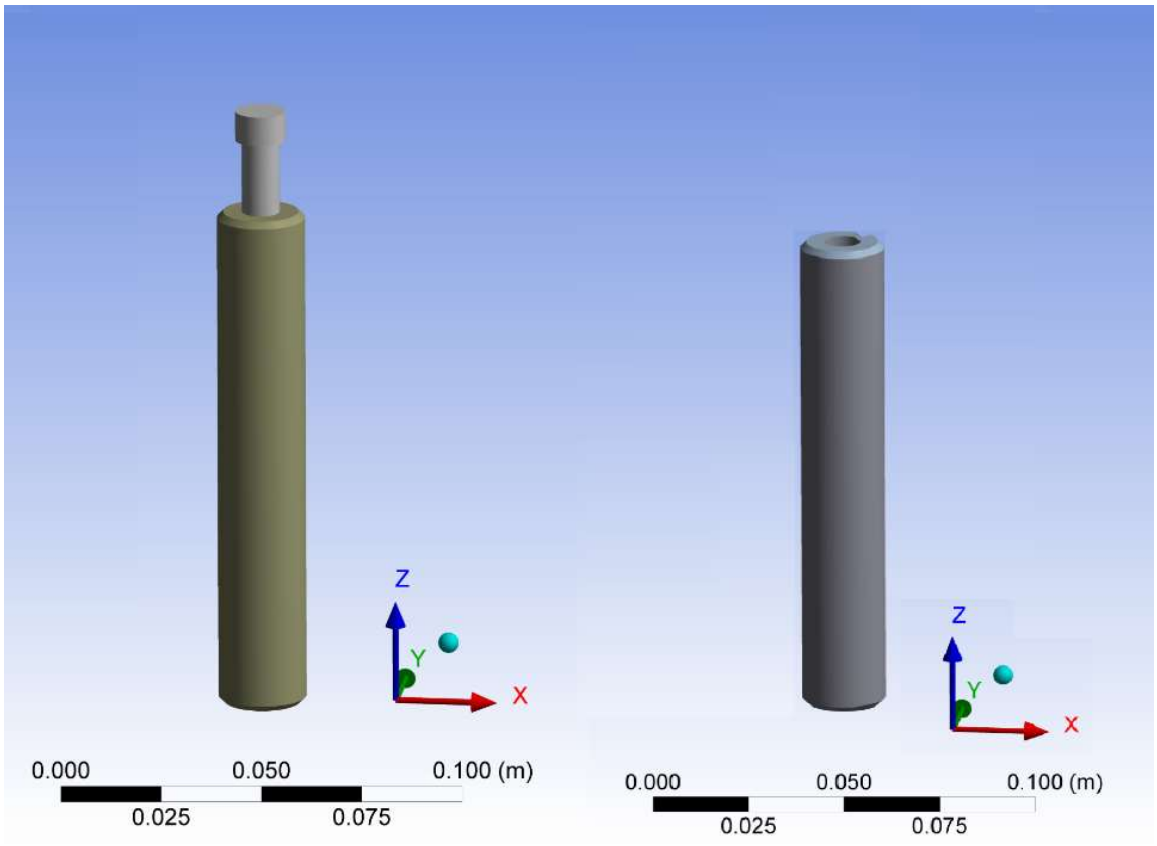

Figure 14. GODIVA-IV burst rod on the left and control rod on the right.

The ANSYS model uses these real data as an input into a transient thermal analysis model of GODIVA-IV. ANSYS then breaks the timeline into small time-steps. Each time-step sets the boundary conditions for the mesh matrix and solves that matrix to calculate temperature and thermal conduction. It iterates on this matrix until a converged solution is found. The next small time-step uses the previous solution as an input and then re-solves the matrix. ANSYS performs these iterations until a defined end; for the GODIVA-IV model, this is approximately 0 to 300 seconds. The approximation is because of the differences in time-steps between the input data, transient thermal analysis, and transient

structural analysis. Figure 15 shows the temperature input data from a GODIVA-IV burst that occurred on May 24, 2016. [a]

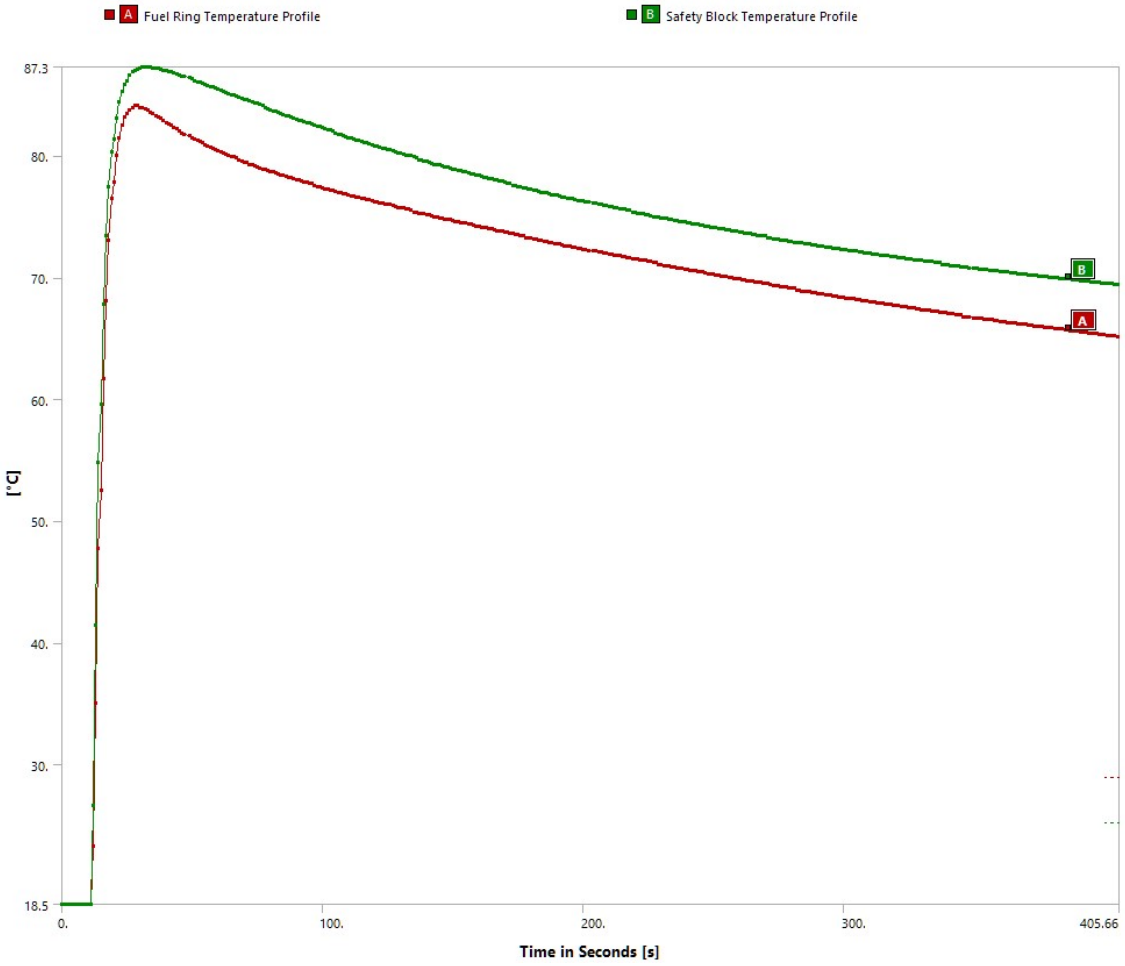

Figure 15. Temperature data input.

The calculated temperatures for each mesh element, node, and time-step are automatically fed into ANSYS transient structural analysis. These data are used as an input for the structural expansion and stress/strain calculations. Every defined time-step for the structural expansion has a corresponding input from the solution calculated in the transient

[a] These results were provided through communication with Joetta M Goda at Los Alamos National Laboratory.

thermal analysis. This makes the structural analysis dependent on the thermal solution. Figure 16 shows an example of a thermal solution mesh that would be fed into the structural analysis at 302 sec. Figure 17 is an exaggerated deformation result from the input thermal solution in figure 16. The legend located on the left of figure 17 is the non-exaggerated numerical answers for the expansion.

The GODIVA-IV model show in figure 16 has had some simplifications applied. The mounting base, clamp legs, and the ring shown in figure 13 were not used in the analysis because they underwent zero thermal expansion.

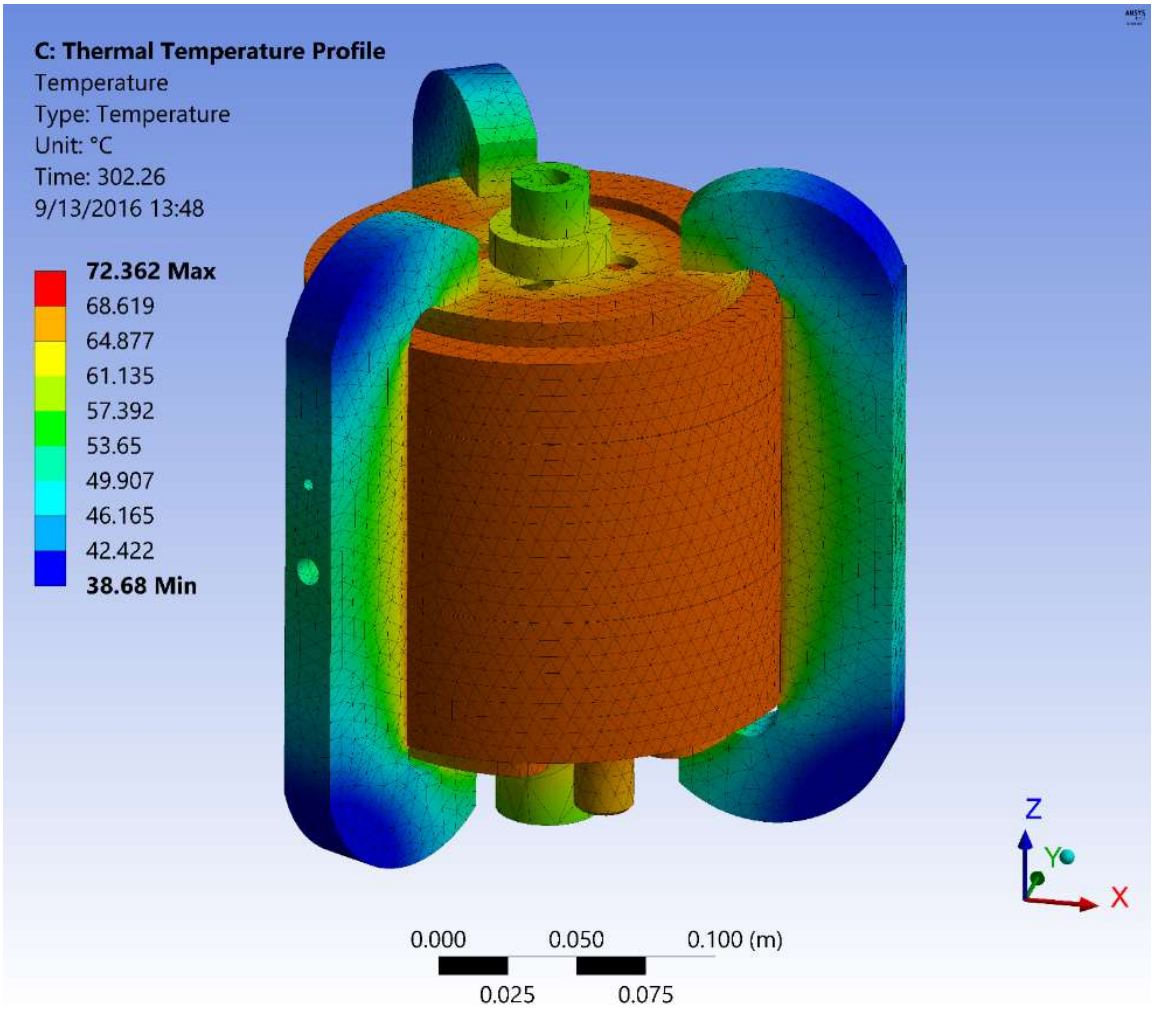

Figure 16. Calculated thermal solution at the end of the analysis (302 sec).

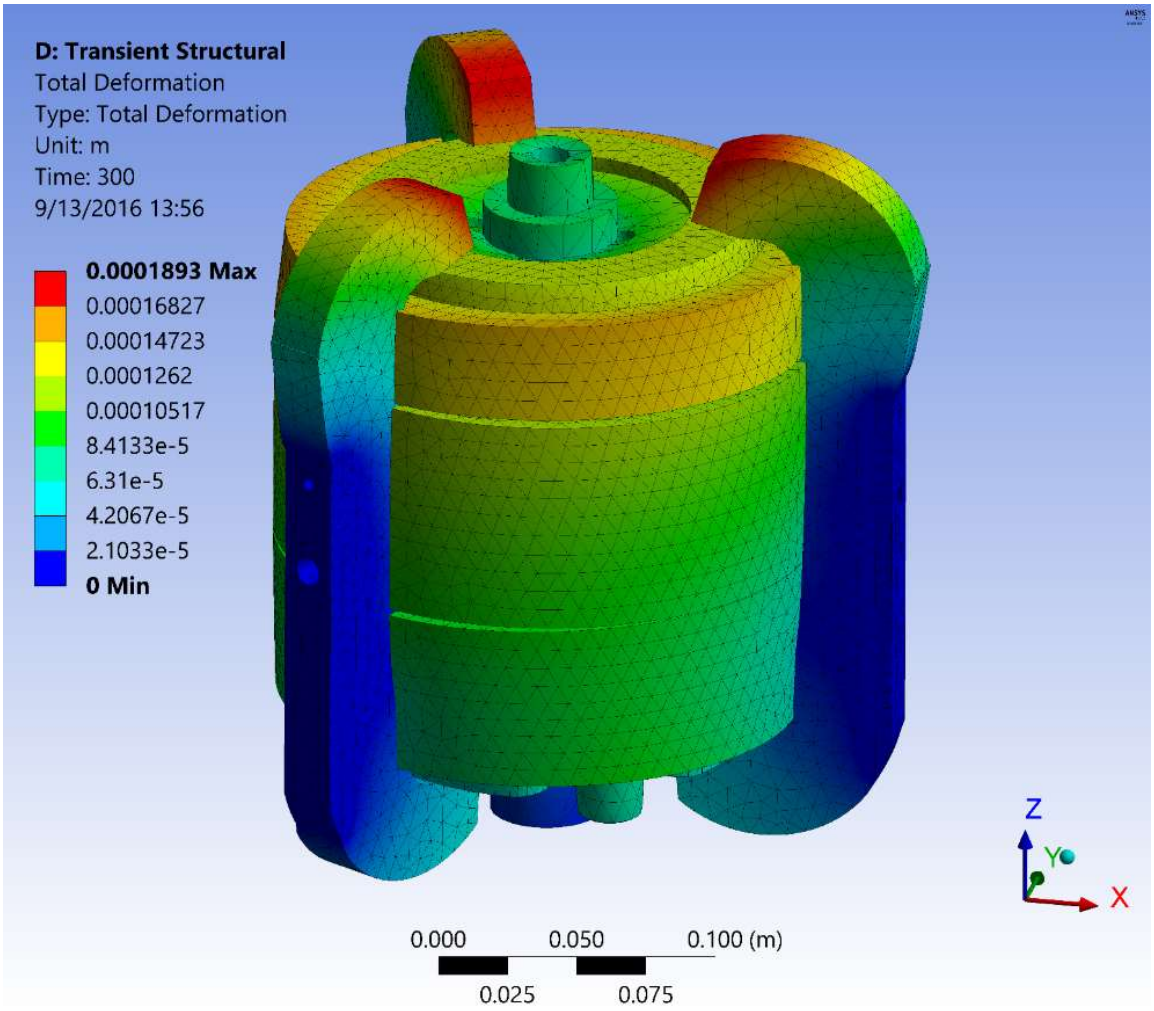

Figure 17. Exaggerated structural deformation at the end of the analysis (300 sec).

These calculations are performed for all defined time-steps. The results are compiled and exported into spreadsheets. Figure 18 and figure 19 are plots of the displacement points defined over the deformed surfaces.

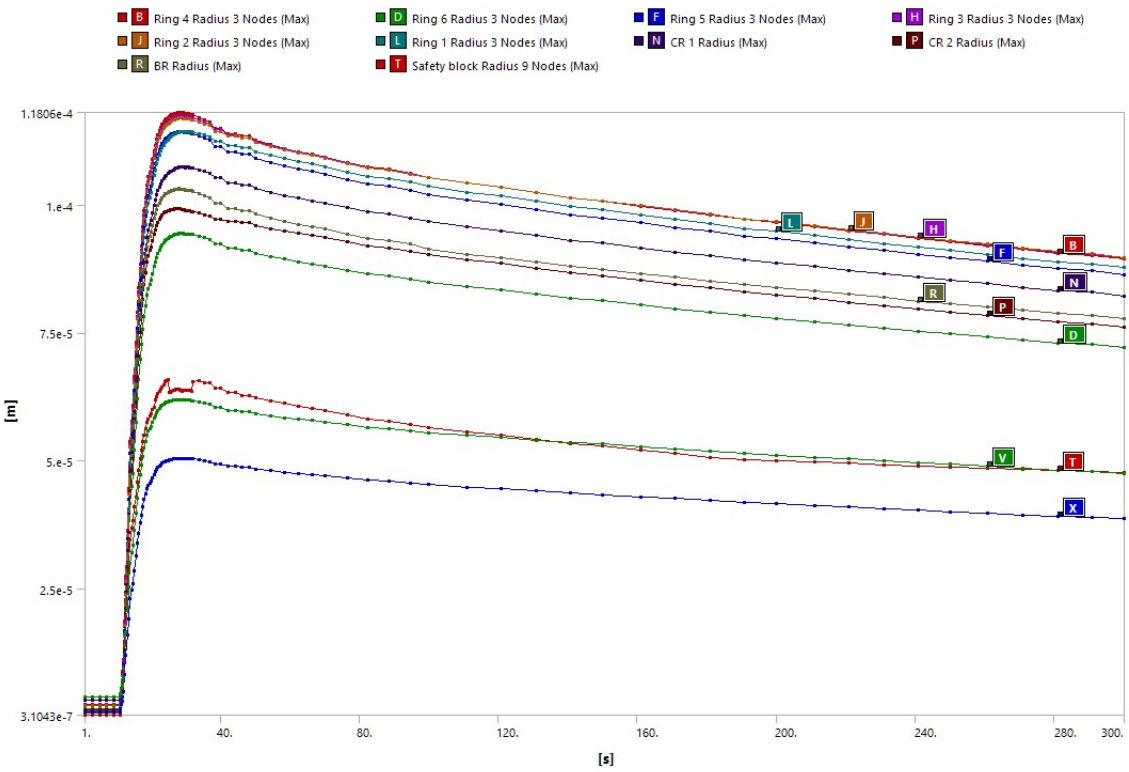

Figure 18. Displacement of radii from beginning of experiment.

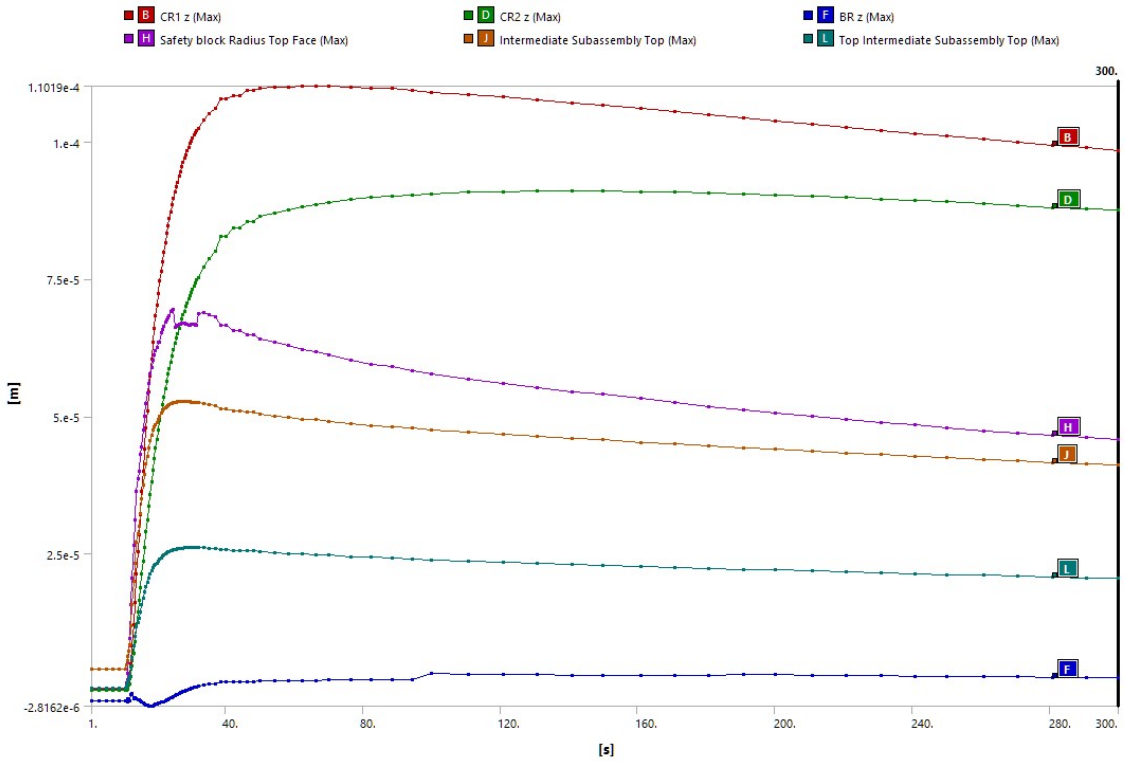

Figure 19. Displacement of Z lengths from beginning of experiment.

These data were then placed back into the original CAD model of GODIVA-IV to determine the change in volume of each of the critical components in GODIVA-IV. This was necessary for adjustment of the atom density. With the new volumes and dimensions, these were used to change the original MCNP input file for GODIVA-IV. Several time-steps were compiled and run to see the change in k-effective. Figure 20 shows the computed data points with a trendline through them.

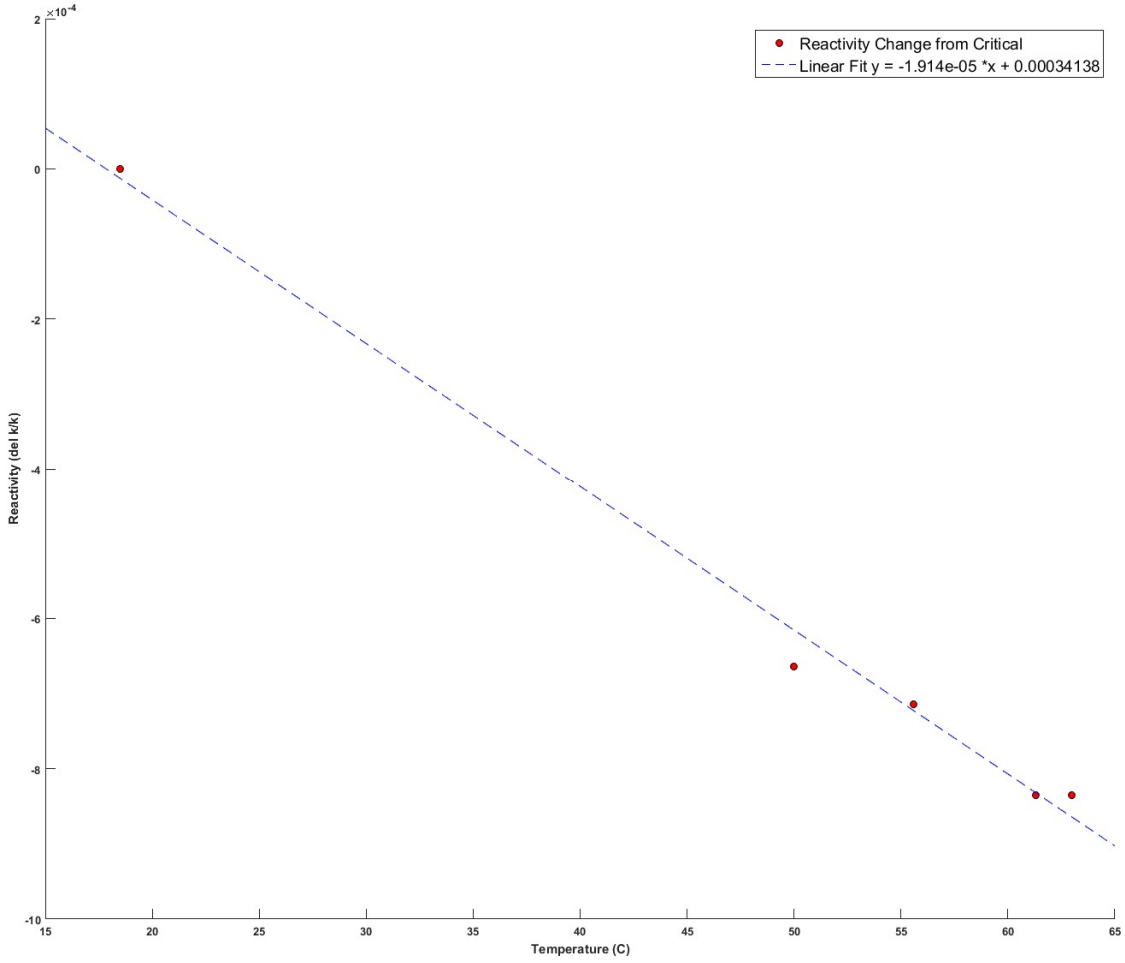

Figure 20. GODIVA-IV Δk/k vs temperature.

The following table shows the results of the ANSYS MCNP model to simulate a burst. It shows that using ANSYS to model the mechanical interactions, and subsequently MCNP to determine the reactivity change, can yield accurate results compared to the real experiment. The uncertainty is derived from the linear-least-squares fit of the data.

Table 5. Temperature Coefficient Results Using β=0.0065

Result Set	Temperature Coefficient (¢/°C)
Los Alamos National Laboratory	-0.3 [12]
ANSYS, MCNP	-0.2945 ± 4.9 %[a]

[a] These results have been submitted for publication in Nuclear Engineering and Design Journal [25]

5.0 MODELING REQUIREMENTS AND CHALLENGES

5.1 Model Simplifications

Simplifications found in previous thermal-mechanical codes were shown to be too aggressive when modeling the katana effect. The simplifications were required because the old methods involved limitations in computing technology and modeling methods. Great strides have been made in both areas since then; computers today are significantly faster and able to take in much larger models. However, even with modern advances, the problem of modeling the katana effect still requires some simplifications to keep runtime reasonable.

5.1.1 Temperature Input

The perfect temperature input would use measured data from multiple temperature probes during a run of EBR-II. Each subassembly would have a centerline temperature with multiple axial locations. These data would then be compiled into a three-dimensional heat transfer model which would subsequently calculate the temperature everywhere. The experimental data from EBR-II was significantly less copious. There were a few core probes which measured temperature of the bulk sodium and two instrumented subassemblies. These were not enough to generate a global temperature profile. In place of the measurements of temperature, a thermal hydraulics model was used to calculate the power generated for each core subassembly. Previous work had been published on a thermal hydraulics benchmark for run 138B. [13] These data were used as an input for the heat transfer model. Figure 21 shows the power produced per subassembly for run 138B.

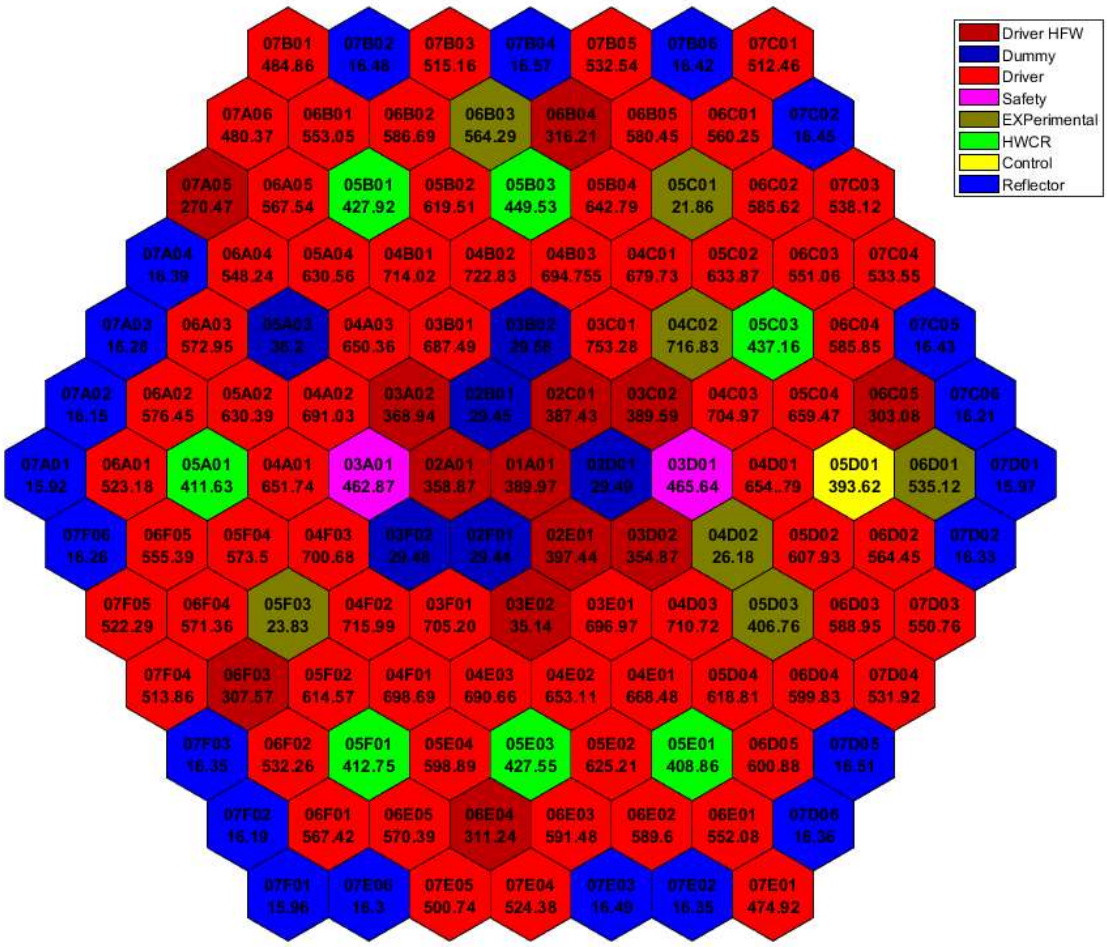

Figure 21. Power (kw) per subassembly (127 total) for run 138B at 65 MWth

The thermal hydraulics model was verified by comparing against the measured outlet temperature and was found to agree. The problem with these powers was that they cannot be used without other information. This led to the creation of a separate heat transfer model that utilized the power per subassembly generated by the thermal hydraulics model. Ultimately the temperature profile was calculated since there was no existing data to compare against. The only verification available was to compare the outlet temperatures of a couple of subassemblies to measured values. This comparison is a weak method of verification because the desired model output is a highly detailed temperature profile. Comparing against a few measured outlet temperatures is insufficient given the breadth of the data. Ultimately using power per subassembly is a simplification because localized

temperature gradients were simplified into an average temperature, and these missing gradients will affect the thermal expansion of the duct.

5.1.2 Discretized Model

Mentioned in section 2.4.3 was the requirement that the model will need to resolve net stress and displacement. This necessitates an iterative process since both quantities are independent variables. Neither is known but both require the other for a solution. Knowing iteration will be necessary, the eventual solution will be a series of discontinuous points. The overall uncertainty being the convergence criteria established prior to the solution.

With enough points, the solution will appear like a continuous function across the entire structure. Due to the discontinuous nature of the function, the solution is still an approximation of a continuous function leading to missing information in the discontinuities.

5.1.3 Structural Displacements to Neutron Transport Geometry

Modeling the katana effect requires creating a geometry in a neutron transport code that simulates a bowed duct. Multiple simulations will be done at various stages of bowing to calculated the reactivity response of the bowed core. A translation between mechanical analysis result data and Monte Carlo neutron transport geometry is required. Due to the complexities of the result data and the limitation of Monte Carlo code geometry capabilities, a simplification is required to transfer the displacement data from FEA to neutron transport.

Neutron transport codes are advancing rapidly to incorporate more exotic geometries and common file formats. The primary change is the incorporation of unstructured meshes. The primary purpose is to streamline the development of engineering design without the need to recreate complex designs in neutron transport codes. There have been some gains in this incorporation of unstructured meshes, but the method is still new and not fully resolved.

Stated in section 2.4, the katana effect leads to a highly deformed geometry that cannot be modeled with a simple formula. Most neutron transport codes do not have an ability to model such geometry. Therefore, some simplification or approximation must be done. Specifically, for EBR-II hexagonal ducts can be modeled in most codes, but a non-symmetric bow cannot be modeled. The most accurate way to simplify the katana effect is to discretize the hexagonal duct into axial slices. Figure 22 shows a representation of the axial slices.

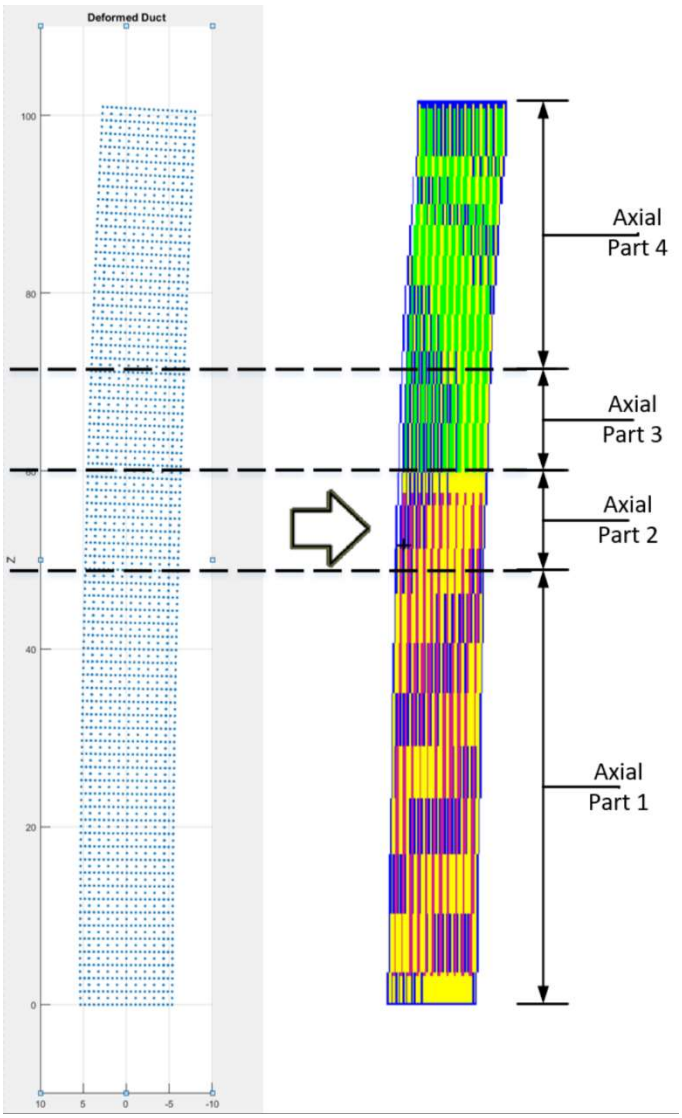

Figure 22. Process for converting displacement data to axial slices.

These axial slices will be moved according to the net displacement of the nodes between each slice. The primary problem with this method is that it takes very precise displacement information and performs a gross average to obtain a net displacement. The number of nodes for each slice varied but generally was several thousand. This led to a low uncertainty for the average displacement. The loss of the individual nodal movements was still present and was considered a simplification.

5.2 Katana Effect Analysis Flow Chart

Figure 23 and figure 24 are flow charts of the process to calculating the katana effect temperature coefficient. More detailed description of the method is in section 6.0.

5.2.1 FEA Model Flowchart

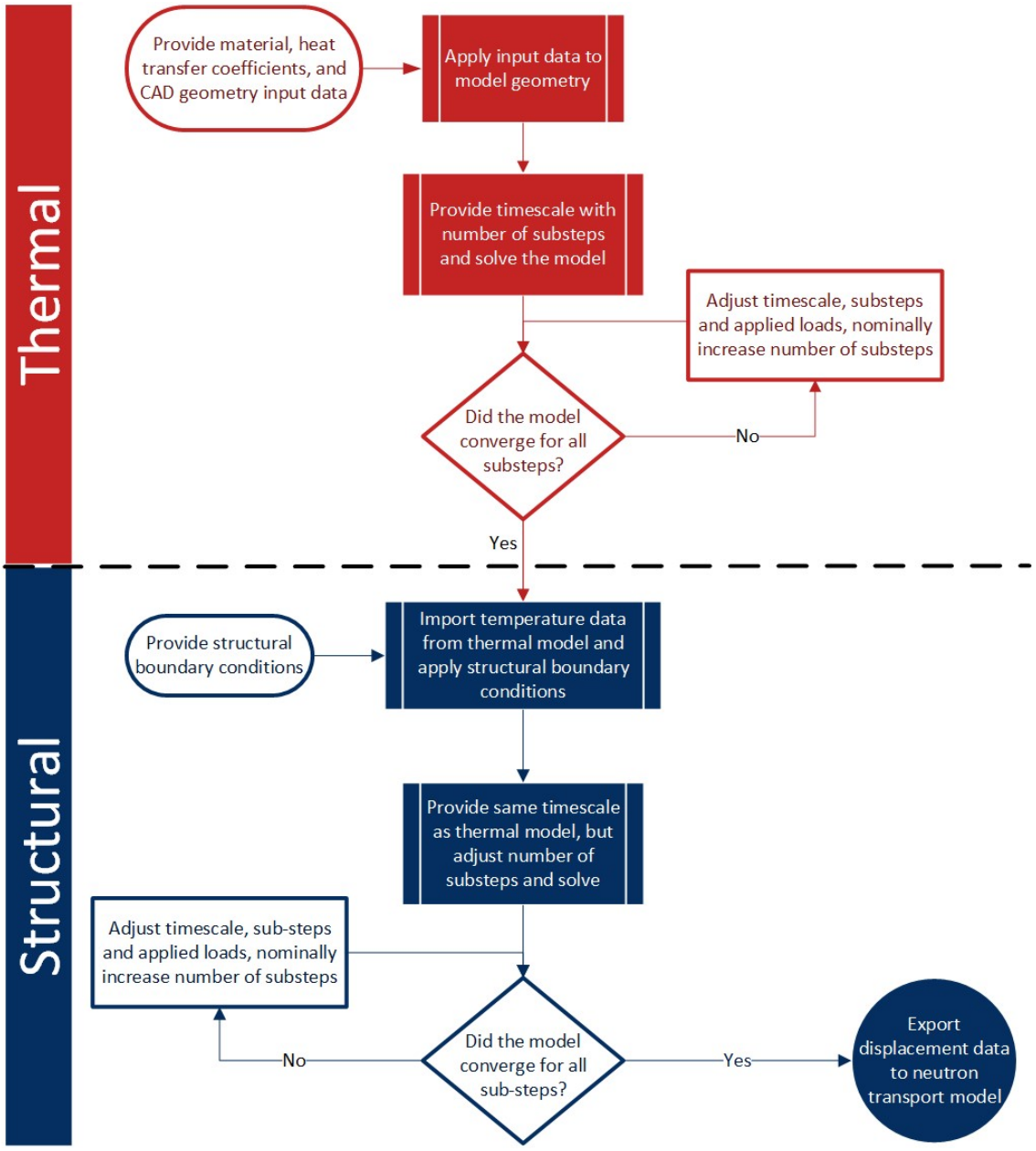

Figure 23. FEA model flow chart.

5.2.2 Katana Effect Temperature Coefficient Calculation Flow Chart

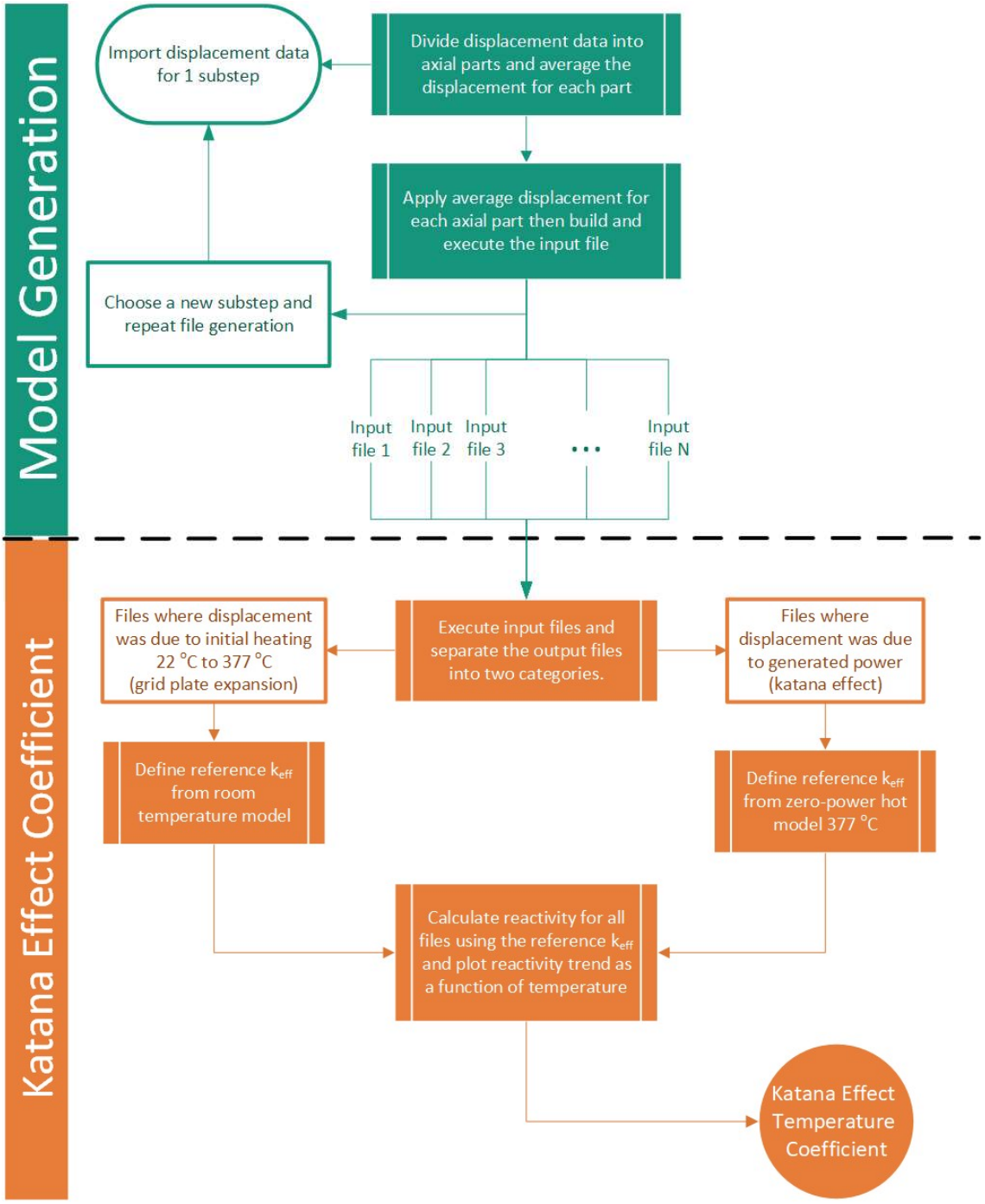

Figure 24. Neutron transport model and katana effect calculation flow chart.

6.0 SIMULATION METHOD

The following sections describe the EBR-II katana model and the process for achieving the solution. The analysis was run on a Dell T7610 Precision Workstation, the specifications for the workstation are shown in table 6.

Table 6. Computer Specifications

CPU Type	Dual Socket Xeon E2697
Threads	48
Memory Type	DDR3 1866 MHz ECC
Memory	256 GB
OS Drive	RAID 10 256 GB SSDs
Storage Drive	RAID 10 3.6 TB Hard Disk
Scratch Disk	Intel 750 Series NVME 1.2 TB
GPU Computing	2 x NVIDIA K40 Teslas

6.1 The Katana Effect ANSYS Model

6.1.1 EBR-II Katana Effect CAD Model

Modeling the katana effect first required creating a CAD model of EBR-II. Previous work has shown that duct-bowing was primarily led by the duct itself and was not impacted in a significant way by the internal structures or the fuel pins. This is justified due to where the internal components were welded. The lower extension and adapter were welded in one location at the bottom of the duct. The upper extension was welded at the top of the ducts. The fuel elements were attached to the lower extension but were not rigidly connected. The only restraints for the internal structures were welds at the top and bottom of the ducts. Subsequently the components could slide freely against the walls of the duct leading to no internal restraint for duct-bowing. This is important to note because only modeling the duct and not the internal components significantly reduces the problem size. Therefore, a model was constructed based upon the engineering drawings of only the ducts and the upper part

of the reactor grid plate. Originally, the ducts and the lower adapters were modeled with a boundary condition where the lower adapters were not allowed to thermally expand. Figure 25 shows the CAD model with the lower adapters modeled. Initially restraining the lower adapters proved incorrect because one of the significant negative reactivity effects was the thermal expansion of the upper part of the reactor grid plate and the grid plate was the primary lower restraint of the subassemblies. This meant that the upper part of the reactor grid plate not only thermally expanded but also underwent mechanical stress as the ducts bowed. Subsequently, the CAD model was redone to remove the lower adapters and model the grid plate. Figure 26 shows the redone CAD model with the reactor grid plate modeled.

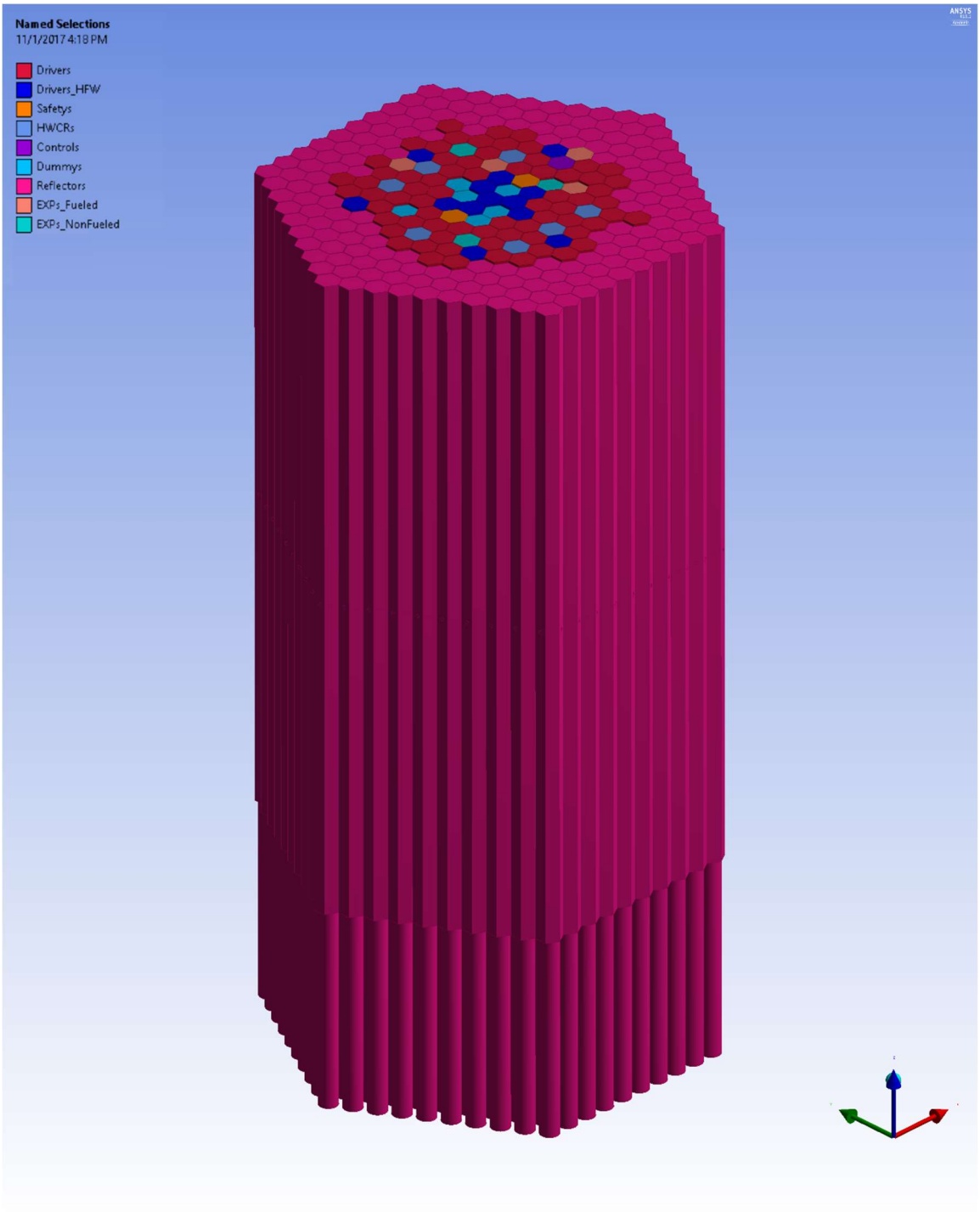

Figure 25. EBR-II katana effect CAD model with lower adapters.

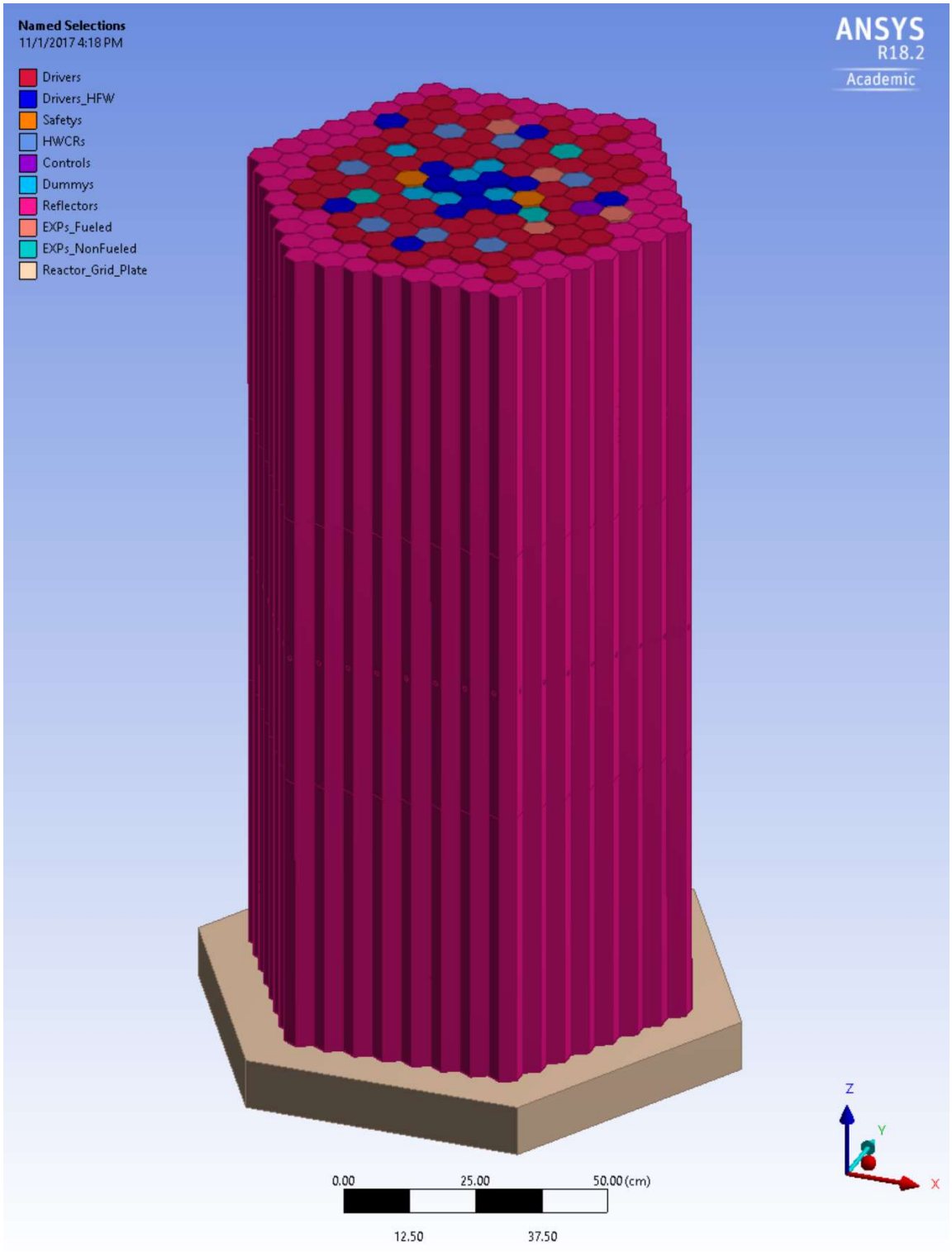

Figure 26. EBR-II katana effect CAD model with reactor grid plate.

Another change that was made to the CAD model was the reduction in the number of subassemblies. The original model had 331 subassemblies; these included the core region, stainless-steel reflector, and the first row of blankets. It became clear with initial testing that modeling 331 subassemblies would lead to an unreasonably large model that would necessitate the use of a large scale high performance cluster (HPC) to solve. The model was reduced to 169 subassemblies. The main loss of detail from this reduction in subassemblies was the inverted bowing direction of row 10. This is a significant loss for achieving an exact result for the katana effect negative reactivity, but it only affects the magnitude of the answer and not the trend. This was shown in the initial runs 1 through 24 of EBR-II compared against run 25. Refer to section 2.2 for more information on the inverted bowing phenomena.

The second modeling component was simulation of the thermal profile. An initial investigation of achieving the thermal profile through Computational Fluid Dynamics (CFD) was done. This was quickly dismissed because of the significant modeling complexity that integration would have caused. Instead a heat transfer model was used. That model used powers calculated from an external thermal hydraulics model. [13] Combining the subassembly powers with hand-calculated heat transfer coefficients, enough boundary conditions could be created to solve for a thermal profile. Additions were required to be made to the CAD model to incorporate the boundary conditions. The main addition being the sodium inside of the ducts. Since it was not necessary to model the internal structures, the sodium fills the internal space. The sodium is divided into three axial sections. This allowed the power generation volume to match the axial position of the fuel pins. It also allowed for control rod movement. Figure 27 shows the EBR-II CAD model broken down into its constituent parts.

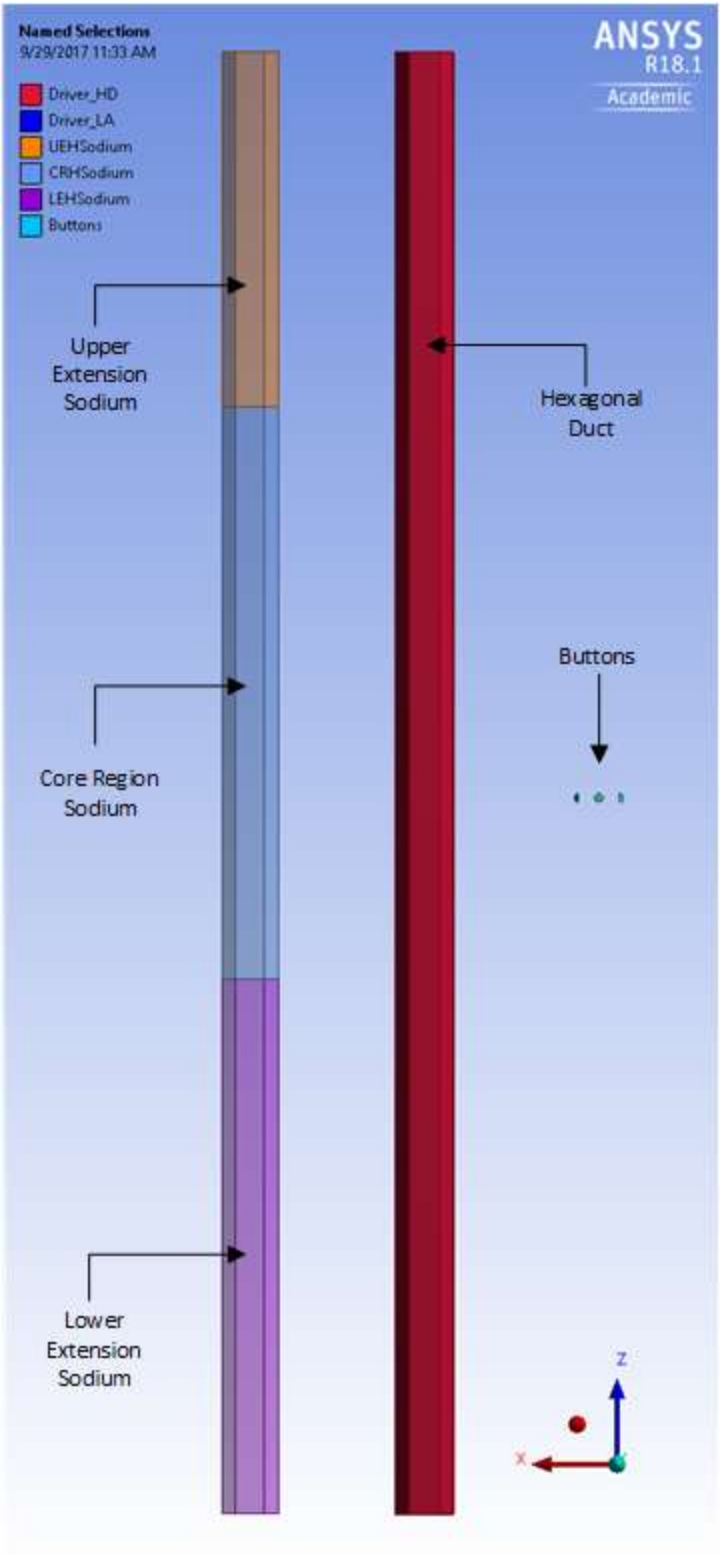

Figure 27. Driver subassembly CAD model part breakdown.

6.1.2 EBR-II Katana Effect Mesh Model

As stated in section 2.4, FEA can lead to a model that can exceed reasonable computing capability. This is caused by the sheer size of the data that needs to be retained. Most FEA codes, ANSYS included, have several algorithms that reduce the extraneous information from the matrices and streamline the final matrix solution such that the size is made as small and as efficient as possible. Even with these algorithms, the bare minimum of information can still be large. The size of the problem is directly related to the size of the mesh that is used to cover the geometry. Specifically, the number of nodes of the mesh determines size because an element can be made of a significantly different number of nodes depending on the type of element. A fully converged solution can lead to an incorrect answer if the mesh is poorly designed. What constitutes a good mesh varies from problem to problem. Most codes have an auto-meshing function where the user does not need to precisely tell the code what elements and where. The auto-meshing function is not a panacea to meshing a geometry. Small problems can use auto-meshing that yields an acceptable result without the need for further modification. Larger problems when auto-meshed will nominally yield a mesh that will be either too coarse or too fine depending on the problem geometry. Most FEA codes give the user tools to manipulate the auto-meshing algorithm to manipulate the mesh to a more acceptable state. For example, the EBR-II thermal model auto-meshed, without any mesh modifications applied, leads to a mesh that has 739,796 nodes and 327,804 elements. Figure 28 is a figure of the EBR-II model using only the auto-mesh function.

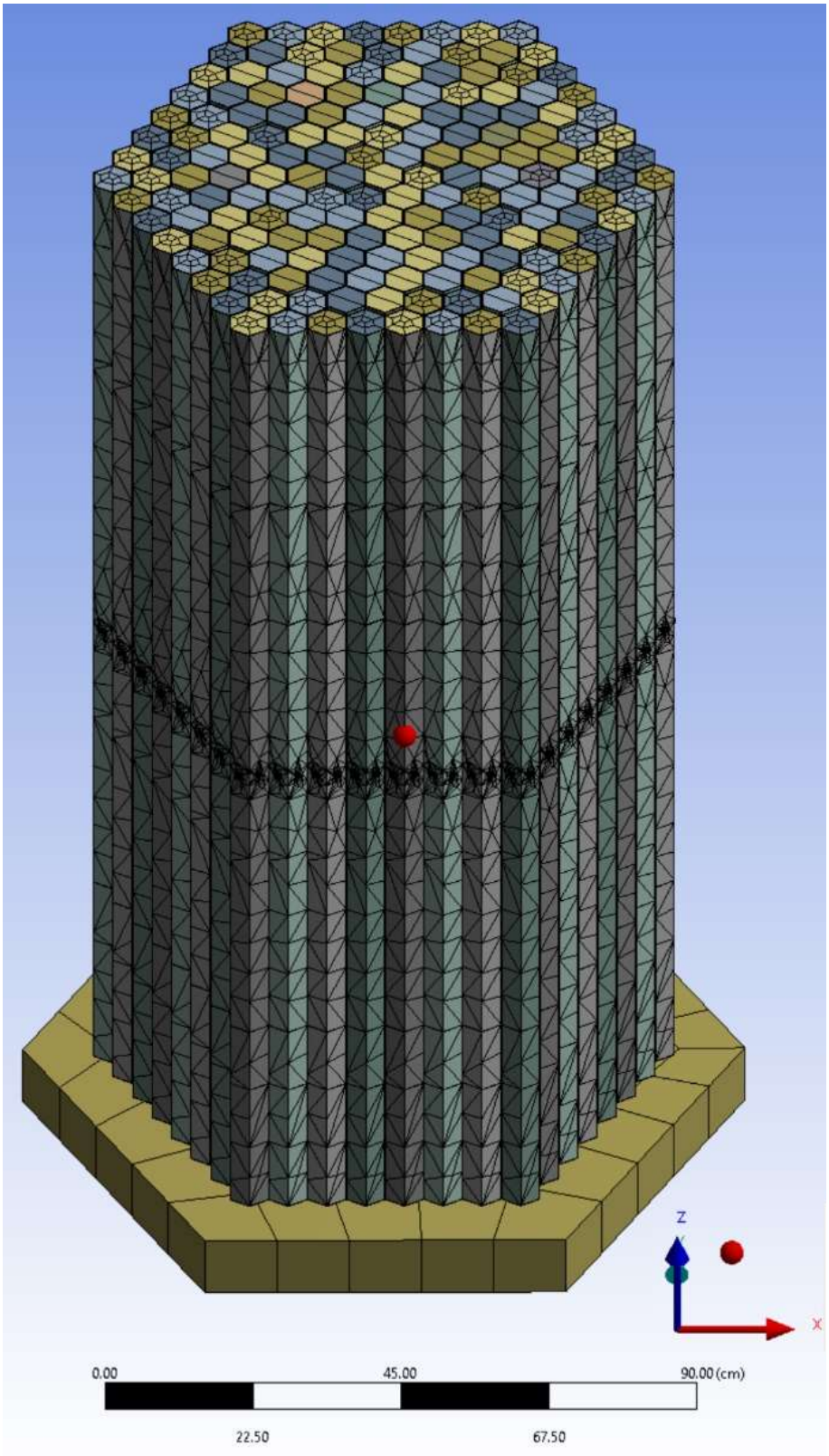

Figure 28. EBR-II ANSYS auto-mesh.

The auto-mesh for EBR-II was inadequate and attempting the analysis with the auto-mesh led to an un-converged solution. The primary problem with the mesh was the element size. Using figure 28 as a reference, most of the faces of the ducts only contain one element across the smaller dimension. One element means that there are only two points where the temperature would be calculated. The shape functions assumed in section 3.0 only account for the change between the nodes; the functions do not create nodes where the temperature would be calculated. This problem with the mesh nearly guarantees non-convergence.

The second problem with the mesh is the shape of the elements. Most FEA codes have dozens of different element shapes such that nearly any geometry can be meshed without severe distortion of the elements. When there are a significant number of distorted elements, node solution contribution can be highly skewed. Figure 29 shows a node where a distorted element is contributing to the node. The center node in the figure has seven solution contributions. Distorted elements create an out-of-balance condition where the center node solution is impossible to achieve given the seven surrounding solutions. Ultimately this leads to a load difference that cannot be resolved, ergo non-convergence.

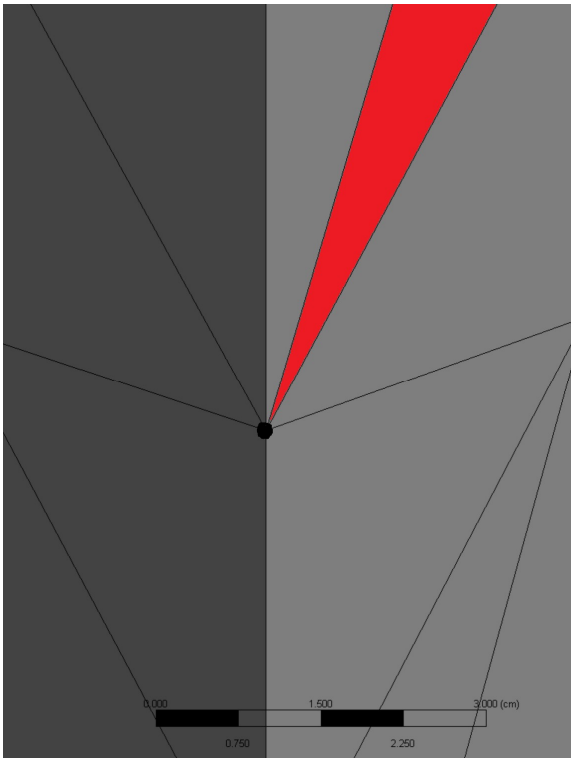

Figure 29. Mesh element distortion.

What constitutes a good mesh is highly specific to the given geometry and analysis. There are a couple of guidelines to follow which are applicable to all geometries. The first guideline is creating a mesh that appears "even." Evenness is a qualitative measure; more specifically, the appearance of the meshed faces should not have sharp changes in node density. Figure 30 shows an example of an even mesh. The nodes are equally distributed from node-to-node and there are no sharp changes in node density. The correction applied to the mesh for figure 30 was to modify the auto-mesh to use only tetrahedrons and brick type elements of a certain size.

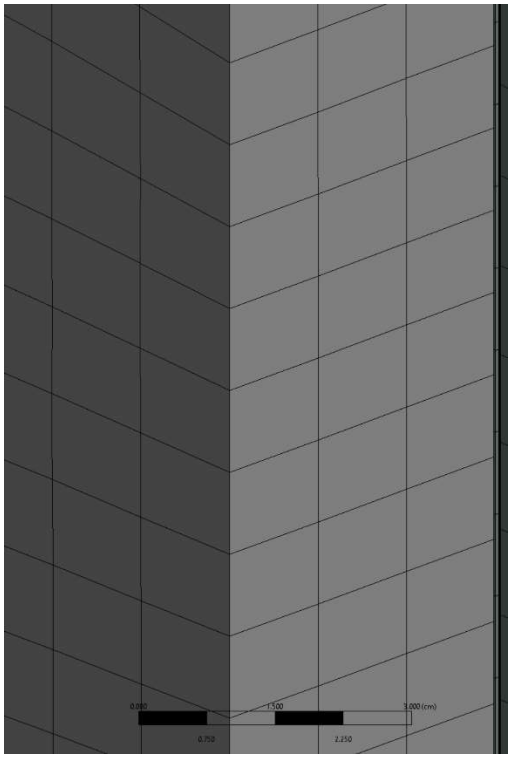

Figure 30. Even mesh example.

The second guideline refers to node density in critical areas. Most analyses have sections of the geometry where a change is occurring. For a thermal model, this can be power generation leading to a hot body conducting heat to a cold body. The interface between those bodies should contain enough nodes such that the solution can make a steady transition from hot to cold. A lack of nodes leads to large discontinuities such that even a correct description of the solution function will have node-to-node changes which exceed the convergence criteria. Therefore, a good mesh will have higher node density in areas where large changes in the node-to-node solution are occurring. For EBR-II the main interaction area where large changes in stress and temperature occur is where the buttons are in contact. Figure 31 shows the mesh across the buttons. There are some distorted elements surrounding the button, but the node density is relatively symmetric surrounding the button. There are also many nodes on the buttons themselves to account for the high stress that occurs during duct-bowing.

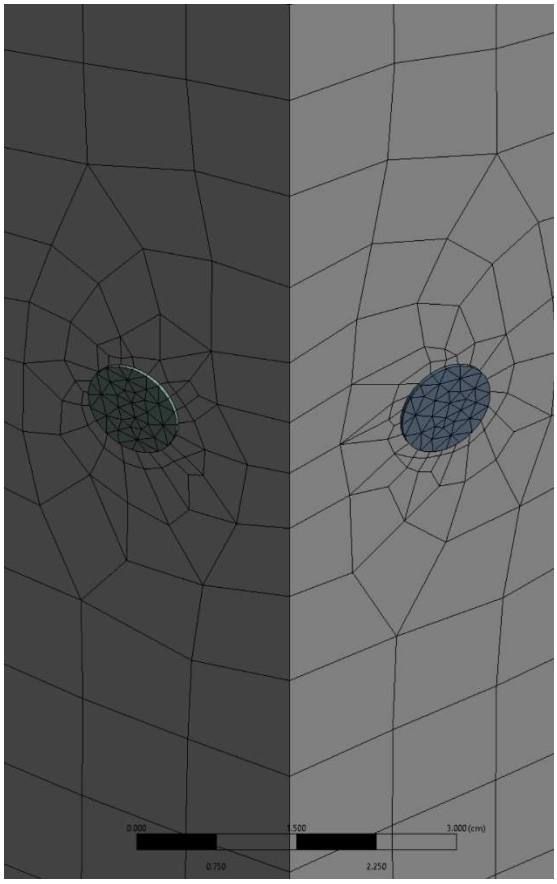

Figure 31. Mesh density.

Using the previous guidelines, it could be assumed that any geometry can easily have a good mesh if the node count were increased significantly. This is a bad assumption because every node is a variable that needs to be solved. Increasing node count increases solution and problem size. Section 2.4 first stated that FEA model size can easily exceed computing capacity. The EBR-II final mesh was 6.6 x 10^6 nodes. Assuming 8 bytes per number, the solution size for one iteration in the thermal model is as follows:

$$nodes * DOF\ (unknowns) * 8\ bytes = Solution\ Size$$

$$6.6\ x\ 10^6\ nodes * 1\ DOF * 8\ bytes = 52\ MB$$

The previous storage space is only for the solution; it does not include all the coefficient values or the non-linear material properties that are interpolated at non-

reference temperatures, and applied heat generation. The final random-access memory (RAM) requirement for the thermal solution was:

$$52\ MB * \sim 20 = 1\ GB$$

Finally, there are the storage requirements for the code itself to keep a record of node numbers, elements, contact information, boundary conditions, iterative copies of the mesh, and variables used for code functioning:

$$1\ GB * \sim 100 = 100\ GB$$

The final RAM usage for the EBR-II thermal model was ~128 GB. The relationship between mesh size (number of nodes) and RAM usage is linear. A doubling of the mesh size would double the RAM usage, far exceeding reasonable computing capability.

Several modifications were done to the EBR-II mesh. With these mesh modifications, a good mesh was established. A higher fidelity mesh would have yielded better results, but increasing the mesh size would have exceed the resources of the machine listed in table 6. Table 7 shows a list of mesh modifications used for the katana effect model. Figure 32 shows a figure of the mesh that was used for the katana effect analysis. It was chosen as the final mesh because it had a good balance between mesh density and mesh size.

Table 7. Thermal Mesh Modifications

Property	Applicable Geometry	Value
Meshing Method	Hex Ducts and Reactor Grid Plate	Use a combination of mostly tetrahedrons and brick elements
Mesh Element Sizing	Hex Ducts and Reactor Grid Plate	1.2 cm
Meshing Method	Buttons	Use only tetrahedron-type elements
Mesh Element Sizing	Buttons	2.0e-2 cm

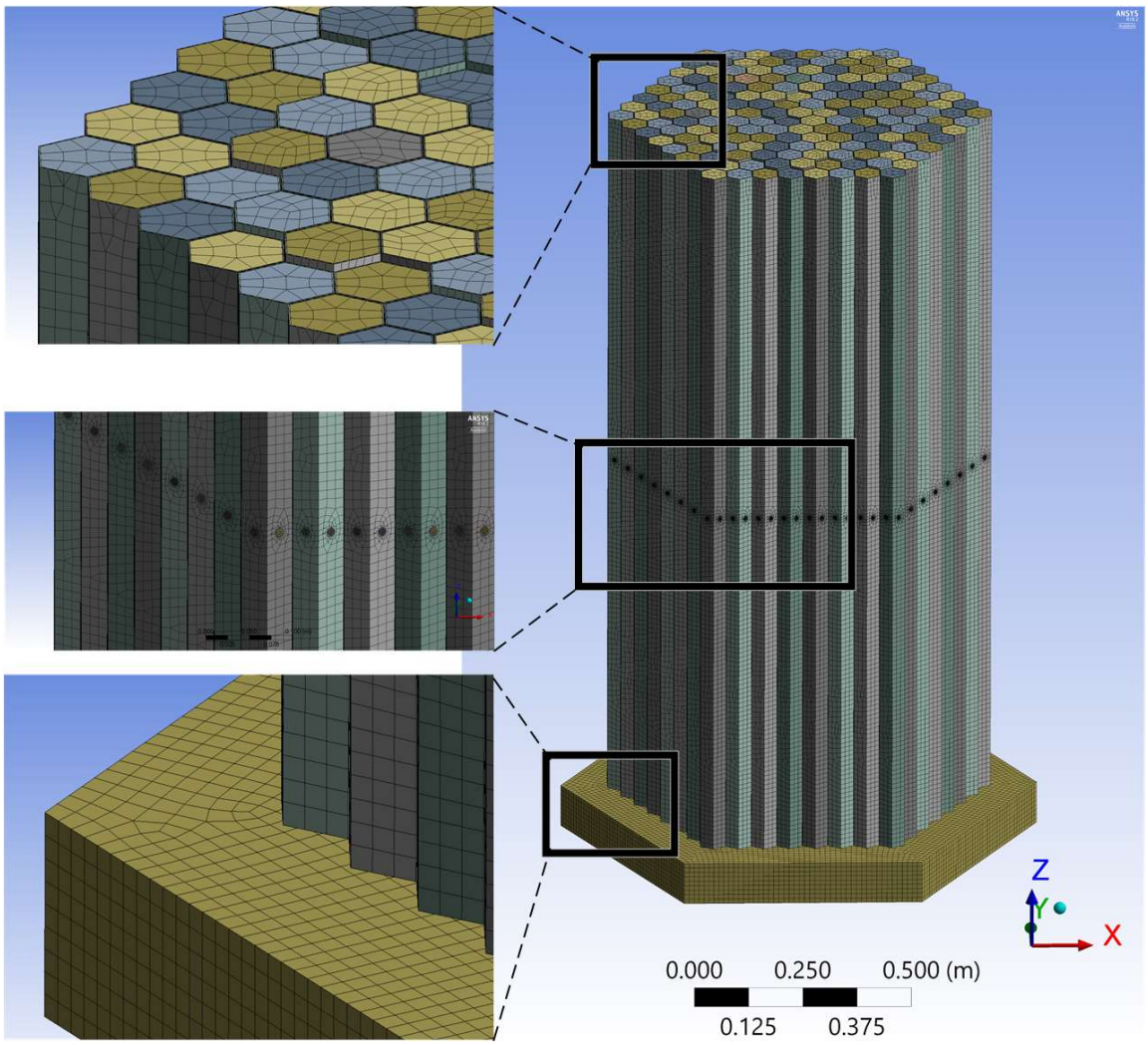

Figure 32. EBR-II final thermal mesh.

With the EBR-II geometry successfully meshed, it was loaded into the thermal analysis model. Table 8 shows the final mesh statistics for the thermal model.

Table 8. Thermal Mesh Statistics

Property	Value
Number of Nodes	5,806,191
Number of Elements	1,661,623

6.1.3 EBR-II Katana Effect Thermal Model

The next stage in the model analysis was to apply the boundary conditions for the thermal analysis. Thermal FEA analysis does not need many boundary conditions to achieve a converged solution, but it can easily give incorrect answers. Sometimes the incorrect answers are not obvious given the copious amount of results data. A solution can look correct for 99% of the mesh, but then have 10 elements which yield absurd answers. This indicates a poorly defined mesh. Nominally greater node count where the absurd answer is located can resolve this issue. Inspection of the results is important such that nonsensical answers can be found and corrected.

The boundary conditions for thermal analysis are divided into three groups: global conductivity matrix, temperature unknowns, and global applied load (power generation). The temperature unknowns are the solution vector, due to boundary conditions, some of the node temperatures are given as an input. Ambient temperature is not used here because it is considered the reference temperature and is used in conductivity formulation. The main boundary conditions for EBR-II were the heat transfer coefficient of the sodium flowing over the hexagonal ducts, power generation from the fuel region, and a globally applied temperature increase from 22ºC to 377ºC. Table 44 in appendix D.3 shows the temperature dependent heat transfer coefficient applied to the outside of the ducts simulating the flow of sodium over the ducts. Table 45 in appendix D.3 shows the change in temperature over time of the inlet sodium and the outside walls of the sodium pool. Table 43 in appendix D.2 shows a list of the power change over time for individual subassemblies.

The katana effect model is divided into two stages: initial warming and power-up. The initial warming stage is a linear increase in temperature from the reference temperature of 22ºC to 377ºC. The temperature increase is applied to all bodies in the model. This method of warming deviates from run 138B. The sodium in the pool was never replaced and was kept in a liquid state at a temperature of 377ºC. When subassemblies were placed into the core they were at room temperature 22ºC and then heated up as they were submerged in the liquid sodium. The katana effect model does not simulate the placing of subassemblies into hot sodium. The model simulates the heating of subassemblies from

22ºC to 377ºC, assuming they were all loaded at the same time and then evenly heated. The freezing point of the sodium does not matter in the analysis because the initial heating is a boundary condition which forces the temperature solution to follow the trend in table 45 appendix D.3. When the model reaches 377ºC, the sodium is in a liquid state and normal heat transfer resumes.

The next stage in the analysis required choosing the solution method and solver type. The defaults for multi-core thermal analysis are Distributed Direct Sparse Solver using Full Newton Raphson Method. While these were the defaults, investigation showed that the defaults were the best choice for solving this type of problem. They were left unchanged.

After the solution method is chosen, steps and substeps need to be configured. "Problem stepping" determines how many substeps should be used to apply a given load and the overall timescale. The primary goal is to use the minimum number of substeps possible without losing solution fidelity. Some nomenclature is required. An iteration is one attempt at solving for the solution vector u. A substep is where multiple iterations lead to a converged solution. A converged substep will write out solution information for the user to inspect and appears in the final solution. A step is a series of converged substeps defined over a given timescale. A series of steps with converged substeps is usually what is required for a successful solution. The defined timescale for each step can be different depending on the problem needs. For example, one step can have a timescale of 0 sec to 0.1 sec and another step can have a timescale of 0.1 sec to 10 sec. The number of substeps can also change depending on problem requirements. Continuing with the same example, the step with a timescale of 0.1 sec to 10 secs can have 100 substeps whereas the same timescale can have 10 substeps. There are no limits on how many substeps can be applied. The time-stepping also determines how the boundary conditions are applied. For example, heat generation is time dependent, which means using a small timescale for a step can apply the load incrementally such that convergence can be achieved more easily. Some boundary conditions are step independent because they depend on temperature. Some boundary conditions, like the heat transfer coefficient, can use both timescale and/or temperature.

Figure 33 below shows the EBR-II model time scale and step information for the overall problem and for the time-dependent boundary conditions.

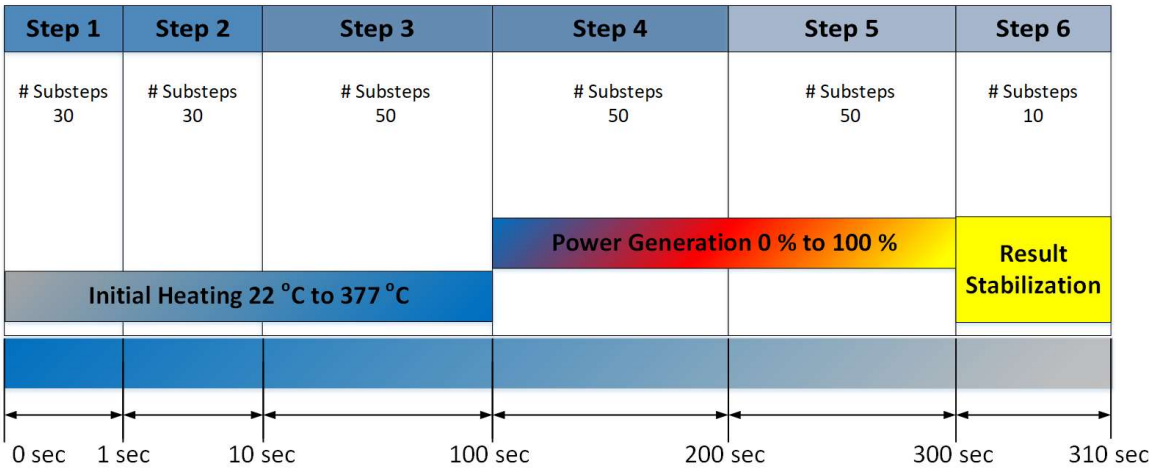

Figure 33. Step and timescale for katana effect thermal model.

With a fully converged thermal solution, the results and information must be transferred to the structural analysis. This process is computationally intensive because the result data must be interpolated to a new meshing scheme. For EBR-II the major change between the two analyses was that modeling the sodium was no longer necessary for the structural model since the sodium only contributes to heat transfer.

Figure 34 and figure 35 show a flowchart of the thermal analysis process. The circle with a large “A” connects the two flow charts together.

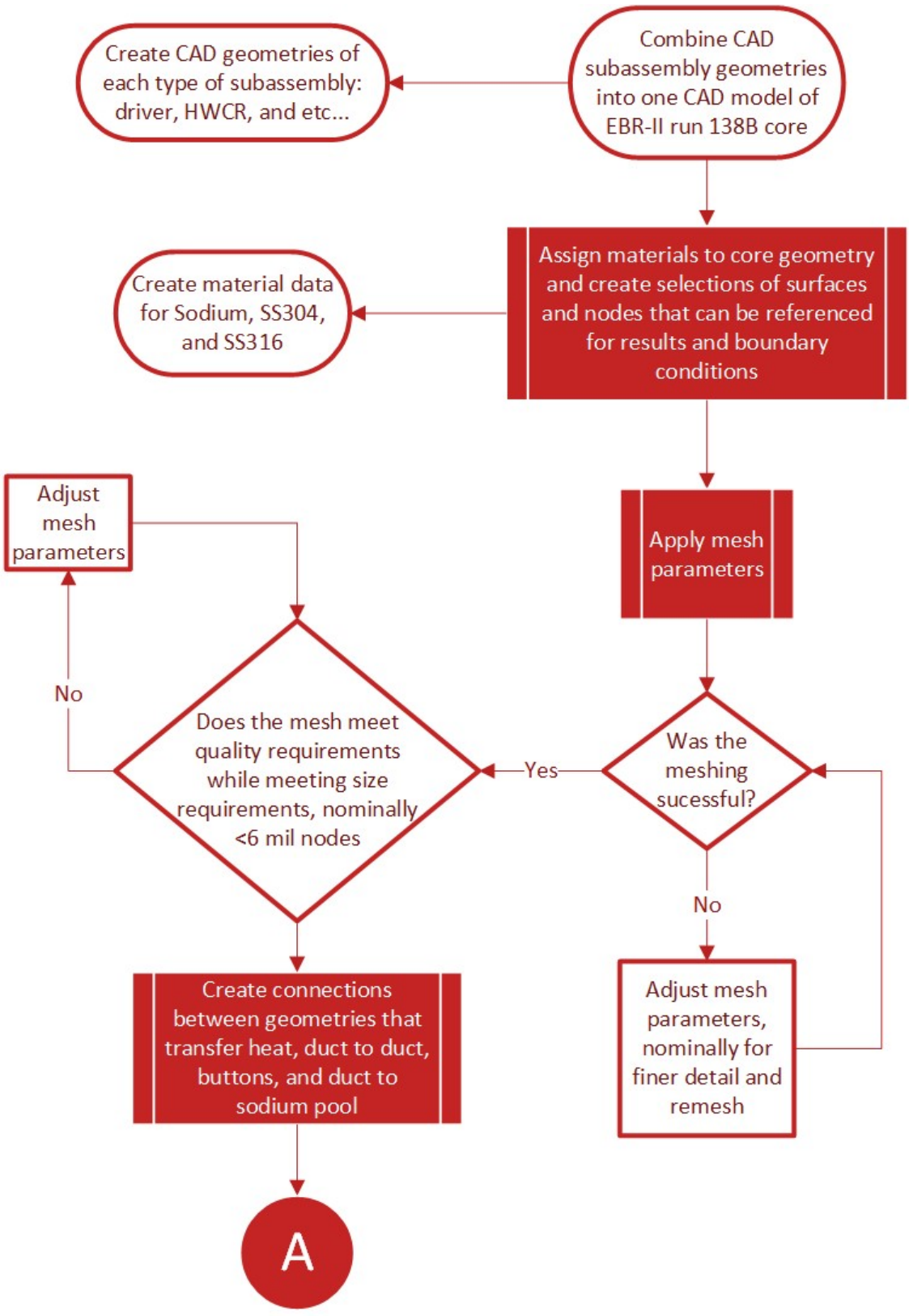

Figure 34. ANSYS thermal model flowchart part 1.

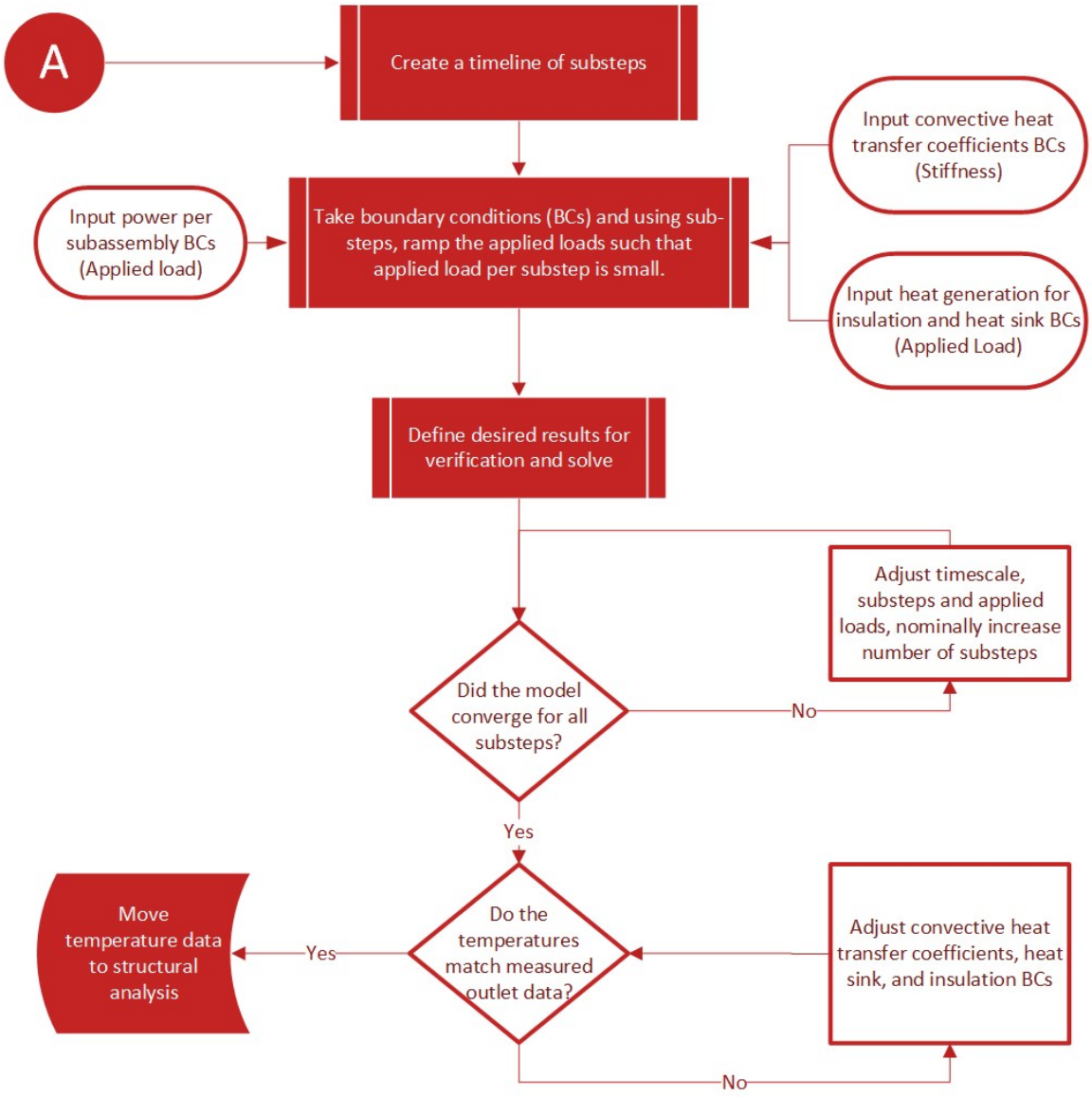

Figure 35. ANSYS thermal model flowchart part 2.

6.1.4 EBR-II Katana Effect Structural Model

The structural analysis follows a similar methodology as the thermal analysis. A mesh needs to be established and modified. Boundary conditions need to be applied, then a stepping regime needs to be established. Table 9 shows the mesh settings used for the structural analysis.

Table 9. Structural Mesh Modifications

Property	Applicable Geometry	Value
Meshing Method	Hex Ducts and Reactor Grid Plate	Use a combination of mostly tetrahedrons and brick elements
Mesh Element Sizing	Hex Ducts and Reactor Grid Plate	1.2 cm
Meshing Method	Buttons	Use only tetrahedron type elements
Mesh Element Sizing	Buttons	2.0e-2 cm

The mesh settings used for the structural model were identical to the thermal model. In some circumstances, the structural mesh can be different than the thermal mesh. In the katana effect model, the meshes were identical in form. The only difference between the two meshes was the reduction in mesh size. The reduction was due to not including the sodium bodies inside of the ducts.

Table 10. Structural Mesh Statistics

Property	Value
Number of Nodes	4,903,839
Number of Elements	1,214,449

The major difference between the two analyses is that the structural analysis requires a minimum of boundary conditions to properly converge. This minimum is not a set number and can only be determined through parametric study. A well-stated analysis should only require the boundary conditions that match the experiment. For complex analyses, however, more boundary conditions beyond the experiment are usually required. This can be attributed to either missing geometry that was not modeled or, due to the complexity of the problem, the solver has a difficult time converging on one solution.

EBR-II's structural experiment boundary conditions were a mechanical restraint where the hexagonal duct met the upper part of the reactor grid plate and the ducts that touched the reactor liner. The latter of these boundary conditions was not used for the model because only 169 of 637 subassemblies were modeled. The reduction in the number of subassemblies led to a problem with the structural model. The subassemblies that were in contact with the reactor liner were now missing. A similar boundary condition could have been used for the outer row in the 169-subassembly model, but this would have invalidated the katana effect results because row 8 underwent a bowing effect. Removing this boundary condition led to only the boundary condition of upper part of the reactor grid plate restraint being used.

The upper part of the reactor grid plate restraint had two characteristics. The first was to prevent movement in the Z axis. FEA is a powerful tool because it simulates all physics effects including gravity. Without a restraint in the Z axis, the entire geometry falls without stopping, leading to non-convergence. The second characteristic was free movement in the X and Y axis. Free movement was required to ensure that thermal expansion could still occur unhindered. The upper part of the reactor grid plate is the geometry which effectively sets the pitch of the subassemblies. It is also the part that sets the clearance between the subassemblies. It was important to ensure thermal expansion of the reactor grid plate would still occur, otherwise the effect of the bow would be overestimated.

Only providing one boundary condition that restrained the Z axis proved impossible to solve. Some intermediate data showed ducts being displaced and warped in absurd ways indicating multiple solutions were being found, eventually leading to non-convergence. After some significant parametric study, it was decided that introduction of an artificial boundary condition was needed. Some previous work had demonstrated that restraining the very center assembly 01A01 would be enough to allow the solver to converge. The first problem with this boundary condition was the assumption of no temperature gradient across the center subassembly that would cause the subassembly to bow. There was not enough experimental data to check against, but it was unlikely that there was no temperature gradient across the duct. The second problem was that the boundary condition

did not allow the duct to thermally expand in the axial direction. This is not a significant modeling defect since axial thermal expansion only acts like a control rod. The overall effect of not including axial thermal expansion for the center assembly is a slight loss of reactivity. This happens because the center assembly would be acting like a control rod which has been marginally removed from the core. Figure 36 shows the step information for the ANSYS structural model.

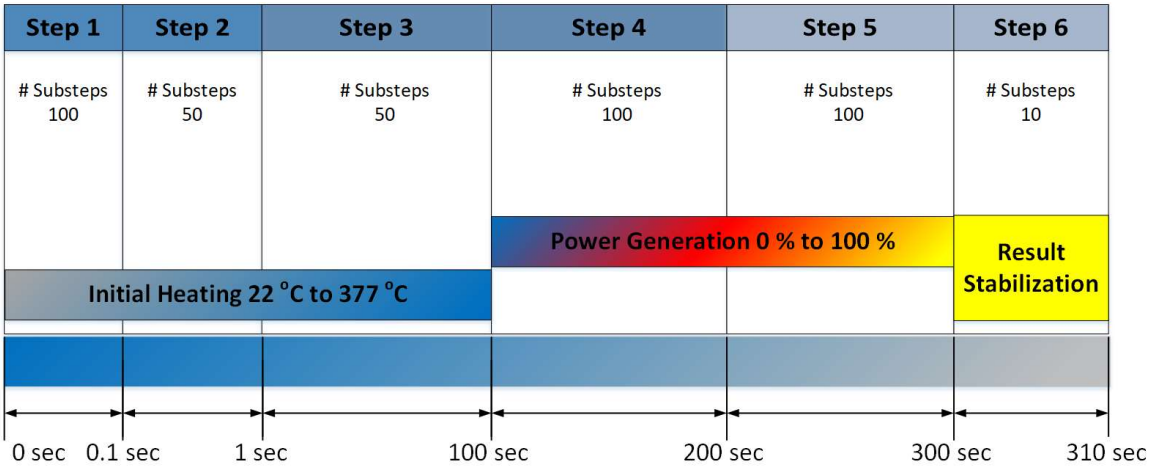

Figure 36. Step and timescale for katana effect structural model.

A fully converged solution was obtained from the ANSYS structural model. Results were exported from ANSYS, specifically the displacement data in the X, Y, and Z axis. These data have the format of "node number X, Y, Z and directional deformation." Each displacement direction is output as a separate file. One of the difficulties encountered early in the analysis was how to relate node numbers with position in the EBR-II structural model and then convert those positions to the MCNP model. One of the tools ANSYS provides to aid in referencing parts of the model is called "Named Selections." These named selections allow the user to select bodies, faces, nodes, elements, and edges. They can be manually selected by the user or found by using a search algorithm. For example, a selection can be made using surface area range or from what material the body is made. These selections can be cascaded to refine the selection down to precisely what is desired. The named selections can then be converted to nodes. This is important because ANSYS then tracks the named selections through the problem to be used as a reference in boundary

conditions or for any use that requires a geometry selection. The named selections are recorded inside of the input file. This provided a method to relate node numbers to body selections, more specifically EBR-II subassemblies. The displacement output data was then paired with the displacement information to relate subassembly to points and then to displaced points.

Due to the considerable complexity of the model and the numerous named selections of nodes that were required, the input file was 50 gigabytes of ~1.1 billion lines of text. Due to the size, the file was unable to be opened in any word processing program. A special reader program was written such that it would search the input file, line by line, and extract the needed information. The program would keep a database of subassembly position and the node numbers that corresponded with them. The reader program would also read in the displacement data and match the node numbers and assembly position with the node position and then the directional displacement information. New points were generated such that the program had a record of assembly positions, node numbers per assembly, position before displacement, and position after displacement. After this information was gathered, the MCNP model could be constructed. Figure 37 and figure 38 show process flowcharts of the structural analysis. The letter “B” located in the circle connects the two flow charts together.

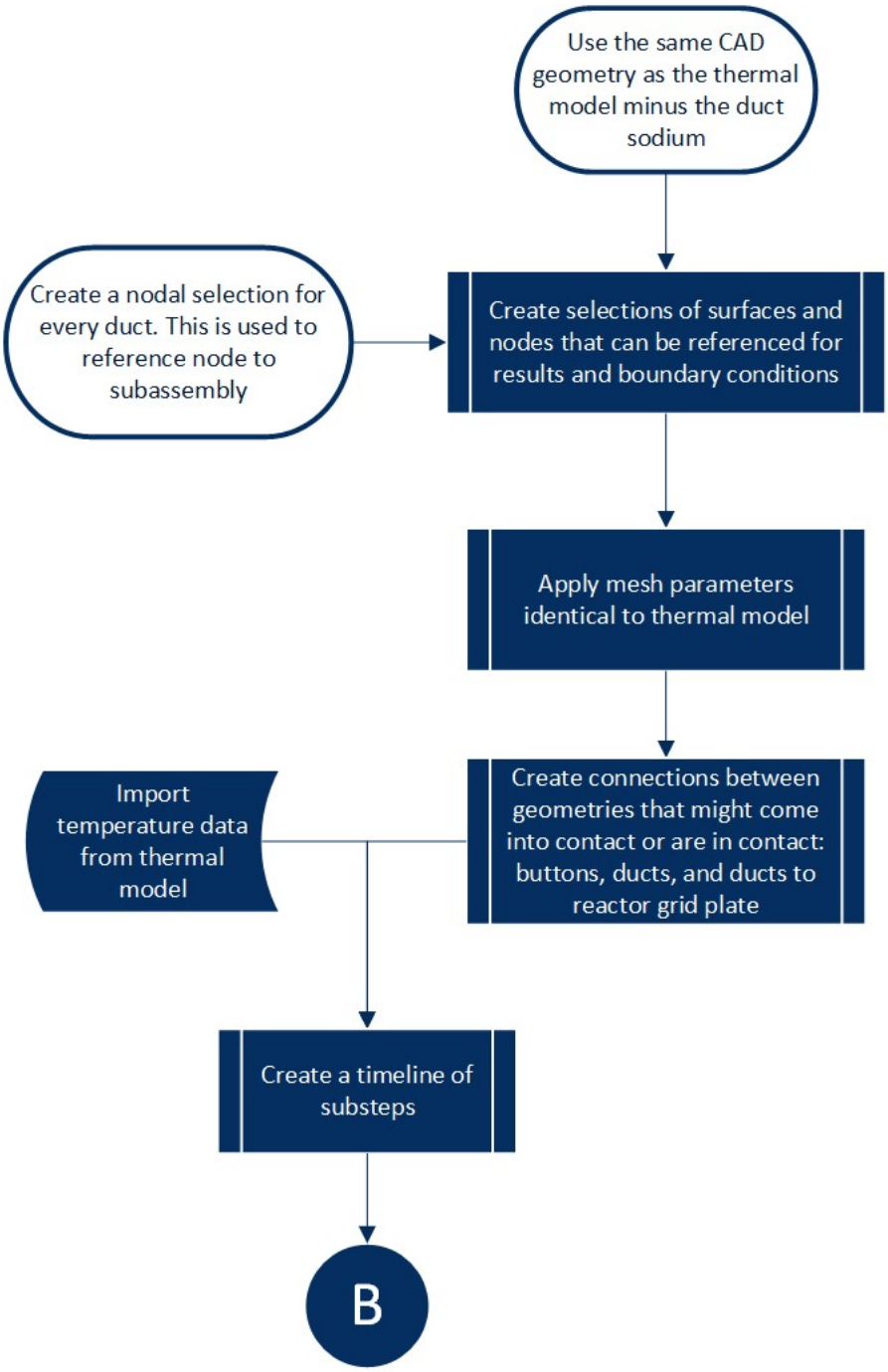

Figure 37. ANSYS structural model flowchart part 1.

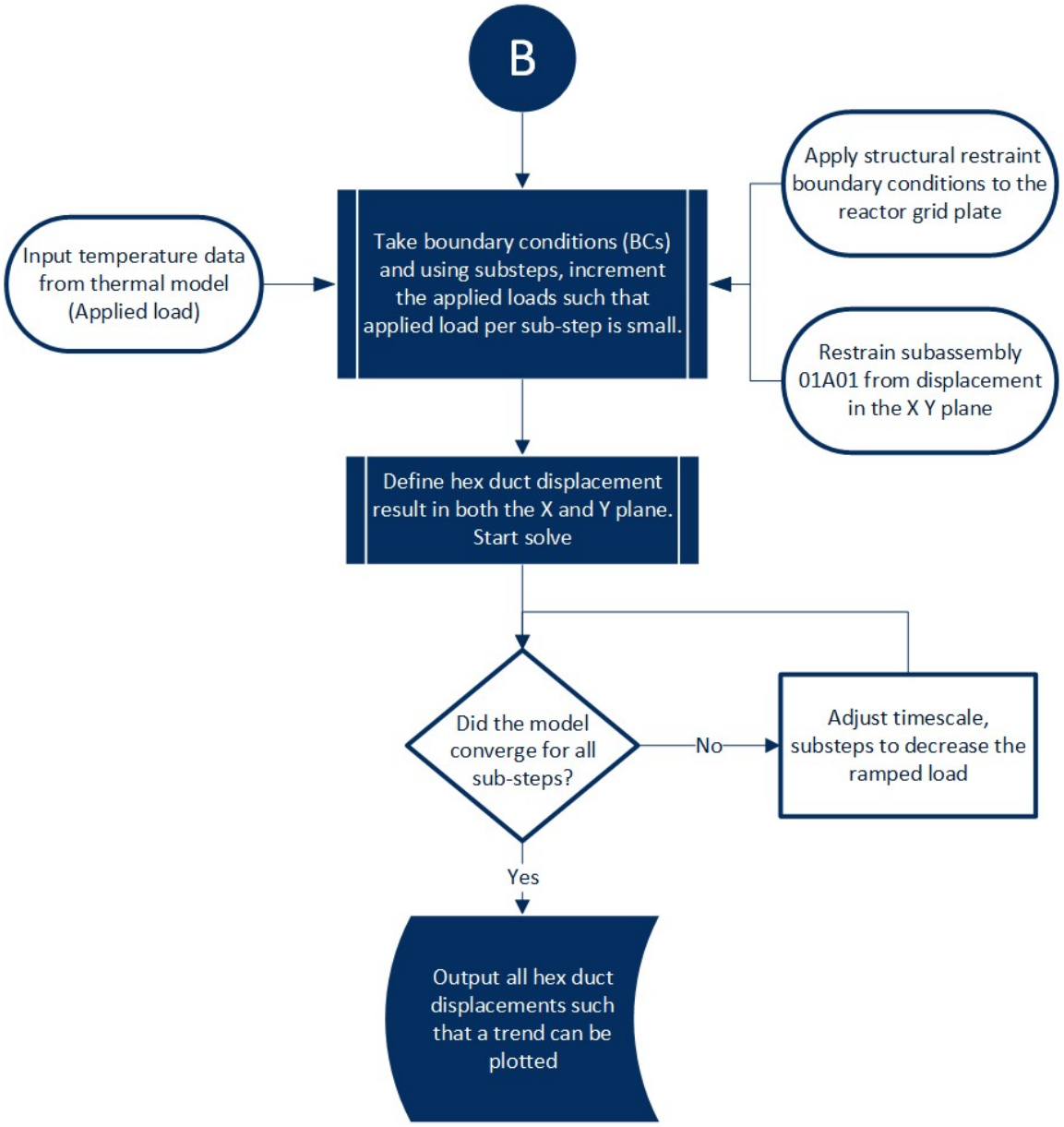

Figure 38. ANSYS structural model flowchart part 2.

6.2 The Katana Effect MCNP Model

6.2.1 MCNP Input Card Kcode Architect (MICKA)

The katana effect MCNP model of EBR-II run 138B was derived from a model used in an evaluated benchmark. [14] The major difference between the benchmark model and the katana effect model is the code which simulates a bow on the subassemblies.

A special program was written in MATLAB called MCNP Input Card KCODE Architect (MICKA) whose initial purpose was to simulate the katana effect, but later was purposed to perform perturbations for the evaluated benchmark. [15] More information on MICKA's benchmark capabilities is available in published work. [16] Figure 39 is a flowchart of the basic operations that MICKA performs to create an input file.

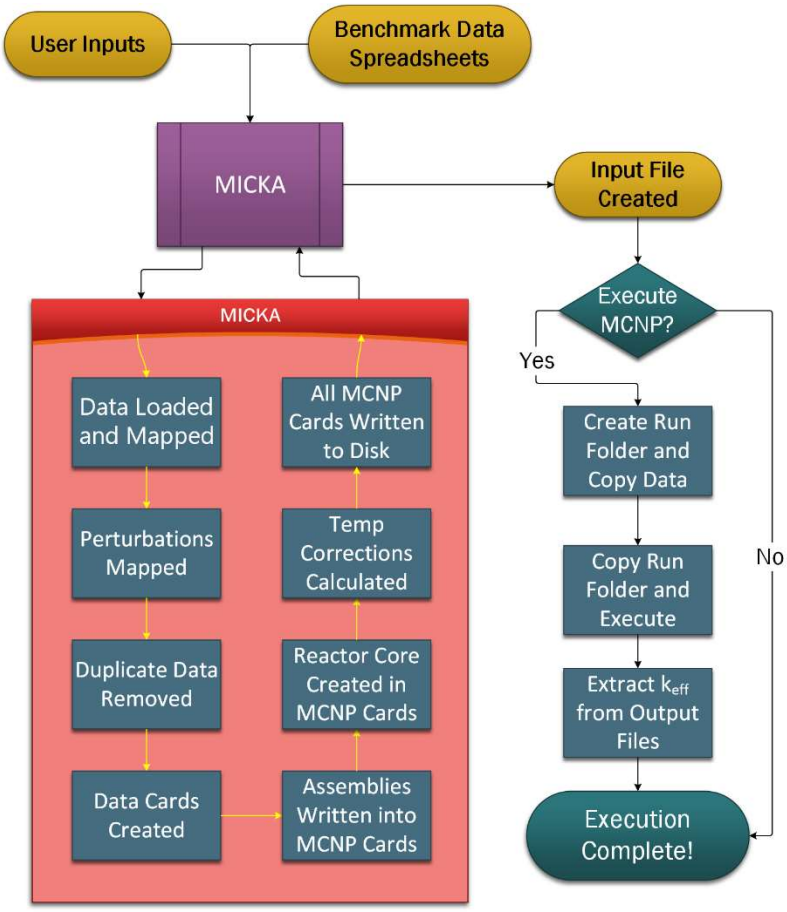

Figure 39. MICKA flowchart.

6.2.2 MCNP Katana Geometry Solution

As stated in section 5.1.3, the problem of MCNP's inability to model a bowed duct would need to be solved using axial slices. MICKA's initial purpose was to perform this slicing. The basic operation was that the core was sliced at the boundaries between the fuel slug sections and the boundary for the gas plenum leading to four distinct core parts. Every subassembly had the same axial slices. This led to every subassembly consisting of three

fueled parts and one part that consists of everything above the fuel. The database of ANSYS displacements was then modified to divide the displacement data per subassembly into the four axial parts. Node numbers and positions were sorted by the Z axis to determine where they were in relation to the slices. Each subassembly now had four sets of node numbers sorted by axial location. For each set of node numbers, an average displacement was calculated. This would simulate the katana effect by displacing each subassembly axial part by the net average displacement. With the part displacements sorted and calculated, MICKA could build the katana core.

The main difference between the katana core versus the benchmark core was the subassembly construction. Instead of a once-through process which writes the surface and cell cards, a loop is used and three subassemblies are constructed. Each displaced part is constructed as one whole displaced subassembly. Then after the three subassemblies are created, the correct parts of each subassembly are selected according to the axial slices. This process leaves only the bowed subassembly. Figure 40 shows a figure of the selected axial parts.

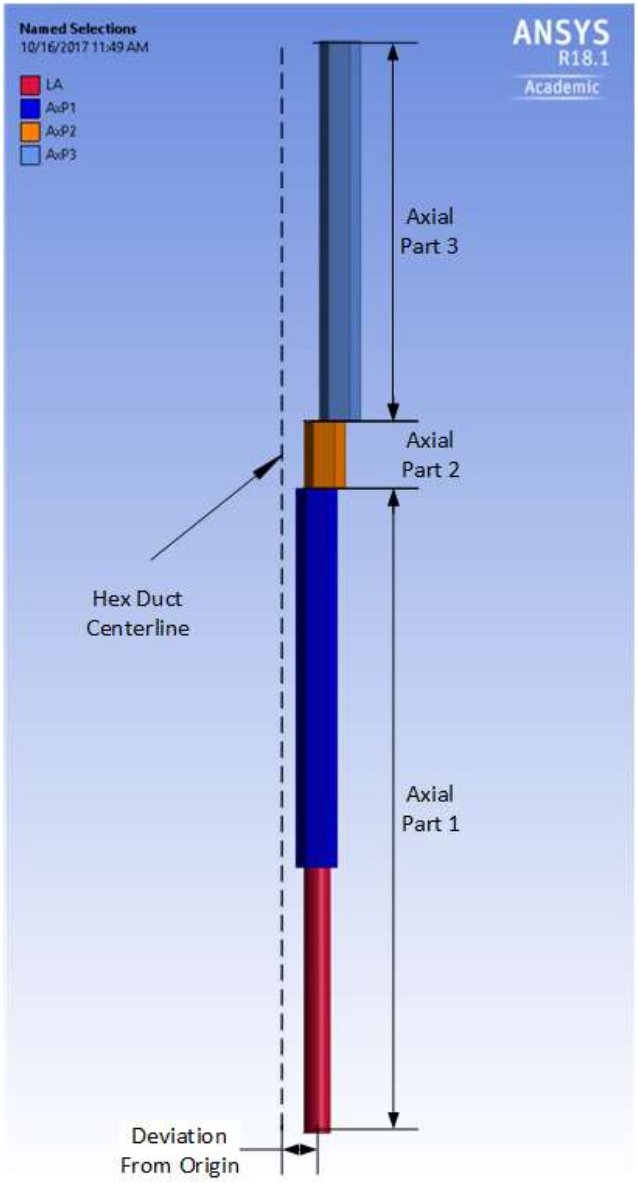

Figure 40. Axial slices representation of the katana effect MCNP model.

The displaced axial parts are then added to a lattice according to their axial height. In summary, three separate cores are created, each with the subassembly displacements modeled. The subassembly axial parts are then stacked to create the whole core. Figure 41 shows an example of the axial part movement in the MCNP model.

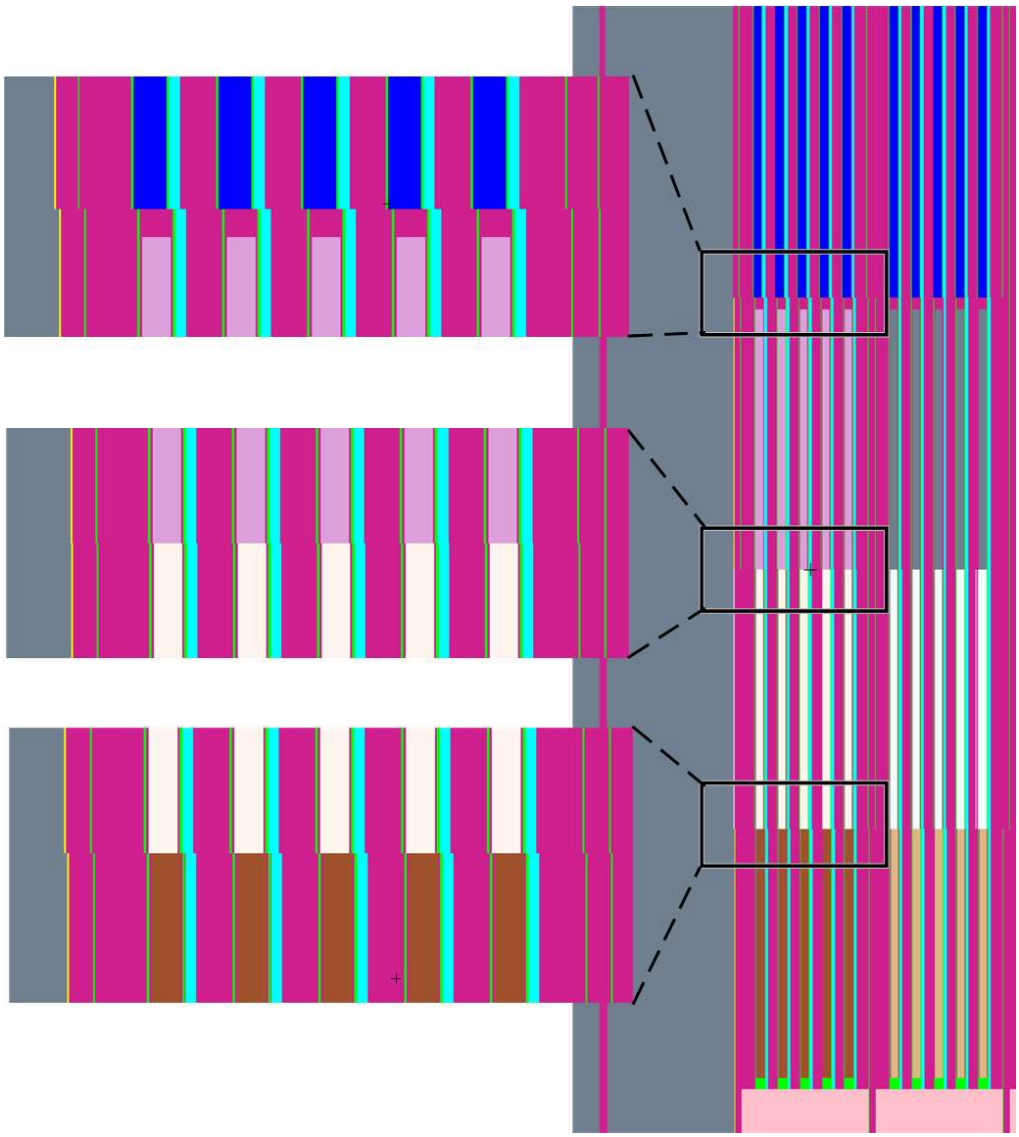

Figure 41. MCNP katana effect model axial part displacements.

Figure 42 shows three subassemblies which exhibit zero bowing. Using figure 42 as a reference, figure 43 shows how the subassemblies in the MCNP model move due to the katana effect. These figures are located at the top of the subassemblies where the greatest movement takes place.

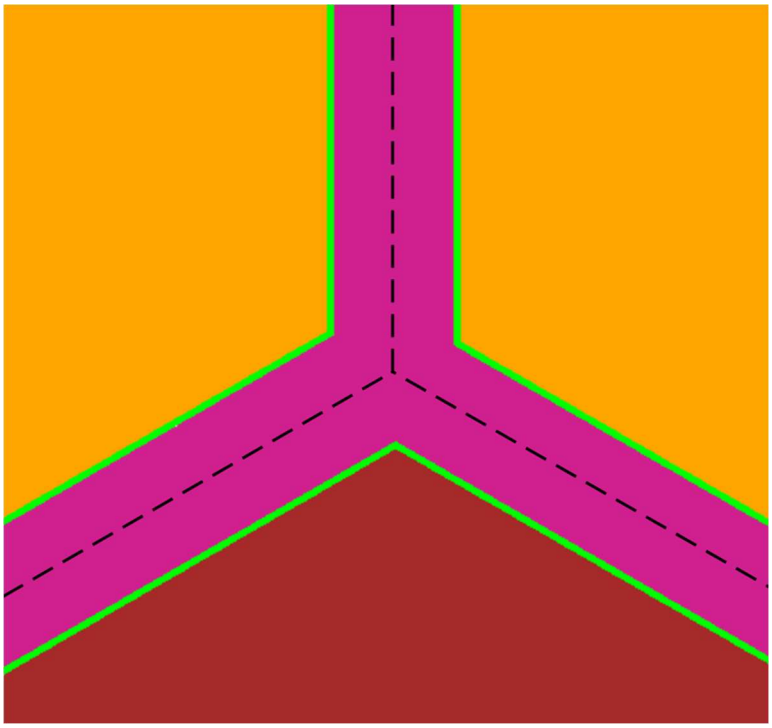

Figure 42. XY plot of no duct movement.

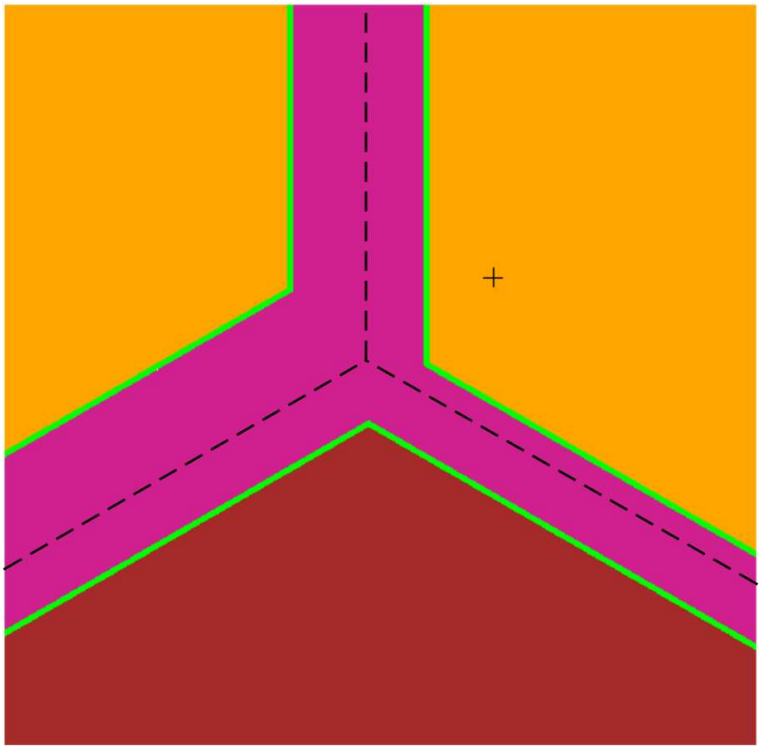

Figure 43. XY plot of duct movement.

The MCNP model of the katana effect was a series of static cores at various stages of EBR-II power-up. The power-up sequence was in two stages. The first stage was the initial heating of the core. This was required because run 138B was conducted beyond the freezing point of sodium ~370ºC. Initial heating was an important effect because the subassemblies had undergone some thermal expansion before initial power was applied. At zero power, the core had undergone significant geometric changes due to thermal expansion; modeling the katana effect would need to consider the base core configuration such that the subsequent flowering had the correct initial starting point. Figure 44 is a plot of the MCNP model of the katana effect at 100% power. The subassembly displacements are small (~1 mm) in comparison to the outer diameter of the duct.

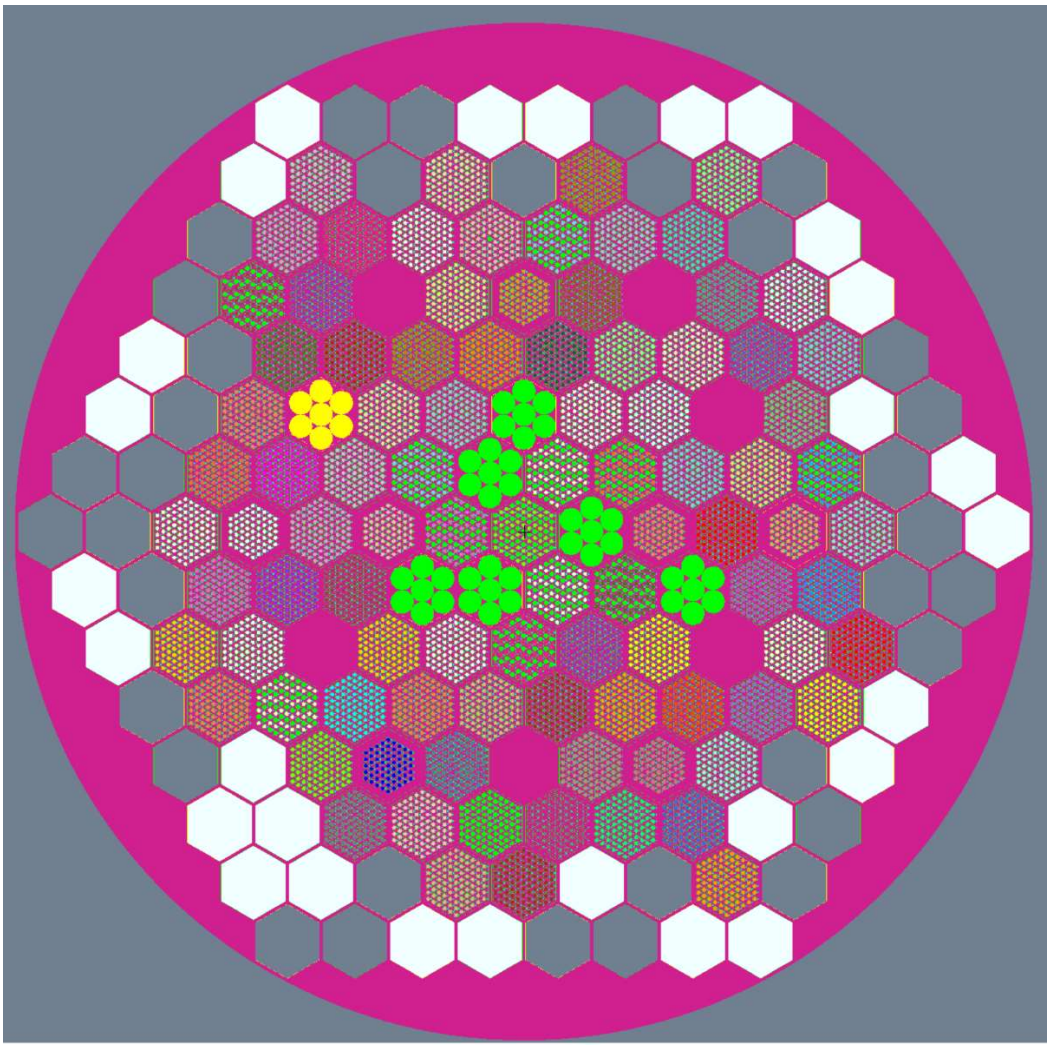

Figure 44. Plot of the MCNP katana effect at 100% power.

The second stage was where power was applied. It was applied linearly from 0% to 100% power. Various times were selected during the analysis where the deformations were exported and then imported into the MCNP model. The MCNP models were then executed and reactivity changes were calculated against the cold zero power core.

Table 11. Katana Effect MCNP Model Statistics at 100% Power

Property	Value
Cells	5485
Surfaces	4034
Materials	298
Max average x displacement	0.4197 cm ± 0.0026 max
Max average y displacement	0.3483 cm ± 0.0026 max
Lines of input	43787
Number of particles per generation	150,000
Number of generations	1030
Number of generations skipped	30

7.0 RESULTS

The following results calculate the temperature coefficient for each effect by calculating the reactivity change from a reference k_{eff}, then performing a linear least-squares-fit of temperature versus reactivity. The beta effective used was $\beta = 0.0068$. Inhours were calculated from % Δk/k using the relation $441\ Ih\ per\ \%\ \Delta k/k$. [17]

The equation used to calculate change in reactivity was:

$$\Delta\rho = \frac{k_1 - k_{ref}}{k_1 * k_{ref}} \tag{8}$$

The uncertainty of these results has three components. The first is the uncertainty in the temperature input. The temperature input uncertainty is unknown because of the dearth of data available. As was stated in section 2.3, the new method of ANSYS to MCNP has the same weakness of uncertainty of the input data. The next component is due to the stochastic uncertainty from MCNP which for the following results was 5e-5 on k_{eff}. The last component was the linear fit. The fit has the highest known uncertainty due to the spread of the MCNP results.

The ANSYS uncertainty is not applied to these results because the uncertainty at its maximum value is the convergence tolerance which in most cases is 1% of the applied load. That value is an order-of-magnitude higher than the values calculated for the out-of-balance condition on the final iteration. Using the maximum value would overestimate the uncertainty. The ANSYS katana effect model uncertainty was 1e-4 cm and was considered negligible given the size of the displacements.

7.1 Upper Part of the Reactor Grid Plate Feedback Reactivity Coefficient

The upper part of the reactor grid plate expansion simulation consisted of MCNP input files created for approximately every 32 ºC change in temperature. Only the geometry displacements were changed between each run. The reason was to isolate the thermal expansion and not to calculate the integrated effect due to multiple other expansions. The

reference value used was the room temperature model at 22 ºC. Table 12 shows a list of all the MCNP runs that were performed and their corresponding k_{eff} values.

Table 12. Upper Grid Plate Expansion MCNP Results

Temperature (ºC)	k_{eff}	MCNP k_{eff} uncertainty	Reactivity (% Δk/k)	Reactivity ($)	Reactivity (Ih)
22	0.99754	0.00005	NA	NA	NA
54	0.99724	0.00005	-0.0302	-0.044	-13.2993
86	0.99700	0.00005	-0.0543	-0.080	-23.9446
119	0.99647	0.00005	-0.1076	-0.158	-47.4709
151	0.99619	0.00005	-0.1359	-0.200	-59.9101
183	0.99600	0.00005	-0.1550	-0.228	-68.3549
215	0.99563	0.00005	-0.1923	-0.283	-84.8093
247	0.99534	0.00005	-0.2216	-0.326	-97.7146
280	0.99509	0.00006	-0.2468	-0.363	-108.8459
312	0.99480	0.00005	-0.2761	-0.406	-121.7652
344	0.99444	0.00005	-0.3125	-0.460	-137.8134

Figure 45 and 46 show the k_{eff} results plotted. The error bars for both figures are the stochastic uncertainties from MCNP on k_{eff}.

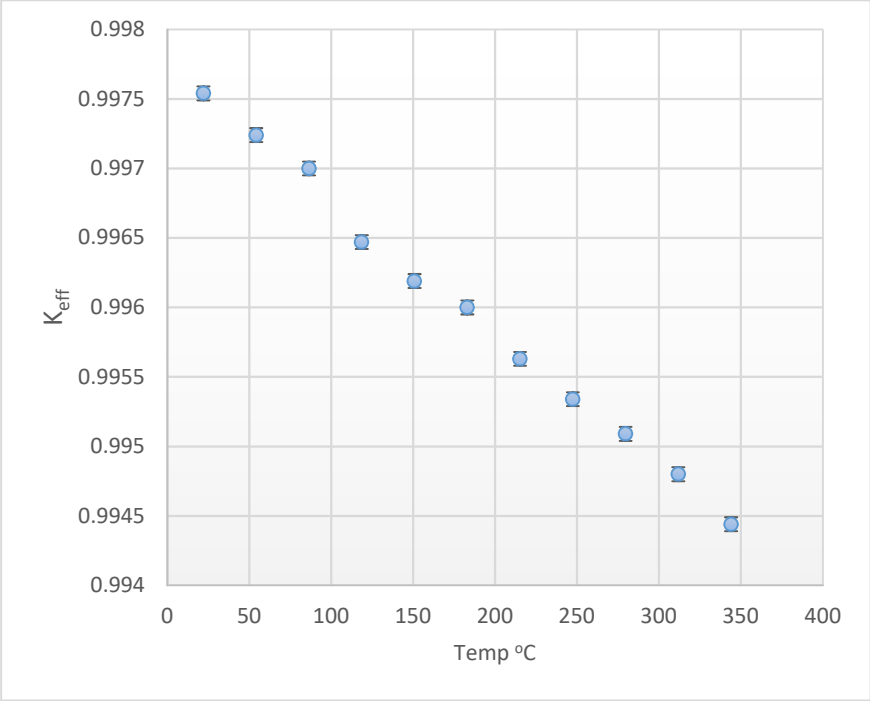

Figure 45. MCNP results for the reactor grid plate expansion.

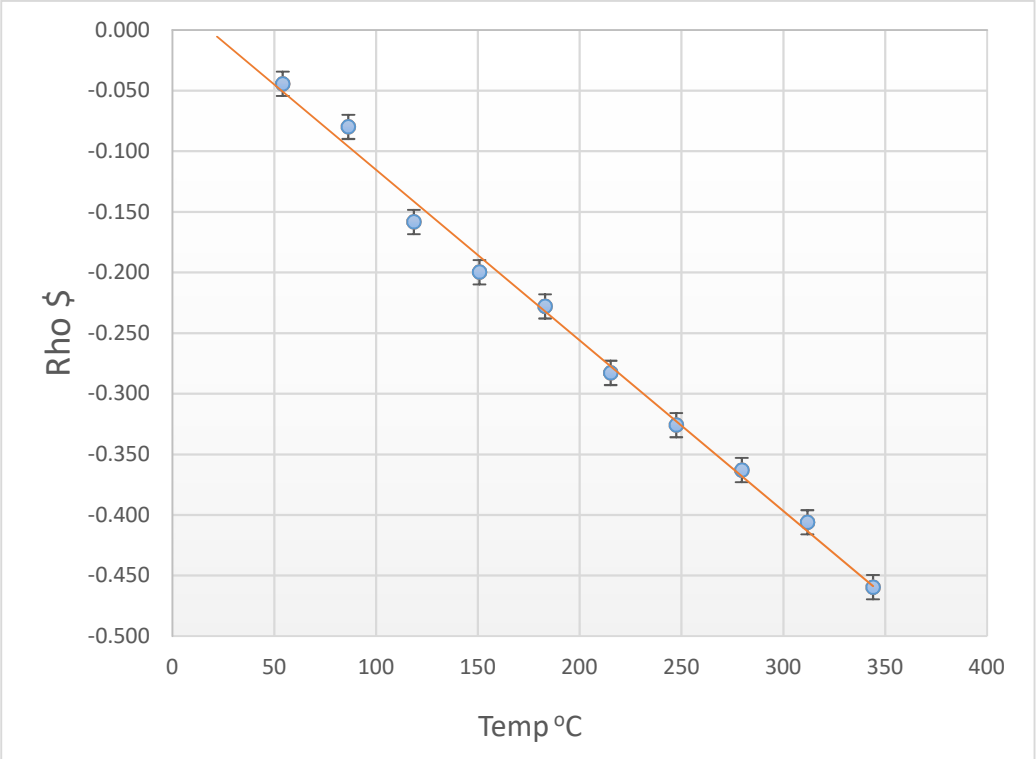

Figure 46. Upper grid plate expansion feedback reactivity coefficient.

Using a linear least-squares-fit of the data in figure 46, the upper grid plate feedback reactivity coefficient was calculated in Table 13.

Table 13. Upper Grid Plate Expansion Reactivity Coefficient Comparison

Method	Feedback coefficient $(10^{-4}\ \$/\ ^{o}C)$	Estimated uncertainty (%)
Values used for run 138B [2]	-14.5	20
ANSYS coupled MCNP	-14.1	2.6

Table 13 shows that the ANSYS MCNP method of calculating the reactivity coefficient agrees with the established value.

7.2 Katana Effect Temperature Coefficient

The katana effect MCNP models were created for every 3 ºC to 4 ºC change in temperature. Like the upper grid plate expansion calculation, only the displacements were changed for each run. The reference value used to calculate the reactivities was the displacements associated at 344 ºC. Table 14 shows the MCNP results of the katana effect.

Table 14. Katana Effect MCNP Results

Temperature (°C)	k_{eff}	MCNP k_{eff} uncertainty	Reactivity (% Δk/k)	Reactivity ($)	Reactivity (Ih)
344	0.99444	0.00005	NA	NA	NA
347	0.99434	0.00005	-0.0101	-0.0149	-4.4599
351	0.99442	0.00005	-0.0020	-0.0030	-0.8919
354	0.99438	0.00006	-0.0061	-0.0089	-2.6758
358	0.99427	0.00005	-0.0172	-0.0253	-7.5824
361	0.99434	0.00005	-0.0101	-0.0149	-4.4599
365	0.99424	0.00005	-0.0202	-0.0297	-8.9207
368	0.99421	0.00005	-0.0233	-0.0342	-10.2591
372	0.99413	0.00005	-0.0314	-0.0461	-13.8286
375	0.99408	0.00005	-0.0364	-0.0536	-16.0598
379	0.99409	0.00005	-0.0354	-0.0521	-15.6136
382	0.99411	0.00005	-0.0334	-0.0491	-14.7211
386	0.99403	0.00005	-0.0415	-0.0610	-18.2913
389	0.99405	0.00005	-0.0395	-0.0580	-17.3987
393	0.99396	0.00005	-0.0486	-0.0714	-21.4157
396	0.99393	0.00005	-0.0516	-0.0759	-22.7549
400	0.99396	0.00005	-0.0486	-0.0714	-21.4157
403	0.99385	0.00005	-0.0597	-0.0878	-26.3264
407	0.99382	0.00005	-0.0627	-0.0923	-27.6658
410	0.99376	0.00005	-0.0688	-0.1012	-30.3450
414	0.99372	0.00005	-0.0729	-0.1071	-32.1313

The displacements at 344 °C represent the zero-power hot thermal expansion of the core. Not including these initial displacements would over-estimate the katana effect. The thermal expansion essentially is a pitch change leading to an increase in the gap between subassemblies. Increasing this gap changes the katana effect reactivity coefficient by allowing more bowing to take place before initial contact occurs between the ducts. More bowing before contact would mean an increase in the positive reactivity effect leading to

an overall decrease in the reactivity effect. Not including the extra gap would decrease positive reactivity and would allow for more flowering to take place. Figure 47 is a plot of the data shown in table 14. Figure 48 shows the calculated reactivities versus temperature. The error bars for both figures are the stochastic uncertainties from MCNP on k_{eff}.

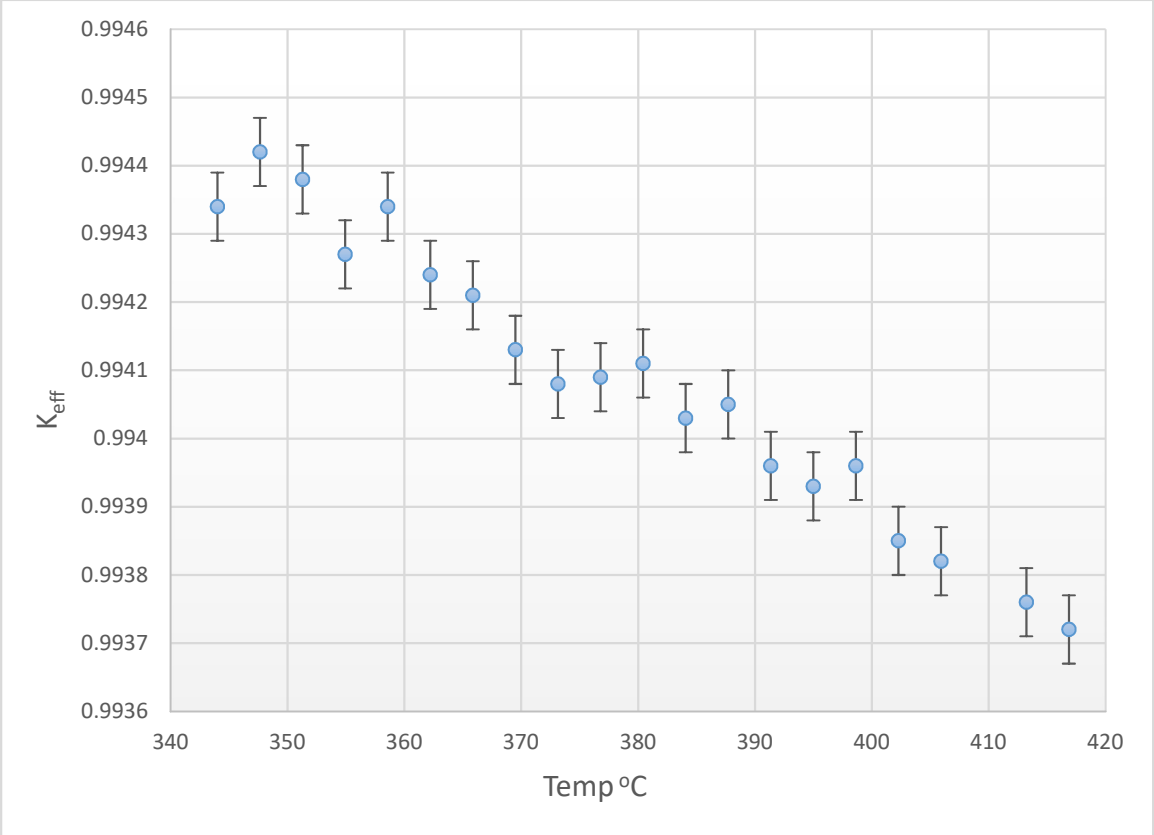

Figure 47. Katana effect MCNP results.

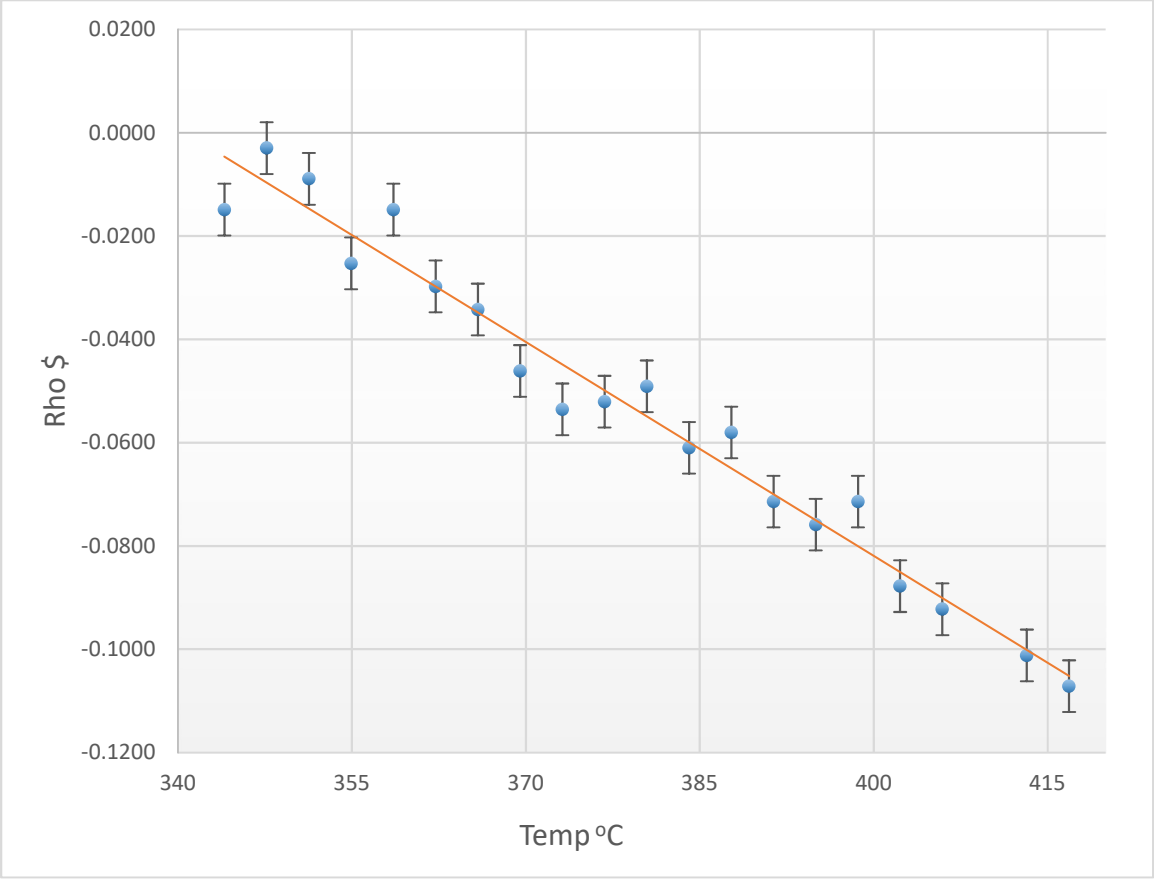

Figure 48. Katana effect reactivity results.

Using a linear least-squares-fit of the data in figure 48, the katana effect feedback reactivity coefficient was calculated in table 15.

Table 15. Katana Effect Reactivity Coefficient Comparison

Method	Feedback coefficient (10^{-4} \$/ ^{o}C)	Estimated uncertainty (%)
ANSYS coupled MCNP	-14.5	4.4

A definitive result for the katana effect reactivity feedback coefficient does not exist, but observances on the results data can be made. One of the measured observances from the older EBR-II runs was the positive reactivity caused by duct-bowing. The results

in table 14 show in temperatures 344 ^{O}C and 351 ^{O}C there is little change in k_{eff} while surrounding temperatures show large changes. The small changes could be due to the overall changes in k_{eff} are smaller than the stochastic uncertainty of MCNP. Compared to the higher power data whose trend is more linear, the separation in the data points could contain a positive reactivity between the temperature points. No information is available at which power the positive reactivity occurred, but it was known to occur at lower power.

8.0 CONCLUSION

The ANSYS coupled MCNP method provide superior analysis and quantification of the katana effect; however, this method suffers from the same issues as previous ones; mainly, the requirement of accurate and highly specific input data. The primary improvement has been the structural expansion. The specificity of the duct movements and their subsequent mechanical stresses has increased in orders-of-magnitude compared to older techniques. The method has also been performed using codes which are not custom codes for the work. ANSYS coupled with MCNP is important because future development of this method should not be limited to a custom code which is not always available.

Continued work for this method would be creation of a computational fluid dynamics (CFD) model to address the simplification of the thermal input. A CFD model would dramatically improve the quality of the katana effect result because the temperature data would be more accurate for local effects. The temperature data would also have a superior verification because more measured thermal hydraulic data was available during run 138B.

A second improvement would be to include the first row of blankets outside of the reflector. The blanket would have power calculated from a thermal hydraulic model such that the power would be included in the ANSYS thermal model. Including the blanket row would enable the simulation of the inverted bow at row 8 discussed in section 2.2.

The advances of the work done by EBR-II have given three decades has resulted in a great worth of information to the growth of studying liquid metal fast reactors. The amount of information learned was so prodigious that it was not always recorded. The lack of data has caused the coupling of ANSYS with MCNP verification to be weak, but the results calculated, along with the GODIVA-IV work, show the value of using these new methods to seek the reactivity changes due to structural displacement.

9.0 REFERENCES

[1] C. L. Pope, "Experimental Breeder Reactor II Benchmark Evaluation," Idaho State University, Pocatello, ID, 2015.

[2] D. Mohr, L. Chang, E. Feldman, P. Betten and H. Planchon, "Loss-of-primary-flow-without-scram tests: Pretest predicitions and preliminary results," *Nuclear Engineering and Design,* vol. 101, no. 1, pp. 45-56, 1987.

[3] F. Metcalf, J. Natoce, W. T. F. Rupp and E. Hutter, in *EBR-II System Design Descriptions*, vol. II Primary System, Argonne, Argonne National Laboratory, 1971, pp. 2.1-1 - 2.12-21.

[4] L. J. Koch, Experimental Breeder Reactor-II (EBR-II) An Integrated Experimental Fast Reactor Nuclear Power Station, Argonne: Argonne National Laboratory, 1987.

[5] W. B. Loewenstein, "The Physics Design of the EBR-II," Argonne National Laboratory, Argonne Il, July 1961.

[6] J. P. Finck, "A Technique for Computing the Reactivity Feedback Due to Core-Assembly Bowing in LMFBR's," Argonne National Laboratory, Argonne, 1987.

[7] D. A. Kucera and D. Mohr, "BOW-V: A CDC-3600 Program to Calculate the Equilibrium Configurations of a Thermally Bowed Reactor Core," Argonne National Laboratory, Argonne, 1970.

[8] S. Moaveni, Finite Element Analysis, Theory and Application with ANSYS, Minnesota State University, Mankato: Prentice Hall, 1999.

[9] S. A. Dupree and S. K. Fraley, A Monte Carlo Primer, a Practical Approach to Radiation Transport, Albuquerque NM: Kluwer Academic / Plenum Publishers, 2002.

[10] B. F. Brown, "Fundamentals of Monte Carlo Particle Transport," in *Lecture note for Monte Carlo Course*, Los Alamos, 2014.

[11] International Handbook of Evaluated Criticality Safety Benchmark Experiments, INEA/NCS/DOC(2006)1, NEA No. 7258: OECD-NEA, 2015.

[12] J. M. Goda, J. A. Bounds, W. L. Myers and R. G. Sanchez, "Module 8: Godiva-IV Critical Assembly Demonstration, LA-UR-13-28223," LANL, Los Alamos, 2013.

[13] W. van Rooijen and H. Mochizuki, "Analysis of the EBR-II SHRT-45R Unprotected Loss of Flow," *Science and Technology of Nuclear Installations,* vol. 2015, p. 14, 2015.

[14] E. Lum, C. Pope, R. Stewart, B. Byambadorj and Q. Beaulieu, "Evaluation of Run 138B At Experimental Breeder Reactor II, a Prototypic Liquid Metal Fast Breeder Reactor," International Reactor Physics Experiment Evaluation Project (IRPhEP), (Submitted for Publication).

[15] *MATLAB The Language of Technical Computing,* MathWorks, 2016.

[16] R. Stewart, E. Lum and C. Pope, "MCNP Input Card and Kcode Architect (M.I.C.K.A)," *Annals of Nuclear Energy,* vol. TBD, no. TBD, p. TBD, (Submitted for Publication) 2017.

[17] D. Meneghetti and D. A. Kucerfa, "Comparisons of PRD Componenets for Various EBR-II Configurations," in *ANS Topical Meeting on Advances in Reactor Physics*, Saratoga Springs, New York, 1986.

[18] R. L. McVean and et al, "EBR-II Dry Critical Experiments," Argonne National Laboratory, Argonne, 1962.

[19] P. R. Betten, R. M. Singer, M. J. Lee, E. E. Feldman, L. K. Chang, D. Mohr and H. P. Planchon, "Conceptual Design Basis and Temperature Predictions in a Simulated Instrumented LMFBR Blanket Subassembly," in *Third International Meeting on Reactor Thermal Hydraulics*, Newport, Rhode Island, 1985.

[20] C. L. Pope and M. J. Lineberry, "Comparison of Measured and Monte Carlo Results for a Neutron Beam Transmission Through an Irradiated Nuclear Fuel Assembly," *Nuclear Technology,* 2013.

[21] Argonne National Laboratory, "EBR-II Specification E0288-002-SF-06".

[22] J. K. Fink and L. Leibowitz, "Thermodynamic and Transport Properties of Sodium Liquid and Vapor," Argonne National Laboratory, Argonne, 1995.

[23] MatWeb, "MatWeb Material Property Data," MatWeb, LLC, 2017. [Online]. Available: http://www.matweb.com/tools/contents.aspx#reference. [Accessed 29 09 2017].

[24] "ANSYS® Academic Research Mechanical, Release 18.1, Help System, Theory Reference, ANSYS, Inc.".

[25] E. Lum and C. Pope, "GODIVA-IV Reactivity Temperature Coefficient Calculation Using Finite Element and Monte Carlo Techniques," *Nuclear Engineering and Design,* vol. TBD, no. TBD, p. TBD, (Submitted for Publication) 2017.

APPENDIX A GODIVA-IV ANSYS GEOMETRY IMPORT FILE

```
Godiva-IV  HEU-MET-FAST-086  Case 5 PSC  BR In  LDN Masses  Detailed  E-VI  RN2
c  Core
c  Spindle
1        1    0.088296      1   -3    6   -7          $ Spindle Bottom
2        1    0.088296      1   -4    7   -8          $ Spindle Bottom Middle
3        1    0.088296      1   -4    8   14  -39     $ Spindle Bottom Indented
4        1    0.088296      1   -5  -10   39          $ Spindle Top Middle
5        1    0.088296      1   -5   10  -11   15     $ Spindle Top Indented
6        1    0.088296      1   -5   11  -12          $ Spindle Top Part 1
7        1    0.088296      2   -5   12  -13          $ Spindle Top Part 2
c  Intermediate Inner Subassembly Plate
8       14    0.046841     16  -18   20  -21          $ ISP Bottom Region
9       14    0.046841      4  -18   21  -22          $ ISP Lwr Middle Region
10       0                -19  -38   54   56          $ Align Pin Hole Cylinder
11       0                -19  -29  -56  #23          $ Align Pin Hole Tip Gap
12      14    0.046841      4  -17   22  -24   26
                         #10  #11  #22  #23          $ ISP Top Middle Region
13      14    0.046841      4  -17   19   24  -25
                         -27   28  #10               $ ISP Top Region
c  Top Intermediate Subassembly Plate
15      15    0.048380      4   19   25  -34  -37
                         -42   43   44               $ Lower Part of Top ISP
16      15    0.048380      4   19  -35   37  -38     $ Lwr Mdl Part of Top ISP
17      15    0.048380      4  -35   36   38  -39     $ Middle Part of Top ISP
18      15    0.048380      5  -35   36   39  -40     $ Upr Mdl Part of Top ISP
19      15    0.048380     33  -35   40  -41          $ Upper Part of Top ISP
c  Long Alignment Pin
20       0                -36  -40   57   58  -59     $ Slit in Align Pin Cap
21       1    0.088296    -36   38  -40  #20          $ Alignment Pin Cap
22       1    0.088296    -38  -54   56               $ Alignment Pin Barrel
23       1    0.088296     55  -56  -60               $ Alignment Pin Tip
c  Safety Block
24       0                 -4  -67   75   83          $ Lwr SB/Base Gap
25       0                -64   65  -66               $ Lwr Thermocouple Hole
26       0                -63   66  -68               $ Upr Thermocouple Hole
27       0                -62   65   70  -71  -72 -73 $ Safety Block Notch
28      13    0.047127      4  -62   64   65  -66
                         -73  #27                    $ Safety Block Bottom
29      13    0.047127      4  -62   63   66  -67     $ Safety Blk Lwr Middle
30      13    0.047127     61  -62   63   67  -68     $ Safety Blk Upr Middle
31      13    0.047127     61  -62   68  -69  -74     $ Safety Block Top
c  Safety Block Base
32       0                 -4  -87                    $ Indentation in SB Base
33       0                 -4   79  -80               $ SB Base Bottom Hole
34       0                -65   70  -71  -78   82
                         -84  -85                    $ SB Base Notch
35       1    0.088296      4  -76   79  -80          $ SB Base Bottom
36       1    0.088296    -76   80  -81               $ SB Base Solid Round
37       1    0.088296    -77   81  -82   86          $ Curved SB Base
38       1    0.088296    -65  -78   82  -85  #34     $ SB Base Platform
39       1    0.088296     -4   65  -83  #32          $ Lower SB Base Barrel
40       1    0.088296    -69  -75   83               $ Upper SB Base Barrel
c  Notches for Fuel Rings
41       0                -94  108 -117  123 -124
                         125 #591                    $ Right Notch, Rings 2-5
42       0                -94  108 -117 -141 -143 144 $ FL Notch, Rings 2-5
43       0                -94  108 -117 -142 -145 146 $ BL Notch, Rings 2-5
c  Fuel Ring 1 (101)
44       7    0.047898     90  -93  104 -105 -137
                         138 #125                    $ Middle, Right Pad Slot
45       7    0.047898     90  -93  -99  104 -105
                         137 #126                    $ Front End, Rt Pad Slot
46       7    0.047898     90  -93  -98  104 -105
                        -138 #127                    $ Back End, Rt Pad Slot
47       7    0.047898     90  -93  104 -105 -133
                        -134 #128                    $ Middle, FL Pad Slot
48       7    0.047898     90  -93 -100  104 -105
                         133 #129                    $ Rt End, FL Pad Slot
49       7    0.047898     90  -93 -102  104 -105
                         134 #130                    $ Lt End, FL Pad Slot
50       7    0.047898     90  -93  104 -105  136
                         135 #131                    $ Middle, BL Pad Slot
51       7    0.047898     90  -93 -101  104 -105
                        -136 #132                    $ Rt End, BL Pad Slot
52       7    0.047898     90  -93 -103  104 -105
                        -135 #133                    $ Lt End, BL Pad Slot
53       0                 -94  104 -108  123 -124
                         126 #199 #591               $ Right Notch, Ring 1
54       0                 -94  104 -108 -139 -143
                         144
55       0                 -94  104 -108 -140 -145
                         146
56 7 0.047898 89 -94 95 96 97 104 -105 #44  #45  #46  #47 #48 #49  #50  #51
     #52 #53  #54  #55 #125 #126 #127 #128 #129 #130 #131 #132 #133 #199 #591
57       7    0.047898     89  -94   95   96   97 105
                         -106  #53  #54  #55 #591 $ Middle Annulus, Ring 1
58       7    0.047898     92  -94   95   96   97
                         106 -108  #53  #54  #55
                        #591             $ Top Annulus, Ring 1
c  Fuel Ring 2 (102)
```

```
59       0                89  -92   95   96   97
                         106 -107                    $ Gap inside Ring 2
60       0                91  -92   95   96   97
                         107 -108                    $ Internal Gap, Rings 1, 2
61       8    0.048260    89  -91   95   96   97
                         107 -108                    $ Bottom Annulus, Ring 2
62       8    0.048260    89  -94   95   96   97
                         108 -109  #41  #42  #43 #591  $ Middle Annulus, Ring 3
63       8    0.048260    92  -94   95   96   97
                         109 -110  #41  #42  #43 #591  $ Top Annulus, Ring 2
c  Fuel Ring 3 (103)
64       0                91  -94   95   96   97
                         110 -111  #41  #42  #43
                         #591  $ Outer Edge Gap, Rng 2, 3
65       0                91  -92   95   96   97
                         109 -110                    $ Internal Gap, Rings 2, 3
66       9    0.048087    89  -91   95   96   97
                         109 -111                    $ Bottom Annulus, Ring 3
67       9    0.048087    89  -94   95   96   97
                         111 -112  #41  #42  #43
                         #591  $ Middle Annulus, Ring 3
68       9    0.048087    92  -94   95   96   97
                         112 -114  #41  #42  #43
                        #591             $ Top Annulus, Ring 3
c  Fuel Ring 4 (104)
69       0                91  -92   95   96   97
                         113 -114                    $ Gap, Rings 3 and 4
70       0                89  -92   95   96   97
                         112 -113                    $ Gap Inside Ring 4
721      0               -720  722 -723               $ Smaller Ring 4 TC Hole
722      0                -94 -721  723               $ Larger Ring 4 TC Hole
71      10    0.047905    89  -91   95   96   97
                         113 -114                    $ Bottom Annulus, Ring 4
72      10    0.047905    89  -94   95   96   97
                         114 -115  #41  #42  #43
                         #591  #721 #722  $ Middle Annulus, Ring 4
73      10    0.047905    89  -91   95   96   97
                         115 -116                    $ Top Annulus, Ring 4
c  Fuel Ring 5 (105)
74       0                91  -92   95   96   97
                         115 -116                    $ Gap inside Ring 5 Slot
75       0                -22   89  -92   95   96
                          97  116                    $ Gap inside Ring 5
76      11    0.047712   -22   92  -94   95   96
                          97  115  #41  #42  #43
                          #591                $ Bottom Annulus, Ring 5
77      11    0.047712    22   88  -94   95   96
                          97 -117  #41  #42  #43
                          #591               $ Middle Annulus, Ring 5
78      11    0.047712    88  -91   95   96   97
                         117  -37                    $ Top Annulus, Ring 5
c  Fuel Ring 6 (106)
79       0                91  -94   95   96   97
                         117 -118 #591                   $ Gap inside Ring 6 Slot
80       0               -37   91  -92  118          $ Gap, Rings 5 and 6
81       0               -37  -91  -95  118          $ Left Hole above C/B Rod
82       0               -37  -91  -96  118          $ Back Hole above C/B Rod
83       0               -37  -91  -97  118          $ Right Hole above C/B Rod
84       0               -94  118 -122  123 -124
                         126 #201 #591               $ Right Notch, Ring 6
85       0               -94  118 -122 -139 -143 144 $ Fr Left Notch, Ring 6
86       0               -94  118 -122 -140 -145 146  $ Bk Left Notch, Ring 6
87       0                90  -93  121 -122 #139     $ Shallow Indent, Ring 6
88      12    0.048206   -37   92  -94  118
                         #84  #85  #86 #591 $ Bottom Annulus, Ring 6
89      12    0.048206    37   89  -94 -121  #84
                         #85  #86 #136 #137 #138
                         #201 #591 $ Middle Annulus, Ring 6
90      12    0.048206    89  -90  121 -122          $ Inr Top Annulus, Ring 6
91      12    0.048206    93  -94  121 -122
                         #84  #85  #86 #201 #591  $ Otr Top Annulus, Ring 6
c  Control and Burst Rods
92       0               -95  166 -118               $ Hole above Control Rod 1
93       0               -95  148  156 -166          $ Gap between CR1 & Rings
94       0              -147  164 -166               $ Top Hole in CR1
95       0              -148 -166  168 -170  171     $ Control Rod 1 Notch
96       0              -148  156  174               $ Gap for Lower CR1 Slant
97       0              -148 -166  175  #95          $ Gap for Upper CR1 Slant
98      16    0.048057   147 -148  156 -159 -174     $ Bottom Region, CR1
99      16    0.048057  -148  159 -164               $ Solid Middle Region, CR1
100     16    0.048057   147 -148  164 -166 -175 #95 $ Top Region, CR1
101      0               -97  167 -118               $ Hole above Control Rod 2
102      0               -97  153  157 -167          $ Gap between CR2 & Rings
103      0              -152  165 -167               $ Top Hole in CR2
104      0              -153 -167  169 -172  173     $ Control Rod 2 Notch
105      0              -153  157  179               $ Gap for Lower CR1 Slant
106      0              -153 -167  180 #104          $ Gap for Upper CR1 Slant
107     16    0.048057   152 -153  157 -160 -179     $ Bottom Region, CR2
108     16    0.048057  -153  160 -165               $ Solid Middle Region, CR2
109     16    0.048057   152 -153  165 -167 -180 #104  $ Top Region, CR2
110      0               -96  163 -118               $ Hole above Burst Rod
111      0               -96  150  162 -163          $ Gap, BR Pin Head & HEU
112      0               -96  149  161 -162          $ Gap, BR Pin Cyl & HEU
113      0               -96  151  104 -161          $ Gap between BR & Rings
114      0               229 -96  #116 #117
```

```
                          -104 $ Gap for Lower BR Slant
115      0                -151 -161  177                $ Gap for Upper BR Slant
116     17    0.048941     149 -151  154 -155 -176     $ Bottom Region, Burst Rod
117     17    0.048941    -151  155 -158  178          $ Middle Region, BR
118     17    0.048941     149 -151  158 -161 -177     $ Top Region, Burst Rod
119      0                -158 -178                    $ Tip of BR Pin Hole
120      2    0.087654    -149  158 -162                $ Cylinder, BR Pin
121      2    0.087654    -150  162 -163                $ Head, BR Pin
122      0                -147  156 -159                $ Bottom Hole, CR1
123      0                -152  157 -160                $ Bottom Hole, CR2
c  Support Pads
125      2    0.087654    -105 -137  138  181 -182
                           189                          $ Middle, Right Pad
126      2    0.087654    -105  137  181 -182 -184
                           189                          $ Back End, Right Pad
127      2    0.087654    -105 -138  181 -182 -183
                           189                          $ Front End, Right Pad
128      2    0.087654    -105 -133 -134  181 -182
                           189                          $ Middle, Front Pad
129      2    0.087654    -105  133  181 -182 -185
                           189                          $ Right End, Front Pad
130      2    0.087654    -105  134  181 -182 -187
                           189                          $ Left End, Front Pad
131      2    0.087654    -105  135  136  181 -182
                           189                          $ Middle, Back Pad
132      2    0.087654    -105 -136  181 -182 -186
                           189                          $ Right End, Back Pad
133      2    0.087654    -105 -135  181 -182 -188
                           189                          $ Left End, Back Pad
136      2    0.087654      90  -93  120 -121  127
                          -128                          $ Left Deep Indent, Ring 6
137      2    0.087654      90  -93  120 -121 -129
                           130                          $ Bk Rt Deep Indent, Ring 6
138      2    0.087654      90  -93  120 -121  131
                          -132                          $ FR Rt Deep Indent, Ring 6
139      2    0.087654    121  190 -191 -192          $ Solid Bearing Ring
c  Subassembly Cover Plate
140      0                   5   41 -193 -199          $ Central Hole in SCP
141      0                  41  -89  193 -198          $ Inner Void in SCP
142      0                -194  198 -199                $ Back Hole in SCP
143      0                -195  198 -199                $ Right Hole in SCP
144      0                -196  198 -199                $ Front Hole in SCP
145      0                -197  198 -199                $ Left Hole in SCP
146      2    0.087654      89  122  -90 -199          $ Edge of SCP
147      2    0.087654     -89  193  194  195  196
                           197  198 -199                $ Center of SCP
c Mounting Plate
148      0                -200  226 -229                $ Empty Central Hole
149      0                -201  226 -229          $ Empty Hole for CR1
150      0                -202  226 -229          $ Empty Hole for Burst Rod
151      0                -203  226 -229          $ Empty Hole for CR2
152      0                -204  226 -229                $ Empty Right Hole
153      0                -205  226 -229                $ Empty Front Left Hole
154      0                -206  226 -229                $ Empty Back Left Hole
176      4    0.059890     200  201  202  203  204
                           205  206
                           226 -229  232 -233  234
                           235 -236 -237
c  Right Clamp Support (Leg)
181      0                 248  123 -124  257 -258
                           269 -189                     $ Upper Inset below Clamp
182      0                -243 -248  257 -258          $ Inset Bottom for Rt Leg
192      1    0.088296     248 -251  252 -123  257
                          -258                          $ Left Side of Insert
193      1    0.088296     248 -251  124 -256  257
                          -258                          $ Right Side of Insert
195      1    0.088296     229 -249  252 -256  258
                          -263                          $ Upr Mdl Region, Rt Leg
c  Right Clamp
198      0                 123 -124 -267 -189          $ Void in Btm Inside Crnr
199      0                 123 -124 -267  284          $ Void in Btm Inside Crnr
200      0                 123 -124 -268  192          $ Void in Top Inside Crnr
201      0                 123 -124 -268  284          $ Void in Top Inside Crnr
204      1    0.088296    -244 -258  285                $ Locking Bolt
205      0                -271 -258  285                $ Hole for Long Roll Pin
206      3    0.083675     123 -124 -258  267 -269
                          -189  284                     $ Bottom of Clamp Back
591      3    0.083675     123 -124 -258  267  268
                           189 -192  284 -285          $ Middle of Clamp Back
592      3    0.083675     123 -124 -258  267  268
                           189 -192  285  244  271     $ Middle of Clamp Back
208      3    0.083675     123 -124 -258  268 -270
                           192  284                     $ Top of Clamp Back
209      3    0.083675     123 -124  267 -269 -189
                           282 -284                     $ Bottom Prong
210      3    0.083675     123 -124  267  189 -276
                           283 -284                     $ Crnr Fill for Btm Prong
211      3    0.083675     123 -124  268 -270  192
                           282 -284                     $ Top Prong
212      3    0.083675     123 -124  268  277 -192
                           283 -284                     $ Crnr Fill for Top Prong
c  Belly Band
216      1    0.088296     286 -287  288 -251          $ Solid Belly Band
c  Nuts and Bolts
217      1    0.088296       5  199 -292 -293          $ Nut on Spindle
```

```
218      1    0.088296    261 -294 -295              $ Cap of Rt Locking Bolt
c
c *************** The following cell cards were added by me *****************
c ****** to remove the ridiculous detail in the thumb screw cards ************
c ***********************from the Right clamp ******************************
c
504     1   0.088296     249   252 -256  258
                        -261  -251                    $ Btm Top Region, Rt Leg
289     1   0.088296    -286   261 -245

c  Surfaces for Spindle
1     cz   0.31750                              $ Glory-Hole Radius
2     cz   0.714375                             $ Glory-Hole Top Radius
3     cz   1.15570                              $ Spindle Bottom OR
4     cz   1.27000                              $ Spindle Bottom Middle OR
5     cz   1.42875                              $ Spindle Top Middle OR
6     pz   0.00254                              $ Bottom of Spindle
7     pz   1.59004                              $ Top of Spindle Bottom
8     pz   5.71754                              $ Top of Spindle Bot Middle
10    pz   8.89254                              $ Top of Spindle Top Middle
11    pz   9.21004                              $ Bottom of Spindle Top
12    pz  10.79754                              $ Bottom of Wide Glory Hole
13    pz  11.43254                              $ Top of Spindle
14    tz   0.0  0.0  5.87629  1.27000  0.15875  0.15875  $ Bottom Spindle Torus
15    tz   0.0  0.0  9.05129  1.42875  0.15875  0.15875  $ Top Spindle Torus
c  Surfaces for Intermediate Inner Subassembly Plate
16    cz   1.15951                              $ OR of Bottom IISP Hole
17    cz   3.92430                              $ OR of Upper Int Sub Plate
18    cz   4.38150                              $ OR of Lower Int Sub Plate
19    c/z  2.54000  0.0  0.31750                $ OR of Alignment Pin Hole
20    pz   0.0                                  $ Bottom of Inter Inner SA Pl
21    pz   1.27000                              $ Top of Bottom IISP Hole
22    pz   2.54000                              $ Top of IISP Wide Region
24    pz   3.27422                              $ Bot of IISP Hole Slant
25    pz   3.49250                              $ Top of Int Inr Subassby Plate
26    tz   0.0  0.0  2.619375  3.92430  0.079375  0.079375 $ Int Sub Plate Torus
27    kz   7.25805   1  -1                     $ Outer Slant for Int ISP
28    kz   2.06375   1   1                     $ Inner Slant for Int ISP
29    k/z  2.54000  0.0  2.5400   1  1         $ Tip of Alignment Pin Hole
c  Surfaces for Top Inner Subassembly Plate
33    cz   3.50520                              $ Upper IR of UISP
34    cz   3.94208                              $ Lower OR of UISP
35    cz   4.415155                             $ Upper OR of UISP
36    c/z  2.54000  0.0  0.396875               $ OR of Alignment Pin Cap Hole
37    pz   4.96570                              $ Top of UISP Indentation
38    pz   6.02361                              $ Bottom of Alignment Pin Cap
39    pz   6.03504                              $ Bottom of UISP Middle IR
40    pz   6.98754                              $ Bottom of UISP Inner Annulus
41    pz   7.64794                              $ Top of Top Inr Subasmbly Pl
42    kz  -0.29083   1   1                     $ Outer Slant for Top ISP
43    kz   4.92125   1  -1                     $ Inner Slant for Top ISP
44    tz   0.0  0.0  4.80695  3.94208  0.15875  0.15875  $ Torus for Top ISP
c  Surfaces for Long Alignment Pin
54    c/z  2.54000  0.0  0.31369                $ OR of Alignment Pin Barrel
55    pz   2.806065                             $ Bottom of Alignment Pin
56    pz   2.88544                              $ Bottom of Align Pin Barrel
57    pz   7.568565                             $ Bottom of Slit in Align Pin
58    px  -0.079375                             $ Left Edge of Align Pin Slot
59    px   0.079375                             $ Right Edge of Align Pin Slot
60    k/z  2.54000  0.0  2.57175  1  1          $ Tip of Alignment Pin
c  Surfaces for Safety Block
61    cz   1.11125                              $ SB Hole Upper Radius
62    cz   4.29260                              $ Safety Block Outer Radius
63    c/z  3.97510  0.0  0.08509                $ Smaller SB TC Outer Radius
64    c/z  3.97510  0.0  0.15588                $ Larger SB TC Outer Radius
65    pz  -7.87400                              $ Bottom of Safety Block
66    pz  -6.60400                              $ Interface between TC ORs
67    pz  -5.01650                              $ Interface between SB Hole ORs
68    pz  -0.73025                              $ Top of Thermocouple Hole
69    pz  -0.25400                              $ Top of Safety Block
70    py  -0.238125                             $ Front of Safety Block Notch
71    py   0.238125                             $ Back of Safety Block Notch
72    p    1.0    0.0    1.0    -11.63960       $ Bottom of Safety Block Notch
73    kz -12.00785   1   1                     $ Slant for Safety Block Bottom
74    kz  -0.11511   130.646     -1            $ Slant for Saftey Bock Top
c  Surfaces for Safety Block Base
75    cz   1.10871                              $ OR of Upper SB Base
76    cz   1.90500                              $ OR of SB Base Bottom
77    cz   2.54000                              $ OR of Platform Curve
78    cz   3.65760                              $ OR of SB Base Platform
79    pz -13.11148                              $ Bottom of SB Base
80    pz -10.57148                              $ Top of SB Base Bottom Hole
81    pz  -9.92648                              $ Bottom of Platform Curve
82    pz  -9.30148                              $ Bottom of SB Base Platform
83    pz  -5.33400                              $ Top of Threaded SB Base
84    p    1.0    0.0   -1.0      4.59740       $ Slant for SB Base Notch
85    kz  -4.37515   1  -1                     $ Slant for SB Base Platform
86    tz   0.0  0.0 -9.92648   2.54000  0.62500   0.62500  $ Platform Crv Torus
87    tz   0.0  0.0 -7.794625  1.27000  0.079375  0.079375 $ Torus for SB Base
c  Surfaces for Fuel Rings
88    cz   3.97510                              $ IR of Ring 5
89    cz   4.44500                              $ IR of Rings 1-4, 6
90    cz   5.08000                              $ IR of S Pad Slots, BR Annulus
91    cz   6.34365                              $ OR of Protrusions
92    cz   6.35000                              $ IR of Slots
```

```
93    cz   7.30250                                $ OR of S Pad Slots, BR Annulus
94    cz   8.89000                                $ OR of Rings
95    c/z  3.33375   -5.774224   1.11125         $ FR Hole, Control or Burst Rod
96    c/z -6.66750    0.0        1.11125         $ Lt Hole, Control or Burst Rod
97    c/z  3.33375    5.774224   1.11125         $ BR Hole, Control or Burst Rod
98    c/z  5.249401  -3.280190   1.11125         $ Front End of Right Pad Slot
99    c/z  5.249401   3.280190   1.11125         $ Back End of Right Pad Slot
100   c/z  0.216027  -6.186209   1.11125         $ Right End of Front Pad Slot
101   c/z  0.216027   6.186209   1.11125         $ Right End of Back Pad Slot
102   c/z -5.466549  -2.906616   1.11125         $ Left End of Front Pad Slot
103   c/z -5.466549   2.906616   1.11125         $ Left End of Back Pad Slot
720   c/y  0.0        1.55702    0.08509         $ Smaller FR4 TC Outer Radius
721   c/y  0.0        1.55702    0.15588         $ Larger FR4 TC Outer Radius
104   pz  -7.80455                                $ Bottom of Ring 1
105   pz  -7.43839                                $ Top of Ring 1 Slots for Pads
106   pz  -5.21208                                $ Bottom of Ring 1 Slot
107   pz  -5.18668                                $ Bottom of Ring 2
108   pz  -4.89458                                $ Bottom of Ring 2 Outer Edge
109   pz  -2.62128                                $ Bottom of Ring 3
110   pz  -2.32918                                $ Top of Ring 2
111   pz  -2.30378                                $ Bottom of Ring 3 Outer Edge
112   pz  -0.08128                                $ Bottom of Ring 3 Slot
113   pz  -0.05588                                $ Bottom of Fuel Ring 4
114   pz   0.23622                                $ Bottom of Ring 4 Outer Edge
115   pz   2.21742                                $ Bottom of Ring 5 Outer Edge
116   pz   2.50952                                $ Top of Fuel Ring 4
117   pz   4.64820                                $ Top of Ring 5 Outer Edge
118   pz   4.67360                                $ Bottom of Ring 6 Outer Edge
120   pz   7.16661                                $ Bottom of Deep Indent in 6
121   pz   7.32409                                $ Bottom of Shallow Indent in 6
122   pz   7.78416                                $ Top of Fuel Ring 6
123   py  -1.11125                                $ Front Edge of Left Notch
124   py   1.11125                                $ Back Edge of Left Notch
722   py   5.08000                                $ End of Small TC Hole
723   py   7.93750                                $ End of Larger TC Hole
125   px   8.45820                                $ Back Edge of Left Notch(2-5)
126   px   8.49376                                $ Back Edge of Left Notch(1&6)
127   p   -0.3639702  1.0        0.0   0.0       $ Left Edge of Top Deep Indent
128   p    0.3639702  1.0        0.0   0.0       $ Right Edge of Top Deep Indent
129   p   -5.6712818  1.0        0.0   0.0       $ Top Edge of BR Deep Indent
130   p   -0.8390996  1.0        0.0   0.0       $ Bottom Edge of BR Deep Indent
131   p    5.6712818  1.0        0.0   0.0       $ Top Edge of BL Deep Indent
132   p    0.8390996  1.0        0.0   0.0       $ Bottom Edge of BL Deep Indent
133   p   28.6362533  1.0        0.0   0.0       $ Rt Flat Edge, Front Pad Slot
134   p   -0.5317094  1.0        0.0   0.0       $ Lt Flat Edge, Front Pad Slot
135   p    0.5317094  1.0        0.0   0.0       $ Lt Flat Edge, Back Pad Slot
136   p  -28.6362533  1.0        0.0   0.0       $ Rt Flat Edge, Back Pad Slot
137   p   -0.6248694  1.0        0.0   0.0       $ Bk Flat Edge, Right Pad Slot
138   p    0.6248694  1.0        0.0   0.0       $ Fr Flat Edge, Right Pad Slot
139   p    1.0        1.7320508  0.0 -16.9857200 $ Back Edge, BL Notch (1 & 6)
140   p    1.0       -1.7320508  0.0 -16.9857200 $ Back Edge, BR Notch (1 & 6)
141   p    1.0        1.7320508  0.0 -16.9164000 $ Back Edge, BL Notch (2 - 5)
142   p    1.0       -1.7320508  0.0 -16.9164000 $ Back Edge, BR Notch (2 - 5)
143   p    1.7320508 -1.0        0.0   1.9247415 $ Lower Edge, BL Notch
144   p    1.7320508 -1.0        0.0  -1.9247415 $ Upper Edge, BL Notch
145   p    1.7320508  1.0        0.0   1.9247415 $ Lower Edge, BR Notch
146   p    1.7320508  1.0        0.0  -1.9247415 $ Upper Edge, BR Notch
c  Surfaces for Control Rods and Burst Rod
147   c/z  3.33375   -5.774224   0.47625         $ Control Rod 1 IR
148   c/z  3.33375   -5.774224   1.09220         $ Control Rod 1 OR
149   c/z -6.66750    0.0        0.47625         $ Burst Rod IR
150   c/z -6.66750    0.0        0.63500         $ BR Pin Head OR (Est.)
151   c/z -6.66750    0.0        1.09220         $ Burst Rod OR
152   c/z  3.33375    5.774224   0.47625         $ Control Rod 2 IR
153   c/z  3.33375    5.774224   1.09220         $ Control Rod 2 OR
154   pz -10.79627                                $ Bottom of Burst Rod (Full In)
155   pz  -8.89127                                $ Top of Burst Rod Bottom Hole
156   pz -10.48893                                $ Bottom of Control Rod 1
157   pz -12.54633                                $ Bottom of Control Rod 2
158   pz  -1.27127                                $ Flat Bottom of BR Top Hole
159   pz  -8.58393                                $ Top of CR1 Bottom Hole
160   pz -10.64133                                $ Top of CR2 Bottom Hole
161   pz   1.90373                                $ Top of Burst Rod (Full In)
162   pz   3.76301                                $ Bottom of BR Pin Head
163   pz   4.51231                                $ Top of BR Pin
164   pz   0.30607                                $ Bottom of CR1 Top Hole
165   pz  -1.75133                                $ Bottom of CR2 Top Hole
166   pz   2.21107                                $ Top of Control Rod 1
167   pz   0.15367                                $ Top of Control Rod 2
168   p    1.0       -1.7320508  2.0  18.99920   $ Back Edge, CR1 Notch
169   p    1.0        1.7320508  2.0  14.88440   $ Back Edge, CR2 Notch
170   p    1.7320508  1.0        0.0   0.4179439 $ Lower Edge, CR1 Notch
171   p    1.7320508  1.0        0.0  -0.4179439 $ Upper Edge, CR1 Notch
172   p    1.7320508 -1.0        0.0   0.4179439 $ Lower Edge, CR2 Notch
173   p    1.7320508 -1.0        0.0  -0.4179439 $ Upper Edge, CR2 Notch
174   k/z  3.33375 -5.774224 -11.42873   1  1    $ Lower Slant for Control Rod 1
175   k/z  3.33375 -5.774224   3.15087   1 -1    $ Upper Slant for Control Rod 1
176   k/z -6.66750  0.0      -11.73607   1  1    $ Lower Slant for Burst Rod
177   k/z -6.66750  0.0        2.84353   1 -1    $ Upper Slant for Burst Rod
178   k/z -6.66750  0.0       -1.54115   3  1    $ Tip of Burst Rod
179   k/z  3.33375  5.774224 -13.48613   1  1    $ Lower Slant for Control Rod 2
180   k/z  3.33375  5.774224   1.09347   1 -1    $ Upper Slant for Control Rod 2
c  Surfaces for Support Pads
181   cz   5.08508                               $ Inner Radius of Support Pads
182   cz   7.29488                               $ Outer Radius of Support Pads
```

```
183   c/z  5.249401  -3.280190   1.10490         $ Left End of Front Pad
184   c/z  5.249401   3.280190   1.10490         $ Right End of Front Pad
185   c/z  0.216027  -6.186209   1.10490         $ Front End of Left Pad
186   c/z  0.216027   6.186209   1.10490         $ Front End of Right Pad
187   c/z -5.466549  -2.906616   1.10490         $ Back End of Left Pad
188   c/z -5.466549   2.906616   1.10490         $ Back End of Right Pad
189   pz  -9.02589                               $ Bottom of Support Pads
c  Surfaces for Bearing Ring
190   cz   5.08381                               $ Inner Radius of Bearing Ring
191   cz   7.29615                               $ Outer Radius of Bearing Ring
192   pz   8.75411                               $ Top of Bearing Ring
c  Surfaces for Subassembly Cover Plate
193   cz   1.43510                               $ Radius of Central Hole in SCP
194   c/z  0.0        2.54       0.68199         $ Back Hole, Subassby Cvr Plate
195   c/z  2.54       0.0        0.68199         $ Right Hole, Subassby Cvr Plat
196   c/z  0.0       -2.54       0.68199         $ Front Hole, Subassby Cvr Plat
197   c/z -2.54       0.0        0.68199         $ Left Hole, Subassby Cvr Plate
198   pz   8.00680                               $ Top of SCP Raised Edge
199   pz   8.32430                               $ Top of Subassembly Cvr Plate
c  Surfaces for Mounting Plate
200   cz   5.23875                               $ Radius of Central Hole in MP
201   c/z  3.33375   -5.774224   1.30467         $ FR Hole for Control Rod 1
202   c/z -6.66750    0.0        1.30467         $ Left Hole for Burst Rod
203   c/z  3.33375    5.774224   1.30467         $ BR Hole for Control Rod 2
204   c/z  7.16788    0.0        1.30467         $ Open Right Hole
205   c/z -3.58394   -6.20757    1.30467         $ Open Front Left Hole
206   c/z -3.58394    6.20757    1.30467         $ Open Back Left Hole
226   pz  -20.37715                              $ Bottom of Mounting Plate
229   pz  -16.56715                              $ Top of Mounting Plate
230   pz  -14.02715                              $ Top of Support Nuts
232   px  -44.45000                              $ Left (short) Edge of Mntg Pl
233   px   25.40000                              $ Right (long) Edge of Mtng Pl
234   p    1.0       -1.7320508  0.0 -50.80000   $ BL (long) Edge of Mntg Plate
235   p    1.0        1.7320508  0.0 -50.80000   $ FL (long) Edge of Mntg Plate
236   p    1.0       -1.7320508  0.0  88.90000   $ FR (short) Edge of Mntg Plate
237   p    1.0        1.7320508  0.0  88.90000   $ BR (short) Edge of Mntg Plate
c  Surfaces for Clamp Support (Leg) Right
238   s    15.35430   0.0     2.80035  0.79375   $ End for BB Bolt Hole, Rt Leg
241   c/z  14.84630   0.0        0.47625         $ Top Hole for Right Leg
242   c/y  14.84630  -0.05715    0.47625         $ Front Hole for Right Leg
243   c/x   0.0     -10.85215    1.11125         $ Bottom Edge of Rt Leg Insert
244   c/x   0.0      -0.05715    0.555625        $ Hole for Locking Bolt
245   c/x   0.0       2.80035    0.79375         $ Belly Band Bolt Cylinder
246   c/x   0.0       3.355975   0.238125        $ Left Hole for Right Leg
248   pz  -10.85215                              $ Flat Bottom of Rt Leg Inset
249   pz   -1.32715                              $ Top of Slant for Right Leg
250   pz    1.53035                              $ Flat Bottom of Top Hole Rt Lg
251   pz    4.38785                              $ Top of Right Leg
252   py   -1.42875                              $ Front Edge of Right Leg
254   py   -0.55563                              $ Back Edge of Front Hole Rt Lg
256   py    1.42875                              $ Back Edge of Right Leg
257   px   12.30630                              $ Left Edge of Right Leg
258   px   12.94130                              $ Right Edge of Right Leg Inset
259   px   13.57630                              $ Right Edge of Left Pin Hole
261   px   17.38630                              $ Right Edge of Right Leg Top
263   p   22.0    0.0    3.0   378.51715         $ Slant for Right Leg
265   k/z 14.84630  0.0        1.05410   1       $ Tip of Top Hole, Rt Leg Bolt
c  Surfaces for Clamp
267   c/y   7.62127  -8.15086    1.11125         $ Rounded Btm Corner of Clamp
268   c/y   7.62127   7.87908    1.11125         $ Rounded Top Corner of Clamp
269   c/y   8.32927  -7.77494    4.76250         $ Curved Bottom of Clamp
270   c/y   8.32927   7.34568    4.76250         $ Curved Top of Clamp
271   c/x   0.0       3.35407    0.238125        $ Hole for Long Roll Pin
276   pz   -8.15086                              $ Top of Bottom Corner Fill
277   pz    7.87908                              $ Bottom of Top Corner Fill
282   px    5.08000                              $ Left Edge of Prongs
283   px    7.62127                              $ Left Edge of Corner Fill
284   px    8.49630                              $ Left Edge of Clamp Back
285   px   11.67130                              $ Front Edge of Holes
c  Surfaces for Belly Band
286   cz   20.32000                              $ IR for Belly Band
287   cz   22.22500                              $ OR for Belly Band
288   pz    1.21285                              $ Bottom of Belly Band
c  Surfaces for Nuts and Bolts
292   cz    2.22250                              $ OR for Nut on Spindle
293   pz    9.59430                              $ Top of Nut on Spindle
294   c/x   0.0      -0.05715    1.11125         $ OR of Rt Locking Bolt Cap
295   px   18.49755                              $ Rt Edge of Rt Locking Blt Cap
c  Surfaces for Right Belly Band Bolt
296   c/x   0.0       2.80035    1.190625        $ Cyl Head for Belly Band Bolt
297   c/x   0.0       2.80035    1.250786        $ Hex Head for Belly Band Bolt
298   px   14.75994                              $ Left Edge of BB Chamfer
299   px   14.84376                              $ Right Edge of BB Chamfer
300   px   19.28876                              $ Left Edge of BB Head
301   px   20.15998                              $ Right Edge of BB Hex Head
303   k/x  14.05001  0.0        2.80035   1      $ Slant for Right BB Bolt Hole
c  Surfaces for Core Cover
315   pz   26.61285                              $ Top of Core Cover Lid
c  Surfaces for Contamination Shield (Lexguard)
c  Positions are rotated 90 degrees clockwise relative to description in Sec 1
c  viz., "front" becomes "left", "left" becomes "back," etc.
333   pz  -42.92965                              $ Bottom of Contam Shield
c  Surfaces for Bottom Steel Plate in Contamination Shield
c
400   cz   45.0                                  $ Radius for imp=1 Regions
```

```
401   py    0.0
402   px    0.0
403   pz  -13.39215                              $ Top of Leg Bolts
404   pz    1.05410                              $ Bottom of Top Leg Hole Tip
405   cz   20.31900                              $ OR for Cyl Head of BB Bolt
c
c ***************** The following cell cards were added by me *****************
c ************ to remove the trcl cards from the back left clamp *************
c
1123 P   -0.866025 -0.5 0   -1.11125
1124 P   -0.866025 -0.5 0    1.11125
1267 RCC  39.49062 31.60021 -8.15086  -86.6025 -50 0  1.11125
1284 P   -0.5       0.866025 0        8.4963
1268 RCC  0.519495  9.10021 7.87908  -8.66025 -5       0  1.11125
1244 RCC  25 -43.30125 -0.05715  -50 86.6025 0  0.555625
1245 RCC  25 -43.30125 3.35598  -50 86.6025 0  0.79375
1258 P   -0.5       0.866025 0       12.94130
1285 P   -0.5       0.866025 0       11.67130
1271 RCC  25 -43.30125 3.35407  -50 86.6025 0  0.238125
1269 RCC  0.165495  9.71336 -7.77494  -8.66025 -5 0 4.7625
1270 RCC  0.165495  9.71336  7.34568  -8.66025 -5 0 4.7625
1282 P   -0.5       0.866025 0         5.08
1283 P   -0.5       0.866025 0         7.62127
1252 P   -0.866025 -0.5      0        -1.42875
1256 P   -0.866025 -0.5      0         1.42875
1263 P   -11       19.0526   3       378.517
1261 P   -0.5       0.866025 0        17.3863
c
c ***************** The above cell cards were added by me *********************
c ************ to remove the trcl cards from the front left clamp *************
c
2123 P 0.866025 -0.5 0  -1.11125
2124 P 0.866025 -0.5 0   1.11125
2267 RCC -47.11188 18.39979 -8.15086  86.6025 -50 0  1.11125
2284 P  -0.5 -0.866025 0  8.49630
2268 RCC -8.140755 -4.10021 7.87908  8.66025 -5 0  1.11125
2244 RCC 25 43.30125 -0.05715  -50 -86.6025 0  0.555625
2245 RCC 25 43.30125 3.35598  -50 -86.6025 0  0.79375
2258 P -0.5 -0.866025 0  12.9413
2285 P -0.5 -0.866025 0  11.67130
2271 RCC  25 43.30125 3.35407  -50 -86.6025 0  0.238125
2269 RCC -8.494755 -4.71336 -7.77494  8.66025 -5 0  4.7625
2270 RCC -8.494755 -4.71336 7.34568  8.66025 -5 0  4.7625
2282 P  -0.5 -0.866025 0  5.08000
2283 P  -0.5 -0.866025 0  7.62127
2252 P  0.866025 -0.5 0  -1.42875
2256 P  0.866025 -0.5 0  1.42875
2263 P -11 -19.0526 3 378.51715
2261 P -0.5 -0.866025 0  17.3863

kcode    100000    1.0    50    6050
imp:n    1.0   173r    0.0     2r
ksrc     0.0    0.0   -1.0
c        SS 303  (8.0 g/cc)
m1        6000.66c     3.0083e-4
         14028.62c     1.5821e-3       14029.62c     8.0109e-5
         14030.62c     5.3177e-5
         15031.66c     1.5554e-4
         16000.62c     4.5067e-4
         24050.62c     7.2466e-4       24052.62c     1.3974e-2
         24053.62c     1.5844e-3       24054.62c     3.9443e-4
         25055.62c     8.7693e-4
         26054.62c     3.5742e-3       26056.62c     5.5564e-2
         26057.62c     1.2722e-3       26058.62c     1.6962e-4
         28058.62c     5.0437e-3       28060.62c     1.9282e-3
         28061.62c     8.3482e-5       28062.62c     2.6522e-4
         28064.62c     6.7229e-5
         42000.66c     1.5065e-4
c        SAE 4340  (7.85 g/cc)
m2        6000.66c     1.5940e-3
         14028.62c     3.4929e-4       14029.62c     1.7686e-5
         14030.62c     1.1740e-5
         15031.66c     2.7472e-5
         16000.62c     2.9481e-5
         24050.62c     3.1603e-5       24052.62c     6.0944e-4
         24053.62c     6.9097e-5       24054.62c     1.7202e-5
         25055.62c     6.2385e-4
         26054.62c     4.7824e-3       26056.62c     7.4345e-2
         26057.62c     1.7022e-3       26058.62c     2.2696e-4
         28058.62c     9.8985e-4       28060.62c     3.7842e-4
         28061.62c     1.6384e-5       28062.62c     5.2051e-5
         28064.62c     1.3194e-5
         42000.66c     1.2319e-4
c        VascoMax 300  (8.0 g/cc)
m3        6000.66c     8.0221e-5
         13027.62c     1.7855e-4
         14028.62c     7.9104e-5       14029.62c     4.0054e-6
         14030.62c     2.6588e-6
         15031.66c     7.7770e-6
         16000.62c     7.5112e-6
         22000.62c     7.3453e-4
         25055.62c     4.3847e-5
         26054.62c     3.4070e-3       26056.62c     5.2965e-2
         26057.62c     1.2127e-3       26058.62c     1.6169e-4
         27059.66c     7.1938e-3
```

```
         28058.62c     1.0367e-2       28060.62c     3.9635e-3
         28061.62c     1.7160e-4       28062.62c     5.4518e-4
         28064.62c     1.3819e-4
         42000.66c     2.4103e-3
c        Aluminum 6061-T6 (2.70 g/cc * 0.99925)
m4       12000.62c     6.6898e-4
         13027.62c     5.8593e-2
         14028.62c     3.2037e-4       14029.62c     1.6222e-5
         14030.62c     1.0768e-5
         22000.62c     2.5469e-5
         24050.62c     2.6495e-6       24052.62c     5.1093e-5
         24053.62c     5.7929e-6       24054.62c     1.4421e-6
         25055.62c     2.2197e-5
         26054.62c     6.0121e-6       26056.62c     9.3463e-5
         26057.62c     2.1399e-6       26058.62c     2.8532e-7
         29063.62c     4.8671e-5       29065.62c     2.1695e-5
c *****  Zinc not available in ACTI or ENDF66
c          30000.nnc     3.1082e-5
c        Lexguard  (1.2 g/cc)
m5        1001.62c     3.3161e-2
          6000.66c     3.7894e-2
          8016.62c     7.1026e-3
c        AISI 1019 Carbon Steel  (7.87 g/cc)
m6        6000.66c     6.7080e-4
         15031.66c     3.0603e-5
         16000.62c     3.6946e-5
         25055.62c     7.3328e-4
         26054.62c     4.9536e-3       26056.62c     7.7008e-2
         26057.62c     1.7632e-3       26058.62c     2.3509e-4
c        HEU + Mo (93.17 wt.% for Ring 101)
m7       42000.66c     1.7013e-3
         92233.66c     4.6624e-6       92234.66c     4.7311e-4
         92235.66c     4.3069e-2       92236.66c     3.1300e-4
         92238.66c     2.3373e-3
c        HEU + Mo (93.15 wt.% for Ring 102)
m8       42000.66c     1.7366e-3
         92233.66c     4.6953e-6       92234.66c     4.7635e-4
         92235.66c     4.3364e-2       92236.66c     3.1521e-4
         92238.66c     2.3631e-3
c        HEU + Mo (93.08 wt.% for Ring 103)
m9       42000.66c     1.7209e-3
         92233.66c     4.6795e-6       92234.66c     4.7439e-4
         92235.66c     4.3186e-2       92236.66c     3.1416e-4
         92238.66c     2.3876e-3
c        HEU + Mo (93.18 wt.% for Ring 104)
m10      42000.66c     1.6974e-3
         92233.66c     4.6634e-6       92234.66c     4.7327e-4
         92235.66c     4.3083e-2       92236.66c     3.1307e-4
         92238.66c     2.3332e-3
c        HEU + Mo (93.18 wt.% for Ring 105)
m11      42000.66c     1.7104e-3
         92233.66c     4.6427e-6       92234.66c     4.7116e-4
         92235.66c     4.2892e-2       92236.66c     3.1168e-4
         92238.66c     2.3228e-3
c        HEU + Mo (93.14 wt.% for Ring 106)
m12      42000.66c     1.7319e-3
         92233.66c     4.6903e-6       92234.66c     4.7580e-4
         92235.66c     4.3313e-2       92236.66c     3.1488e-4
         92238.66c     2.3652e-3
c        HEU + Mo (93.17 wt.% for Safety Block)
m13      42000.66c     1.1406e-3
         92233.66c     4.6411e-6       92234.66c     4.7095e-4
         92235.66c     4.2873e-2       92236.66c     3.1158e-4
         92238.66c     2.3266e-3
c        HEU + Mo (93.18 wt.% for IISP)
m14      42000.66c     1.6964e-3
         92233.66c     4.5562e-6       92234.66c     4.6239e-4
         92235.66c     4.2092e-2       92236.66c     3.0587e-4
         92238.66c     2.2795e-3
c        HEU + Mo (93.15 wt.% for TISP)
m15      42000.66c     1.7520e-3
         92233.66c     4.7059e-6       92234.66c     4.7743e-4
         92235.66c     4.3462e-2       92236.66c     3.1593e-4
         92238.66c     2.3684e-3
c        HEU + Mo (93.16 wt.% for Control Rods 1 and 2)
m16      42000.66c     1.5428e-3
         92233.66c     4.6944e-6       92234.66c     4.7631e-4
         92235.66c     4.3360e-2       92236.66c     3.1516e-4
         92238.66c     2.3580e-3
c        HEU + Mo (93.16 wt.% for Burst Rod)
m17      42000.66c     1.5712e-3
         92233.66c     4.7807e-6       92234.66c     4.8507e-4
         92235.66c     4.4157e-2       92236.66c     3.2095e-4
         92238.66c     2.34013-3
c
totnu
prdmp     j   575
print

 end of input
```

APPENDIX B MICKA SOURCE CODE

The following sections show the source code for the MICKA.

B.1. Main.m

```
% MICKAs Original Author: Edward Lum
% MICKA Code Updates and Revisions by: Ryan Stewart
%{

Title: MCNP Input Card & Kcode Architect (M.I.C.K.A)
Author: Edward Lum
Revisions: Ryan Stewart
Date Started: 1/13/2015
Date Revisions: 8/22/2016

This program was written for a PHD thesis authored by Edward Lum.
The purpose of which was to determine the negative reactivity coefficient
associated with the flowering of the EBR-II core.
The program writes the surface, data and cell cards for
an MCNP model of the EBR-II core.

Ryan Stewart was brought onto the EBR-II benchmarking project in Fall 2016
and he has made contributions to the code inorder to lessen the burden on
the original author.

%}

%%

%{

The code makes references to circles in order to represent depth
into the program. See Dantes Inferno for the reference.

Each circle increment is a new subfunctional depth. For example main calls
hexmaker which calls drivermaker which calls pinmaker. The levels would
respectively be 1st > 2nd > 3rd > 4th circles.

%}

%%

%{

%This is the main control script which will eventually write the mcnp input
%file and allow for perturbations of the pins.

%Flow Chart

%}

%% General MICKA notes

%{

Hard codes dimensions:

AtomDenCalc

    if strcmp(Comptyp(end),'F'); LatHeight=34.29/NumSec; end
    if strcmp(Comptyp(end),'B'); LatHeight=139.7/NumSec; end

    These are the pre programmed ARC data slug heights.

MaxP=68; MinP=0.2; These are the max and min power data based upon rod position
that arte found in the logbook


%}

%% *************************** 1st circle ********************************

%% ************************MICKA Initialization***************************


%format shorte  changes the format matlab uses to store sig figs

format shortE

% Clears all variables in memeory and close all open figures
% clearvars
% clearvars -global
% close all
% fclose('all');
```

```
clear all

% Turn off text syntax warning

warning('off','MATLAB:handle_graphics:exceptions:SceneNode')

% This line adds the working functions to the matlab path
addpath(genpath('..\MICKA'))

%% **********************Parameter Definitions*****************************

%Gui prompt for filename and run parameters
[ MICKAin, MCNPin, FeaturesM, DebugM, filenameprefix, ProbSet, MoveDefault, KatanaIn]=...
    UserInputs(  );

%% **********************Variable Definitions*****************************

%Pre define some global variables
%surnum is the surface number. It is being defined as a global variable so
%that no matter how many times surfaces are written, they are written
%sequentially. This is the same for the cellnum
%A global array is needed to keep a list of the outisde hex surfaces to be
%used in the final core cell card.

global Title MatLabOld
global debugMICKA SaveSAMakerVars KeyboardStop InducedErrors
global MSATyC MNumC Perturb
global mcnpfix comout
global PrintMaps PrintForPub nsaBplot
global cellnum planesurfnum NumSlice
global DelTemp CtoKconv KtoMevConv RoomTemp SmearSATYs

% This following function sets all of the internal switches to have MICKA
% functions and what features it has.
[ nsa ]=SetGlobals( MICKAin, MCNPin, FeaturesM, DebugM, filenameprefix, ProbSet, KatanaIn );

%% ***************************Debug Settings*******************************

if debugMICKA; nsa=str2double(DebugM{3});  end

%% **********************Perturbation Codes*******************************

[ PertsM, PertsD, PertsS, PertD, PertsOM, PertsT ] = LoadPerturb( filenameprefix, ProbSet );

%% ************************Reactor Map Import*****************************
%This portion will read a file that links assembly types with MICKA numbers
%For now this will be hard coded. This generates
% The if statement switch to debugging mode where all different types of
% assemblies are made with only 1 pin in each assembly

ReactMap = ReactorMapper(nsa);

%% **************************Control and Safe rod positions***************

%This function calls basically hardcoded values for subassembly movement.

[MoveMap, Crit, MoveDesc ]= MoveMapper(ReactMap,nsa,MoveDefault);

%% ****************************Description Input**************************
% This appears here because of the rod heights being applied to the top
% description.

[ DescC ] =...
    DescWriter( Title, MCNPin, PertsM, PertsD, PertsS, PertD, PertsOM, PertsT, MoveDesc, FeaturesM, Crit, ProbSet);

%% ************************Dimension Map Import***************************
%Dimensions are read from the master Dimensions database. They are converted
%for MICKA and sorted into their respective assemblies.

[DimMap, hdp]=DimMapper(ReactMap,MoveMap,PertsD);

%% ************************Material Map Import*****************************

%Materials are read from the master materials database. They are converted
%for MICKA and sorted into their respective assemblies.
% Dim Map is changed inside of MatMapper because of the swell code.
[MatMap, ReactMap, BurnMap, MassMap, DimMap]=MatMapper(ReactMap,DimMap,Title);

%% **********************Hex Duct Origin Calculation**********************
%The origins of all of the is calculated in HexOrg
[hdorg,OrgMap]=HexOrg(nsa,hdp,ReactMap);

%% ************************Plane maker for pin boundary and division*******
%Plane maker creates planes that are used commonly between many different
%subassemblies. It is also the code which creates the slices for the
%Katana Effect.
pmsurfc=PlaneMaker(NumSlice,DimMap);

%% ******************Initialize Bending Origins*****************************
%This code loads the ANSYS data into MICKA to be used for the KATANA
%effect.
[ ~ ] = ThermalXHex( ReactMap,hdorg,nsa );

%% **********************Garbage Colelction*******************************
```

```
%This section is meant to read in all the data for materials and such, and
%remove duplicate materials etc...
[MatMap,MatChngMap]=GarbageCollect(MatMap);

%% **********************Data Card Maker**********************************

% This writes the data cards using information from MatMapper
[sourcenum, datac ]=DataMaker(MCNPin,hdorg,ReactMap,MatMap,DimMap);

%% ********************Create Plots*************************

if PrintMaps && ~MatLabOld

    Plotters( MatMap, BurnMap, MassMap, MoveMap, hdorg )

end

%% **********************Hex Duct card maker*****************************

[ cellc, surfc, datac ] = ...
    AssemblyMaker( hdp, hdorg, ReactMap, DimMap, MatMap, nsa, pmsurfc, datac, DescC  );

%% Core maker creates the mcnp cards that describe the outside of the core

[corecellc, coresurfc]=CoreMaker(hdp,nsa,DimMap(end,:),ReactMap,MatMap(end,:));

    surfc=[surfc; coresurfc];
    cellc=[cellc; corecellc];

%% Free Gas Thermal Treatment

if strcmp('Y',FeaturesM{1})

    [ inds ] = cellfind( datac,'c TMP',1,'Match' );
    [ numcell ] = cellfind( cellc,'IMP:N',1,'NoMatch' );

    datac{inds,1}='TMP';
    datac{inds,2}=[Num2StrM((DelTemp+CtoKconv+RoomTemp)*KtoMevConv) ' ' Num2StrM(length(numcell)-1) 'R'];

end

%% Output problem parameters

dispPrint(' ');
dispPrint([Title ' Parameters'])
dispPrint(' ')
dispPrint(['Problem Title: ' Title]);
dispPrint(['Number of Slices: ' Num2StrM(length(planesurfnum))]);
dispPrint(['Number of Assemblies: ' MICKAin{4}]);
dispPrint(['Number of Starting Sources: ' Num2StrM(sourcenum)]);

%% MCNP execution

MCNPStart( MCNPin, FeaturesM, cellc, surfc, datac, ProbSet );

disp('Calculation Correct')

%% Any output messages

if InducedErrors

   for gg=1:10

        disp('**************WARNING!!!!!**************')
        disp('Induced errors is on! Check set globals')
        disp('to see what errors are being applied.')
        disp('**************WARNING!!!!!**************')
        disp(' ')

   end

end
```

B.2. UserInputs.m

```
function [ MICKAin, MCNPpara, FeaturesM, DebugM, filenameprefix, ProbSet, MoveDefault, KatanaIn ] = UserInputs(  )
%USERINPUTS Summary of this function goes here
%   Detailed explanation goes here

global UseDefault
global CompName
global debugMICKA
global BorD

% Set the computer name, this determines computer specific parameters

[~,hostname]=dos('hostname');

CompName=hostname(1:end-1);

UseDefault=false;

debugMICKA=false;
```

```
% Problem title prefix
filenameprefix='EBRII138B';

% Problem Set
ProbSet='Detailed';

%% Variables hard coded

MaxP=68; MinP=0.2; % Max and Min power data logs
MaxMove=8.97; MinMove=4.81; % The Control Rod Positon assiciated with the power above.

%% Default Configuration

DefaultsinPrompt={'Enter a title:',...
                  'Reactor','Run Set',...
                  '# of processors:',...
                  'Use Default MICKA parameters set in SetGloabls.m:',...
                  'Engage Debug Settings:',...
                  'Default Core Layout:'};

Defaultsdefault={'NoTitle',filenameprefix,ProbSet,'26','Y','N','Y'};

Defaults=inputdlg(DefaultsinPrompt,'MICKA',1,Defaultsdefault);

UseDefault=strcmp(Defaults{5},'Y');
debugMICKA=strcmp(Defaults{6},'Y');

ProbSet=[Defaults{2} Defaults{3}];

if ~UseDefault

    %% Basic Problem Parameters

    MICKAinprompt={'Enter a title:',...
                   '# of processors:',...
                   'Katana Effect:',...
                   '# of Sub Assemblies:',...
                   'Set Defaults to "as-built":',...
                   'Print Information Maps?:'...
                   'Print Maps that can be published?:',...
                   'Default Core Layout?',...
                   'Perform Perturbations?:',...
                   'Power Level in MW, Max 68, Min 0.2, or LP Crit:'};

    MICKADefault={Defaults{1},Defaults{4},'N','721','N','N','N',Defaults{7},'N','LP Crit'};

    MICKAin=inputdlg(MICKAinprompt,'MICKA',1,MICKADefault);

    %% Sets the rod movement

    if strcmp(MICKAin{10},'LP Crit')

        MoveDefault={'14','14','0','14','14','3.01','14','14','0','14'};

    else

        % The following lines assume a proportional control rod to power
        % curve. This is not correct, this will need replaced with a model,
        % but for now it will be linear.

        CurP=str2double(MICKAin{10});
        DiffP=MaxP-MinP;
        DiffMove=MaxMove-MinMove;

        CurMove=Num2StrM(((CurP-MinP)/DiffP)*DiffMove+MinMove);

        MoveDefault={'8','8','9','9','9',CurMove,'9','9','9','9'};

    end

    %% Sets the default presented features

    if strcmp(MICKAin{5},'Y')

            FeaturesDefault={'Y','Y','Y','Y','N','Y','Y','N','N','Y','Y'};

    else

            FeaturesDefault={'Y','Y','N','N','N','Y','N','Y','Y','N','Y'};

    end

    %% MCNP Parameters

    MCNPPrompt={'MCNP NPG:',...
                'MCNP NSK:',...
                'MCNP K guess:'...
                'MCNP NG:',...
                'Execute MCNP [Y/N]:',...
                'Execute MCNP Plotter [Y/N]:',...
                'Commented Output File [Y/N]:',...
                'Core Lattice:',...
                'Pin Lattice:',...
                'Cross Section Set:',...
```

```
                'Bulk Absolute Temperature:',...
                'Generate New Cross Section Set:',...
                '# of Significant figures used for materials:'};

    MCNPDefault={'150000','30','1','1030','Y','N','Y','Y','Y','.70c','343','N','5'};

    MCNPin=inputdlg(MCNPPrompt,'MICKA',1,MCNPDefault);

    %% Features

    FeaturesPrompt={'Temp Corrs Density (Na,SS), Xsec?:',...
                    'Small Value Cutoff:',...
                    'Fuel Element Swelling:',...
                    'Fuel Slug Swelling:',...
                    'Sodium porosity:',...
                    'S, Alpha, Beta Correction?:',...
                    'Boron Burnup:',...
                    'Remove Lower Adpaters:',...
                    'Smear Blanket Rows 13-16:',...
                    'Sodium Above Slug Height ThermX Correction:',...
                    'MKII versus MKIIA Distinction:'};

    FeaturesM=inputdlg(FeaturesPrompt,'MICKA',1,FeaturesDefault);

else % Default settings

    MICKAin={Defaults{1},Defaults{4},'N','721','N','N','N',Defaults{7},'Y','LP Crit'};

    MCNPin={'150000','30','1','1030','Y','N','Y','Y','Y','.70c','343','N','5'};

    FeaturesM={'Y','Y','N','N','N','Y','N','Y','Y','N','Y'};

    MoveDefault={'14','14','0','14','14','3.01','14','14','0','14'};

end

    %% Code for Movement uncertainty
    % Uncertainty +-0.0402in

    MoveUnc=false;

    if MoveUnc

        MoveDefault=CTRLrodPOS(MoveDefault);

    end

%% Debug Settings

if debugMICKA

    DebugPrompt={'Total Debug:',...
                 'Use a debug reactor Map:',...
                 '# of Debug Assm:',...
                 'Save Material Vars:',...
                 'Save Dimension Vars:',...
                 'Save Movement Vars:',...
                 'Save All Other Main Function Vars:',...
                 'Save All SaMakerVars:',...
                 'Save SA Mats to spreadsheets:',...
                 'NPE:',...
                 'NFE:',...
                 'NDE:'};

    DebugDefault={'N','Y','61','N','N','N','N','N','N','1','1','1'};

    DebugM=inputdlg(DebugPrompt,'MICKA',1,DebugDefault);

    MICKAin(4)=DebugM(3,1);

else

    DebugM={'N','N','61','N','N','N','N','N','N','N','N','N'};

end

%% Dissertation settings

if strcmp(MICKAin{3},'Y')

    %Dissertation

    ProbSet='Katana Effect';

    KatanaInPrompt={'Number of Slices:',...
                    'Use Example Data:',...
                    'Exaggeration Factor:'};

    KatanaDefault={'3','Y','10'};

    KatanaIn=inputdlg(KatanaInPrompt,'MICKA',1,KatanaDefault);

else

    %Benchmark
```

```
    KatanaIn=cell(3,1);

end

%% MCNP Parameters Set

MCNPpara={MICKAin{2};...         % Tasks
       str2double(MCNPin{1});... % NPG
       str2double(MCNPin{3});... % K guess
       str2double(MCNPin{2});... % NSK
       str2double(MCNPin{4});... % NG
                  MCNPin{5};...  % MCNPST
                  MCNPin{6};...  % MCNPPlot
                  MCNPin{7};...  % MCNP Comments
                  MCNPin{8};...  % Core Lattice
                  MCNPin{9};...  % Pin Lattice
                  MCNPin{10};... % Cross section set
                  MCNPin{11};... % Bulk absolute Temperature
                  MCNPin{12};... % Generate a new cross section set
                  MCNPin{13}};   % Number of significant digits

%% Override all inputs
 [ MICKAin, MCNPpara, FeaturesM, DebugM, filenameprefix, ProbSet, MoveDefault, KatanaIn ]...
     = InputOverride( MICKAin, MCNPpara, FeaturesM, DebugM, filenameprefix, ProbSet, MoveDefault, KatanaIn );

end

function MoveDefault=CTRLrodPOS(MoveDefIn)

MoveDir=-1;
MoveUnc=0.102/2.54;
ScaleF=1;

Pert=ScaleF*MoveUnc*MoveDir;

% Rod insertion distances, var name means which rod.

Safe10 =str2double(MoveDefIn{1})+Pert;
Safe16 =str2double(MoveDefIn{2})+Pert;
HWCR40 =str2double(MoveDefIn{3})+Pert;
HWCR42 =str2double(MoveDefIn{4})+Pert;
HWCR46 =str2double(MoveDefIn{5})+Pert;
HWCR48 =str2double(MoveDefIn{6})+Pert;
HWCR50 =str2double(MoveDefIn{7})+Pert;
HWCR54 =str2double(MoveDefIn{8})+Pert;
HWCR58 =str2double(MoveDefIn{9})+Pert;
HWCR60 =str2double(MoveDefIn{10})+Pert;

MoveDefault{1}  =Num2StrM(Safe10);
MoveDefault{2}  =Num2StrM(Safe16);
MoveDefault{3}  =Num2StrM(HWCR40);
MoveDefault{4}  =Num2StrM(HWCR42);
MoveDefault{5}  =Num2StrM(HWCR46);
MoveDefault{6}  =Num2StrM(HWCR48);
MoveDefault{7}  =Num2StrM(HWCR50);
MoveDefault{8}  =Num2StrM(HWCR54);
MoveDefault{9}  =Num2StrM(HWCR58);
MoveDefault{10} =Num2StrM(HWCR60);

end
```

B.3. SetGlobals.m

```
function [ nsa ] = SetGlobals( MICKAin, MCNPpara, FeaturesM, DebugM, filenameprefix, ProbSet, KatanaIn )
% This function sets all the globalvariables

%% Initialize variables

%% File I/O

global Title

% Sets the filename and Title for the problem.
Title=[filenameprefix MICKAin{1}];

%% Initialize starting surface numbers
global surfnum cellnum SAsurfnum SAUnivnum hexsurfnum

% nsa stands for "Number of Sub Assemblies". This variable stands outside
% of set globals because it is used in too many functions to be made a
% seperate global variable
nsa=str2double(MICKAin{4});

surfnum=nsa+1;

%These must be set to one.

cellnum=1;
SAsurfnum=1;
SAUnivnum=1;
```

```
% Set to 0 and this variable is changed if Katana is engaged
hexsurfnum=cell(1,1);

% The katanapls variable is used to set the plane numbers used to deliniate
% katana slicing planes.
% It contains the plane number and the z height at which the surface is
% made

global KatanaPls
KatanaPls=1;

%% Reactor Map Switches

global DefaultLayout
global OrgOut

OrgOut=true;
DefaultLayout=strcmp(MICKAin{8},'Y');

%% Cross section sets

global xcrossset
global sabxcrossset

% Nominal '.70c'
% s(a,b) '.20t'

xcrossset=MCNPpara{11};

% SAlphaBetas is used to store the thermal scattering cross sections to be
% written to the data cards. I do not like using a global for this, for the
% sake of time I am just going to use a gloabl. This can be fixed easily by
% passing the variable back and fourth.

global sabDefs
global SABdefault
global SABcorr
global SABSet

SABSet=1;   % 1=ENDF70SaB (default) 2=ENDF71SAB 3=ENDF6SAB
sabDefs=1; % This means iron SAB
SABdefault=true; % Sets the defaults for which zaid to which assembly.
SABcorr=strcmp(FeaturesM{6},'Y');

%% Determine matlab version

global MatLabOld

if verLessThan('matlab','8.4')

    MatLabOld=true;

else

    MatLabOld=false;

end

%% Max generated card length
global CharLen

CharLen=78;

%% Lattice Codes
global LAT
global LATp

LAT=strcmp('Y',MCNPpara{9});
LATp=strcmp('Y',MCNPpara{10});

%% Small Value cutoff
global SmallValCutOff

SmallValCutOff=strcmp('Y',FeaturesM{2});

%% Debugging Parameters

 global KeyboardStop

 % This is used to debug for a particular fault.

 KeyboardStop=false;

  global SaveDebugMatVars
  global SaveDebugDimVars
  global SaveDebugMoveVars
  global SaveOtherFuncVars
  global SaveSAMakerVars
  global debugMICKA % This is set in the user inputs function
  global debugCore
  global SaveSAMats
  global debug1SAMode
  global ExportOtherMats

 TotdebugMICKA=strcmp('Y',DebugM{1});
```

```
 if str2double(DebugM{3})>2

     debug1SAMode=false;

 else

     debug1SAMode=true;

 end

    ExportOtherMats=false;

 if TotdebugMICKA

    debugCore=strcmp('Y',DebugM{2});
    SaveDebugMatVars=true;
    SaveDebugDimVars=true;
    SaveDebugMoveVars=true;
    SaveOtherFuncVars=true;
    SaveSAMakerVars=true;
    SaveSAMats=true;
    ExportOtherMats=true;

 else

    debugCore=strcmp('Y',DebugM{2});
    SaveDebugMatVars=strcmp('Y',DebugM{4});
    SaveDebugDimVars=strcmp('Y',DebugM{5});
    SaveDebugMoveVars=strcmp('Y',DebugM{6});
    SaveOtherFuncVars=strcmp('Y',DebugM{7});
    SaveSAMakerVars=strcmp('Y',DebugM{8});
    SaveSAMats=strcmp('Y',DebugM{9});

 end

 global Dnp Dnpe Dnde

 % Reset the default core layout

 if debugMICKA; DefaultLayout=false; end

 Dnpe=str2num(DebugM{10});
 Dnp=str2num(DebugM{11});
 Dnde=str2num(DebugM{12});

if SaveSAMakerVars || SaveSAMats

    warndlg('Save all SAs will create .mat file for EVERY posistion in the Core.','!! Warning !!');
    pause(5)

end

% The following veriables determine the number of pins to print for each
% type of SA

global DebugDrNFE
global DebugDrHFWNFE DebugDrHFWNDE
global DebugSafeNFE
global DebugHWCRNFE DebugHWCRNPE
global DebugContNFE
global DebugDumNDE
global DebugBlankNDE
global DebugX402ANFE DebugX402ANDE
global DebugXX09NFE DebugXX09NDE
global DebugXX10NFE DebugXX10NDE
global DebugXY16NDE

% These values cannot be less than 1

DebugDrNFE=91;
DebugDrHFWNFE=46; DebugDrHFWNDE=45;
DebugSafeNFE=61;
DebugHWCRNFE=61; DebugHWCRNPE=7;
DebugContNFE=61;
DebugDumNDE=7;
DebugBlankNDE=19;
DebugX402ANFE=90; DebugX402ANDE=1;
DebugXX09NFE=59; DebugXX09NDE=2;
DebugXX10NFE=18; DebugXX10NDE=1; % These numbers are weird, check the function to see why
DebugXY16NDE=61;

%% Induced errors

% This section controls errors that were found and corrected but then
% reenabled. The reason for this is to make sure perturbation models were
% always run with the same model.

global InducedErrors
global OldControlSA
global BadBoronMat

InducedErrors=false;

if InducedErrors
```

```
    OldControlSA=true;
    BadBoronMat=true;

else

    OldControlSA=false;
    BadBoronMat=false;

end

%% MCNP Settings
%This is a global switch to turn comments on and off.

global comout
global mcnpfix

comout=MCNPpara{8};

% Turns on comments if debug core engaged

if debugCore; comout='Y'; end

%Mcnp fix is a gap that is introduced when macrobodies share boundaries.
%It makes it so that round edges are not coincident.

mcnpfix=0.00001;

%% Temperature Corrections

global TempCor
global RoomTemp

RoomTemp=23;

TempCor=false;

if strcmp('Y',FeaturesM{1}); TempCor=true; end

% The temp at critical form the logbook was 650.6F and the reference temp
% was 23c
global DelTemp
global CtoKconv
global LogTemp

LogTemp=321;

%DelTemp=321; Logbook

DelTemp=str2double(MCNPpara{12})-RoomTemp; % Logbook
CtoKconv=273;

global FindOldLib

% This instructs MICKA to create a new temperature library
FindOldLib=strcmp('N',MCNPpara{13});

% This converison can be multiplied by kelvin to change a temperature into
% MeV
global KtoMevConv

KtoMevConv=8.617e-11;

%% Print Settings
global SAtocol

 SAtocol=[1,[1 0 0];...  % Driver     (Red)
     2,[0.75 0 0];...    % DriverHFW  (Light Red)
     3,[1 0 1];...       % Safety     (Magenta)
     4,[0 1 0];...       % HWCR       (Green)
     5,[1 1 0];...       % Control    (Yellow)
     6,[0 0 0.75];...    % Dummy      (Light Blue)
     7,[0 0 1];...       % Reflector  (Blue)
     8,[0 1 1];...       % Blanket    (Cyan)
     9,[0 0 1];...       % Spare      (Blue)
     10,[0.5 0.5 0];...  % Experiment (Light Yellow)
     11,[0 0 0];];       % Wall       (Black)

global PrintMaps PrintForPub nsaBplot

% Print the maps

PrintMaps=strcmp('Y',MICKAin{6});

% This variable controls how the maps are printed, publications require
% bigger text and no captions.

PrintForPub=strcmp('Y',MICKAin{7});

% nsaBplot sets the numbers of assemblies to plot in the maps
nsaBplot=127;

 if debugMICKA; nsaBplot=91; PrintMaps=false; end

%% Sets the number of material significant digits and Avogadros #
```

```
 global MatSigFig

 MatSigFig=str2double(MCNPpara{14});

 global NAv

 % Variable initialization
NAv=0.602214199; %Avogadros number in Atoms / bcm

%% Temperature Swelling parameters
 global SwellPin
 global SwellFuelSlug

 % These variables are used to engage the pin swelling code.

 SwellPin=strcmp('Y',FeaturesM{3});
 SwellFuelSlug=strcmp('Y',FeaturesM{4});

%% Sodium Bond parameters

 global NaPen
 global NaHTempCorr

 NaPen=strcmp('Y',FeaturesM{5});
 NaHTempCorr=strcmp('Y',FeaturesM{10});

%% Create individual planes for pins

 global createplane
 % This variable controls if one plane is used to define pin sections or if
 % the pins determine their own planes. The threshold is greater than 10
 % uniquelevels determined in PlaneMaker

 createplane=false;

%% Perturbation code settings
 global PerturbMat
 global PerturbDim
 global Perturb
 global LFPPerturb
 global SmearPerturb
 global DenPerturb
 global OtherMatsPerturb
 global BoronPerturb
 global TempPerturb
 global PitchPerturb

 PerturbMat=strcmp(MICKAin{9},'Y');
 PerturbDim=strcmp(MICKAin{9},'Y');
 Perturb=strcmp(MICKAin{9},'Y');
 LFPPerturb=false;
 SmearPerturb=false;
 DenPerturb=false;
 OtherMatsPerturb=false;
 BoronPerturb=false;
 TempPerturb=false;
 PitchPerturb=false;

%% Bias Controls

global  B4CBurnB
global  RemoveLAB
global  SmearBlk1316

B4CBurnB=strcmp(FeaturesM{7},'Y');
RemoveLAB=strcmp(FeaturesM{8},'Y');
SmearBlk1316=strcmp(FeaturesM{9},'Y');

%% Other Variable Input

% These two values distinguish the differences in FEH and FSLAS between
% MKII and MKIIA fuel pins. These are no in StandardDims because that file
% does not distinguish between MKII types. This might get corrected later
% but for now this is easier

global MKIIvsMKIIA
global MKIIfeh MKIIfslas
global MKIIAfeh MKIIAfslas

MKIIvsMKIIA=strcmp('Y',FeaturesM{11});

MKIIfeh=23.814;
MKIIfslas=1.5;
MKIIAfeh=25.05;
MKIIAfslas=0.25;

% These will be converted into CM

MKIIfeh=MKIIfeh*2.54;
MKIIfslas=MKIIfslas*2.54;
MKIIAfeh=MKIIAfeh*2.54;
MKIIAfslas=MKIIAfslas*2.54;

%% Dissertation settings
```

```
global NumSlice
global BorD
global UseKatanaExDat
global KatanaExagPlot

BorD=strcmp(MICKAin{3},'N');

% Determines how many "slices" to make through the core. 2 slices means 3
% sections. Better to think of them as cuts
NumSlice=str2double(KatanaIn{1});

% Use data that is just an example of flowered data.
UseKatanaExDat=strcmp(KatanaIn{2},'Y');

% Add an exaggeration factor, all displacements will be *10 larger. This
% should only be used for plotting purposes.
KatanaExagPlot=str2double(KatanaIn{3});

% Many features need to be turned off because they cause too many issues
% when attempting to slice the geometry

if BorD

    NumSlice=1;
    UseKatanaExDat=true;
    KatanaExagPlot=1;

else

    SwellPin=false;
    SwellFuelSlug=false;
    NaHTempCorr=false;
    RemoveLAB=false;
    MKIIvsMKIIA=false;

end

%% Reactor Map Vars Breakout

global MNumC MSATyC SAPosC SANomen NomenTyC

MNumC=1;
MSATyC=2;
SAPosC=3;
SANomen=4;
NomenTyC=5;

%% Dim Vars breakout
% This section breaks out all of the variables into seperate variables.

global upphC uppdC hdhC hdodC hdidC hdwtC bhC crhC uehC lehC lcdC lchC
global ihdhC ihdodC ihdidC ihdwtC ihdlchC ihdlcdC sazoC saatC sazmC pshC psdC
global pszoC pslasC pehC peidC peodC pewtC peoC pewwhC pewwdC npeC pepC fshC
global fsdC fszoC fslasC fehC feidC feodC fewtC feozC feoxC fewwhC fewwdC
global nfeC fepC eshC esdC eszoC eslasC dehC deidC deodC dewtC deoC ndeC depC
global usbhC pesbhC pebphC fetphC fesphC

upphC=6;
uppdC=9;
hdhC=12;
hdodC=15;
hdidC=18;
hdwtC=21;
bhC=24;
crhC=27;
usbhC=30;
uehC=33;
lehC=36;
lcdC=39;
lchC=42;
ihdhC=45;
ihdodC=48;
ihdidC=51;
ihdwtC=54;
ihdlchC=57;
ihdlcdC=60;
sazoC=63;
saatC=66;
sazmC=69;
pshC=72;
psdC=75;
pszoC=78;
pslasC=81;
pehC=84;
peidC=87;
peodC=90;
pewtC=93;
peoC=96;
pewwhC=99;
pewwdC=102;
pesbhC=105;
pebphC=108;
npeC=111;
pepC=114;
fshC=117;
```

```
fsdC=120;
fszoC=123;
fslasC=126;
fehC=129;
feidC=132;
feodC=135;
fewtC=138;
feozC=141;
feoxC=144;
fewwhC=147;
fewwdC=150;
fetphC=153;
fesphC=156;
nfeC=159;
fepC=162;
eshC=165;
esdC=168;
eszoC=171;
eslasC=174;
dehC=177;
deidC=180;
deodC=183;
dewtC=186;
deoC=189;
ndeC=192;
depC=195;

%% Mat Vars Breakout
global FmC FdC FdatC FNamC FNadC FNadatC FPGmC FPGdC FPGdatC FCladmC FCladdC
global FCladdatC PmC PdC PdatC FPwwrmC FPwwdC FPwwdatC PPGmC PPGdC PPGdatC
global PCladmC PCladdC PCladdatC DmC DdC DdatC DuctmC DuctdC DuctdatC
global DnamC DnadC DnadatC SUPmC SUPdC SUPdatC SLOmC SLOdC SLOdatC LAmC LadC
global LadatC IHDLAmC IHDLAdC IHDLAdatC OPmC OPdC OPdatC ShieldPmC ShieldPdC
global ShieldPdatC SparemC SparedC SparedatC Sparem2C Spared2C Sparedat2C

FmC=6;
FdC=7;
FdatC=8;
FNamC=9;
FNadC=10;
FNadatC=11;
FPGmC=12;
FPGdC=13;
FPGdatC=14;
FCladmC=15;
FCladdC=16;
FCladdatC=17;
PmC=18;
PdC=19;
PdatC=20;
FPwwrmC=21;
FPwwdC=22;
FPwwdatC=23;
PPGmC=24;
PPGdC=25;
PPGdatC=26;
PCladmC=27;
PCladdC=28;
PCladdatC=29;
DmC=30;
DdC=31;
DdatC=32;
DuctmC=33;
DuctdC=34;
DuctdatC=35;
DnamC=36;
DnadC=37;
DnadatC=38;
SUPmC=39;
SUPdC=40;
SUPdatC=41;
SLOmC=42;
SLOdC=43;
SLOdatC=44;
LAmC=45;
LadC=46;
LadatC=47;
IHDLAmC=48;
IHDLAdC=49;
IHDLAdatC=50;
OPmC=51;
OPdC=52;
OPdatC=53;
ShieldPmC=54;
ShieldPdC=55;
ShieldPdatC=56;
SparemC=57;
SparedC=58;
SparedatC=59;
Sparem2C=60;
Spared2C=61;
Sparedat2C=62;

end
```

B.4. LoadPerturb.m

```
function [ PertsM, PertsD, PertsS, PertD, PertsOM, PertsT ] = LoadPerturb( filenameprefix, ProbSet )
% This function is a master perturb function which initializes all of the
% perturbation codes

global Perturb

if Perturb

    % This sets the intial perturbation values for dimensions

    [ ~, ~, PertsD] = DimsPerturber( 0, 0, 0);

    % This sets the intial perturbation values for materials

    [ ~, PertsM] = MatsPerturber( filenameprefix, ProbSet, 0);

    % This sets the initial perturbation values for the temperature

    PertsT = TempPerturber( );

    % This sets the intial perturbation values for mass density

    [ ~, ~,PertD,SmearChng] = DensPerturber( 0, 0,'Load');

    % This sets the intial perturbation values for the Smears

    [ ~, ~,PertsS] = SmearPerturber( 0, 0,'Load',SmearChng, PertD);

    % This creates a description file for OtherMats perturbation

    [ PertsOM ] = OtherMatsPerturber(  );

else

    % The global variables which swtich on perturbations are located in the
    % setGlobals.m function. They are all off by default requiring the user
    % to turn them on through the prompts.

    PertsD=0; PertsM=0; PertsS=0; PertD=0; PertsOM=0; PertsT=0;

end

end
```

B.5. DimsPerturber.m

```
function [ NumObjPert, DimMapPerts, PertsD] = DimsPerturber( DimMapUnPert, var2, var3 )

global PerturbMat
global LFPPerturb
global SmearPerturb
global DenPerturb
global PerturbDim
global OtherMatsPerturb
global TempPerturb
global PitchPerturb
global OtherMatsPath
global FindOldLib
global Title
persistent SetPerturb
persistent DimPert
persistent PertSign
persistent SAPertType
persistent SAPertEXPName
persistent NotSAPertEXPType
persistent ScaleF
persistent SelectMade
persistent DumorBen
persistent BoundLim
persistent ManUnc

if isempty(SetPerturb); SetPerturb=true; end

% These lines make sure something is printed as output
DimMapPerts=DimMapUnPert;
AskUserForSAtype=false;
SAtype=0;
PertsD=0;
NumObjPert=0;

if SetPerturb

    Pertchoice = questdlg('What kind of Perturbation would you like to perform?', ...
        'Pert Choice', ...
        'Dimension','Material','No perturbation','Dimension');

    switch  Pertchoice
```

```
case 'Dimension'

    Pertchoice4 = questdlg('Perturb Pitch?', ...
    'Pert Choice', ...
    'Yes','No','No');

switch Pertchoice4

    case 'Yes'

        PerturbMat=false;
        LFPPerturb=false;
        SmearPerturb=false;
        DenPerturb=false;
        PerturbDim=false;
        OtherMatsPerturb=false;
        PitchPerturb=true;

    case 'No'

        PerturbMat=false;
        LFPPerturb=false;
        SmearPerturb=false;
        DenPerturb=false;
        PerturbDim=true;
        OtherMatsPerturb=false;
        PitchPerturb=false;

end

case 'Material'

    PerturbDim=false;

    Pertchoice2 = questdlg('Specific ZAID, Composition or Other?', ...
        'Pert Choice', ...
        'ZAID','Other','Composition','ZAID');

    switch  Pertchoice2

        case 'ZAID'

            PerturbMat=true;
            LFPPerturb=false;
            SmearPerturb=false;
            DenPerturb=false;
            OtherMatsPerturb=false;

        case 'Composition'

            PerturbMat=false;
            LFPPerturb=false;
            SmearPerturb=false;
            DenPerturb=false;
            OtherMatsPerturb=true;

            % This works just by engaging the code which then
            % allows the user to select a new excel file that
            % contains Othermats. These have have been modified by
            % the user.

            [ filename, filepath ] = FileSelect( 'Specs\Perturbations\OtherMats' );

            curFile=[filepath '\' filename ];

            while isdir(curFile)

               [ filename, filepath ] = FileSelect(curFile);

               curFile=[filepath '\' filename ];

            end

            OtherMatsPath=curFile;

        case 'Other'

            PerturbMat=false;
            OtherMatsPerturb=false;

            Pertchoice3 = questdlg('LFP, NA SS Smear Ratio, or Mass Density?', ...
                'Pert Choice', ...
                'LFP','Smear','Density','LFP');

            switch Pertchoice3

                case 'LFP'

                    LFPPerturb=true;
                    SmearPerturb=false;
                    DenPerturb=false;

                case 'Smear'
```

```
                        LFPPerturb=false;
                        SmearPerturb=true;
                        DenPerturb=false;

                    case 'Density'

                        LFPPerturb=false;
                        SmearPerturb=false;
                        DenPerturb=true;

                end

            end

        case 'No perturbation'

            PerturbMat=false;
            LFPPerturb=false;
            SmearPerturb=false;
            DenPerturb=false;
            PerturbDim=false;
            OtherMatsPerturb=false;

    end

   % Need to remove the unneeded titles

   ExcludedDims={'upph';'uppd';'hdh';'hdid';'ihdh';'ihdid';...
       'sazo';'saat';'sazm';'pslas';'peid';'pszo';'peo';'npe';...
       'fszo';'feoz';'feox';'nfe';'eszo';'deo';'nde';'esh';'deh';...
       'deid';'dep';'dewt'};

   % Perturb the pitch

   if PitchPerturb

       % Pitch uncertainty

        TitleS=Title(end);

        if strcmp(TitleS,'P'); PertSign2='+'; else; PertSign2='-'; end

        prompt={'Sign for Perturbation (+ or -):','Bounding Limit Multiple:','Dist, 1=1, 2=sqrt(3), 3=sqrt(6)...
etc','Manual Pert Amount? In cm'};
        name = 'Dim Perturbations';
        PertsAns = inputdlg(prompt,name,1,{PertSign2,'1','1','cm'});

        BoundLim=str2double(PertsAns{2,1});
        Pertsign=PertsAns{1,1};
        DistF=str2double(PertsAns{3,1});

        ManUnc=str2double(PertsAns{4,1});
        ManUncT=PertsAns{4,1};

        DistFnum=sqrt((DistF-1)*3);

        if DistFnum==0; DistFnum=1; end

        ScaleF=DistFnum*BoundLim;

        if strcmp(Pertsign,'+'); PertSign=1; else PertSign=-1; end

        PertsD={'hdp';PertSign;SAPertType;ManUncT;ManUnc;ScaleF};

   end

    % Handle response
    if PerturbDim

        % Read in the Standard Dims Data

        [~,~,dimdbintemp]=xlsread('Specs\Dimensions\StandardDims.xlsx');
        [~,~,dimdbintempexp]=xlsread('Specs\Dimensions\CustomDims.xlsx');

        %Extract Dim Variables titles
        DimVarTitles=dimdbintemp(3,3:3:end)';

       for tt=1:length(ExcludedDims)

           row=cellfind(DimVarTitles,ExcludedDims{tt},1,'Match');

           if row~=0

               if row==1

                   DimVarTitles=DimVarTitles(2:end);

               else if row==length(DimVarTitles)

                        DimVarTitles=DimVarTitles(1:end-1);

                    else

                        DimVarTitles=[DimVarTitles(1:row-1);DimVarTitles(row+1:end)];
```

```
                end
            end

        end

    end

    dispPrint('Dimensional Perturbation will be performed.')

    [DimVarSelectInd,SelectMade] = listdlg('PromptString','Select Dimension Variable',...
        'SelectionMode','multiple',...
        'ListString',DimVarTitles);

    DimPert=DimVarTitles{DimVarSelectInd};

    if (strcmp(DimPert(1),'e') || strcmp(DimPert(1:2),'de'))

        % 1=dummy 2=blanket 3=both

        DnBSet={'Dummy';'Blanket';'Both'};

        DumOrBlk=questdlg(['What does ' DimPert ' perturb?'],...
            'Perturbation Choice', 'Dummy', 'Blanket', 'Both', 'Both');

        switch DumOrBlk
            case 'Blanket'
                DumorBen=8;

            case 'Dummy'
                DumorBen=6;

            case 'Both'
                DumorBen=[8;6];

        end

    end

    % Standard Assemblies
    DimPertCols=cellfind(dimdbintemp,DimPert,2,'Match');

    % Extract all values
    RowOff=5;
    DimPertVals=cell2mat(dimdbintemp(RowOff:end,DimPertCols));
    DimPertVals(isnan(DimPertVals))=0;
    [DimPertValsRows,DimPertValsCols]=find(DimPertVals);
    AssmChoices=dimdbintemp(DimPertValsRows+RowOff-1,1);

    % Experimental Assemblies

    DimPertEXPCols=cellfind(dimdbintempexp,DimPert,2,'Match');

    % Extract all values
    DimPertEXPVals=cell2mat(dimdbintempexp(RowOff-1:end,DimPertEXPCols));
    DimPertEXPVals(isnan(DimPertEXPVals))=0;
    [DimPertValsEXPRows,DimPertValsEXPCols]=find(DimPertEXPVals);
    [NotDimPertValsEXPRows,NotDimPertValsEXPCols]=find(~DimPertEXPVals);

    SAPertType=DimPertValsRows;

    SAPertEXPName=dimdbintempexp(DimPertValsEXPRows+RowOff-2,1);

    %% Scalf and Sign

    TitleS=Title(end);

    if strcmp(TitleS,'P'); PertSign2='+'; else; PertSign2='-'; end

    prompt={'Sign for Perturbation (+ or -):','Bounding Limit Multiple:','Dist, 1=1, 2=sqrt(3), 3=sqrt(6)...
etc','Manual Pert Amount? In cm'};
    name = 'Dim Perturbations';
    PertsAns = inputdlg(prompt,name,1,{PertSign2,'1','1','0'});

    if str2double(PertsAns{4,1})~=0

        ManUnc=str2double(PertsAns{4,1});
        ManUncT=PertsAns{4,1};

    else

        ManUnc=0;
        ManUncT='From StandardDims.xlsx';

    end

    BoundLim=str2double(PertsAns{2,1});
    Pertsign=PertsAns{1,1};
    DistF=str2double(PertsAns{3,1});

    DistFnum=sqrt((DistF-1)*3);

    if DistFnum==0; DistFnum=1; end

    ScaleF=DistFnum*BoundLim;
```

```
        if strcmp(Pertsign,'+'); PertSign=1; else PertSign=-1; end

        PertsD={DimPert;PertSign;SAPertType;ManUncT;NotSAPertEXPType;ScaleF};

    else

            dispPrint('Dimensional Perturbation will NOT be performed.')
            dispPrint(' ')

            SelectMade=0;

    end

    SetPerturb=false;

else if SelectMade==1

        DimVarTitles=DimMapPerts(1,:);

        cn=cellfind(DimVarTitles,DimPert,2,'Match');

        FirstFind=true;

        for ii=2:size(DimMapPerts,1)-1

            curSATY=DimMapPerts{ii,2};
            curName=DimMapPerts{ii,4};

            if curSATY==10; NoExcluidedEXP=false; else NoExcluidedEXP=true; end

            for kk=1:size(SAPertEXPName)

               if strcmp(curName,SAPertEXPName{kk,1});

                   NoExcluidedEXP=true;

               end

            end

            if any(curSATY==SAPertType) & NoExcluidedEXP

                curDim=DimMapPerts{ii,cn};
                curUnc=DimMapPerts{ii,cn+1}*PertSign*ScaleF;
                curTol=DimMapPerts{ii,cn+2}*PertSign*ScaleF;

                if isempty(DumorBen)

                    if isempty(curUnc) && isempty(curTol);

                        Pert=0;

                        dispPrint(' ')
                        dispPrint('There is a problem with the uncertainty from DimMap')
                        dispPrint('There is a missing uncertainty or tolerance')
                        dispPrint(['MICKA ID:' Num2StrM(DimMapPerts{ii,1})...
                                   ' SA Type:' Num2StrM(DimMapPerts{ii,2})...
                                   ' Position:' DimMapPerts{ii,3}...
                                   ' Identifier:' DimMapPerts{ii,4}...
                                   ' Nomenclature:' DimMapPerts{ii,5}...
                                   curDim])
                        dispPrint(' ')
                        dispPrint('Check the command window to troubleshoot.')

                        keyboard

                    end

                    if ~isempty(curUnc); Pert=curUnc; end
                    if ~isempty(curTol); Pert=curTol; end

                    if  ManUnc~=0;

                        Pert=ManUnc*PertSign*ScaleF;
                        DimMapPerts{ii,cn+1}=Pert;
                        DimMapPerts{ii,cn+1}='Manual Pert!';

                    end

                    DimMapPerts=DimsException(DimMapPerts, DimPert, ii,cn, Pert,ScaleF);

                    if FirstFind

                        NumObjPert=1;
                        FirstFind=false;

                    else

                        NumObjPert=NumObjPert+1;

                    end

                else

                      if any(curSATY==DumorBen)
```

```
                            if isempty(curUnc) && isempty(curTol);

                                Pert=0;

                                dispPrint(' ')
                                dispPrint('There is a problem with the uncertainty from DimMap')
                                dispPrint('There is a missing uncertainty or tolerance')
                                dispPrint(['MICKA ID:' Num2StrM(DimMapPerts{ii,1})...
                                           ' SA Type:' Num2StrM(DimMapPerts{ii,2})...
                                           ' Position:' DimMapPerts{ii,3}...
                                           ' Identifier:' DimMapPerts{ii,4}...
                                           ' Nomenclature:' DimMapPerts{ii,5}...
                                           curDim])
                                dispPrint(' ')
                                dispPrint('Check the command window to troubleshoot.')

                                keyboard

                            end

                            if ~isempty(curUnc); Pert=curUnc; end
                            if ~isempty(curTol); Pert=curTol; end

                            if  ManUnc~=0;

                                % The in to cm conversion needs to be
                                % placed in.

                                Pert=ManUnc*PertSign*ScaleF*2.54;
                                DimMapPerts{ii,cn+1}=Pert;
                                DimMapPerts{ii,cn+1}='Manual Pert!';

                            end

                            DimMapPerts=DimsException(DimMapPerts, DimPert, ii,cn, Pert,ScaleF);

                            if FirstFind

                                NumObjPert=1;
                                FirstFind=false;

                            else

                                NumObjPert=NumObjPert+1;

                            end

                        end

                    end

                end

                dispPrint([['Perturbing ' DimPert];{[ii,size(DimMapPerts,1)-1]}]);

            end

            CurVar=DimMapPerts{1,cn};

            DimMapPerts{1,cn}=[CurVar ' Perturbed ' Num2StrM(NumObjPert) ' times'];

            PertsD={DimPert;PertSign;SAPertType;ScaleF};

            dispPrint(' ')

        end

    end

end

function [DimMapP]=DimsException(DimMapPertsUnCorr, DimPert, rn, cn, Pert, ScaleF)
% This function is a catch all to correct other dims that are perturb.
% For example, a change in hdod requires that hdwt is changed to be
% thicker, otherwise a change in hdid or hdod would result in an autochange
% of the other.

% DimMapPertsUnCorr, the entire dimmap

% PertVarVal is the variable value plus perturbation, aka value to use in
% the input file.

% Pert the value of just the perturbation

% rn, MICKA number = rn-1.

% Copy the incoming data

DimMapP=DimMapPertsUnCorr;
ModelDimViolation=false;
NewVal=0;
OldVal=DimMapP{rn,cn};

% Define variables
```

```
global upphC uppdC hdhC hdodC hdidC hdwtC bhC crhC uehC lehC lcdC lchC
global ihdhC ihdodC ihdidC ihdwtC ihdlchC ihdlcdC sazoC saatC sazmC pshC psdC
global pszoC pslasC pehC peidC peodC pewtC peoC pewwhC pewwdC npeC pepC fshC
global fsdC fszoC fslasC fehC feidC feodC fewtC feozC feoxC fewwhC fewwdC
global nfeC fepC eshC esdC eszoC eslasC dehC deidC deodC dewtC deoC ndeC depC
global usbhC pesbhC pebphC fetphC fesphC

persistent SinDimVar

% There are a set of similar checks that need to happen.

LatEerO=false; % This switch is used to initiate a lattice check.
LatEerI=false;
HeightEer=false;    % A height violation has occured

% Of the previous catagories, they are subdivided into diameter and height.
DiaErr=false;

% for diameters, is od or id changing? odc, outter diameter
% change
odc=false;

% Swtich for WW detection against an inner duct.
IsWWR=false;

% Error Flags
wallerr=false;

% This switch catches What perturbation is getting applied

    switch DimPert

        case 'hdod'

            % values to be checked for this are the core lattice boundary
            % violation. Since there are no physical buttons in the model,
            % the expansion does not affect pitch


            %% Gather Variables

            od=DimMapP{rn,hdodC};
            id=DimMapP{rn,hdidC};
            wt=DimMapP{rn,hdwtC};

            latd=DimMapP{end,ihdodC}+2*DimMapP{end,bhC};

            %% New Value Calcs

            % New od
            nod=od+Pert;

            % Declare the new value
            NewVal=nod;

            % Copy id
            nid=id;

            % New wall thickness
            nwt=(nod-nid)/2;

            % Non scaled pert
            NoSFPert=Pert/ScaleF;

            % New Scaling factor
            NScaleF=floor(abs((latd-od)/(NoSFPert/2)));

            %% Model Checks

            if Pert<0
                % Negative pert

                if nwt<=0;

                   ModelDimViolation=true;
                   VarInEr='Wall Thickness';
                   wallerr=true;
                   DiaErr=true;

                end

            else
                % Positive Pert

                if nod>=latd;

                    ModelDimViolation=true;
                    VarInEr='OD exceeds Outter Lattice Diamter';
                    LatEerO=true;
                    DiaErr=true;

                end

            end
```

```
    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,cn}=NewVal;
    DimMapP{rn,cn+1}='This dimension has been perturbed by:';
    DimMapP{rn,cn+2}=Pert;

    % Write any values that have been changed.

    DimMapP{rn,hdwtC}=nwt;
    DimMapP{rn,hdwtC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,hdwtC+2}=DimPert;

case 'hdwt'

    % values to be checked for this are the fuel lattice boundary
    % violation. The wall expands equally for both outter and inner

    %% Checks

    %% Gather Variables

    od=DimMapP{rn,hdodC};
    id=DimMapP{rn,hdidC};
    wt=DimMapP{rn,hdwtC};

    latd=DimMapP{end,ihdodC}+2*DimMapP{end,bhC};
    latdp=DimMapP{rn,hdidC}-(DimMapP{rn,hdidC}/100);

    %% New Value Calcs

    % New Wallthickness
    nwt=wt+Pert;

    % Declare the new value
    NewVal=nwt;

    nod=od+(Pert);
    nid=id-(Pert);

    % Non scaled pert
    NoSFPert=Pert/(ScaleF);

    %% Model Checks

    if Pert<0
        % Negative pert

        if nwt<=0;

               NScaleF=wt/NoSFPert;
               ModelDimViolation=true;
               VarInEr='Wall Thickness';
               wallerr=true;

        end

    else
        % Positive Pert

        if nod>=latd;

            % New Scaling factor
            NScaleF=floor(abs((latd-od)/(2*NoSFPert)));

            ModelDimViolation=true;
            VarInEr='OD exceeds Outter Lattice Diamter';
            LatEerO=true;
            DiaErr=true;

        end

        if nid<=latdp

            % New Scaling factor
            NScaleF=floor(abs((nid-latdp)/(2*NoSFPert)));

            ModelDimViolation=true;
            VarInEr='ID exceeds Fuel Lattice Diamter';
            LatEerI=true;
            DiaErr=true;

        end

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,hdwtC}=NewVal;
    DimMapP{rn,hdwtC+1}='This dimension has been perturbed by:';
    DimMapP{rn,hdwtC+2}=Pert;
```

```
    % Write any values that have been changed.

    DimMapP{rn,hdodC}=nod;
    DimMapP{rn,hdodC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,hdodC+2}=DimPert;

    DimMapP{rn,hdidC}=nid;
    DimMapP{rn,hdidC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,hdidC+2}=DimPert;

case 'bh'

    % The button height only affects the pitch and since there is
    % no hard limit on tank diameter, the expansion is free.

    %% Checks

    SinDimVarChk=true;

    %% Gather Variables

    bh=DimMapP{rn,bhC};

    %% New Value Calcs

    % New button heigh
    nbh=bh+Pert;

    % Declare the new value
    NewVal=nbh;

    % Calculate new pitch for core.
    nhdp=DimMapP{end,ihdodC}+2*nbh;

    % Check for sim dim var changes

    if isempty(SinDimVar)

        SinDimVar=NewVal;

    else

        SinDimVar=[SinDimVar;NewVal];

    end

    if RoundM(mean(SinDimVar),5)~=NewVal;

        ModelDimViolation=true;
        VarInEr='A single unique dimension is changing depending on the assembly.';

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,bhC}=NewVal;
    DimMapP{rn,bhC+1}='This dimension has been perturbed by:';
    DimMapP{rn,bhC+2}=Pert;

    % Write any values that have been changed.

    DimMapP{end,ihdidC}=nhdp;
    DimMapP{end,ihdidC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{end,ihdidC+2}=DimPert;

case 'crh'

    disp('This perturbation can cause a problem with the "control" rod')
    disp('More specifically a rounding error on the heights of')
    disp('the order of 1e-6 on cards "Upper Ext Inner Hex"')
    disp('the order of 1e-6 on cards "Shield Block Top"')
    disp('the order of 1e-6 on cards "Inner Wall Inner Hex Duct"')
    disp('To correct make sure the sum of z+h equals the z')
    disp('of the part above. This ONLY applies to the control rod')
    keyboard

    % The core region height change also needs to be applied to
    % HWCR pszo to move the poison pins.

    %% Checks

    %% Gather Variables

    crh=DimMapP{rn,crhC};
    ezo=DimMapP{rn,pszoC};

    %% New Value Calcs

    % New button heigh
    ncrh=crh+Pert;

    nezo=ezo+Pert;
```

```
    % Declare the new value
    NewVal=ncrh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,crhC}=NewVal;
    DimMapP{rn,crhC+1}='This dimension has been perturbed by:';
    DimMapP{rn,crhC+2}=Pert;

    DimMapP{rn,pszoC}=nezo;
    DimMapP{rn,pszoC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,pszoC+2}=DimPert;

case 'usbh'

    usb=DimMapP{rn,usbhC};

    % Declare the new value

    nusb=usb+Pert;

    NewVal=nusb;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,cn}=NewVal;
    DimMapP{rn,cn+1}='This dimension has been perturbed by:';
    DimMapP{rn,cn+2}=Pert;

case 'ueh'

    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    ueh=DimMapP{rn,uehC};

    %% New Value Calcs

    % New button heigh
    nueh=ueh+Pert;

    % Declare the new value
    NewVal=nueh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,uehC}=NewVal;
    DimMapP{rn,uehC+1}='This dimension has been perturbed by:';
    DimMapP{rn,uehC+2}=Pert;

case 'leh'

    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    leh=DimMapP{rn,lehC};

    %% New Value Calcs

    % New button heigh
    nleh=leh+Pert;

    % Declare the new value
    NewVal=nleh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,lehC}=NewVal;
    DimMapP{rn,lehC+1}='This dimension has been perturbed by:';
    DimMapP{rn,lehC+2}=Pert;

case 'lcd'
    % The button height only affects the pitch and since there is
    % no hard limit on tank diameter, the expansion is free.

    %% Checks


    %% Gather Variables
```

```
    lcd=DimMapP{rn,lcdC};
    hdid=DimMapP{rn,hdidC};

    %% New Value Calcs

    % New button heigh
    nlcd=lcd+Pert;

    % Declare the new value
    NewVal=nlcd;

    if nlcd>=hdid

        ModelDimViolation=true;
        VarInEr='Lower adapter diammeter is greater than hdid';
        DiaErr=true;

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,lcdC}=NewVal;
    DimMapP{rn,lcdC+1}='This dimension has been perturbed by:';
    DimMapP{rn,lcdC+2}=Pert;

case 'lch'
    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    lch=DimMapP{rn,lchC};

    %% New Value Calcs

    % New button heigh
    nlch=lch+Pert;

    % Declare the new value
    NewVal=nlch;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,lchC}=NewVal;
    DimMapP{rn,lchC+1}='This dimension has been perturbed by:';
    DimMapP{rn,lchC+2}=Pert;

case 'ihdod'

    % ihdod has to be concerned with hitting ihdid and then not
    % causing a negative wall thickness


    %% Switches for checks

    DiaChk=true;

    %% Gather Variables

    od=DimMapP{rn,ihdodC};
    id=DimMapP{rn,ihdidC};
    wt=DimMapP{rn,ihdwtC};

    oid=DimMapP{rn,hdidC};

    %% New Value Calcs

    % New od
    nod=od+Pert;

    % Declare the new value
    NewVal=nod;

    % Copy id
    nid=id;

    % New wall thickness
    nwt=(nod-nid)/2;

    % Non scaled pert
    NoSFPert=Pert/ScaleF;

    % New Scaling factor
    NScaleF=floor(abs((oid-od)/(NoSFPert/2)));

    %% Model Checks

    if Pert<0
        % Negative pert
```

```
        if nwt<=0;

           ModelDimViolation=true;
           VarInEr='Wall Thickness';
           wallerr=true;
           DiaErr=true;

        end

    else
        % Positive Pert

        if nod>=oid;

            ModelDimViolation=true;
            VarInEr='OD exceeds outter hdid';
            LatEerO=true;
            DiaErr=true;

        end

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,cn}=NewVal;
    DimMapP{rn,cn+1}='This dimension has been perturbed by:';
    DimMapP{rn,cn+2}=Pert;

    % Write any values that have been changed.

    DimMapP{rn,ihdwtC}=nwt;
    DimMapP{rn,ihdwtC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,ihdwtC+2}=DimPert;

case 'ihdwt'

    % values to be checked for this are the fuel lattice boundary
    % violation. The wall expands equally for both outter and inner

    %% Checks

    %% Gather Variables

    od=DimMapP{rn,ihdodC};
    id=DimMapP{rn,ihdidC};
    wt=DimMapP{rn,ihdwtC};

    oid=DimMapP{rn,hdidC};
    latdp=DimMapP{rn,ihdidC}-(DimMapP{rn,ihdidC}/100);

    %% New Value Calcs

    % New Wallthickness
    nwt=wt+Pert;

    % Declare the new value
    NewVal=nwt;

    nod=od+(Pert);
    nid=id-(Pert);

    % Non scaled pert
    NoSFPert=Pert/(ScaleF);

    %% Model Checks

    if Pert<0
        % Negative pert

        if nwt<=0;

                NScaleF=wt/NoSFPert;
                ModelDimViolation=true;
                VarInEr='Wall Thickness';
                wallerr=true;

        end

    else
        % Positive Pert

        if nod>=oid;

            % New Scaling factor
            NScaleF=floor(abs((oid-od)/(2*NoSFPert)));

            ModelDimViolation=true;
            VarInEr='OD exceeds Outter Lattice Diamter';
            LatEerO=true;
            DiaErr=true;

        end
```

```
        if nid<=latdp

            % New Scaling factor
            NScaleF=floor(abs((nid-latdp)/(2*NoSFPert)));

            ModelDimViolation=true;
            VarInEr='ID exceeds Fuel Lattice Diamter';
            LatEerI=true;
            DiaErr=true;

        end

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,ihdwtC}=NewVal;
    DimMapP{rn,ihdwtC+1}='This dimension has been perturbed by:';
    DimMapP{rn,ihdwtC+2}=Pert;

    % Write any values that have been changed.

    DimMapP{rn,ihdodC}=nod;
    DimMapP{rn,ihdodC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,ihdodC+2}=DimPert;

    DimMapP{rn,ihdidC}=nid;
    DimMapP{rn,ihdidC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,ihdidC+2}=DimPert;

case 'ihdlch'

    % The cylinder height needs to be checked against outer
    % cylinder hight, allowable travel.

    %% Checks

    %% Gather Variables

    ihdlch=DimMapP{rn,ihdlchC};
    lch=DimMapP{rn,lchC};
    saoz=DimMapP{rn,saatC}+DimMapP{rn,sazoC};
    leh=DimMapP{rn,lehC};

    %% New Value Calcs

    % New button heigh
    nihdlch=ihdlch+Pert;

    % Declare the new value
    NewVal=nihdlch;

    % Non scaled pert
    NoSFPert=Pert/ScaleF;

    % New Scaling factor
    NScaleF=floor(abs(((ihdlch+saoz)-(leh+lch))/(NoSFPert)));

    %% Model Checks

    if (ihdlch+saoz)>=(leh+lch)

            ModelDimViolation=true;
            VarInEr='Inner lower cylinder exceeds height of outter duct';
            HeightEer=true;

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,ihdlchC}=NewVal;
    DimMapP{rn,ihdlchC+1}='This dimension has been perturbed by:';
    DimMapP{rn,ihdlchC+2}=Pert;

case 'ihdlcd'

    % Inner cylinder diameter needs to be checked to see if it
    % exceeds the diameter of the outter duct

    %% Checks


    %% Gather Variables

    ihdlcd=DimMapP{rn,ihdlcdC};
    lcd=DimMapP{rn,lcdC};

    %% New Value Calcs

    % New button heigh
```

```
    nihdlcd=ihdlcd+Pert;

    % Declare the new value
    NewVal=nihdlcd;

    if nihdlcd>=lcd

        ModelDimViolation=true;
        VarInEr='Inner cylinder diameter is greater than lcd';
        DiaErr=true;

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,ihdlcdC}=NewVal;
    DimMapP{rn,ihdlcdC+1}='This dimension has been perturbed by:';
    DimMapP{rn,ihdlcdC+2}=Pert;

case 'psh'

    % The poison slug height needs to add the additional height
    % to the overall element length.

    %% Gather Variables

    peh=DimMapP{rn,pehC};
    psh=DimMapP{rn,pshC};

    %% New Value Calcs

    % New od
    npsh=psh+Pert;

    % Declare the new value
    NewVal=npsh;

    % New Element Height
    npeh=peh+Pert;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,cn}=NewVal;
    DimMapP{rn,cn+1}='This dimension has been perturbed by:';
    DimMapP{rn,cn+2}=Pert;

    % Write any values that have been changed.

    DimMapP{rn,pehC}=npeh;
    DimMapP{rn,pehC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,pehC+2}=DimPert;

case 'psd'
    % Poison slug diameter needs to be checked against the inside
    % diameter.

    %% Checks


    %% Gather Variables

    od=DimMapP{rn,psdC};

    wt=DimMapP{rn,pewtC};
    id=DimMapP{rn,peodC}-(2*DimMapP{rn,pewtC});
    nid=id;
    nwt=wt;

    %% New Value Calcs

    % New button heigh
    nod=od+Pert;

    % Declare the new value
    NewVal=nod;

    if nod>=id

        ModelDimViolation=true;
        VarInEr='New OD exceeds Inner Diameter';
        DiaErr=true;
        odc=true;


    end

    % Non scaled pert
    NoSFPert=Pert/ScaleF;

    % New Scaling factor
    NScaleF=floor(abs((id-od)/(NoSFPert)));
```

```
    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,psdC}=NewVal;
    DimMapP{rn,psdC+1}='This dimension has been perturbed by:';
    DimMapP{rn,psdC+2}=Pert;

case 'peh'
    % poison element height

    %% Gather Variables

    peh=DimMapP{rn,pehC};

    %% New Value Calcs

    % New Element Height
    npeh=peh+Pert;

    % Declare the new value
    NewVal=npeh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,pehC}=NewVal;
    DimMapP{rn,pehC+1}='This dimension has been perturbed by:';
    DimMapP{rn,pehC+2}=Pert;

case 'peod'

    % peod needs to check to see if the OD will intersect the hex
    % element window.

    %% Gather variables

    od=DimMapP{rn,peodC};
    wt=DimMapP{rn,pewtC};
    nid=od-2*wt;

    sd=DimMapP{rn,psdC};

    wwd=DimMapP{rn,pewwdC};

    ox=wwd/3;

    Pitch=DimMapP{rn,pepC};

    %% New Value Calcs

    nod=od+Pert;

    NewVal=nod;

    % New wall thickness
    nwt=(nod-nid)/2;

    NoSFPert=Pert/ScaleF;

    %% Model Checks

    [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

    if ModelDimViolation

          % New Scaling factor
          NScaleF=ceil(abs((ExD)/(NoSFPert)));
          VarInEr='Element OD exceeds hex window OD';

    end

    if nid<=sd

        % New Scaling factor
        NScaleF=floor(abs((id-sd)/(2*NoSFPert)));

        ModelDimViolation=true;
        VarInEr=['ID exceeds Boron Slug psd:' Num2StrM(sd)];
        DiaErr=true;
        odc=true;

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,peodC}=NewVal;
    DimMapP{rn,peodC+1}='This dimension has been perturbed by:';
    DimMapP{rn,peodC+2}=Pert;

    DimMapP{rn,pewtC}=nwt;
```

```
    DimMapP{rn,pewtC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,pewtC+2}=DimPert;

case 'pewt'

    %  The wall expands equally for both outter and inner and outer
    %  diameter

    %% Checks

    %% Gather Variables

    od=DimMapP{rn,peodC};
    wt=DimMapP{rn,pewtC};
    id=od-2*wt;

    sd=DimMapP{rn,psdC};

    wwd=DimMapP{rn,pewwdC};

    ox=wwd/3;

    Pitch=DimMapP{rn,pepC};

    %% New Value Calcs

    % New Wallthickness
    nwt=wt+Pert;

    % Declare the new value
    NewVal=nwt;

    nod=od+(Pert);
    nid=id-(Pert);

    % Non scaled pert
    NoSFPert=Pert/(ScaleF);

    %% Model Checks

    if Pert<0
        % Negative pert

        if nwt<=0;

               NScaleF=wt/NoSFPert;
               ModelDimViolation=true;
               VarInEr='Wall Thickness';
               wallerr=true;

        end

    else
        % Positive Pert

        [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

        if ModelDimViolation

              % New Scaling factor
              NScaleF=ceil(abs((ExD)/(NoSFPert)));
              VarInEr='Element OD exceeds hex window OD';

        end

        if nid<=sd

            % New Scaling factor
            NScaleF=floor(abs((id-sd)/(2*NoSFPert)));

            ModelDimViolation=true;
            VarInEr=['ID exceeds Boron Slug psd:' Num2StrM(sd)];
            DiaErr=true;
            odc=true;

        end

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,pewtC}=NewVal;
    DimMapP{rn,pewtC+1}='This dimension has been perturbed by:';
    DimMapP{rn,pewtC+2}=Pert;

    % Write any values that have been changed.

    DimMapP{rn,peodC}=nod;
    DimMapP{rn,peodC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,peodC+2}=DimPert;

    DimMapP{rn,peidC}=nid;
    DimMapP{rn,peidC+1}='Dimension has been change because of a perturbation in:';
```

```
    DimMapP{rn,peidC+2}=DimPert;

case 'pewwh'
    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    wwh=DimMapP{rn,pewwhC};

    %% New Value Calcs

    % New button heigh
    nwwh=wwh+Pert;

    % Declare the new value
    NewVal=nwwh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,pewwhC}=NewVal;
    DimMapP{rn,pewwhC+1}='This dimension has been perturbed by:';
    DimMapP{rn,pewwhC+2}=Pert;

case 'pewwd'

    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    od=DimMapP{rn,pewwdC};
    oxd=DimMapP{rn,peodC}-od/3;
    Pitch=DimMapP{rn,pepC};

    ox=oxd/2;

    IsWWR=true;

    % Declare the new value

    nod=od+(Pert);

    NewVal=nod;


    %% New Value Calcs

        % Positive Pert

        [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

        if ModelDimViolation

              % New Scaling factor
              NScaleF=ceil(abs((ExD)/(NoSFPert)));
              VarInEr='Element OD exceeds hex window OD';

        end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,pewwdC}=NewVal;
    DimMapP{rn,pewwdC+1}='This dimension has been perturbed by:';
    DimMapP{rn,pewwdC+2}=Pert;

case 'pesbh'

    % This shield block should not be affected by anything

    %% Checks

    %% Gather Variables

    sbh=DimMapP{rn,pesbhC};

    %% New Value Calcs

    % New button heigh
    nsbh=sbh+Pert;

    % Declare the new value
    NewVal=nsbh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,pesbhC}=NewVal;
```

```
    DimMapP{rn,pesbhC+1}='This dimension has been perturbed by:';
    DimMapP{rn,pesbhC+2}=Pert;

case 'pebph'
    % Bottom plug for the poison element

    %% Checks

    %% Gather Variables

    pebph=DimMapP{rn,pebphC};

    %% New Value Calcs

    % New button heigh
    npebph=pebph+Pert;

    % Declare the new value
    NewVal=npebph;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,pebphC}=NewVal;
    DimMapP{rn,pebphC+1}='This dimension has been perturbed by:';
    DimMapP{rn,pebphC+2}=Pert;

case 'pep'

case 'fsh'

    % The poison slug height needs to add the additional height
    % to the overall element length.

    %% Gather Variables

    fsh=DimMapP{rn,fshC};

    %% New Value Calcs

    % New od
    nfsh=fsh+Pert;

    % Declare the new value
    NewVal=nfsh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,fshC}=NewVal;
    DimMapP{rn,fshC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fshC+2}=Pert;

case 'fsd'

    % Poison slug diameter needs to be checked against the inside
    % diameter.


    %% Gather Variables

    od=DimMapP{rn,fsdC};

    wt=DimMapP{rn,fewtC};
    id=DimMapP{rn,feodC}-(2*DimMapP{rn,fewtC});
    nid=id;
    nwt=wt;

    %% New Value Calcs

    % New button heigh
    nod=od+Pert;

    % Declare the new value
    NewVal=nod;

    if nod>=id

        ModelDimViolation=true;
        VarInEr='New OD exceeds Inner Diameter';
        DiaErr=true;
        odc=true;


    end

    % Non scaled pert
    NoSFPert=Pert/ScaleF;

    % New Scaling factor
    NScaleF=floor(abs((id-od)/(NoSFPert)));

    %% Write the new data to dimmap
```

```
    % Write the perturbation

    DimMapP{rn,fsdC}=NewVal;
    DimMapP{rn,fsdC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fsdC+2}=Pert;

case 'fslas'
    % Fslas does not appear to be affected by anything else.

    %% Gather Variables

    slas=DimMapP{rn,fslasC};

    %% New Value Calcs

    % New od
    nslas=slas+Pert;

    % Declare the new value
    NewVal=nslas;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,fslasC}=NewVal;
    DimMapP{rn,fslasC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fslasC+2}=Pert;

case 'feh'

    % This perturbation is applied ontop of the element swelling

    %% Gather Variables

    feh=DimMapP{rn,fehC};

    %% New Value Calcs

    % New Element Height
    nfeh=feh+Pert;

    % Declare the new value
    NewVal=nfeh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,fehC}=NewVal;
    DimMapP{rn,fehC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fehC+2}=Pert;

case 'feod'

    % peod needs to check to see if the OD will intersect the hex
    % element window.

    %% Gather variables

    od=DimMapP{rn,feodC};
    wt=DimMapP{rn,fewtC};
    nid=od-2*wt;

    sd=DimMapP{rn,fsdC};

    wwd=DimMapP{rn,fewwdC};

    ox=wwd/3;

    Pitch=DimMapP{rn,fepC};

    %% New Value Calcs

    nod=od+Pert;

    NewVal=nod;

    % New wall thickness
    nwt=(nod-nid)/2;

    NoSFPert=Pert/ScaleF;

    %% Model Checks

    [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

    if ModelDimViolation

          % New Scaling factor
          NScaleF=ceil(abs((ExD)/(NoSFPert)));
          VarInEr='Element OD exceeds hex window OD';

    end
```

```
    if nid<=sd

        % New Scaling factor
        NScaleF=floor(abs((id-sd)/(2*NoSFPert)));

        ModelDimViolation=true;
        VarInEr=['ID exceeds Boron Slug psd:' Num2StrM(sd)];
        DiaErr=true;
        odc=true;

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,feodC}=NewVal;
    DimMapP{rn,feodC+1}='This dimension has been perturbed by:';
    DimMapP{rn,feodC+2}=Pert;

    DimMapP{rn,fewtC}=nwt;
    DimMapP{rn,fewtC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,fewtC+2}=DimPert;

case 'fewt'

    %  The wall expands equally for both outter and inner and outer
    %  diameter

    %% Checks

    %% Gather Variables

    od=DimMapP{rn,feodC};
    wt=DimMapP{rn,fewtC};
    id=od-2*wt;

    sd=DimMapP{rn,fsdC};

    wwd=DimMapP{rn,fewwdC};

    ox=wwd/3;

    Pitch=DimMapP{rn,fepC};

    %% New Value Calcs

    % New Wallthickness
    nwt=wt+Pert;

    % Declare the new value
    NewVal=nwt;

    nod=od+(Pert);
    nid=id-(Pert);

    % Non scaled pert
    NoSFPert=Pert/(ScaleF);

    %% Model Checks

    if Pert<0
        % Negative pert

        if nwt<=0;

               NScaleF=wt/NoSFPert;
               ModelDimViolation=true;
               VarInEr='Wall Thickness';
               wallerr=true;

        end

    else
        % Positive Pert

        [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

        if ModelDimViolation

              % New Scaling factor
              NScaleF=ceil(abs((ExD)/(NoSFPert)));
              VarInEr='Element OD exceeds hex window OD';

        end

        if nid<=sd

            % New Scaling factor
            NScaleF=floor(abs((id-sd)/(2*NoSFPert)));

            ModelDimViolation=true;
            VarInEr=['ID exceeds Boron Slug psd:' Num2StrM(sd)];
            DiaErr=true;
```

```
            odc=true;

        end

    end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,fewtC}=NewVal;
    DimMapP{rn,fewtC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fewtC+2}=Pert;

    % Write any values that have been changed.

    DimMapP{rn,feodC}=nod;
    DimMapP{rn,feodC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,feodC+2}=DimPert;

    DimMapP{rn,feidC}=nid;
    DimMapP{rn,feidC+1}='Dimension has been change because of a perturbation in:';
    DimMapP{rn,feidC+2}=DimPert;

case 'fewwh'

    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    wwh=DimMapP{rn,fewwhC};

    %% New Value Calcs

    % New button heigh
    nwwh=wwh+Pert;

    % Declare the new value
    NewVal=nwwh;

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,fewwhC}=NewVal;
    DimMapP{rn,fewwhC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fewwhC+2}=Pert;

case 'fewwd'

    % The core region height does not affect anything else.

    %% Checks

    %% Gather Variables

    od=DimMapP{rn,fewwdC};
    oxd=DimMapP{rn,feodC}-od/3;
    Pitch=DimMapP{rn,fepC};

    ox=oxd/2;

    IsWWR=true;

    % Declare the new value

    nod=od+(Pert);

    NewVal=nod;


    %% New Value Calcs

        % Positive Pert

        [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

        if ModelDimViolation

              % New Scaling factor
              NScaleF=ceil(abs((ExD)/(NoSFPert)));
              VarInEr='Element OD exceeds hex window OD';

        end

    %% Write the new data to dimmap

    % Write the perturbation

    DimMapP{rn,fewwdC}=NewVal;
    DimMapP{rn,fewwdC+1}='This dimension has been perturbed by:';
    DimMapP{rn,fewwdC+2}=Pert;
```

```
      case 'fep'

      case 'esd'

% Poison slug diameter needs to be checked against the inside
          % diameter.


          %% Gather Variables

          od=DimMapP{rn,esdC};

          wt=DimMapP{rn,dewtC};
          id=DimMapP{rn,deodC}-(2*DimMapP{rn,dewtC});
          nid=id;
          nwt=wt;

          %% New Value Calcs

          % New button heigh
          nod=od+Pert;

          % Declare the new value
          NewVal=nod;

          if nod>=id

              ModelDimViolation=true;
              VarInEr='New OD exceeds Inner Diameter';
              DiaErr=true;
              odc=true;


          end

          % Non scaled pert
          NoSFPert=Pert/ScaleF;

          % New Scaling factor
          NScaleF=floor(abs((id-od)/(NoSFPert)));

          %% Write the new data to dimmap

          % Write the perturbation

          DimMapP{rn,esdC}=NewVal;
          DimMapP{rn,esdC+1}='This dimension has been perturbed by:';
          DimMapP{rn,esdC+2}=Pert;

      case 'eslas'

          % Fslas does not appear to be affected by anything else.

          %% Gather Variables

          slas=DimMapP{rn,eslasC};

          %% New Value Calcs

          % New od
          nslas=slas+Pert;

          % Declare the new value
          NewVal=nslas;

          %% Write the new data to dimmap

          % Write the perturbation

          DimMapP{rn,eslasC}=NewVal;
          DimMapP{rn,eslasC+1}='This dimension has been perturbed by:';
          DimMapP{rn,eslasC+2}=Pert;

      case 'deod'

          % peod needs to check to see if the OD will intersect the hex
          % element window.

          %% Gather variables

          od=DimMapP{rn,deodC};

          ox=0;

          Pitch=DimMapP{rn,depC};

          %% New Value Calcs

          nod=od+Pert;

          NewVal=nod;

          NoSFPert=Pert/ScaleF;

          %% Model Checks
```

```
        [ModelDimViolation,errplot,ExD]=PinODtoHexIDchk(nod,ox,Pitch,IsWWR);

        if ModelDimViolation

              % New Scaling factor
              NScaleF=ceil(abs((ExD)/(NoSFPert)));
              VarInEr='Element OD exceeds hex window OD';

        end

        %% Write the new data to dimmap

        % Write the perturbation

        DimMapP{rn,deodC}=NewVal;
        DimMapP{rn,deodC+1}='This dimension has been perturbed by:';
        DimMapP{rn,deodC+2}=Pert;

    otherwise

        dispPrint('No check was found.')
        dispPrint('Double check to see if this is a real dimension.')
        dispPrint(' ')
        dispPrint(['Variable Pert:' DimPert])
        keyboard

end

%% Violations and error messages

% Throws and error for a model violation
if ModelDimViolation

    NoSFPert=Pert/ScaleF;

    if LatEerO || wallerr; DiaErr=true; end

    if DiaErr

        dispPrint('A perturbed dimension has violated the real core.')
        dispPrint(' ')
        dispPrint(['Variable Pert:' DimPert])
        dispPrint(' ')
        dispPrint(['Specific Error in:' VarInEr])
        dispPrint(' ')
        dispPrint(['New OD:' Num2StrM(nod)])
        dispPrint(['New ID:' Num2StrM(nid)])
        dispPrint(['New WT:' Num2StrM(nwt)])
        dispPrint(' ')
        dispPrint(['Old OD:' Num2StrM(od)])
        dispPrint(['Old ID:' Num2StrM(id)])
        dispPrint(['Old WT:' Num2StrM(wt)])
        dispPrint(' ')
        dispPrint(' ')
        dispPrint(['Scaled Pert:' Num2StrM(Pert)])
        dispPrint(['Pert:' Num2StrM(NoSFPert)])
        dispPrint(' ')

        if LatEerO

            dispPrint('Lattice check error.')
            dispPrint(['Outer Lattice Diameter: ' Num2StrM(latd)]);
            dispPrint(['New recommended Scaling Factor: ' Num2StrM(NScaleF)]);

        end

        if LatEerI

            dispPrint('Lattice check error.')
            dispPrint(['Inner Lattice Diameter: ' Num2StrM(latdp)]);
            dispPrint(['New recommended Scaling Factor: ' Num2StrM(NScaleF)]);

        end

        if wallerr

            dispPrint('Wall Thickness Error Check.')
            dispPrint(['New recommended Scaling Factor: ' Num2StrM(NScaleF)]);

        end

        if odc

            dispPrint('An inner/outter diameter has exceeded the other')
            dispPrint(['New recommended Scaling Factor: ' Num2StrM(NScaleF)]);

        end

        keyboard % ***This keyboard has a purpose not to do with debugging

    end

    if HeightEer
```

```
            dispPrint('A perturbed dimension has violated the real core.')
            dispPrint(' ')
            dispPrint(['Variable Pert:' DimPert])
                      dispPrint(' ')
            dispPrint(['Specific Error in:' VarInEr])
            dispPrint(' ')

        end

        if  ~isempty(SinDimVar)

            dispPrint('A perturbed dimension has violated the real core.')
            dispPrint(' ')
            dispPrint(['Variable Pert:' DimPert])
            dispPrint(' ')
            dispPrint(['Specific Error in:' VarInEr])
            dispPrint(' ')

        end

        if ~isempty(errplot)

            dispPrint('A perturbed dimension has violated the real core.')
            dispPrint(' ')
            dispPrint(['Variable Pert:' DimPert])
            dispPrint(' ')
            dispPrint(['Specific Error in:' VarInEr])
            dispPrint(' ')
            dispPrint(['Reduce Scaling Factor by: ' Num2StrM(NScaleF)]);

            keyboard

        end

    end


    % Throws an error for negative dimensions
    if NewVal<0;

        dispPrint('There is a problem with the uncertainty from DimMap')
        dispPrint('There is a problem with a dimension causing a negative number')
        dispPrint(['Variable that is negaitve:' DimPert])
        dispPrint('Check the command window to troubleshoot.')

        keyboard

    end

    % Throws and error incase the new val was not added to DimMap

    if NewVal==OldVal

        dispPrint('The newval was not added to DimMap')
        dispPrint('There is a problem with a dimension causing a negative number')
        dispPrint(['Variable Pert:' DimPert])
        dispPrint(['Scaled Pert:' Num2StrM(Pert)])
        dispPrint(['NewVal: ' NewVal ' OldVal: ' OldVal])
        dispPrint('Check the command window to troubleshoot.')

        keyboard
    end


[~,hostname]=dos('hostname');

hostname=hostname(1:end-1);

% if strcmp(hostname,'TechRed')
%
%     keyboard
%
% end

end

function [modelviolation,f3,Circ2LnD]=PinODtoHexIDchk(od,ox,Pitch,IsWWR)
%  if the pin is bigger than the lattice element

f3=[];
modelviolation=false;
Circ2LnD=[];

% Create a circle of points based upon feod
or=od/2;

% Need to also calculate the shift of the pin this will be the difference
% in radii between the old and the new element

% Switch for WWR detection or not.
if IsWWR; NEfacet=1; X0=0; Xe=or; else NEfacet=-1; X0=-or; Xe=0; end


Xcirc=linspace(X0,Xe,100)'+(NEfacet*ox);
Ycirc=sqrt((or.^2)-((Xcirc-ox).^2));
```

```
% Create a line based upon the north west facet

EleWinS=tand(30)*Pitch;

y1=sind(60)*EleWinS;
x1=NEfacet*cosd(60)*EleWinS;

y2=0;
x2=2*x1;

% find an equation for the line of that facet

coefln = polyfit([x1, x2], [y1, y2], 1);
yln=coefln(1).*Xcirc+coefln(2);

% Now compare the y values, if any of the y values from the circle are greater
% than the values from the line, then we have violated our model

if sum(Ycirc>=yln)>=1;

    modelviolation=true;
    f3=figure;
    errax=axes('Parent',f3);
    plot(errax,Xcirc,Ycirc,Xcirc,yln);
    axis('equal')
    hold(errax,'on')

    %% Find what radii change would be acceptable

    [ExCircR,~]=find(Ycirc>=yln);

    YindMaxR=max(ExCircR);
    YindMinR=min(ExCircR);

    ExCircX=Xcirc(ExCircR);
    ExCircY=Ycirc(ExCircR);
    ExLnY=yln(ExCircR);
    ExLnX=Xcirc(ExCircR);

    LnCircDiff=ExCircY-ExLnY;

    [YmaxDiff,YMaxExInd]=max(LnCircDiff);

    YmaxIRow=YindMinR+YMaxExInd;

    YmaxCirc=Ycirc(YmaxIRow);
    XmaxCirc=Xcirc(YmaxIRow);

    XmaxLn=(YmaxCirc-coefln(2))/coefln(1);
    YmaxLn=yln(YmaxIRow);

    scatter(errax,[XmaxCirc;XmaxLn;XmaxCirc],[YmaxCirc;YmaxCirc;YmaxLn],...
                'MarkerEdgeColor',[0 .5 .5],...
                'MarkerFaceColor',[0 .7 .7],...
                'LineWidth',1.5)

   Xln=[XmaxCirc,YmaxCirc]-[XmaxLn,YmaxCirc];
   Hln=[XmaxLn,YmaxCirc]-[XmaxCirc,YmaxLn];

   Angle=180-rad2deg(acos( dot(Xln,Hln)/norm(Xln)/norm(Hln) ));

   Ln2CircXD=sqrt((XmaxCirc-XmaxLn)^2+(YmaxCirc-YmaxCirc)^2);

   % This value is the distance the radius has exceed the hexduct by

   Circ2LnD=sind(Angle)*Ln2CircXD;

end


end
```

B.6. MatsPerturber.m

```
function [ ZAIDdensPert, Perts] = MatsPerturber( ZAIDdensUnPert, ChOn, Volume, PerturbCalcWeight )

%% *************************** 5th circle ********************************

% This function will perturb the materials.

global     NAv
global     PerturbMat
global     Title ProbSet
global     FindOldLib
global     TempPerturb
persistent SetPerturb
persistent ZAIDPertn
persistent ZAIDPerts
persistent ZnumPerts
persistent ZISOPerts
persistent PertAmount
persistent ScaleF
```

```
persistent SelectMadeZ
persistent SelectMadeD
persistent SelectMadeS

SelectMadeS=false;
SelectMadeD=false;

if isempty(SetPerturb); SetPerturb=true; end

if ~PerturbMat; SetPerturb=false; SelectMadeZ=0; end

% These lines make sure something is printed as output
ZAIDdensPert=ZAIDdensUnPert;
Perts=0;

if SetPerturb

    % User Select which ISOTOPE

    if PerturbMat

            % Construct a questdlg with three options
             Pertchoice2 = questdlg('Perturb a ZAID, Density, or Smear?', ...
             'Pert Choice', ...
             'ZAID','Density','Temperature','ZAID');

        if strcmp(Pertchoice2,'Density') || strcmp(Pertchoice2,'Smear')

            % Import the material names.

            % Need to load the other materials from OtherMats.xlsx
            indata=importdata('Specs\Materials\RealDb\Other Mats.xlsx');

            mattempdata=indata.textdata(4:end,1);

            [MatInd,SelectMadeIm] = listdlg('PromptString',...
                      'Select A material to Perturb',...
                      'SelectionMode','multiple',...
                      'ListString',mattempdata);

            ZAIDPerts=mattempdata{MatInd};


        end

           switch Pertchoice2

               case 'ZAID'

                   TitleZT=strrep(Title,ZAIDdensUnPert,'');

                   TitleZ=TitleZT(1:end-1);

                   TitleS=TitleZT(end);

                   if strcmp(TitleS,'P'); PertSign=''; else; PertSign='-'; end

                   dispPrint('Material Perturbation will be performed.')

                   prompt={'ZAID:','Amount to perturb in %:','Scaling Factor:','Dist, 1=1, 2=sqrt(3), 3=sqrt(6)... etc'};
                   name = 'Perturbations';
                   PertsAns = inputdlg(prompt,name,1,{TitleZ, [PertSign '1.06'],'1','1'});

                   DistF=str2double(PertsAns{4,1});
                   DistFnum=sqrt((DistF-1)*3);

                   if DistFnum==0; DistFnum=1; end

                   ZAIDPerts=PertsAns{1,1};
                   ZAIDPertn=str2double(ZAIDPerts);
                   ScaleF=str2double(PertsAns{3,1})*DistFnum;
                   PertAmount=str2double(PertsAns{2,1})/100;

                   ZISOPerts=ZAIDPerts(end-2:end);
                   ZnumPerts=strrep(ZAIDPerts,ZISOPerts,'');

                   Perts={ZAIDPerts; ZISOPerts; ZnumPerts; PertAmount; ScaleF};

                   SelectMadeZ=1;

               case 'Density'

                   dispPrint('Material Perturbation will be performed.')

                   prompt={'Amount to perturb the mass density in %:','Scaling Factor:'};
                   PertsAns = inputdlg(prompt,ZAIDPerts,1);

                   ScaleF=str2double(PertsAns{3,1});
                   PertAmount=str2double(PertsAns{2,1})/100;

                   Perts={ZAIDPerts; 0; 0; PertAmount; ScaleF};

                   SelectMadeZ=0;

               case 'Temperature'
```

```
                    TempPerturb=true;
                    FindOldLib=false;

            end


    else

            dispPrint('Material Perturbation will NOT be performed.')
            dispPrint(' ')

            SelectMadeZ=0;
            SelectMadeD=0;
    end

    SetPerturb=false;

else if SelectMadeZ==1

        if SelectMadeZ==1

            Znums=ZAIDdensUnPert{1,2};
            ZComps=ZAIDdensUnPert{3,2};
            ZAdens=ZAIDdensUnPert{4,2};
            ZMWs=ZAIDdensUnPert{2,2};


            for kk=1:size(Znums,2)

               curzaid=Num2StrM(Znums(1,kk));

               curziso=curzaid(end-2:end);

               Zlist{1,kk}=strrep(curzaid,curziso,'');

            end

            Znumcols=cellfind(Zlist,ZnumPerts,2,'No Match');

            [~, ZAIDlistnCols]=find(Znums==ZAIDPertn);

            if isempty(ZAIDlistnCols); ZAIDlistnCols=0; end

            if Znumcols~=0 & ZAIDlistnCols~=0

                ZDens=ZAdens.*ZMWs./NAv;

                ZWgtComps=ZDens./sum(ZDens);

                ZPertWgts=ZWgtComps(1,Znumcols);

                ZAIDPertwgt=Znums(ZAIDlistnCols);
                ZAIDnPertwgt=ZPertWgts(ZPertWgts~=ZAIDPertwgt);

                ZAIDnPertwgtComp=ZAIDnPertwgt./sum(ZAIDnPertwgt);

                Pert=1+PertAmount*ScaleF;

                % Performs the change of the selected isotope
                nZDens=ZDens;
                nZDens(1,ZAIDlistnCols)=ZDens(1,ZAIDlistnCols).*Pert;

                nZMass=nZDens.*Volume;

                % Performs the reduce of the other isotopes
                %ZAIDnlistnCols=Znumcols(Znumcols~=ZAIDlistnCols);
                %nZDens(1,ZAIDnlistnCols)=ZDens(1,ZAIDnlistnCols).*NPert;

                nZAdens=nZDens.*NAv./ZMWs;

                if PerturbCalcWeight

                    % Since materials not fuel are defined by weight
                    % composition, need to use that as the comps

                    nZComps=nZDens./sum(nZDens);

                else

                    nZComps=nZAdens./sum(nZAdens);

                end


                ZAIDdensPert{3,2}=nZComps;
                ZAIDdensPert{4,2}=nZAdens;
                ZAIDdensPert{5,2}=nZMass;
                ZAIDdensPert{6,2}=nZDens;

                % Totals

                ZAIDdensPert{3,3}=sum(nZComps);
                ZAIDdensPert{4,3}=sum(nZAdens);
```

```
                ZAIDdensPert{5,3}=sum(nZMass);
                ZAIDdensPert{6,3}=sum(nZDens);

            end

        end

        if SelectMadeD==1

            Znums=ZAIDdensUnPert{1,2};
            ZComps=ZAIDdensUnPert{3,2};
            ZAdens=ZAIDdensUnPert{4,2};
            ZMWs=ZAIDdensUnPert{2,2};

            ZDens=ZAdens.*ZMWs./NAv;

            Pert=1+PertAmount*ScaleF;

            [ ZAIDS, ZAIDSnot] = Nomen2ZAID( ZAIDPerts);

           kk=0;
           jj=0;

           for cc=1:length(ZAIDS);

               if sum(ZAIDS(cc)==Znums); kk=kk+1; end

           end

           for gg=1:length(ZAIDSnot);

               if sum(ZAIDSnot(gg)==Znums); jj=jj+1; end

           end

           if length(ZAIDS)==kk;

                if length(ZAIDSnot)==jj;

                    nZDens=ZDens*Pert;
                    nZMass=nZDens.*Volume;
                    nZAdens=nZDens.*NAv./ZMWs;

                    if PerturbCalcWeight

                        % Since materials not fuel are defined by weight
                        % composition, need to use that as the comps

                        nZComps=nZDens./sum(nZDens);

                    else

                        nZComps=nZAdens./sum(nZAdens);

                    end

                    ZAIDdensPert{3,2}=nZComps;
                    ZAIDdensPert{4,2}=nZAdens;
                    ZAIDdensPert{5,2}=nZMass;
                    ZAIDdensPert{6,2}=nZDens;

                    % Totals

                    ZAIDdensPert{3,3}=sum(nZComps);
                    ZAIDdensPert{4,3}=sum(nZAdens);
                    ZAIDdensPert{5,3}=sum(nZMass);
                    ZAIDdensPert{6,3}=sum(nZDens);

                end

           end

        end

    end

end


end

function [ ZAIDS, ZAIDSnot] = Nomen2ZAID( Nomen)

switch Nomen

    case 'Sodium'
        ZAIDS=11023;
    case 'Argon'

    case 'Helium'

    case 'Pgas'

    case 'SS304'
        ZAIDS=[6000;15031;26056];
```

```
        ZAIDSnot=11023;

    case 'B4C'

    case 'SS316'
        ZAIDS=[6000;26056];
        ZAIDSnot=11023;

    case 'SS304NaSmear'
        ZAIDS=[6000;15031;26056];
        ZAIDSnot=[];

    case 'SS316NaSmear'
        ZAIDS=[6000;26056];
        ZAIDSnot=[];

end

end
```

B.7. TempPerturber.m

```
function [ PertsT ] = TempPerturber( in1,in2,in3 )
% This function perturbs the overall temperature of everything in the
% reacotr.


global TempPerturb
global TempCor
global RoomTemp
global DelTemp
global CtoKconv
global LogTemp

% The temperature uncertainty in the SHRT book is 0.7C
PertT=LogTemp;

if TempPerturb

    prompt={'Sign for Perturbation (+ or -):','Bounding Limit (0.7C) Multiple:','Dist, 1=1, 2=sqrt(3), 3=sqrt(6)... etc'};
    name = 'Dim Perturbations';
    PertsAns = inputdlg(prompt,name,1,{'+','10','1'});

    BoundLim=str2double(PertsAns{2,1});
    Pertsign=PertsAns{1,1};
    DistF=str2double(PertsAns{3,1});

    DistFnum=sqrt((DistF-1)*3);

    if DistFnum==0; DistFnum=1; end

    ScaleF=DistFnum*BoundLim;

    if strcmp(Pertsign,'+'); PertSign=1; else PertSign=-1; end

    Pert=0.7*ScaleF*PertSign;

    PertT=DelTemp+Pert;

    PertsT={DelTemp; PertT; Pert; ScaleF};

    DelTemp=PertT;

else

    PertsT=0;

end

end
```

B.8. DensPerturber.m

```
function [ MatOut, MassDOut, PertD, SmearChngOut ] = DensPerturber( MatIn, MassDIn, MatName )
% This function perturbs the smears by changing the amount of sodium in
% the smear.

% ComboN switch is used to declare if a material is used in other
% components or not. This only applies to sodium and steel.

%% ************************** 5th circle *********************************

% This function will perturb the materials.

global     NAv
global     DenPerturb
global     BoronPerturb
global     Title
```

```
persistent SetPerturb
persistent ZAIDPertn
persistent ZAIDPerts
persistent ZnumPerts
persistent ZISOPerts
persistent PertAmount
persistent ScaleF
persistent SelectMadeD
persistent mSS304
persistent dSS304
persistent mSS316
persistent dSS316
persistent mSodium
persistent dSodium
persistent EleAbun
persistent SmearChng
persistent BoundLim

% Copy the incoming data

MatOut=MatIn;
MassDOut=MassDIn;

if isempty(SetPerturb); SetPerturb=true; end

if ~DenPerturb; SetPerturb=false; SelectMadeD=0; end

PertD=[];

if SetPerturb

    if DenPerturb

           indata=importdata('Specs\Materials\RealDb\Other Mats.xlsx');

            mattempdata=indata.textdata;
            mattempnum=indata.data;

            EleTit=mattempdata(1,:);
            MatTit=mattempdata(:,1);

            indatatemp=num2cell(mattempnum);

            OtherMats=[EleTit; [MatTit(2:end,1) indatatemp(2:end,:)]];

            [MatInd,SelectMadeIm] = listdlg('PromptString',...
                    'Select a Density to Perturb',...
                    'SelectionMode','multiple',...
                    'ListString',OtherMats(4:end,1));

            dispPrint('Material Perturbation will be performed.')

            % Make an exception for B4C

            ZAIDPerts=OtherMats{MatInd+3,1};

            TitleS=Title(end);

            if strcmp(TitleS,'P'); PertSign=''; else; PertSign='-'; end

            prompt={'Amount to change density in g/cc:','Bounding Limit Multiple:','Dist, 1=1, 2=sqrt(3), 3=sqrt(6)...
etc'};
            PertsAns = inputdlg(prompt,ZAIDPerts,1,{[PertSign '0.1'],'1','1'});

            BoundLim=str2double(PertsAns{2,1});
            PertAmount=str2double(PertsAns{1,1});
            DistF=str2double(PertsAns{3,1});

            DistFnum=sqrt((DistF-1)*3);

            if DistFnum==0; DistFnum=1; end

            ScaleF=DistFnum*BoundLim;

            if MassDIn==0; MassDIn=OtherMats{MatInd+3,2};  end

            PertD={ZAIDPerts; MassDIn+(PertAmount*ScaleF); MassDIn; PertAmount; ScaleF; BoundLim};

            if strcmp(ZAIDPerts,'B4C')

               BoronPerturb=PertD;
               PertD{2}='New density is burnup dependant';

            end

            SetPerturb=false;

            SelectMadeD=true;

            % Need to grab SS304L and SS316

            SS304Row=8; SS304Cols=[7 9:16 20:33];
            SS316Row=10; SS316Cols=[7 9:11 20:33 40:46];
            SodiumRow=4; SodiumCols=8;
```

```
            mSodium=[OtherMats(2,SodiumCols);OtherMats(SodiumRow,SodiumCols)];
            dSodium=OtherMats{SodiumRow,2};

            mSS304=[OtherMats(2,SS304Cols);OtherMats(SS304Row,SS304Cols)];
            dSS304=OtherMats{SS304Row,2};

            mSS316=[OtherMats(2,SS316Cols);OtherMats(SS316Row,SS316Cols)];
            dSS316=OtherMats{SS316Row,2};

            EleAbun=[OtherMats(2,SS316Cols);OtherMats(3,SS316Cols)];

            SmearChng=false;

    else

        dispPrint('Density Perturbation will NOT be performed.')
        dispPrint(' ')

    end

else if SelectMadeD==1



        if strcmp(MatName(end), 'C');

            Combo=true;

            MatNameOld=MatName;

            MatName=MatNameOld(1:end-1);

            if strcmp(MatName(1:2),'SS')

                Steel=true;
                Sodium=false;

            else

                Steel=false;
                Sodium=true;

            end

            SmearChng=true;

        else

            Combo=false;

        end

        if SelectMadeD==1 && strcmp(MatName,ZAIDPerts)


            if Combo

                if Steel

                    % Only 304 is used in the smears, but 316 is used int
                    % he fuel pin wirewrap

                    if strcmp(MatName(3:5),'304')

                        MassDOut=MassDIn+(PertAmount*ScaleF);

                    end

                    if strcmp(MatName(3:5),'316')

                        MassDOut=MassDIn+(PertAmount*ScaleF);

                    end

                end

                if Sodium

                   MassDOut=MassDIn+(PertAmount*ScaleF);

                end

            else

                MassDOut=MassDIn+(PertAmount*ScaleF);

            end

            PertD={ZAIDPerts; MassDIn; MassDOut; PertAmount; ScaleF; BoundLim};

        end

    end

end
```

```
SmearChngOut=SmearChng;

end
```

B.9. SmearPerturber.m

```
function [ MatOut, MassDOut, PertS ] = SmearPerturber( MatIn, MassDIn, MatName,SmearChng, PertD)
% This function perturbs the smears by changing the amount of sodium in
% the smear.

%% *************************** 5th circle *********************************

% This function will perturb the materials.

global     NAv
global     SmearPerturb
global     Title
persistent SetPerturb
persistent ZAIDPertn
persistent ZAIDPerts
persistent ZnumPerts
persistent ZISOPerts
persistent PertAmount
persistent ScaleF
persistent SelectMadeS
persistent mSS304
persistent dSS304
persistent mSS316
persistent dSS316
persistent mSodium
persistent dSodium
persistent EleAbun
persistent OtherMats

% Copy the incoming data

MatOut=MatIn;
MassDOut=MassDIn;

if isempty(SetPerturb);

    % Other Mats Import
    indata=importdata('Specs\Materials\RealDb\Other Mats.xlsx');

    mattempdata=indata.textdata;
    mattempnum=indata.data;

    EleTit=mattempdata(1,:);
    MatTit=mattempdata(:,1);

    indatatemp=num2cell(mattempnum);

    OtherMats=[EleTit; [MatTit(2:end,1) indatatemp(2:end,:)]];

    SmearMatNames=mattempdata(find(mattempnum(5:end,7))+4);

    % Need to grab SS304L and SS316

    SS304Row=8; SS304Cols=[7 9:16 20:33];
    SS316Row=10; SS316Cols=[7 9:11 20:33 40:46];
    SodiumRow=4; SodiumCols=8;

    mSodium=[OtherMats(2,SodiumCols);OtherMats(SodiumRow,SodiumCols)];
    dSodium=OtherMats{SodiumRow,2};

    mSS304=[OtherMats(2,SS304Cols);OtherMats(SS304Row,SS304Cols)];
    dSS304=OtherMats{SS304Row,2};

    mSS316=[OtherMats(2,SS316Cols);OtherMats(SS316Row,SS316Cols)];
    dSS316=OtherMats{SS316Row,2};

    EleAbun=[OtherMats(2,SS316Cols);OtherMats(3,SS316Cols)];

    SetPerturb=true;

end

if ~SmearPerturb; SetPerturb=false; SelectMadeS=0;  end

PertS=0;

if SetPerturb

    % Construct a questdlg with three options
    Pertchoice = questdlg('Would you like to perturb a Smear?', ...
        'Pert Choice', ...
        'Yes','No','No');

    % Handle response
    switch Pertchoice
```

```
        case 'Yes'

            [MatInd,SelectMadeIm] = listdlg('PromptString',...
                    'Select a smear to Perturb',...
                    'SelectionMode','multiple',...
                    'ListString',SmearMatNames);

            dispPrint('Material Perturbation will be performed.')

            ZAIDPerts=SmearMatNames{MatInd};

            TitleS=Title(end);

            if strcmp(TitleS,'P'); PertSign=''; else; PertSign='-'; end


            prompt={'Amount to change sodium content in wt%:','Scaling Factor:'};
            PertsAns = inputdlg(prompt,ZAIDPerts,1,{[PertSign '0.1'],'10'});


            ScaleF=str2double(PertsAns{2,1});
            PertAmount=str2double(PertsAns{1,1})/100;

            PertS={ZAIDPerts; 0; 0; PertAmount; ScaleF};

            SetPerturb=false;

            SelectMadeS=true;

        case 'No'

            dispPrint('Smear Perturbation will NOT be performed.')
            dispPrint(' ')

    end

else if SelectMadeS==1 & strcmp(MatName,ZAIDPerts)


        if SelectMadeS==1

            MatInL=size(MatIn,2);

            if MatInL==size(mSS316,2)+1

                OldSSm=mSS316;
                OldSSd=dSS316;

            end

            if MatInL==size(mSS304,2)+1

                OldSSm=mSS304;
                OldSSd=dSS304;

            end

            NaWt=MatIn{2,cell2mat(MatIn(1,:))==11023};
            SSWt=abs(1-NaWt);

            NewNaWt=NaWt+ScaleF*PertAmount;
            NewSSWt=abs(1-NewNaWt);

            if NewNaWt<=0; keyboard; end

            clear MatOut

            MatOut=cell(2,size(MatIn,2));

            MoffSet=0;

            for jj=1:size(MatIn,2)

                CurZAID=MatIn{1,jj};
                MatOut{1,jj}=CurZAID;

                if CurZAID==11023

                    MatOut{2,jj}=NewNaWt;
                    MoffSet=1;

                else

                    MatOut{2,jj}=OldSSm{2,jj-MoffSet}*NewSSWt;

                end

            end

            % Recalculate the mass density

            MassDOut=dSodium*NewNaWt+OldSSd*NewSSWt;

            % Perform a check
```

```
            if RoundM(sum(cell2mat(MatOut(2,:))),5)~=1

                % Error in composition
                keyboard

            end

        end

    end

end

% The following lines are to adjust the density of either sodium or steel
% if they have been perturbed by DensPerturber

if SmearChng


    if strcmp('SUP',MatName); dSSmat=dSS304; end
    if strcmp('Slo',MatName); dSSmat=dSS304; end
    if strcmp('SLC',MatName); dSSmat=dSS304; end
    if strcmp('SIHDLC',MatName); dSSmat=dSS316; end
    if strcmp('SafetySUP',MatName); dSSmat=dSS304; end
    if strcmp('B4CShieldSmear',MatName); dSSmat=dSS304; end
    if strcmp('RefSS304s',MatName); dSSmat=dSS304; end
    if strcmp('RefSS316s',MatName); dSSmat=dSS316; end


    NaWt=MatIn{2,cell2mat(MatIn(1,:))==11023};
    SSWt=abs(1-NaWt);

    MassDOut=(dSodium*NaWt)+(dSSmat*SSWt);

end

end
```

B.10. OtherMatsPerturber.m

```
function [ Perts ] = OtherMatsPerturber(  )
% Simple file to ask the user about what was perturbed and how

global OtherMatsPerturb
global OtherMatsPath
global BoronPerturb
Perts=0;

if OtherMatsPerturb

    SlInd=strfind(OtherMatsPath,'\');
    PInd=strfind(OtherMatsPath,'.');

    ElePert=OtherMatsPath(SlInd(end)+1:PInd(end)-2);

    MatPert=OtherMatsPath(SlInd(end-1)+1:SlInd(end)-1);

    SignIn=OtherMatsPath(PInd(end)-1);

    if strcmp(MatPert,'B4C');

        BoronPerturb=true;

        ChangeLabel='Change in grams of ';

    else

        ChangeLabel='% Change in WT% of ';

    end

    if strcmp(SignIn,'M'); SignO='-'; end
    if strcmp(SignIn,'P'); SignO='+'; end

    prompt={'Other Material',[ChangeLabel ElePert ':'],'Sacling Factor:','Sign:'};

    PertsAns = inputdlg(prompt,ElePert,1,{MatPert,'5','1',SignO});


    Perts={PertsAns{1,1}; ElePert; PertsAns{3,1}; PertsAns{2,1}; PertsAns{4,1}};

end

end
```

B.11. ReactorMapper.m

```
function [ ReactMap ] = ReactorMapper( nsa )

%This program must also provide a translator for EBR-II positions to MICKA
```

```
%positions.

global BorD
global debugMICKA
global debugCore
global DefaultLayout
global SmearSATYs
global SmearBlk1316
global debug1SAMode

MapPath='Specs\CoreMaps';
KatanaCorePath=[MapPath '\Katana Cores'];

if DefaultLayout && ~debugMICKA && ~debugCore

    if SmearBlk1316

        [MICKAnums,ReactMapTemp]=xlsread([MapPath '\SHRT45_BS_IR.xlsx']);

    else

        [MICKAnums,ReactMapTemp]=xlsread([MapPath '\SHRT45Map.xlsx']);

    end

end

if ~DefaultLayout && ~debugMICKA && ~debugCore

    [ filename, filepath ] = FileSelect( MapPath );

    [MICKAnums,ReactMapTemp]=xlsread([filepath '\' filename]);

end

if debugMICKA && ~debugCore

    [ filename, filepath ] = FileSelect( MapPath );

    [MICKAnums,ReactMapTemp]=xlsread([filepath '\' filename]);

end

if debugMICKA && debugCore

   [MICKAnums,ReactMapTemp]=xlsread([MapPath '\DebugMap.xlsx']);

end

if ~BorD && ~debugCore

    [ filename, filepath ] = FileSelect( KatanaCorePath );

    [MICKAnums,ReactMapTemp]=xlsread([filepath '\' filename]);

end

if debug1SAMode

   [MICKAnums,ReactMapTemp]=xlsread([MapPath '\1SADebugMap.xlsx']);

end

% Search for smeared SATYs

SmearSATYs=zeros(1,3);

for ii=1:size(MICKAnums+1,1)

    SATY=MICKAnums(ii,2);
    Mnum=MICKAnums(ii,1);

    if SATY>10 && SATY~=30

       if SmearSATYs(1)==0

           SmearSATYs(end,3)=SATY;

       else

           SmearSATYs(end+1,3)=SATY;

       end

       SATYtemp=Num2StrM(SATY);

       SATY=str2num(SATYtemp(2));

       % Adds the standard type number.

       SmearSATYs(end,2)=SATY;
       SmearSATYs(end,1)=Mnum;

    end
```

```
    ReactMapTemp{ii+1,1}=MICKAnums(ii,1);
    ReactMapTemp{ii+1,2}=SATY;

end

% SmearSATYs=unique(SmearSATYs,'rows');

ReactMap=ReactMapTemp(1:nsa+1,:);

%Output the ReactMap
dispPrint('Exporting Reactor Map');
dispPrint(' ')
delete('Maps\ReactMap.xls');
ReactMapTemp=ReactMap;
ReactMapTemp(2:end,3)=strcat('''',ReactMapTemp(2:end,3));
xlswrite('Maps\ReactMap.xls',ReactMapTemp);

global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotReactDat.mat')
    dispPrint('     TotReactDat saved....');

end

end
```

B.12. MoveMapper.m

```
function [ MoveMap, Crit, MoveSAs ] = MoveMapper( ReactMap , nsa, MoveDefault )
%This function outputs the inches of insertion for the controls and
%safeties.

%% *************************** 2nd circle ********************************

global UseDefault
global SaveDebugMoveVars

Crit=false;

%Creates The id part of the movement map
saIDmove={'MATLAB ID','MICKA Type','Position','Identifier','Type of Assembly';...
             10    ,     3      ,'03D01'   ,'S678'      ,'Safety';...
             16    ,     3      ,'03A01'   ,'S677'      ,'Safety';...
             40    ,     4      ,'05C03'   ,'L4228S'    ,'HWCR';...
             42    ,     5      ,'05D01'   ,'L4302N'    ,'Control';...
             46    ,     4      ,'05E01'   ,'L4224S'    ,'HWCR';...
             48    ,     4      ,'05E03'   ,'L4225S'    ,'HWCR';...
             50    ,     4      ,'05F01'   ,'L4226S'    ,'HWCR';...
             54    ,     4      ,'05A01'   ,'L4227S'    ,'HWCR';...
             58    ,     4      ,'05B01'   ,'L4230S'    ,'HWCR';...
             60    ,     4      ,'05B03'   ,'L4229S'    ,'HWCR';};

% It is unlcear whether the experimental assemblies need to move,
% but I am going to include code for them anyway. The format is the same as saIDmove

saIDmoveEXP={38    ,     10     ,'05C01'   ,'XX10'      ,'INST 10';...
             44    ,     10     ,'05D03'   ,'XX09'      ,'INST 09';...
             52    ,     10     ,'05F03'   ,'XY-16'     ,'Dummy Control';};

%Move Map
%The following values are in inches!!!

%Total movement value, adjust individual assemblies by change the value
%after TotMov

% % % Critical
% TotMov=14;                      % Crit: 14
%
%   samove={'Insertion Distance';
%         TotMov-0;... %10 Safety    Crit: 0
%         TotMov-0;... %16 Safety    Crit: 0
%         TotMov-14;... %40 HWCR     Crit: -14
%         TotMov+0;... %42 Control   Crit: 0
%         TotMov+0;... %46 HWCR      Crit: 0
%         TotMov-10.99;... %48 HWCR  Crit: -10.99
%         TotMov-0;... %50 HWCR      Crit: 0
%         TotMov-0;... %54 HWCR      Crit: 0
%         TotMov-14;... %58 HWCR     Crit: -14
%         TotMov+0};   %60 HWCR      Crit: 0

% Custom

for kk=2:size(saIDmove,1)

    Rods{kk-1,1}=['M ID:',Num2StrM(saIDmove{kk,1}),' ',saIDmove{kk,3},' ',saIDmove{kk,4},' ',saIDmove{kk,5},':'];

end

dispPrint('Rod Movements will be applied')
dispPrint(' ')
```

```
% 68 MW
% MoveDefault={'8','8','9','9','9','8.97','9','9','9','9'};

% 200 kW
% MoveDefault={'8','8','9','9','9','4.81','9','9','9','9'};

% 0 MW

name = 'Rod Movement';

if UseDefault

    MoveAns = MoveDefault;

else

    MoveAns = inputdlg(Rods,name,1,MoveDefault);

end

samove={'Insertion Distance';
    str2double(MoveAns{1});... %10 Safety    Crit: 14
    str2double(MoveAns{2});... %16 Safety    Crit: 14
    str2double(MoveAns{3});... %40 HWCR     Crit: 0
    str2double(MoveAns{4});... %42 Control   Crit: 14
    str2double(MoveAns{5});... %46 HWCR      Crit: 14
    str2double(MoveAns{6});... %48 HWCR  Crit: 3.01
    str2double(MoveAns{7});... %50 HWCR      Crit: 14
    str2double(MoveAns{8});... %54 HWCR      Crit: 14
    str2double(MoveAns{9});... %58 HWCR     Crit: 0
    str2double(MoveAns{10})};   %60 HWCR      Crit: 14

if str2double(MoveAns{1})==14   && str2double(MoveAns{2})==14   &&...
   str2double(MoveAns{3})==0    && str2double(MoveAns{4})==14   &&...
   str2double(MoveAns{5})==14   && str2double(MoveAns{6})==3.01 &&...
   str2double(MoveAns{7})==14   && str2double(MoveAns{8})==14   &&...
   str2double(MoveAns{9})==0    && str2double(MoveAns{10})==14;

Crit=true;

end


%
%  TotMov=14;                       % Crit: 14
%
%   samove={'Insertion Distance';
%         TotMov-0;... %10 Safety    Crit: 0
%         TotMov-0;... %16 Safety    Crit: 0
%         TotMov-14;... %40 HWCR     Crit: -14
%         TotMov-0;... %42 Control   Crit: 0
%         TotMov-14;... %46 HWCR     Crit: 0
%         TotMov-0;... %48 HWCR      Crit: -10.99%
%         TotMov-14;... %50 HWCR     Crit: 0
%         TotMov-0;... %54 HWCR      Crit: 0
%         TotMov-0;... %58 HWCR      Crit: -14
%         TotMov-10.99};   %60 HWCR  Crit: 0

%*************** There should be no need to change the EXPs **************

TotMovEXP=14;

samoveEXP={TotMovEXP+0;... %38 INST 10
           TotMovEXP+0;... %44 INST 09
           TotMovEXP+0};   %52 Dummy Control

MoveSAs=[saIDmove, samove];

MoveMapT=[[saIDmove; saIDmoveEXP], [samove; samoveEXP]];

%Output the MoveMap
xlswrite('Maps\MoveMap.xls',MoveMapT)

VarTitles={'Insertion Distance'};

MoveMap=[ReactMap, [VarTitles; cell(nsa,length(VarTitles))]];

for mm=2:length(MoveMapT)

    % Creates the move map variabe

    SAfoundRows=cellfind(MoveMap(:,3),MoveMapT{mm,3},1,'Match');

    if ~SAfoundRows==0
       %This writes insertion position to the assemblies

       MoveMap(SAfoundRows,6)=MoveMapT(mm,6);

    end

end

if SaveDebugMoveVars
```

```
    % Save TotDimDat
    save('Debug\TotMoveDat.mat')
    dispPrint('     TotDat saved....');

end

end
```

B.13. cellfind.m

```
function [ inds ] = cellfind( cellin,varin,rowcol,Match )
%This function takes in a cell array and a variable to be found and outputs
%its index number inside of the cell.
%This only works for cell arrays that do not contain subcell arrays.
%multiple finds will yeild a numerical array with rows in column 1 and
%columns in column 2.

%rowcol=1 means output row number
%rowcol=2 means output col number
%rowcol=3 means output both numbers

%An Output of 0 means it did not find it.


% Match means match exact case and length

if strcmp(Match,'Match')

    Matchcase=true;

else

    Matchcase=false;

end

%Subindex
kk=1;

if isnumeric(varin)


    for ii=1:size(cellin,1)

        %Extracts lines from the cell array
        cellintemp=cellin(ii,:);

        %converts the finds in 0 and the emptycells into 1
        cellinindarnum=cellfun(@isnumeric,cellintemp);

        %Gets the column number
        colnum=find(cellinindarnum==1);

        if ~(colnum==0)

            if cellintemp{colnum}==varin

                  if kk>1

                      indouttemp=[ii,colnum];

                      indout=[indout; indouttemp];

                  end

                  if kk==1

                     indout=[ii,colnum];

                     kk=kk+1;

                  end

            end

        end


    end

end

if ischar(varin)


    for ii=1:size(cellin,1)

        %Extracts lines from the cell array
        cellintemp=cellin(ii,:);

        %converts everything to a string
        cellintemp2=cellfun(@(x) Num2StrM( x,'%.8g'),cellintemp,'un',0);
```

```
        %Finds if varin is in the line
        cellinindar=strfind(cellintemp2,varin);

        %converts the finds in 0 and the emptycells into 1
        cellinindarnum=cellfun(@isempty,cellinindar);

        %Gets the column number
        colnum=find(cellinindarnum<1);

        if isempty(colnum)

            colnum=0;

        end

        if ~(colnum(1,1)==0)

            for ll=1:length(colnum)

                if Matchcase

                    if length(cellintemp{colnum(ll)})==length(varin)

                           if kk>1

                               indouttemp=[ii,colnum(ll)];

                               indout=[indout; indouttemp];

                           end

                           if kk==1

                              indout=[ii,colnum(ll)];

                              kk=kk+1;

                           end

                    end

                else

                           if kk>1

                               indouttemp=[ii,colnum(ll)];

                               indout=[indout; indouttemp];

                           end

                           if kk==1

                              indout=[ii,colnum(ll)];

                              kk=kk+1;

                           end

                end

            end

        end


    end

end

if ~exist('indout','var'); indout=[0,0]; end

if rowcol==1; inds=indout(:,1); end

if rowcol==2; inds=indout(:,2); end

if rowcol==3; inds=indout; end

end
```

B.14. DescWriter.m

```
function [ DescOut ] = DescWriter( Title, MCNPin, PertsM, PertsD, PertsS, PertD, PertsOM, PertsT, MoveMap, FeaturesM,
Crit, ProbSet )
% This function will prompt the user to add a description to the input file.

% Combine MCNP parameters into one string.

global comout
global BoronPerturb

%% Set the material paras
```

```
if ~iscell(PertsM)

    PertMpara='No Material Perturbations Performed';

else

    PertMpara={['Perturbed ZAID:' PertsM{1}]; ...
           ['ZAID Isotope:' PertsM{2}]; ...
           ['ZAID Znum:' PertsM{3}]; ...
           ['Fraction Chng:' Num2StrM(PertsM{4})]; ...
           ['Scale Fact:' Num2StrM(PertsM{5})]};

end

%% Sodium Smears

if ~iscell(PertsS)

    PertSpara='No Material Perturbations Performed';

else

    PertSpara={['Perturbed Smear Material:' PertsS{1}]; ...
           ['Change in Sodium WT%:' Num2StrM(PertsS{4}*100)]; ...
           ['Scale Fact:' Num2StrM(PertsS{5})]};

end

%% Density Smears

if ~iscell(PertD)

    PertsDpara='No Material Perturbations Performed';

else

    ScaleF=PertD{5};
    BoundLim=PertD{6};

    if rem(ScaleF,1) == 0 ;

        ScaleFOut=Num2StrM(PertD{5});

    else

        ScaleFOut=[Num2StrM(BoundLim) ' * SQRT(' Num2StrM((ScaleF/BoundLim)^2) ')'];

    end

    PertsDpara={['Perturbed Density Material:' PertD{1}]; ...
           ['New Density:' Num2StrM(PertD{2})]; ...
           ['Old Density:' Num2StrM(PertD{3})]; ...
           ['Change in Density:' Num2StrM(PertD{4})]; ...
           ['Scale Fact:' ScaleFOut]};

end

%% Other MAterials

if ~iscell(PertsOM)

   PertOMpara='No Material Perturbations Performed';

else

    if BoronPerturb

        ChangeLabel='Change in grams of ';

    else

        ChangeLabel='% Change in Wt% of ';

    end

   PertOMpara={['Perturbed Other Material:' PertsOM{1}]; ...
              [ChangeLabel PertsOM{2} ':' PertsOM{4}]; ...
              ['Scaling Factor:' PertsOM{3}];...
              ['Sign:' PertsOM{5}]};

end

%% set the dimensional paras

if ~iscell(PertsD)

    PertDpara='No Dimensional Perturbations Performed';

else

    if PertsD{2}>0; PertDSign='+'; else PertDSign='-'; end

    EXPs=PertsD{4,1};

%    if isempty(EXPsPert); EXPs='Alls EXPs Perturbed'; else; EXPs=strjoin(EXPsPert,','); end
```

```
    PertDpara={['Perturbed Dim:' PertsD{1}]; ...
           ['Sign on Perturbation:' PertDSign]; ...
           ['Pert Amount:' EXPs]; ...
           ['Scale Fact:' Num2StrM(PertsD{6})]};

end

%% Temperature

if ~iscell(PertsT)

   PertTpara='No Temperature Perturbation Performed';

else

   PertTpara={['Logbook Temperature:' Num2StrM(PertsT{1})]; ...
              [ 'New Temperature:' Num2StrM(PertsT{2})]; ...
              [ 'Change in Temperature:' Num2StrM(PertsT{3})];...
              [ 'Scaling Factor:' Num2StrM(PertsT{4})]};

end

%% set the mcnp paras

Mpara={['     NPG=' Num2StrM(MCNPin{2})]; ...
       ['     Kguess=' Num2StrM(MCNPin{3})]; ...
       ['     NSK=' Num2StrM(MCNPin{4})]; ...
       ['     NG=' Num2StrM(MCNPin{5})]; ...
       ['     Cpus=' MCNPin{1}]};

%% Set the features paras

Fpara={['     Temp Corrs Density (Na,SS), Xsec: ' FeaturesM{1}]; ...
       ['     Cross Section Set: ' MCNPin{11}]; ...
       ['     Temperature Change from 20C: ' MCNPin{12}]; ...
       ['     Fuel Element Swelling: ' FeaturesM{3}]; ...
       ['     Fuel Slug Swelling: ' FeaturesM{4}]; ...
       ['     Sodium porosity: ' FeaturesM{5}];...
       ['     S, Alpha, Beta Correction: ' FeaturesM{6}]; ...
       ['     Boron Burnup: ' FeaturesM{7}]; ...
       ['     Remove Lower Adpaters: ' FeaturesM{8}];...
       ['     Smear Blanket Rows 13-16: ' FeaturesM{9}];...
       ['     Sodium Above Slug Height ThermX Correction: ' FeaturesM{10}];...
       ['     MKII versus MKIIA Distinction:' FeaturesM{11}]};

cellc={'c ', 'Cell Cards'};

%% Set the movement paras

MMPOSl=length(MoveMap{1,3})+2;
MMIDl=length(MoveMap{1,4})+2;
MMTypel=length(MoveMap{1,5})+2;
MMInDl=length(MoveMap{1,6})+2;

if Crit

    MovePara={'The Rods are in Critical Position'};

else

    for jj=1:size(MoveMap,1)

        curPOSl=length(MoveMap{jj,3});
        curIDl=length(MoveMap{jj,4});
        curTypel=length(MoveMap{jj,5});
        curInDl=length(Num2StrM(MoveMap{jj,6}));

        curPOSspc=[':' repmat(' ',1,MMPOSl-curPOSl)];
        curIDspc=[':' repmat(' ',1,MMIDl-curIDl)];
        curTypespc=[':' repmat(' ',1,MMTypel-curTypel)];
        curInDspc=[':' repmat(' ',1,MMInDl-curInDl)];

        if jj==1

            MovePara{jj,1}=[MoveMap{1,3} ': ' MoveMap{1,4} ': '...
                            MoveMap{1,5} ': ' MoveMap{1,6}];

        else

            MovePara{jj,1}=[  MoveMap{jj,3} curPOSspc MoveMap{jj,4}...
                    curIDspc MoveMap{jj,5} curTypespc...
                    Num2StrM(MoveMap{jj,6}) curInDspc];

        end

    end

end

%% Prompt user for a narative

N=22;
prompt = ['Enter a Run Description for :' Title...
    ' Run paramters are automatically entered into the input file'];
```

```
dlg_title = 'EBR-II Input File Description';
num_lines = 10;
defans={'';'';'';'';''};
options.Resize='on';
options.WindowStyle='normal';
options.Interpreter='tex';

DescripText=inputdlg(prompt,dlg_title,...
    [num_lines, length(dlg_title)+N],defans,options);


%% Write the paras

NewPara=false;

if iscell(PertMpara); Pertpara=PertMpara; NewPara=true; end
if iscell(PertOMpara); Pertpara=PertOMpara; NewPara=true; end
if iscell(PertsDpara); Pertpara=PertsDpara; NewPara=true; end
if iscell(PertSpara); Pertpara=PertSpara; NewPara=true; end
if iscell(PertMpara); Pertpara=PertMpara; NewPara=true; end

if ~NewPara; Pertpara='No Material Perturbations Performed'; end

RunPara=[ProbSet;...
        {' ';'Rod Insertion Distances';' '};...
        MovePara;...
        {' ';'Model Features';' '};...
        Fpara;...
        {' ';'MCNP Parameters'};...
        Mpara;...
        {' '; 'Perturbation Parameters';'Materials:';' '};...
        Pertpara;...
        {' ';'Dimensions:';' '};...
        PertDpara;...
        {' ';'Temperature:';' '};...
        PertTpara;...
        {' ';'Description of Input:'; ' '}];

if isempty(DescripText)

    DesText=[RunPara; {'Did not write a description.'}];

else

    DesText=[RunPara; DescripText];

end

if size(DesText,2)>78

    kk=1;

    if size(DesText,2)>1; TLen=size(DesText,1); else TLen=1; end

    for ii=1:TLen

       NotDone=true;

       curln=DesText(ii,:);

       curlnLen=length(curln);

       if curlnLen<78; NotDone=false; OutLnT(kk,1:2)={'c',curln}; kk=kk+1; end

       while NotDone

           spcind=strfind(curln,' ');

           spcindslt=spcind(spcind<78);

           NlnL=curln(1,1:spcindslt(end));
           NlnR=curln(1,spcindslt(end)+1:end);

           OutLnT(kk,1:2)={'c',NlnL};

           kk=kk+1;

           curln=NlnR;

           if length(curln)<78; NotDone=false; OutLnT(kk,1:2)={'c',curln}; kk=kk+1; end

       end

    end

else

    TLen=size(DesText,1);

    for jj=1:TLen

        OutLnT(jj,1:2)=[{'c'},DesText(jj,:)];

    end
```

```
end

% Temporarily Enable and then disable comments

ChngComOut=0;

if strcmp(comout,'N'); ChngComOut=1 ;comout='Y'; end

DescTitle1=NameCard([Title ' Input File Description '],'Title');
DescTitle2=NameCard([Title ' Input File Input Cards '],'Title');

if ChngComOut==1 ;comout='N'; end

DescOut=[DescTitle1; OutLnT; DescTitle2; cellc];

global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotDescDat.mat')
    dispPrint('     TotDescDat saved....');

end

end
```

B.15. DimMapper.m

```
function [ DimMap, hdp ] = DimMapper( ReactMap,MoveMap,PertsD)
%Dim map, reads in dimensions from excel files and then uses those value in
%MICKA. They are all written to the master variable DimMap.

global SAlphaBetas
global PerturbDim
global debugMICKA
global SaveDebugDimVars
global MKIIvsMKIIA MKIIAfeh MKIIAfslas fslasC
global LAT
global sabDefs
global SABdefault
global SABcorr
global PitchPerturb

dispPrint('Loading and sorting the dimensional data.')
dispPrint('Please be patient...')
dispPrint(' ')

% Determine dim map size.

nsa=size(ReactMap,1)-1;

% Read in the Standard Dims Data

[~,~,dimdbintemp]=xlsread('Specs\Dimensions\StandardDims.xlsx');

dimdbintemp=delNaN(dimdbintemp,'Convert All to','num');
dimdbintemp2=dimdbintemp(5:end,3:end);

%% Dim Vars breakout
% This section calls the defined columns for the dimensions

global MNumC MSATyC SAPosC SANomen NomenTyC
global upphC ihdodC bhC
global sazmC
global npeC
global nfeC
global ndeC

%% Conversion to cm

%Convert to centimeteres for dimensions and samovement.
dimdbin=cellfun(@(inches) inches*2.54,dimdbintemp2,'un',0);

% The move map needs to be done as well but only once since it is shared
% between the experiments and the other SAS

MoveMap(2:end,6)=cellfun(@(inches) inches*2.54,MoveMap(2:end,6),'un',0);

%Undo the conversion on the number pins
dimdbin(:,ndeC-5)=cellfun(@(inches) round(inches/2.54),dimdbin(:,ndeC-5),'un',0);
dimdbin(:,nfeC-5)=cellfun(@(inches) round(inches/2.54),dimdbin(:,nfeC-5),'un',0);
dimdbin(:,npeC-5)=cellfun(@(inches) round(inches/2.54),dimdbin(:,npeC-5),'un',0);

%Extract Dim Variables titles
DimVarTitles=dimdbintemp(3,3:end);

%Creates dim map
DimMap=[ReactMap, [DimVarTitles; cell(nsa,length(DimVarTitles))]];

% Read in the custom assembly data

[~,~,dimdbintempEXP]=xlsread('Specs\Dimensions\CustomDims.xlsx');
```

```
dimdbintempEXP=delNaN(dimdbintempEXP,'Convert All to','num');
dimdbintempEXP2=dimdbintempEXP(4:end,4:end);

EXPID=dimdbintempEXP(4:end,1);

%Convert to centimeteres for dimensions and samovement.
dimdbinEXP=cellfun(@(inches) inches*2.54,dimdbintempEXP2,'un',0);

%Undo the conversion on the number pins
dimdbinEXP(:,ndeC-5)=cellfun(@(inches) round(inches/2.54),dimdbinEXP(:,ndeC-5),'un',0);
dimdbinEXP(:,nfeC-5)=cellfun(@(inches) round(inches/2.54),dimdbinEXP(:,nfeC-5),'un',0);
dimdbinEXP(:,npeC-5)=cellfun(@(inches) round(inches/2.54),dimdbinEXP(:,npeC-5),'un',0);

%%
%**************************Dimension Writer**********************************

for mm=1:nsa

    %nsa stands for number of assemblies, but it also coincides with the
    %MICKA num so it can also be used as an ID inside of loops.

    %This is sorted by each assembly type.
    SAty=DimMap{mm+1,MSATyC};

    if ~(SAty==0) && ~(SAty==10)

        DimMap(mm+1,upphC:end)=dimdbin(SAty,:);

    end

    % The following writes the experimental dimensions.

    if SAty==10

        % Grab the ID

        SAID=DimMap{mm+1,SANomen};

        EXPcol=cellfind(EXPID,SAID,1,'Match');

        DimMap(mm+1,upphC:end)=dimdbinEXP(EXPcol,:);

    end

    InsDist=MoveMap{mm+1,6};

    if ~isempty(InsDist)
       %This writes insertion position to the assemblies

       DimMap(mm+1,sazmC)={InsDist};

    end

    if mm==nsa
        %Adds the core dimensions
           DimMap(end+1,MNumC)={'Core'};
           DimMap(end,MSATyC)={'NA'};
           DimMap(end,SAPosC)={'Everywhere'};
           DimMap(end,SANomen)={'EBR-II'};
           DimMap(end,NomenTyC)={'Pool Type'};
           DimMap(end,upphC:end)=dimdbin(end,:);

    end

end

%% Load in the data to discriminate MKII from MKIIA

if MKIIvsMKIIA

    [ MKIImap, DimMap ]= MKIIPinTypeLoader(DimMap);

else

    % If not distinguishment the make all sodium levels a MKII sodium level

    for bb=2:size(DimMap,1)

        curfslas=DimMap{bb,fslasC};

        if ~isempty(curfslas)

            DimMap{bb,fslasC}=MKIIAfslas;

        end

    end

end

%% Dimensional Perturber

DimMapFileName='DimMap';
```

```
if PerturbDim

    % Keep a record of the old dimensions

    DimMapUnPert=DimMap;

    [ NumObjPert, DimMap, Perts] = DimsPerturber( DimMap, 0, 0 );

    delete('Maps\DimMap.xls');
    DimMapUnPert(2:end,SAPosC)=strcat('''',DimMapUnPert(2:end,SAPosC));
    xlswrite(['Maps\' DimMapFileName '.xls'],DimMapUnPert);

    DimMapFileName='DimMap_Perturbed';

end

% The reactor pitch needs to be calculated.

% The hdp comes from the system design description
hdp=DimMap{end,ihdodC}+(2*DimMap{end,bhC});

if PitchPerturb

    PitchPert=PertsD{2}*str2double(PertsD{4})*PertsD{6};

    DimMap{end,ihdodC}=DimMap{end,ihdodC}+PitchPert;

    hdp=hdp+PitchPert;

end

if debugMICKA && ~LAT; hdp=10; end

%% S alpha beta selection

if SABcorr

    sab={'26056';'6000'};

    SAtypes=[dimdbintemp(4:end,1:2);[dimdbintempEXP(4:end,1), dimdbintempEXP(4:end,3)]];

    if ~SABdefault

        [SAtypesIND,SelectMade] = listdlg('PromptString',...
            's(a,b) applied to what SA''s','SelectionMode',...
            'multiple','InitialValue',[2:9,12:18],...
            'ListString',SAtypes(:,1));

    else

        SAtypesIND=[2:9,11:18];
        SelectMade=true;

    end

    if SelectMade

        SAtypesSelection=[SAtypes(SAtypesIND,1), SAtypes(SAtypesIND,2)];

        for hh=1:size(SAtypesSelection)

            if ~SABdefault

                [sabSelect,SelectMade] = listdlg('PromptString',...
                    ['s(a,b) applied to ' SAtypesSelection{hh}],...
                'SelectionMode','multiple','InitialValue',[sabDefs],...
                'ListString',sab);

                SAtypesSelection{hh,3}=sab(sabSelect);

            else

                SAtypesSelection{hh,3}=sab(sabDefs);

            end

        end

    end

    SAlphaBetas=SAtypesSelection;

end

if SaveDebugDimVars

    % Save TotDimDat
    save('Debug\TotDimDat.mat')
    dispPrint('     TotDat saved....');

end

%Output the DimMap
dispPrint('Exporting Dimensional Map');
dispPrint(' ')
```

```
delete('Maps\DimMap.xls');
DimMapTemp=DimMap;
DimMapTemp(2:end,SAPosC)=strcat('''',DimMapTemp(2:end,SAPosC));
xlswrite(['Maps\' DimMapFileName '.xls'],DimMapTemp);

end
```

B.16. MatMapper.m

```
function [ MatMap, ReactMap, BurnMap, MassMap, DimMap ] = MatMapper( ReactMap, DimMap, filename )
%MatMap takes in all the positions and matches position with material.

%% *************************** 2nd circle ******************************

global debugMICKA SaveSAMats BadBoronMat ExportOtherMats
global TempCor NaHTempCorr
global KeyboardStop
global SaveDebugMatVars
global DelTemp
global SwellPin
global SwellFuelSlug
global B4CBurnB
global RemoveLAB
global BorD

%% Global Variable List

% Reactor Map

global MNumC MSATyC SAPosC SANomen NomenTyC SmearSATYs

% Dimension Vars

global bhC hdidC hdwtC ihdodC ihdidC ihdwtC crhC fshC fsdC fehC feidC feodC
global fewtC nfeC eshC esdC  ndeC

% Material Vars

global FmC FdC FdatC FNamC FNadC FNadatC FPGmC FPGdC FPGdatC FCladmC FCladdC
global fslasC FCladdatC PmC PdC PdatC FPwwrmC FPwwdC FPwwdatC PPGmC PPGdC
global PPGdatC PCladmC PCladdC PCladdatC DmC DdC DdatC DuctmC DuctdC DuctdatC
global DnamC DnadC DnadatC SUPmC SUPdC SUPdatC SLOmC SLOdC SLOdatC LAmC LadC
global LadatC IHDLAmC IHDLAdC IHDLAdatC OPmC OPdC OPdatC ShieldPmC ShieldPdC
global ShieldPdatC

%%  Initialize variables

dispPrint('Loading and sorting the material data takes time.')
dispPrint('Please be patient...')
dispPrint(' ')

% Initialize the volume change variable. This is used to calculate
% additional sodium added to swelled slugs
VolChng=0;

% Determine material map size.
nsa=size(ReactMap,1)-1;

% Chart of nuclides import
ChONtemp=delNaN( importdata('Specs\Materials\ChON.xlsx'), 'Convert All' , 'num' );
ChON=ChONtemp(:,1:end-1);

%% Load in the data from test directory

dispPrintMats={'Drivers MKIIAI';'Drivers MKIIA';'Safeties';...
    'Experimentals';'Controls';'SST';'SSR';'Blankets';'Custom'};

[ matdbin ] = RealDbLoader( dispPrintMats );

%% Loads the other materials spresheet, B4C SS's etc...

dispPrint('Reading Other Materials and Calculating atom densities');
dispPrint(' ')

[OtherMatsRows, OtherMaterials, ISOadenOtherMats ] = OtherMatsLoader( ChON );

%% Load the B4C for the HWCR

[ B4CDatOut, ISOadenB4C ] = HWCRB4CLoader( ChON );

%% Temp adjustment

% Delta k in kelvin/celcius

OtherMaterialsNoAdj=OtherMaterials;
ISOadenOtherMatsNoAdj=ISOadenOtherMats;

if TempCor

     [OtherMaterials, ISOadenOtherMats]  = DenTempAdj( OtherMatsRows, OtherMaterialsNoAdj, ISOadenOtherMats , DelTemp );

end
```

```
%% Output other mats materials

if ExportOtherMats

    for rr=1:size(ISOadenOtherMats,1)

        MatTitle=ISOadenOtherMats{rr,1};

        ISOAdenT=ISOadenOtherMats{rr,2};

        if ~isempty(MatTitle)

            ZAID=cell2mat(cellfun(@(x) str2double(x),ISOAdenT{1,2}','un',0));

            ADCompM=ISOAdenT{4,2}';

            MatT=flipud([ZAID,ADCompM]);

            csvwrite(['Debug\OtherMatVar\' MatTitle '.csv'],MatT);

        end

    end

end


%% Decompile the information

% Atom Densities

adSodium=   OtherMaterials{OtherMatsRows(1)-3,2};
adArgon=    OtherMaterials{OtherMatsRows(2)-3,2};
adHelium=   OtherMaterials{OtherMatsRows(3)-3,2};
adPgas=     OtherMaterials{OtherMatsRows(4)-3,2};
adSS304=    OtherMaterials{OtherMatsRows(5)-3,2};
adB4Cwwr=   OtherMaterials{OtherMatsRows(6)-3,2};
adSS316=    OtherMaterials{OtherMatsRows(7)-3,2};
adFPwwr=    OtherMaterials{OtherMatsRows(8)-3,2};
adB4C=      OtherMaterials{OtherMatsRows(9)-3,2};
adSUP=      OtherMaterials{OtherMatsRows(10)-3,2};
adSLO=      OtherMaterials{OtherMatsRows(11)-3,2};
adSLC=      OtherMaterials{OtherMatsRows(12)-3,2};
adSIHDLC=   OtherMaterials{OtherMatsRows(13)-3,2};
adSafetySUP=OtherMaterials{OtherMatsRows(14)-3,2};
adB4CSUP=   OtherMaterials{OtherMatsRows(15)-3,2};
adRefSS304s=OtherMaterials{OtherMatsRows(16)-3,2};
adRefSS316s=OtherMaterials{OtherMatsRows(17)-3,2};
adHEU=      OtherMaterials{OtherMatsRows(18)-3,2};

% Material Comps

mSodium=   OtherMaterials{OtherMatsRows(1)-3,3};
mArgon=    OtherMaterials{OtherMatsRows(2)-3,3};
mHelium=   OtherMaterials{OtherMatsRows(3)-3,3};
mPgas=     OtherMaterials{OtherMatsRows(4)-3,3};
mSS304=    OtherMaterials{OtherMatsRows(5)-3,3};
mB4Cwwr=   OtherMaterials{OtherMatsRows(6)-3,3};
mSS316=    OtherMaterials{OtherMatsRows(7)-3,3};
mFPwwr=    OtherMaterials{OtherMatsRows(8)-3,3};
mB4C=      OtherMaterials{OtherMatsRows(9)-3,3};
mSUP=      OtherMaterials{OtherMatsRows(10)-3,3};
mSLO=      OtherMaterials{OtherMatsRows(11)-3,3};
mSLC=      OtherMaterials{OtherMatsRows(12)-3,3};
mSIHDLC=   OtherMaterials{OtherMatsRows(13)-3,3};
mSafetySUP=OtherMaterials{OtherMatsRows(14)-3,3};
mB4CSUP=   OtherMaterials{OtherMatsRows(15)-3,3};
mRefSS304s=OtherMaterials{OtherMatsRows(16)-3,3};
mRefSS316s=OtherMaterials{OtherMatsRows(17)-3,3};
mHEU=      OtherMaterials{OtherMatsRows(18)-3,3};

if BadBoronMat

    % This introduces an old error in the boron material composition

    mB4C=B4CDatOut{1,3};
    mB4C{2,1}=mB4C{2,1}*-1;
    mB4C{2,2}=mB4C{2,2}*-1;
    mB4C{2,3}=mB4C{2,3}*-1;

end

%% Apply 316 and 304 densities to new vars

SwelledadSS304=adSS304;
SwelledadSS316=adSS316;

% Pull off old sodium density to get height difference calculator

OldadSodium=OtherMaterialsNoAdj{OtherMatsRows(1)-3,2};

FPhRat=OldadSodium/adSodium;

%% Apply Lower Adapter Model Bias
```

```
% As a temporary perturbation we swap out the lower adapter with just
% normal sodium. This in essence "removes" the lower adapter. If this is
% proven a good bias, then code will be written to actually remove the
% lower adpater. We apply these to SIHDLC as well.

if RemoveLAB

    mSIHDLC=mSodium;
    mSLC=mSodium;

    adSIHDLC=adSodium;
    adSLC=adSodium;

end

%%
%Debug parameter
TotDat=cell(1,2);
TotDat{1,1}='Position';
TotDat{1,2}='ISOAden';

%*****************************************DEBUG************************
TotDat=[TotDat;ISOadenOtherMats;ISOadenB4C];
%*****************************************DEBUG************************

%% MICKA mat titles

MatVarTitles=importdata('Specs\Materials\MatVarTitles.xlsx')';
dispPrint('Material Variable Titles read..');
dispPrint(' ')

%% %Prepare the MatMap

dispPrint('Preparing the Material Map Variable')
dispPrint(' ')

MatMap=[ReactMap, [MatVarTitles; cell(nsa,length(MatVarTitles))]];

BurnMap=[ReactMap, [{'Sec 1','Sec 2','Sec 3','Total'}; cell(nsa,4)]];
MassMap=[ReactMap, [{'Sec 1','Sec 2','Sec 3','Total'}; cell(nsa,4)]];
MassDenMap=[ReactMap, [{'Sec 1','Sec 2','Sec 3','Total'}; cell(nsa,4)]];
sdViolateMap=[ReactMap, [{'Burnup','New Burnup','feid','sd','osd','Violated'}; cell(nsa,6)]];

ReactMapPOS=ReactMap(2:end,3);

%%  Grab the Lattice element window OD

LatEleOD=DimMap{end,ihdodC}+(2*DimMap{end,bhC});
ff=1;

%% Prepare debug folcer assignments

OldFoldName='';

%% Material Assignment

for mm=1:size(matdbin,1)

    matdbintemp=matdbin{mm,1};
    matdbtitles=matdbintemp(1,:);
    matdb=matdbintemp(2:end,:);
    matdbPOS=matdbintemp(2:end,2);

if length(matdbPOS{1,1})==7

    NumReg=3;

else

    NumReg=1;

end

        %% This is just for the fuel data

%*********************************Reactor Map to Mat Map*******************

    %This for loop opens the first material database (MKIIAI drivers) grabs the
    %first material spec at a reactor position, then finds that position in the
    %reactor map. After the position is found the material data is written to
    %all the positions that correnspond. Each material spec is written through
    %a series of if statements at the end of the for loop.

    for ii=1:NumReg:length(matdbPOS)

        ReactMapPOSind=1;
        SAfound=false;

        for kk=1:nsa

            %The following if finds where the SA from the matdb is located
            %in the reactor map. kk is its row location in react map

            currentPOSMat=matdbPOS{ii};
```

```
        if ~isempty(strfind(ReactMapPOS{kk},currentPOSMat(1:5)))


           SAfound=true;
           ReactMapPOSind=kk;

           break

        end

    end

SATy=MatMap{ReactMapPOSind+1,MSATyC};

% Grab the Dimensions of the pin

if any(SATy==[1,2,3,4,5,8,10]) && SAfound

    % Set the swelling variables

    SwellPinNG=SwellPin;
    SwellFuelSlugNG=SwellFuelSlug;

    FueledEXP=false;
    FueledSA=false;

    LoadDimDat=false;

    % Standard SAs with Fuel

    if any(mm==[1,2,3,4,5])

        sd=DimMap{ReactMapPOSind+1,fsdC};
        sh=DimMap{ReactMapPOSind+1,fshC};
        np=DimMap{ReactMapPOSind+1,nfeC};
        fslas=DimMap{ReactMapPOSind+1,fslasC};

        DimType='DimLoadF';
        FueledSA=true;
        LoadDimDat=true;

        % While open, perform the fslas height correction

        if NaHTempCorr

            DimMap{ReactMapPOSind+1,fslasC}=(fslas+sh)*FPhRat-sh;

        end

    end

    % Blanket

    if SATy==8

        sd=DimMap{ReactMapPOSind+1,esdC};
        sh=DimMap{ReactMapPOSind+1,eshC};
        np=DimMap{ReactMapPOSind+1,ndeC};

        DimType='DimLoadB';
        FueledSA=false;
        LoadDimDat=true;
        VolChng=0;

    end

    % Experimentals

    if SATy==10

       EXPID=DimMap{ReactMapPOSind+1,SANomen};

        if strcmp(EXPID,'C2776A') || strcmp(EXPID,'XX09') || ...
           strcmp(EXPID,'X402A')  || strcmp(EXPID,'X412')

           FueledEXP=true;
           LoadDimDat=true;

           sd=DimMap{ReactMapPOSind+1,fsdC};
           sh=DimMap{ReactMapPOSind+1,fshC};
           np=DimMap{ReactMapPOSind+1,nfeC};

           DimType='DimLoadF';

        end

    end

    % The burnup needs to be calculated to see if the pins need to be
    % adjusted.

    if FueledSA || FueledEXP

        if SwellPinNG || SwellFuelSlugNG
```

```
            % Grab the composition

            CurMat=[matdbtitles(1,3:end); matdb(ii,3:end); matdb(ii+1,3:end);...
                matdb(ii+2,3:end)]; % Material Comp

            % Calculate the Burnup

             AvgBurn = BurnCalc( CurMat );

            % Grab the Inner Clad Radius

            feid=DimMap{ReactMapPOSind+1,feidC};

        end

        if SwellPinNG

            % Swell the Pin

            [DimMap, SwelledadSS304, SwelledadSS316, EleSwellViolate,nfeod  ]...
                = ElementSwell(DimMap, SATy, ReactMapPOSind, AvgBurn, adSS304, adSS316 );

            % Rewrite feid

            feid=DimMap{ReactMapPOSind+1,feidC};

        end

        if SwellFuelSlugNG

            % Swell the Slug

            [sd, osd, sh, VolChng, sdViolate, ViolateBurn]=SlugSwell(AvgBurn,sd,sh,feid);

            % Write the new Swelled Dimensions back to dimmap

            DimMap{ReactMapPOSind+1,fsdC}=sd;
            DimMap{ReactMapPOSind+1,fshC}=sh;

        end

        % Record and pins that violate the model because of swelling

        if SwellFuelSlugNG

            sdViolateMap{ReactMapPOSind+1,6}=AvgBurn;
            sdViolateMap{ReactMapPOSind+1,7}=ViolateBurn;
            sdViolateMap{ReactMapPOSind+1,8}=feid;
            sdViolateMap{ReactMapPOSind+1,9}=sd;
            sdViolateMap{ReactMapPOSind+1,10}=osd;
            sdViolateMap{ReactMapPOSind+1,11}=sdViolate;

        end

    end

    if LoadDimDat

        [~,~,~]=AtomDenCalc([LatEleOD;sd;sh;np;VolChng;adSodium],0,ChON,DimType);

    end

    clear sd sh np

end

%************************Density Calculations*********************

%Cacluates the density for either the driver pins slugs or the blanket
%slugs.

%% Driver Pins Aden

 if any(SATy==[1,2,3,4,5]) && SAfound

    % The slug dimensions are needed to correct the ARC data

    UrMatFd=zeros(NumReg,1);

    if NumReg==3

        UrMatFm=[matdbtitles(1,3:end); matdb(ii,3:end); matdb(ii+1,3:end);...
            matdb(ii+2,3:end)]; % Material Comp

    else

        UrMatFm=[matdbtitles(1,3:end);matdb(ii,3:end);]; % Material Comp

    end

    for jj=1:NumReg

        SecTypeNum=['FuelS' num2str(jj)];

        [ISOaden,BurnUp(jj),UrMatFd(jj)]=...
```

```
                AtomDenCalc( [UrMatFm(1,:); UrMatFm(jj+1,:)],VolChng,ChON,SecTypeNum); % Material Den


            NUrMatFm(jj,:)=num2cell(ISOaden{3,2});

%****************************************DEBUG************************
            TotDat=[TotDat;[currentPOSMat(1:5) '_' num2str(jj)], {ISOaden}];
%****************************************DEBUG************************

            SlugMass(jj)=ISOaden{5,3};

            SlugMassDensity(jj)=ISOaden{6,3};

        end

         %Write to the mass Map

        MassMap(ReactMapPOSind+1,6:9)=...
            num2cell([SlugMass(1:3), sum(SlugMass(1:3))]);

        %Write to the burnup Map

        BurnMap(ReactMapPOSind+1,6:9)=...
            num2cell([BurnUp(1:3)*100, 100*sum(BurnUp(1:3))/3]);

        % Write to Density Map

        MassDenMap(ReactMapPOSind+1,6:9)=...
            num2cell([SlugMassDensity(1:3), mean(SlugMassDensity(1:3))]);

        %Write the new composition

        UrMatFm=[ISOaden{1,2}; NUrMatFm];

        FuelNums=[ str2double([num2str(ReactMapPOSind) '001']),...
                   str2double([num2str(ReactMapPOSind) '002']),...
                   str2double([num2str(ReactMapPOSind) '003'])];

     end

     ff=ff+1;

    %% Blanket Pins Aden

     if SATy==8 && SAfound

        UrMatBd=zeros(NumReg,1);

        if NumReg==3

            UrMatBm=[matdbtitles(1,3:end); matdb(ii,3:end); matdb(ii+1,3:end);...
                matdb(ii+2,3:end)]; % Material Comp

        else

            UrMatBm=[matdbtitles(1,3:end);matdb(ii,3:end);]; % Material Comp

        end

        for jj=1:NumReg

            SecTypeNum=['BlanketS' num2str(jj)];

            [ISOaden,BurnUp(jj),UrMatBd(jj)]=...
                AtomDenCalc( [UrMatBm(1,:); UrMatBm(jj+1,:)],VolChng,ChON, SecTypeNum); % Material Den

            NUrMatBm(jj,:)=num2cell(ISOaden{3,2});

%****************************************DEBUG************************
            TotDat=[TotDat;[currentPOSMat(1:5) '_' num2str(jj)], {ISOaden}];
%****************************************DEBUG************************

            SlugMass(jj)=ISOaden{5,3};

            SlugMassDensity(jj)=ISOaden{6,3};

        end

         %Write to the mass Map

        MassMap(ReactMapPOSind+1,6:9)=...
            num2cell([SlugMass(1:3), sum(SlugMass(1:3))]);

        %Write to the burnup Map

        BurnMap(ReactMapPOSind+1,6:9)=...
            num2cell([BurnUp(1:3)*100, 100*sum(BurnUp(1:3))/3]);

        % Write to Density Map

        MassDenMap(ReactMapPOSind+1,6:9)=...
            num2cell([SlugMassDensity(1:3), mean(SlugMassDensity(1:3))]);

        %Write the new composition
```

```
        UrMatBm=[ISOaden{1,2}; NUrMatBm];

        BlanketNums=[ str2double([num2str(ReactMapPOSind) '0019']),...
                      str2double([num2str(ReactMapPOSind) '0020']),...
                      str2double([num2str(ReactMapPOSind) '0021'])];

     end

    %% Experiment Pins Aden

     if SATy==10 && SAfound &&  mm==4

        UrMatEd=zeros(NumReg,1);

        if NumReg==3

            UrMatEm=[matdbtitles(1,3:end); matdb(ii,3:end); matdb(ii+1,3:end);...
                matdb(ii+2,3:end)]; % Material Comp

        else

            UrMatEm=[matdbtitles(1,3:end);matdb(ii,3:end);]; % Material Comp

        end

        for jj=1:NumReg

            SecTypeNum=['FuelS' num2str(jj)];

            [ISOaden,BurnUp(jj),UrMatEd(jj)]=...
                AtomDenCalc( [UrMatEm(1,:); UrMatEm(jj+1,:)],VolChng,ChON,SecTypeNum ); % Material Den

            NUrMatEm(jj,:)=num2cell(ISOaden{3,2});

%*****************************************DEBUG************************
            TotDat=[TotDat;[currentPOSMat(1:5) '_' num2str(jj)], {ISOaden}];
%*****************************************DEBUG************************

            SlugMass(jj)=ISOaden{5,3};

            SlugMassDensity(jj)=ISOaden{6,3};

        end

         %Write to the mass Map

        MassMap(ReactMapPOSind+1,6:9)=...
            num2cell([SlugMass(1:3), sum(SlugMass(1:3))]);

        % Write the burnup

        BurnMap(ReactMapPOSind+1,6:9)=...
            num2cell([BurnUp(1:3)*100, 100*sum(BurnUp(1:3))/3]);

        % Write to Density Map

        MassDenMap(ReactMapPOSind+1,6:9)=...
            num2cell([SlugMassDensity(1:3), mean(SlugMassDensity(1:3))]);

        %Write the new composition

        UrMatEm=[ISOaden{1,2}; NUrMatEm];

        EXPNums= [ str2double([num2str(ReactMapPOSind) '001']),...
                   str2double([num2str(ReactMapPOSind) '002']),...
                   str2double([num2str(ReactMapPOSind) '003'])];


     end

    % Write Excel Spreadsheet of data

    if SaveSAMats

        if (exist('NUrMatFm','var') || exist('NUrMatEm','var') || exist('NUrMatBm','var'))

            % Grab the Last Three ISOadens

            S3ISOAden=TotDat{end,2};
            S2ISOAden=TotDat{end-1,2};
            S1ISOAden=TotDat{end-2,2};

            S3Mat=S3ISOAden{4,2}';
            S2Mat=S2ISOAden{4,2}';
            S1Mat=S1ISOAden{4,2}';

            NS3Mat=cellstr(num2str(S3Mat,'%.5e'))';
            NS2Mat=cellstr(num2str(S2Mat,'%.5e'))';
            NS1Mat=cellstr(num2str(S1Mat,'%.5e'))';

            CurPOSTitle=TotDat{end,1}(1:5);

            % Grab the ZAIDS
            ZAIDScel=S1ISOAden{1,2};
```

```
        % Create the double array for the spreadsheet
        Adens=[[{'ZAIDS'};{'S1'};{'S2'};{'S3'}],...
            [ZAIDScel; NS1Mat; NS2Mat; NS3Mat]];

        if ~strcmp(strrep(dispPrintMats{mm},' ',''),OldFoldName)

            NewFold=['Debug\SAmakerVar\' strrep(dispPrintMats{mm},' ','')];

            mkdir(NewFold);

            addpath(NewFold);

            OldFoldName=strrep(dispPrintMats{mm},' ','');

        end

        % Write the spreadsheet
        %xlswrite(['Debug\SAmakerVar\' strrep(dispPrintMats{mm},' ','') '\' CurPOSTitle '.xls'],Adens);

        % Csvwrite

        curFileID=fopen(['Debug\SAmakerVar\' strrep(dispPrintMats{mm},' ','') '\' CurPOSTitle '.csv'],'w');

        fscca='%s,%s,%s,%s\r\n';

        for qq=1:size(Adens,1)

            fprintf(curFileID,fscca,Adens{qq,:});

        end

        fclose(curFileID);

        clear curFileID

    end

end

% Need to clear this variable since it can change size
clear  NUrMatFm  NUrMatEm NUrMatBm

%% HWCR B4C Adens

if SATy == 4

   if B4CBurnB

       for qq=1:size(B4CDatOut)

           if strcmp(B4CDatOut{qq,1},currentPOSMat(1:5))

               mB4CBurned=B4CDatOut{qq,3};
               adB4CBurned=B4CDatOut{qq,2};
               break

           end

       end

   else

   % If no Boron burning is requested then the BOL values are used, but
   % the BOL values can still be modified by a perturbatiopn

        mB4CBurned=B4CDatOut{1,3};
        adB4CBurned=B4CDatOut{1,2};

   end


end

%% ***************************Composition and Den Writer***************

    if SAfound

        switch SATy
        case 1  %*************************Driver**********************

            % Fuel Slug

            MatMap{ReactMapPOSind+1,FdatC}=UrMatFm;
            MatMap{ReactMapPOSind+1,FdC}=UrMatFd;
            MatMap{ReactMapPOSind+1,FmC}=FuelNums;


            % Fuel Bond

            MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
            MatMap{ReactMapPOSind+1,FNadC}=adSodium;
            MatMap{ReactMapPOSind+1, FNamC}=...
                str2double([num2str(ReactMapPOSind) '004']);

           % Plenum Gas
```

```
    MatMap{ReactMapPOSind+1,FPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,FPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,FPGmC}=...
        str2double([num2str(ReactMapPOSind) '005']);

    % SS Cladding

    MatMap{ReactMapPOSind+1,FCladdatC}=mSS316;
    MatMap{ReactMapPOSind+1,FCladdC}=SwelledadSS316;
    MatMap{ReactMapPOSind+1,FCladmC}=...
        str2double([num2str(ReactMapPOSind) '006']);

    % SS Cladding Wire Wrap

    MatMap{ReactMapPOSind+1,FPwwdatC}=mFPwwr;
    MatMap{ReactMapPOSind+1,FPwwdC}=adFPwwr;
    MatMap{ReactMapPOSind+1,FPwwrmC}=...
        str2double([num2str(ReactMapPOSind) '008']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    % Smeared Upper Extension

    MatMap{ReactMapPOSind+1,SUPdatC}=mSUP;
    MatMap{ReactMapPOSind+1,SUPdC}=adSUP;
    MatMap{ReactMapPOSind+1,SUPmC}=...
        str2double([num2str(ReactMapPOSind) '0014']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    % Smeared Lower Adapter

    MatMap{ReactMapPOSind+1,LadatC}=mSLC;
    MatMap{ReactMapPOSind+1,LadC}=adSLC;
    MatMap{ReactMapPOSind+1,LAmC}=...
        str2double([num2str(ReactMapPOSind) '0016']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS316;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);
case 2  %***********************Driver HFW*******************

    % Fuel Slug

    MatMap{ReactMapPOSind+1,FdatC}=UrMatFm;
    MatMap{ReactMapPOSind+1,FdC}=UrMatFd;
    MatMap{ReactMapPOSind+1,FmC}=FuelNums;


    % Fuel Bond

    MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
    MatMap{ReactMapPOSind+1,FNadC}=adSodium;
    MatMap{ReactMapPOSind+1, FNamC}=...
        str2double([num2str(ReactMapPOSind) '004']);

   % Plenum Gas

    MatMap{ReactMapPOSind+1,FPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,FPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,FPGmC}=...
        str2double([num2str(ReactMapPOSind) '005']);

    % SS Cladding

    MatMap{ReactMapPOSind+1,FCladdatC}=mSS316;
    MatMap{ReactMapPOSind+1,FCladdC}=SwelledadSS316;
    MatMap{ReactMapPOSind+1,FCladmC}=...
        str2double([num2str(ReactMapPOSind) '006']);

    % SS Cladding Wire Wrap

    MatMap{ReactMapPOSind+1,FPwwdatC}=mFPwwr;
    MatMap{ReactMapPOSind+1,FPwwdC}=adFPwwr;
    MatMap{ReactMapPOSind+1,FPwwrmC}=...
        str2double([num2str(ReactMapPOSind) '008']);

    % Dummy Pins Material

    MatMap{ReactMapPOSind+1,DdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DdC}=adSS316;
```

```
    MatMap{ReactMapPOSind+1,DmC}=...
        str2double([num2str(ReactMapPOSind) '0011']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    % Smeared Upper Extension

    MatMap{ReactMapPOSind+1,SUPdatC}=mSUP;
    MatMap{ReactMapPOSind+1,SUPdC}=adSUP;
    MatMap{ReactMapPOSind+1,SUPmC}=...
    str2double([num2str(ReactMapPOSind) '0014']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    % Smeared Lower Adapter

    MatMap{ReactMapPOSind+1,LadatC}=mSLC;
    MatMap{ReactMapPOSind+1,LadC}=adSLC;
    MatMap{ReactMapPOSind+1,LAmC}=...
        str2double([num2str(ReactMapPOSind) '0016']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS316;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);
case 3  %*************************Safety**********************

    % Fuel Slug

    MatMap{ReactMapPOSind+1,FdatC}=UrMatFm;
    MatMap{ReactMapPOSind+1,FdC}=UrMatFd;
    MatMap{ReactMapPOSind+1,FmC}=FuelNums;

    % Fuel Bond

    MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
    MatMap{ReactMapPOSind+1,FNadC}=adSodium;
    MatMap{ReactMapPOSind+1, FNamC}=...
        str2double([num2str(ReactMapPOSind) '004']);

   % Plenum Gas

    MatMap{ReactMapPOSind+1,FPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,FPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,FPGmC}=...
        str2double([num2str(ReactMapPOSind) '005']);

    % SS Cladding

    MatMap{ReactMapPOSind+1,FCladdatC}=mSS316;
    MatMap{ReactMapPOSind+1,FCladdC}=SwelledadSS316;
    MatMap{ReactMapPOSind+1,FCladmC}=...
        str2double([num2str(ReactMapPOSind) '006']);

    % SS Cladding Wire Wrap

    MatMap{ReactMapPOSind+1,FPwwdatC}=mFPwwr;
    MatMap{ReactMapPOSind+1,FPwwdC}=adFPwwr;
    MatMap{ReactMapPOSind+1,FPwwrmC}=...
        str2double([num2str(ReactMapPOSind) '008']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    % Smeared Upper Extension

    MatMap{ReactMapPOSind+1,SUPdatC}=mSafetySUP;
    MatMap{ReactMapPOSind+1,SUPdC}=adSafetySUP;
    MatMap{ReactMapPOSind+1,SUPmC}=...
        str2double([num2str(ReactMapPOSind) '0014']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    if ~BorD
```

```
        % For Katana the HWCR Safety and Control are just
        % drivers, a lower adapter materials needs to be added
        % Smeared Lower Adapter

        MatMap{ReactMapPOSind+1,LadatC}=mSLC;
        MatMap{ReactMapPOSind+1,LadC}=adSLC;
        MatMap{ReactMapPOSind+1,LAmC}=...
            str2double([num2str(ReactMapPOSind) '0016']);

    end

    % Smeared Inner Hex Duct Lower Adapter

    MatMap{ReactMapPOSind+1,IHDLAdatC}=mSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAdC}=adSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAmC}=...
        str2double([num2str(ReactMapPOSind) '0017']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS316;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);
case 4  %*************************HWCR**********************

    % Fuel Slug

    MatMap{ReactMapPOSind+1,FdatC}=UrMatFm;
    MatMap{ReactMapPOSind+1,FdC}=UrMatFd;
    MatMap{ReactMapPOSind+1,FmC}=FuelNums;

    % Fuel Bond

    MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
    MatMap{ReactMapPOSind+1,FNadC}=adSodium;
    MatMap{ReactMapPOSind+1, FNamC}=...
        str2double([num2str(ReactMapPOSind) '004']);

   % Plenum Gas Fuel Pin

    MatMap{ReactMapPOSind+1,FPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,FPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,FPGmC}=...
        str2double([num2str(ReactMapPOSind) '005']);

    % SS Cladding

    MatMap{ReactMapPOSind+1,FCladdatC}=mSS316;
    MatMap{ReactMapPOSind+1,FCladdC}=SwelledadSS316;
    MatMap{ReactMapPOSind+1,FCladmC}=...
        str2double([num2str(ReactMapPOSind) '006']);

    % Poison Slug

    MatMap{ReactMapPOSind+1,PdatC}=mB4CBurned;
    MatMap{ReactMapPOSind+1,PdC}=adB4CBurned;
    MatMap{ReactMapPOSind+1,PmC}=...
        str2double([num2str(ReactMapPOSind) '007']);

    % SS Cladding Wire Wrap

    MatMap{ReactMapPOSind+1,FPwwdatC}=mFPwwr;
    MatMap{ReactMapPOSind+1,FPwwdC}=adFPwwr;
    MatMap{ReactMapPOSind+1,FPwwrmC}=...
        str2double([num2str(ReactMapPOSind) '008']);

    % Plenum Gas Poison Pin

    MatMap{ReactMapPOSind+1,PPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,PPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,PPGmC}=...
        str2double([num2str(ReactMapPOSind) '009']);

    % Poison Pin Clad

    MatMap{ReactMapPOSind+1,PCladdatC}=mSS304;
    MatMap{ReactMapPOSind+1,PCladdC}=adSS304;
    MatMap{ReactMapPOSind+1,PCladmC}=...
        str2double([num2str(ReactMapPOSind) '0010']);

    % Poison Pin Wirewrap

    MatMap{ReactMapPOSind+1,OPdatC}=mB4Cwwr;
    MatMap{ReactMapPOSind+1,OPdC}=adB4Cwwr;
    MatMap{ReactMapPOSind+1,OPmC}=...
        str2double([num2str(ReactMapPOSind) '0018']);

    % Poison Pin Shield

    MatMap{ReactMapPOSind+1,ShieldPdatC}=mB4CSUP;
    MatMap{ReactMapPOSind+1,ShieldPdC}=adB4CSUP;
    MatMap{ReactMapPOSind+1,ShieldPmC}=...
        str2double([num2str(ReactMapPOSind) '0019']);
```

```
    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    if ~BorD

        % For Katana the HWCR Safety and Control are just
        % drivers, a lower adapter materials needs to be added
        % Smeared Upper Extension

        MatMap{ReactMapPOSind+1,SUPdatC}=mSafetySUP;
        MatMap{ReactMapPOSind+1,SUPdC}=adSafetySUP;
        MatMap{ReactMapPOSind+1,SUPmC}=...
            str2double([num2str(ReactMapPOSind) '0014']);

    end

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    if ~BorD

        % For Katana the HWCR Safety and Control are just
        % drivers, a lower adapter materials needs to be added
        % Smeared Lower Adapter

        MatMap{ReactMapPOSind+1,LadatC}=mSLC;
        MatMap{ReactMapPOSind+1,LadC}=adSLC;
        MatMap{ReactMapPOSind+1,LAmC}=...
            str2double([num2str(ReactMapPOSind) '0016']);

    end

    % Smeared Inner Hex Duct Lower Adapter

    MatMap{ReactMapPOSind+1,IHDLAdatC}=mSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAdC}=adSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAmC}=...
        str2double([num2str(ReactMapPOSind) '0017']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS316;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);
case 5  %************************Control**********************

    % Fuel Slug

    MatMap{ReactMapPOSind+1,FdatC}=UrMatFm;
    MatMap{ReactMapPOSind+1,FdC}=UrMatFd;
    MatMap{ReactMapPOSind+1,FmC}=FuelNums;

    % Fuel Bond

    MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
    MatMap{ReactMapPOSind+1,FNadC}=adSodium;
    MatMap{ReactMapPOSind+1,FNamC}=...
        str2double([num2str(ReactMapPOSind) '004']);

   % Plenum Gas

    MatMap{ReactMapPOSind+1,FPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,FPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,FPGmC}=...
        str2double([num2str(ReactMapPOSind) '005']);

    % SS Cladding

    MatMap{ReactMapPOSind+1,FCladdatC}=mSS316;
    MatMap{ReactMapPOSind+1,FCladdC}=SwelledadSS316;
    MatMap{ReactMapPOSind+1,FCladmC}=...
        str2double([num2str(ReactMapPOSind) '006']);

    % SS Cladding Wire Wrap

    MatMap{ReactMapPOSind+1,FPwwdatC}=mFPwwr;
    MatMap{ReactMapPOSind+1,FPwwdC}=adFPwwr;
    MatMap{ReactMapPOSind+1,FPwwrmC}=...
        str2double([num2str(ReactMapPOSind) '008']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
```

```
        str2double([num2str(ReactMapPOSind) '0013']);

    % Smeared Upper Extension

    MatMap{ReactMapPOSind+1,SUPdatC}=mSafetySUP;
    MatMap{ReactMapPOSind+1,SUPdC}=adSafetySUP;
    MatMap{ReactMapPOSind+1,SUPmC}=...
        str2double([num2str(ReactMapPOSind) '0014']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS304;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS304;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    if ~BorD

        % For Katana the HWCR Safety and Control are just
        % drivers, a lower adapter materials needs to be added
        % Smeared Lower Adapter

        MatMap{ReactMapPOSind+1,LadatC}=mSLC;
        MatMap{ReactMapPOSind+1,LadC}=adSLC;
        MatMap{ReactMapPOSind+1,LAmC}=...
            str2double([num2str(ReactMapPOSind) '0016']);

    end

    % Smeared Inner Hex Duct Lower Adapter

    MatMap{ReactMapPOSind+1,IHDLAdatC}=mSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAdC}=adSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAmC}=...
        str2double([num2str(ReactMapPOSind) '0017']);
case 6  %************************Dummy SA*********************

    % Determine Which Kind of SS is needed

    % Read Material Map
    HDmat=str2double(matdb{ii,6});
    PCmat=str2double(matdb{ii,7});

    % Change Ducts to the right SS

    if (HDmat==4)

        DummySSdm=mSS304;
        DummySSdd=adSS304;

    else

        DummySSdm=mSS316;
        DummySSdd=adSS316;

    end

    % Change Pin Clad to the right SS

    if (PCmat==4)

        DPSSm=mSS304;
        DPSSd=adSS304;

    else

        DPSSm=mSS316;
        DPSSd=adSS316;

    end

    % Dummy Pins Material

    MatMap{ReactMapPOSind+1,DdatC}=DPSSm;
    MatMap{ReactMapPOSind+1,DdC}=DPSSd;
    MatMap{ReactMapPOSind+1,DmC}=...
        str2double([num2str(ReactMapPOSind) '0011']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    % Dummy Pin Bond

    MatMap{ReactMapPOSind+1,PPGdatC}=mPgas;
```

```
    MatMap{ReactMapPOSind+1,PPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,PPGmC}=...
        str2double([num2str(ReactMapPOSind) '009']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=DummySSdm;
    MatMap{ReactMapPOSind+1,DuctdC}=DummySSdd;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    % Smeared Lower Adapter

    MatMap{ReactMapPOSind+1,LadatC}=mSLC;
    MatMap{ReactMapPOSind+1,LadC}=adSLC;
    MatMap{ReactMapPOSind+1,LAmC}=...
        str2double([num2str(ReactMapPOSind) '0016']);
case 7  %**********************Reflector SA******************

    % Determine Which Kind of SS is needed

    %Read Material Map
    HDmat=str2double(matdb{ii,6});
    HBmat=str2double(matdb{ii,7});

    % Change Ducts to the right SS

    if (HDmat==4)

        RefSSdm=mSS304;
        RefSSdd=adSS304;

    else

        RefSSdm=mSS316;
        RefSSdd=adSS316;

    end

    % Change Pin Clad to the right SS

    if (HBmat==4)

        RefSSbm=mRefSS304s;
        RefSSbd=adRefSS304s;

    else

        RefSSbm=mRefSS316s;
        RefSSbd=adRefSS316s;

    end

    % Hex Blocks Material

    MatMap{ReactMapPOSind+1,DdatC}=RefSSbm;
    MatMap{ReactMapPOSind+1,DdC}=RefSSbd;
    MatMap{ReactMapPOSind+1,DmC}=...
        str2double([num2str(ReactMapPOSind) '0011']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=RefSSdm;
    MatMap{ReactMapPOSind+1,DuctdC}=RefSSdd;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    % Smeared Lower Adapter

    MatMap{ReactMapPOSind+1,LadatC}=mSLC;
    MatMap{ReactMapPOSind+1,LadC}=adSLC;
    MatMap{ReactMapPOSind+1,LAmC}=...
        str2double([num2str(ReactMapPOSind) '0016']);
case 8  %**********************Outter Blanket*****************
```

```
    % Blanket Slug

    MatMap{ReactMapPOSind+1,FdatC}=UrMatBm;
    MatMap{ReactMapPOSind+1,FdC}=UrMatBd;
    MatMap{ReactMapPOSind+1,FmC}=BlanketNums;

    % Blanket Bond

    MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
    MatMap{ReactMapPOSind+1,FNadC}=adSodium;
    MatMap{ReactMapPOSind+1,FNamC}=...
        str2double([num2str(ReactMapPOSind) '004']);

    % SS Cladding

    MatMap{ReactMapPOSind+1,FCladdatC}=mSS304;
    MatMap{ReactMapPOSind+1,FCladdC}=adSS304;
    MatMap{ReactMapPOSind+1,FCladmC}=...
        str2double([num2str(ReactMapPOSind) '006']);

    % Sodium Used in the AS and outside of the SA

    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS304;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS304;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);

    % Smeared Lower Extension

    MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
    MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
    MatMap{ReactMapPOSind+1,SLOmC}=...
        str2double([num2str(ReactMapPOSind) '0015']);

    % Smeared Lower Adapter

    MatMap{ReactMapPOSind+1,LadatC}=mSLC;
    MatMap{ReactMapPOSind+1,LadC}=adSLC;
    MatMap{ReactMapPOSind+1,LAmC}=...
        str2double([num2str(ReactMapPOSind) '0016']);
case 9  %************************Wall Maker*******************
case 10 %************************Experiments******************

    % For this to work, it will have to be a switch function
    % to assign materials to unique SA regions and then share
    % the non uniqure regions.

    if strcmp(EXPID,'C2776A') || ...
       strcmp(EXPID,'XX09')   || ...
       strcmp(EXPID,'X412')   || ...
       strcmp(EXPID,'X402A')

        % Fuel Slug

        MatMap{ReactMapPOSind+1,FdatC}=UrMatEm;
        MatMap{ReactMapPOSind+1,FdC}=UrMatEd;
        MatMap{ReactMapPOSind+1,FmC}=EXPNums;

        % Fuel Bond

        MatMap{ReactMapPOSind+1,FNadatC}=mSodium;
        MatMap{ReactMapPOSind+1,FNadC}=adSodium;
        MatMap{ReactMapPOSind+1,FNamC}=...
            str2double([num2str(ReactMapPOSind) '004']);

       % Plenum Gas

        MatMap{ReactMapPOSind+1,FPGdatC}=mPgas;
        MatMap{ReactMapPOSind+1,FPGdC}=adPgas;
        MatMap{ReactMapPOSind+1,FPGmC}=...
            str2double([num2str(ReactMapPOSind) '005']);

        % SS Cladding

        MatMap{ReactMapPOSind+1,FCladdatC}=mSS316;
        MatMap{ReactMapPOSind+1,FCladdC}=adSS316;
        MatMap{ReactMapPOSind+1,FCladmC}=...
            str2double([num2str(ReactMapPOSind) '006']);

        % SS Cladding Wire Wrap

        MatMap{ReactMapPOSind+1,FPwwdatC}=mFPwwr;
        MatMap{ReactMapPOSind+1,FPwwdC}=adFPwwr;
        MatMap{ReactMapPOSind+1,FPwwrmC}=...
            str2double([num2str(ReactMapPOSind) '008']);

        % Sodium Used in the AS and outside of the SA
```

```
    MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
    MatMap{ReactMapPOSind+1,DnadC}=adSodium;
    MatMap{ReactMapPOSind+1,DnamC}=...
        str2double([num2str(ReactMapPOSind) '0013']);

end

if strcmp(EXPID,'X402A') || ...
   strcmp(EXPID,'XX09')  || ...
   strcmp(EXPID,'XX10')  || ...
   strcmp(EXPID,'XY-16') || ...
   strcmp(EXPID,'X320C')

    % Dummy Pins Material

    MatMap{ReactMapPOSind+1,DdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DdC}=adSS316;
    MatMap{ReactMapPOSind+1,DmC}=...
        str2double([num2str(ReactMapPOSind) '0011']);

    % Dummy Pin Bond

    MatMap{ReactMapPOSind+1,PPGdatC}=mPgas;
    MatMap{ReactMapPOSind+1,PPGdC}=adPgas;
    MatMap{ReactMapPOSind+1,PPGmC}=...
        str2double([num2str(ReactMapPOSind) '009']);

end


if strcmp(EXPID,'C2776A') || ...
   strcmp(EXPID,'X412')   || ...
   strcmp(EXPID,'X402A')  || ...
   strcmp(EXPID,'XX10')

    % Smeared Upper Extension

    MatMap{ReactMapPOSind+1,SUPdatC}=mSafetySUP;
    MatMap{ReactMapPOSind+1,SUPdC}=adSafetySUP;
    MatMap{ReactMapPOSind+1,SUPmC}=...
        str2double([num2str(ReactMapPOSind) '0014']);

end

if strcmp(EXPID,'XX10') || ...
   strcmp(EXPID,'XX09') || ...
   strcmp(EXPID,'XY-16')

    % Smeared Upper Extension

    MatMap{ReactMapPOSind+1,SUPdatC}=mSafetySUP;
    MatMap{ReactMapPOSind+1,SUPdC}=adSafetySUP;
    MatMap{ReactMapPOSind+1,SUPmC}=...
        str2double([num2str(ReactMapPOSind) '0014']);

    % Smeared Inner Hex Duct Lower Adapter

    MatMap{ReactMapPOSind+1,IHDLAdatC}=mSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAdC}=adSIHDLC;
    MatMap{ReactMapPOSind+1,IHDLAmC}=...
        str2double([num2str(ReactMapPOSind) '0017']);

    % Poison Pin Wirewrap

    MatMap{ReactMapPOSind+1,OPdatC}=mB4Cwwr;
    MatMap{ReactMapPOSind+1,OPdC}=adB4Cwwr;
    MatMap{ReactMapPOSind+1,OPmC}=...
        str2double([num2str(ReactMapPOSind) '0018']);


end

if strcmp(EXPID,'C2776A') || ...
   strcmp(EXPID,'XX09')   || ...
   strcmp(EXPID,'XX10')   || ...
   strcmp(EXPID,'XY-16')  || ...
   strcmp(EXPID,'X320C')

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS304;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS304;
    MatMap{ReactMapPOSind+1,DuctmC}=...
        str2double([num2str(ReactMapPOSind) '0012']);

end

if strcmp(EXPID,'X402A') || ...
   strcmp(EXPID,'X412')

    % SS Duct Material

    MatMap{ReactMapPOSind+1,DuctdatC}=mSS316;
    MatMap{ReactMapPOSind+1,DuctdC}=adSS316;
```

```
                    MatMap{ReactMapPOSind+1,DuctmC}=...
                        str2double([num2str(ReactMapPOSind) '0012']);

                end

                % Sodium Used in the AS and outside of the SA

                MatMap{ReactMapPOSind+1,DnadatC}=mSodium;
                MatMap{ReactMapPOSind+1,DnadC}=adSodium;
                MatMap{ReactMapPOSind+1,DnamC}=...
                    str2double([num2str(ReactMapPOSind) '0013']);

                % Smeared Lower Extension

                MatMap{ReactMapPOSind+1,SLOdatC}=mSLO;
                MatMap{ReactMapPOSind+1,SLOdC}=adSLO;
                MatMap{ReactMapPOSind+1,SLOmC}=...
                    str2double([num2str(ReactMapPOSind) '0015']);

                % Smeared Lower Adapter

                MatMap{ReactMapPOSind+1,LadatC}=mSLC;
                MatMap{ReactMapPOSind+1,LadC}=adSLC;
                MatMap{ReactMapPOSind+1,LAmC}=...
                    str2double([num2str(ReactMapPOSind) '0016']);
            case 11 %**************************SPARE**********************
            end

            % The following lines writes out the atomdensity and mass for
            % each pin section.

            % For reference

    % ZAIDdens has the following format DATA is a cell containing the actual
    % data

    %                   ZAID numbers | DATA | Totals
    %              Atom/Weight % Comp | DATA | 'Should equal 1'
    %       Isotope Molecular Weight | DATA | 'N/A'
    %       Initial/Calculated Adens | DATA | 'Total Aden'
    %                Calculated Mass | DATA | 'Total Mass'  **ONLY FOR F/B**

        end

        dispPrint([['Loading...' dispPrintMats{mm}];{[ii,length(matdbPOS)]}]);


    end

        dispPrint(' ')


end

dispPrint('Loading Wall Subassemblies')

for kk=2:size(MatMap,1)

    blankassm=true;

    for ii=6:size(MatMap,2)

        repblank=isempty(MatMap{kk,ii});

        if repblank==0 || MatMap{kk,MSATyC}==0

            blankassm=false;
            break

        end

    end

    if blankassm==true

            MatMap{kk, MSATyC}=0;
            ReactMap{kk, MSATyC}=0;
            ReactMap{kk,NomenTyC}='Reactor Wall';
            Mnum=MatMap{kk, MNumC};

            [Mnrow,~]=find(Mnum==SmearSATYs(1:end,1));

            SmearSATYs(Mnrow,:)=0;


    end

end



%% This section adds generic materials to the material map

%Adds the generic
MatMap(end+1,MNumC)={'Generic'};
```

```
MatMap(end,MSATyC)={'NA'};
MatMap(end,SAPosC)={'Everywhere'};
MatMap(end,SANomen)={'EBR-II'};
MatMap(end,NomenTyC)={'Pool Type'};
MatMap(end,6:end)=[{1}, {adSodium}   , {mSodium},...
                   {2}, {adArgon}    , {mArgon},...
                   {3}, {adHelium}   , {mHelium},...
                   {4}, {adPgas}     , {mPgas},...
                   {5}, {adSS304}    , {mSS304},...
                   {6}, {adB4Cwwr}   , {mB4Cwwr},...
                   {7}, {adSS316}    , {mSS316},...
                   {8}, {adFPwwr}    , {mFPwwr},...
                   {9}, {adB4C}      , {mB4C},...
                   {10},{adSodium}   , {mSodium},...
                   {11},{adSUP}      , {mSUP},...
                   {12},{adSLO}      , {mSLO},...
                   {13},{adSLC}      , {mSLC},...
                   {14},{adSIHDLC}   , {mSIHDLC},...
                   {15},{adSafetySUP}, {mSafetySUP},...
                   {16},{adB4CSUP}   , {mB4CSUP},...
                   {17},{adRefSS304s}, {mRefSS304s},...
                   {18},{adRefSS316s}, {mRefSS316s},...
                   {19},{adHEU}      , {mHEU}];


%% Spreadsheet and variable output

%Output the MatMap
dispPrint('Exporting Material Map');
dispPrint(' ')
delete('Maps\MatMap.xls');
MatMapTemp=MatMap;
MatMapTemp(2:end,SAPosC)=strcat('''',MatMapTemp(2:end,SAPosC));
xlswrite('Maps\MatMap.xls',MatMapTemp);

% Output the New DimMap that has swells
dispPrint('Exporting DimensionalMap');
dispPrint(' ')
delete('Maps\DimMapChng.xls');
DimMapTemp=DimMap;
DimMapTemp(2:end,SAPosC)=strcat('''',DimMapTemp(2:end,SAPosC));
xlswrite('Maps\DimMapChng.xls',DimMapTemp);

%Output the BurnMap
dispPrint('Exporting Burnup Map');
dispPrint(' ')
delete('Maps\MaterialMaps\BurnMap.xls');
BurnMapTemp=BurnMap;
BurnMapTemp(2:end,3)=strcat('''',BurnMapTemp(2:end,3));
xlswrite('Maps\MaterialMaps\BurnMap.xls',BurnMapTemp);

%Output the MassMap
dispPrint('Exporting Mass Map');
dispPrint(' ')
delete('Maps\MaterialMaps\MassMap.xls');
MassMapTemp=MassMap;
MassMapTemp(2:end,3)=strcat('''',MassMapTemp(2:end,3));
xlswrite('Maps\MaterialMaps\MassMap.xls',MassMapTemp);

%Output the MassMap
dispPrint('Exporting Density Map');
dispPrint(' ')
delete('Maps\MaterialMaps\MassDenMap.xls');
MassDenMapTemp=MassDenMap;
MassDenMapTemp(2:end,3)=strcat('''',MassDenMapTemp(2:end,3));
xlswrite('Maps\MaterialMaps\MassDenMap.xls',MassDenMapTemp);

if SwellPin || SwellFuelSlug

    %Output the Violation Maps
    dispPrint('Exporting Density Map');
    dispPrint(' ')
    delete('Maps\MaterialMaps\MassDenMap.xls');
    sdViolateMapTemp=sdViolateMap;
    sdViolateMapTemp(2:end,3)=strcat('''',sdViolateMapTemp(2:end,3));
    xlswrite('Maps\MaterialMaps\sdViolateMap.xls',sdViolateMapTemp);

end

% Save all MatMapper data.
dispPrint('Exporting All Material Data');
dispPrint(' ')
save(['Maps\' filename 'TotMatDat.mat'],'TotDat')
dispPrint([filename 'TotMatDat.mat saved....']);

if SaveDebugMatVars

    %     Create a .mat file for the mats for reading later.
    dispPrint('Saving MatLab Debug Material Variables');
    dispPrint('This is a large amount of data, be patient.');
    dispPrint(' ')

    % Save All material Data
    save('Debug\AllMaterialDat.mat')
    dispPrint('     All Material Data Saved....');
```

```
end

end
```

B.17. RealDBLoader.m

```
function [ matdbin ] = RealDbLoader( dispPrintMats )

% This functions loads the ARC data

dispPrint('Reading in the Material Data...')

matdbin{1,1}=importdata('Specs\Materials\RealDb\MARK-II 2AI.xlsx');
dispPrint([dispPrintMats{1} ' read..']);

matdbin{2,1}=importdata('Specs\Materials\RealDb\MARK-II 2A.xlsx');
dispPrint([dispPrintMats{2} ' read..']);

matdbin{3,1}=importdata('Specs\Materials\RealDb\Safety Rod.xlsx');
dispPrint([dispPrintMats{3} ' read..']);

matdbin{4,1}=importdata('Specs\Materials\RealDb\Experimental.xlsx');
dispPrint([dispPrintMats{4} ' read..']);

matdbin{5,1}=importdata('Specs\Materials\RealDb\Control.xlsx');
dispPrint([dispPrintMats{5} ' read..']);

matdbin{6,1}=importdata('Specs\Materials\RealDb\SST.xlsx');
dispPrint([dispPrintMats{6} ' read..']);

matdbin{7,1}=importdata('Specs\Materials\RealDb\SSR.xlsx');
dispPrint([dispPrintMats{7} ' read..']);

matdbin{8,1}=importdata('Specs\Materials\RealDb\Blanket.xlsx');
dispPrint([dispPrintMats{8} ' read..']);

matdbin{9,1}=importdata('Specs\Materials\RealDb\Custom.xlsx');
dispPrint([dispPrintMats{9} ' read..']);

dispPrint(' ')

end
```

B.18. OtherMatsLoader.m

```
function [OtherMatsRows, OtherMaterials, ISOaden ] = OtherMatsLoader( ChON )

%% *************************** 3rd circle ********************************

% This function loads all of the other materials and calculates their
% densities

global OtherMatsPerturb
global OtherMatsPath
global BoronPerturb

%% Other Mats

% Other Mats Import
indata=importdata('Specs\Materials\RealDb\Other Mats.xlsx');

ISOmats=delNaN( importdata('Specs\Materials\RealDb\ISO Mats.xlsx'), 'Convert All' , 'num' );

if OtherMatsPerturb && ~BoronPerturb

    % Choose a new othermats file.

    indata=importdata(OtherMatsPath);

end

mattempdata=indata.textdata;
mattempnum=indata.data;

EleTit=mattempdata(1,:);
MatTit=mattempdata(:,1);

indatatemp=num2cell(mattempnum);

OtherMats=[EleTit; [MatTit(2:end,1) indatatemp(2:end,:)]];

%% Rows are defined, this is where the variables will be placed inside of the other variables
% When adding a new material, it must be given a unique row.
% The row number must match the row number in othermats.xlsx
% The columns are the locations of the ZAIDs in othermats, if no columns
% are present, then that means it is just a copy of something else, usually
% the material above.
```

```
SodiumRow=4; SodiumCols=8;
ArgonRow=5; ArgonCols=17:19;
HeliumRow=6; HeliumCols=3:4;
PgasRow=7; PgasCols=[3:4 17:19];
SS304Row=8; SS304Cols=[7 9:16 20:33];
B4CwwrRow=9;
SS316Row=10; SS316Cols=[7 9:11 20:33 40:46];
FPwwrRow=11;
B4CRow=12; B4CCols=5:7;
SUPRow=14; SmearSS304Cols=[SS304Cols(1) SodiumCols SS304Cols(2:end)];
SLORow=15;
SLCRow=16;
SIHDLCRow=17;
SafetySUPRow=18;
B4CSUPRow=19;
RefSS304sRow=20;
RefSS316sRow=21; SmearSS316Cols=[SS316Cols(1) SodiumCols SS316Cols(2:end)];
HEURow=22;


OtherMatsRows=[SodiumRow;ArgonRow;HeliumRow;PgasRow;SS304Row;B4CwwrRow;SS316Row;...
    FPwwrRow;B4CRow;SUPRow;SLORow;SLCRow;SIHDLCRow;SafetySUPRow;B4CSUPRow;...
    RefSS304sRow;RefSS316sRow;HEURow];

%Seperates out all of the materials giving them their own variable

%% Sodium

mSodium=[OtherMats(2,SodiumCols);OtherMats(SodiumRow,SodiumCols)]; % Material Read in from Other Mats
dSodium=OtherMats{SodiumRow,2};

[ mSodium, dSodium, PertsD, SmearChng ] = DensPerturber( mSodium, dSodium,'Sodium' );

[ISOaden{SodiumRow-3,2},~,adSodium, mSodium]=AtomDenCalc(mSodium,dSodium,ChON,'OtherMats');
ISOaden{SodiumRow-3,1}='Sodium';

%% Argon

mArgon=[OtherMats(2,ArgonCols);OtherMats(ArgonRow,ArgonCols)];
dArgon=OtherMats{ArgonRow,2};

[ mArgon, dArgon, PertsD, SmearChng ] = DensPerturber( mArgon, dArgon,'Sodium' );

[ISOaden{ArgonRow-3,2},~,adArgon, mArgon]=AtomDenCalc(mArgon,dArgon,ChON,'OtherMats');
ISOaden{ArgonRow-3,1}='Argon';

%% Helium

mHelium=[OtherMats(2,HeliumCols);OtherMats(HeliumRow,HeliumCols)];
dHelium=OtherMats{HeliumRow,2};

[ mHelium, dHelium, PertsD, SmearChng ] = DensPerturber( mHelium, dHelium,'Sodium' );

[ISOaden{HeliumRow-3,2},~,adHelium, mHelium]=AtomDenCalc(mHelium,dHelium,ChON,'OtherMats');
ISOaden{HeliumRow-3,1}='Helium';

%% Plenum Gas

mPgas=[OtherMats(2,PgasCols);OtherMats(PgasRow,PgasCols)];
dPgas=OtherMats{PgasRow,2};

[ mPgas, dPgas, PertsD, SmearChng ] = DensPerturber( mPgas, dPgas,'Sodium' );

[ISOaden{PgasRow-3,2},~,adPgas, mPgas]=AtomDenCalc(mPgas,dPgas,ChON,'OtherMats');
ISOaden{PgasRow-3,1}='Pgas';

%% SS304

mSS304=[OtherMats(2,SS304Cols);OtherMats(SS304Row,SS304Cols)];
dSS304=OtherMats{SS304Row,2};
mB4Cwwr=mSS304;
mShield=mSS304;
dShield=dSS304;

% The C denotes a material that is used in combination with other materials
% the smears.

[ mSS304, dSS304, PertDT, SmearChng ] = DensPerturber( mSS304, dSS304,'SS304LC' );
if ~isempty(PertDT); PertsD=PertDT; end

[ISOaden{SS304Row-3,2},~,adSS304, mSS304]=AtomDenCalc(mSS304,dSS304,ChON,'OtherMats');
ISOaden{SS304Row-3,1}='SS304';

% Wire Wrap Density Adjustment

dB4Cwwr=dSS304*1.343055;
[ISOaden{B4CwwrRow-3,2},~,adB4Cwwr, mB4Cwwr]=AtomDenCalc(mB4Cwwr,dB4Cwwr,ChON,'OtherMats');
ISOaden{B4CwwrRow-3,1}='B4Cwwr';

%% SS316

mSS316=[OtherMats(2,SS316Cols);OtherMats(SS316Row,SS316Cols)];
mFPwwr=mSS316;
dSS316=OtherMats{SS316Row,2};
```

```
[ mSS316, dSS316, PertDT, SmearChng ] = DensPerturber( mSS316, dSS316,'SS304LC' );
if ~isempty(PertDT); PertsD=PertDT; end

[ISOaden{SS316Row-3,2},~,adSS316, mSS316]=AtomDenCalc(mSS316,dSS316,ChON,'OtherMats');
ISOaden{SS316Row-3,1}='SS316';

% Wire Wrap Density Adjustment

dFPwwr=dSS316*1.0056968;
[ISOaden{FPwwrRow-3,2},~,adFPwwr, mFPwwr]=AtomDenCalc(mFPwwr,dFPwwr,ChON,'OtherMats');
ISOaden{FPwwrRow-3,1}='FPwwr';

%% B4C

mB4C=[OtherMats(2,B4CCols); OtherMats(B4CRow,B4CCols)];
dB4C=OtherMats{B4CRow,2};
[ISOaden{B4CRow-3,2},~,adB4C, mB4C]=AtomDenCalc(mB4C,dB4C,ChON,'Ctrl');
ISOaden{B4CRow-3,1}='B4C';

%% Smeared Upper Extension

mSUP=[OtherMats(2,SmearSS304Cols);OtherMats(SUPRow,SmearSS304Cols)];
dSUP=OtherMats{SUPRow,2};

[ mSUP, dSUP, PertsS ] = SmearPerturber( mSUP, dSUP,'SUP',SmearChng,PertsD);

[ISOaden{SUPRow-3,2},~,adSUP, mSUP]=AtomDenCalc(mSUP,dSUP,ChON,'OtherMats');
ISOaden{SUPRow-3,1}='Smeared Upper Extension';

%% Smeared Lower Extension

mSLO=[OtherMats(2,SmearSS304Cols);OtherMats(SLORow,SmearSS304Cols)];
dSLO=OtherMats{SLORow,2};

[ mSLO, dSLO, PertsS ] = SmearPerturber( mSLO, dSLO,'Slo',SmearChng, PertsD);

[ISOaden{SLORow-3,2},~,adSLO, mSLO]=AtomDenCalc(mSLO,dSLO,ChON,'OtherMats');
ISOaden{SLORow-3,1}='Smeared Lower Extension';

%% Smeared Lower Cylinder

mSLC=[OtherMats(2,SmearSS304Cols);OtherMats(SLCRow,SmearSS304Cols)];
dSLC=OtherMats{SLCRow,2};

[ mSLC, dSLC, PertsS ] = SmearPerturber( mSLC, dSLC,'SLC',SmearChng, PertsD);

[ISOaden{SLCRow-3,2},~,adSLC, mSLC]=AtomDenCalc(mSLC,dSLC,ChON,'OtherMats');
ISOaden{SLCRow-3,1}='Smeared Lower Cylinder';

%% Smeared Inner Hex Duct Lower Adapter

mSIHDLC=[OtherMats(2,SmearSS304Cols);OtherMats(SIHDLCRow,SmearSS304Cols)];
dSIHDLC=OtherMats{SIHDLCRow,2};

[ mSIHDLC, dSIHDLC, PertsS ] = SmearPerturber( mSIHDLC, dSIHDLC,'SIHDLC',SmearChng, PertsD);

[ISOaden{SIHDLCRow-3,2},~,adSIHDLC, mSIHDLC]=AtomDenCalc(mSIHDLC,dSIHDLC,ChON,'OtherMats');
ISOaden{SIHDLCRow-3,1}='Smeared Inner Hex Duct Lower Cylinder';

%% Smeared Safety Upper Extension

mSafetySUP=[OtherMats(2,SmearSS304Cols);OtherMats(SafetySUPRow,SmearSS304Cols)];
dSafetySUP=OtherMats{SafetySUPRow,2};

[ mSafetySUP, dSafetySUP, PertsS ] = SmearPerturber( mSafetySUP, dSafetySUP, 'SafetySUP', SmearChng, PertsD);

[ISOaden{SafetySUPRow-3,2},~,adSafetySUP, mSafetySUP]=AtomDenCalc(mSafetySUP,dSafetySUP,ChON,'OtherMats');
ISOaden{SafetySUPRow-3,1}='Safety Smeared Upper Extension';

%% Smeared B4C Shield

mB4CSUP=[OtherMats(2,SmearSS304Cols);OtherMats(B4CSUPRow,SmearSS304Cols)];
dB4CSUP=OtherMats{B4CSUPRow,2};

[ mB4CSUP, dB4CSUP, PertsS ] = SmearPerturber( mB4CSUP, dB4CSUP,'B4CShieldSmear', SmearChng, PertsD);

[ISOaden{B4CSUPRow-3,2},~,adB4CSUP, mB4CSUP]=AtomDenCalc(mB4CSUP,dB4CSUP,ChON,'OtherMats');
ISOaden{B4CSUPRow-3,1}='Smeared B4C Shield';

%% Reflector Stainless Steel 304 Smear

mRefSS304s=[OtherMats(2,SmearSS304Cols);OtherMats(RefSS304sRow,SmearSS304Cols)];
dRefSS304s=OtherMats{RefSS304sRow,2};

[ mRefSS304s, dRefSS304s, PertsS ] = SmearPerturber( mRefSS304s, dRefSS304s, 'RefSS304s', SmearChng, PertsD);

[ISOaden{RefSS304sRow-3,2},~,adRefSS304s, mRefSS304s]=AtomDenCalc(mRefSS304s,dRefSS304s,ChON,'OtherMats');
ISOaden{RefSS304sRow-3,1}='Stainless Steel 304 Reflector Smear';

%% Reflector Stainless Steel 316 Smear

mRefSS316s=[OtherMats(2,SmearSS316Cols);OtherMats(RefSS316sRow,SmearSS316Cols)];
dRefSS316s=OtherMats{RefSS316sRow,2};

[ mRefSS316s, dRefSS316s, PertsS ] = SmearPerturber( mRefSS316s, dRefSS316s, 'RefSS316s', SmearChng, PertsD);
```

```
[ISOaden{RefSS316sRow-3,2},~,adRefSS316s, mRefSS316s]=AtomDenCalc(mRefSS316s,dRefSS316s,ChON,'OtherMats');
ISOaden{RefSS316sRow-3,1}='Stainless Steel 316 Reflector Smear';

%% Highly Enriched Uranium

mHEU={'92238','92235';0.33,0.67};
dHEU=18;
[ISOaden{HEURow-3,2},~,adHEU, mHEU]=AtomDenCalc(mHEU,dHEU,ChON,'OtherMats');
ISOaden{HEURow-3,1}='HEU';

%% Gather the variables into one variable

% initialize Other Materials
OtherMaterials=cell(1,3);

OtherMaterials(SodiumRow-3,:)={'Sodium',adSodium,mSodium};
OtherMaterials(ArgonRow-3,:)={'Argon',adArgon,mArgon};
OtherMaterials(HeliumRow-3,:)={'Helium',adHelium,mHelium};
OtherMaterials(PgasRow-3,:)={'Plenum Gas',adPgas,mPgas};
OtherMaterials(SS304Row-3,:)={'SS304',adSS304,mSS304};
OtherMaterials(SS316Row-3,:)={'SS316',adSS316,mSS316};
OtherMaterials(FPwwrRow-3,:)={'WireWrap Fuel Pins',adFPwwr,mFPwwr};
OtherMaterials(SUPRow-3,:)={'Smeared Upper Extension',adSUP,mSUP};
OtherMaterials(SLORow-3,:)={'Smeared Lower Extension',adSLO,mSLO};
OtherMaterials(SLCRow-3,:)={'Smeared Lower Adapter',adSLC,mSLC};
OtherMaterials(SIHDLCRow-3,:)={'Smeared Inner Hex Duct Lower Adapter',adSIHDLC,mSIHDLC};
OtherMaterials(RefSS304sRow-3,:)={'Stainless Steel 304 Reflector Smear',adRefSS304s,mRefSS304s};
OtherMaterials(RefSS316sRow-3,:)={'Stainless Steel 316 Reflector Smear',adRefSS316s,mRefSS316s};
OtherMaterials(HEURow-3,:)={'HEU',adHEU,mHEU};
OtherMaterials(B4CRow-3,:)={'B4C',adB4C,mB4C};
OtherMaterials(SafetySUPRow-3,:)={'Safety Smeared Upper Extension',adSafetySUP,mSafetySUP};
OtherMaterials(B4CSUPRow-3,:)={'Smeared Shield B4C',adB4CSUP,mB4CSUP};
OtherMaterials(B4CwwrRow-3,:)={'WireWrap Poison Pins',adB4Cwwr,mB4Cwwr};

%% Screen Printing

dispPrint('Other Materials read..');
dispPrint(' ')

%Create a .mat file for the mats for reading later.
dispPrint('Saving Other Materials Material Info');
dispPrint(' ')
save('Debug\ISOadenOtherMats.mat','ISOaden')

end
```

B.19. AtomDenCalc.m

```
function [ISOAden, BurnUp, denout, matdbout ] = AtomDenCalc( matdbintemp,matdbdenin, dummy, Comptyp )

%% *************************** 4th circle ********************************

global         MatLabOld
global         CompCheck
global         KeyboardStop
global         MatSigFig
global         NaPen
global         SmearPerturb
persistent     sd
persistent     sh
persistent     np
persistent     FirstRun
persistent     mFPsFuelS1
persistent     mFPsFuelS2
persistent     mFPsFuelS3
persistent     mFPsBlanketS1
persistent     mFPsBlanketS2
persistent     mFPsBlanketS3
persistent     ChOn
persistent     SecVol
persistent     VolRatio
persistent     LatVol
persistent     NaPenVol
persistent     NaPenAden
persistent     NaZAID

% Load initial data

if isempty(FirstRun)

    % Load the FPs and the Chart of Nuclides

    [ mFPsFuelS1, mFPsFuelS2, mFPsFuelS3,...
        mFPsBlanketS1, mFPsBlanketS2, mFPsBlanketS3 ] = ISOLoader(  );

    [ ChOn ] = ChOnLoader(  );

end

% Check to see if this is a section of a fuel / blanket.
% If it is then it grabs the section number and sets the appropriate LFPs
```

```
NumSec=1;

if strcmp(Comptyp(end-1),'S')

    SecNum=str2num(Comptyp(end));

    Comptyp=Comptyp(1:end-2);

end

% Initialize Variables

ISOAden=0;
BurnUp=0;
denout=0;

% Initialize the function switches

CompCheck=true;
OtherMat=false;
BlanketMat=false;
FuelMat=false;
DimLoad=false;
CtrlMat=false;

if strcmp(Comptyp,'OtherMats');        SATy=0;  OtherMat=true;   end
if strcmp(Comptyp,'Ctrl');             SATy=4;  CtrlMat=true; OtherMat=true;  end
if strcmp(Comptyp,'Fuel');             SATy=1;  FuelMat=true;    end
if strcmp(Comptyp,'Blanket');          SATy=8;  BlanketMat=true; end
if strcmp(Comptyp(1:end-1),'DimLoad'); SATy=0;  DimLoad=true;    end

%% Set the f/b section number and LFPs

if FuelMat

    switch SecNum
        case 1; mFPs=mFPsFuelS1;
        case 2; mFPs=mFPsFuelS2;
        case 3; mFPs=mFPsFuelS3;
    end

end

if BlanketMat

    switch SecNum
        case 1; mFPs=mFPsBlanketS1;
        case 2; mFPs=mFPsBlanketS2;
        case 3; mFPs=mFPsBlanketS3;
    end

end

%% This is loaded intially and then used for the mass calculation later for the fuel slug.

% This calculates the volume of the slug plus the hex area.
if DimLoad

    NumSec=3;

    if strcmp(Comptyp(end),'F'); LatHeight=34.29/NumSec; SATy=1; end
    if strcmp(Comptyp(end),'B'); LatHeight=139.7/NumSec; SATy=8; end

    % *******DATA from material source********

    OR=5.8928/2;

    LatArea=6*tand(30)*(OR^2);

    LatVol=LatArea*LatHeight;

    sd=matdbintemp(2);
    sh=matdbintemp(3)/NumSec;
    np=matdbintemp(4);

    NaPenVol=(matdbintemp(5)/NumSec)*np;

    sr=sd/2;

    PinArea=(pi*(sr^2));

    % Total area of fuel pins across SA

    TotPinVol=PinArea*sh*np;

    TotPinArea=PinArea*np;

    % Volume of a single pin.

    SecVol=(sh*PinArea);

    VolRatio=RoundM(LatVol/TotPinVol,MatSigFig,'S');

    if NaPen && NaPenVol~=0;
```

```
        % Caculate the sodium aden that is to be added to the slug

        GlobalNaAden=matdbintemp(6);

        NaPenAden=(NaPenVol/LatVol)*GlobalNaAden;

        NaZAID='11023';

    else

         NaPenAden=0;

    end

end


%% This is where the volume for the slug is set and the adjustment is performed

if FuelMat || BlanketMat

    % Need to add sodium penetration to the slug if there is enough
    % swelling.

    if NaPen && NaPenAden~=0

        matdbinNoNaPen=matdbintemp;

        matdbintemp=[matdbinNoNaPen,{NaZAID; Num2StrM(NaPenAden)}];


    end

    % Grab the ZAIDS of interest to calculate the un adjusted mass.
    % List these ZAIDS in descending numerical order, if not they wont
    % match column wise with the rest of the generated data.

    UnAdjDatAdensZAID={'94239'; '92235'};
    UnAdjDatAdens=cell(3,size(UnAdjDatAdensZAID,2));

    for gg=1:length(UnAdjDatAdensZAID)

        UnAdjDatAdensCols=cellfind(matdbintemp(1,:),UnAdjDatAdensZAID{gg},2,'Match');

        if gg==1

             UnAdjDatAdens=...
                 [matdbintemp(1,UnAdjDatAdensCols); '1'; matdbintemp(2,UnAdjDatAdensCols)];

        else

             UnAdjDatAdens=...
                 [UnAdjDatAdens, ...
                 [matdbintemp(1,UnAdjDatAdensCols); '1'; matdbintemp(2,UnAdjDatAdensCols)]];

        end

    end

    % Convert the unadjusted composition to numbers
    UnAdjDatAdens=delNaN( UnAdjDatAdens, 'Convert All' , 'num' );

    % Convert to ZAIDdens format

    ZAIDadenTitlesPur={'Zaid Number';'% At Comp';'Initial Adens'};

    ZAIDTotalsPur={'Totals';sum(UnAdjDatAdens{2,1});sum(UnAdjDatAdens{3,1})};

    ZAIDdensPur=[ZAIDadenTitlesPur,...
        [{cell2mat(UnAdjDatAdens(1,:))};...
         {cell2mat(UnAdjDatAdens(2,:))};...
         {cell2mat(UnAdjDatAdens(3,:))}],...
         ZAIDTotalsPur];

    Volume=SecVol;

    % The adjustment is simple, since we are working with atom densities
    % which ultimately are atoms/vol. The vol is comprised height * area,
    % the height is the same for both so the conversion factor is just the
    % ratio of areas between all the slugs and the area of the hexagonal
    % cell.

    matdbtemp=...
        cellfun(@(x) x.*VolRatio,cellfun(@str2double , matdbintemp(2,:),'un',false));

    matdbintemp(2,:)=...
        cellfun(@(x) Num2StrM(x ,'%10.10e\n') ,num2cell(matdbtemp),'un',false);

else

   % Setting the volume to 0 causes the program to not calculated mass for
   % the given atom densitities.
    Volume=0;

end
```

```
%% Fission product removal

% Before anything can be calculated the FPs have to be removed from the
% data. This step only needs to occur if the incomming data is Fuel or
% Blanket

if FuelMat || BlanketMat

    [ FPsAds, NonFPsAds, NaAds ] = FPsDataRemove( matdbintemp );

else

    % If not fuel or blanket, the incomming data needs to change variable
    % name

    NonFPsAds=matdbintemp;

end

%% Convert all in coming data into numbers

if ~DimLoad

    % Convert the incoming composition to numbers
    matdbin=cell2mat(delNaN( NonFPsAds, 'Convert All' , 'num' ));

    if NaPen & NaPenAden~=0

        Namatdb=cell2mat(delNaN( NaAds, 'Convert All' , 'num' ));

    else

        Namatdb=[0;0];

    end

end

%% Fuel Data composition recalc

% With the data converted and the fission products removed, the percent
% composition has to be calculated. For fuel and blanket, this requires
% summing the atom densities of the fissionable material and of the
% fission products then dividing by that sum. Any Other Materials do not
% need it.

if FuelMat || BlanketMat

    % Actinide Zaids and adens

    ActinideZaids=matdbin(1,:); ActinideAdens=matdbin(2,:);

    % Actinide sum

    ActinidesSum=sum(ActinideAdens);

    % Sodium aden

    NaAdZaids=Namatdb(1,1); NaAden=Namatdb(2,1);

    % Uranium sum

    [~, Ucol]=find(ActinideZaids>92000 & ActinideZaids<93000);
    UraniumSum=sum(ActinideAdens(1, Ucol));

    % Fission product sum conversion

    FPsAdSum=FPsAds{2,end}; %Atom density sum from the fuel/blanket data

    % Material Total Aden

    matdbsum=sum([ActinidesSum,FPsAdSum,NaAden]); %Total aden sum from f/b data

    % Percent composition

    FPsAdComp=FPsAdSum/matdbsum; % of FPs in the fuel/blanket data

    % Material Total Aden

    ActinidesComp=ActinideAdens./matdbsum; % of Actinides in f/b data

    if NaPen && NaPenAden~=0

        NaAdComp=NaAden/matdbsum;

    else

        NaAdComp=0;

    end

    % As a self check, we are going to sum the composition and see if they
    % add to 1
```

```
    wgtsum=RoundM(sum([FPsAdComp, ActinidesComp, NaAdComp]),5);

    if wgtsum~=1

            disp('Error in Composition determination!'); keyboard;

    end

else if ~DimLoad

        % To prevent confusion and to make the output variable similar, the
        % other material comp is going to be extracted and then eventually
        % placed in the output variables thrid line.

        OtherMatsZaids=matdbin(1,:);
        OtherMatsComp=matdbin(2,:);

    end

end

%% Addition of fission products

% With % composition calculated there are two different methods than can be
% used. The fuel can utlize both the volume ratio method and calculation of
% atomdensity from % composition. Self checking has to be done be
% performing both methods. This will be done by self contained functions.

% The incoming material data needs to be reformatted so that no matter the
% material type fuel/ss extc... it will have the same format.

% ZAIDdens is the output variable for all sub calculation steps

% ZAIDdens will  have the following format before comp calc.

%                   ZAID numbers | DATA | Totals
%             Atom/Weight % Comp | DATA | 'Should equal 1'
%       Initial/Calculated Adens | DATA | 'Total Aden'  **ONLY FOR F/B**

ZAIDadenTitles={'Zaid Number';'% At Comp';'Initial Adens'};

if FuelMat || BlanketMat

   FPsZaids=mFPs(1,:);                % Zaid numbers from the FP data
   FPsAdens=FPsAdSum.*mFPs(2,:);      % The adens of the FPs in the f/b data.
   FPsComp=FPsAdComp.*mFPs(2,:);    % Percent comps of FPs in the f/b data


   ZAIDadenDat={[ActinideZaids,FPsZaids];...
                [ActinidesComp,FPsComp];...
                [ActinideAdens,FPsAdens]};

    if NaPen && NaPenAden~=0

           ZAIDadenDat={[ActinideZaids,FPsZaids,NaAdZaids];...
                 [ActinidesComp,FPsComp,NaAdComp];...
                 [ActinideAdens,FPsAdens,NaAden]};

    end

   ZAIDTotals={'Totals';sum(ZAIDadenDat{2,1});sum(ZAIDadenDat{3,1})};

   ZAIDdens=[ZAIDadenTitles,ZAIDadenDat,ZAIDTotals];

   % ZAIDdens needs to get passed to the function that actually calculates
   % the atom densities based upon their composition.

else if ~DimLoad

       ZAIDadenDat={OtherMatsZaids;...
                    OtherMatsComp;...
                   []};

       ZAIDTotals={'Totals';sum(ZAIDadenDat{2,1});[]};

       ZAIDdens=[ZAIDadenTitles,ZAIDadenDat,ZAIDTotals];

    end

end

% OtherMats is calculated using the given density from matdbdenin and
% passed to a function

%% Aden Calculation

if FuelMat || BlanketMat
if ZAIDdens{3,2}(end)>1;keyboard;end
    [ ZAIDdens ] = AdenCalc( ZAIDdens, matdbdenin, ChOn, Volume);

    % Turn off Composition Check
    CompCheck=false;

    [ ZAIDMassPur ] = AdenCalc( ZAIDdensPur, matdbdenin, ChOn , LatVol);
```

```
    % Turn on Composition Check
    CompCheck=true;

    temp1=ZAIDdens{1,2};
    temp11=ZAIDdens{5,2};
    temp2=ZAIDMassPur{1,2};
    temp22=ZAIDMassPur{5,2};

    ZAIDMassPur(1,1)={'ZAIDS of mass interest'};
    ZAIDMassPur(3,1)={'Calculated Mass Adjusted 1 Pin'};
    ZAIDMassPur(4,1)={'Calculated Mass Unadjusted 1 Pin'};
    ZAIDMassPur(5,1)={'Calculated Mass Adjusted Ductwide'};
    ZAIDMassPur(6,1)={'Calculated Mass Unadjusted Ductwide'};

    for ff=1:length(UnAdjDatAdensZAID)

        [~,MassCol]=find(temp1==str2num(UnAdjDatAdensZAID{ff}));
        [~,MassColPur]=find(temp2==str2num(UnAdjDatAdensZAID{ff}));

        if ff==1

            temp3=temp11(1,MassCol);
            temp4=temp22(1,MassColPur)/np;
            temp5=temp11(1,MassCol)*np;
            temp6=temp22(1,MassColPur);

        else

            temp3=[temp3, temp11(1,MassCol)];
            temp4=[temp4, temp22(1,MassColPur)/np];
            temp5=[temp5, temp11(1,MassCol)*np];
            temp6=[temp6, temp22(1,MassColPur)];

        end

    end

    ZAIDMassPur(3,2)={temp3};
    ZAIDMassPur(4,2)={temp4};
    ZAIDMassPur(5,2)={temp5};
    ZAIDMassPur(6,2)={temp6};

    % The next few lines are to round the answer to an appropriate number
    % of digits.

    ZAIDMassPur(3:end,3)=num2cell(cellfun(@(x) RoundM(x,MatSigFig,'S'), ZAIDMassPur(3:end,3)));

    ZAIDdens(4:end,3)=num2cell(cellfun(@(x) RoundM(x,MatSigFig,'S'), ZAIDdens(4:end,3)));

    ZAIDdens=[ZAIDdens;ZAIDMassPur(1,:); ZAIDMassPur(3:end,:)];

else if ~DimLoad

        % Debugging for HWCR slug weight.

        if CtrlMat; Volume=52.493384; end

        % The output of this makes the composition numbers ngeative, this is
        % for MCNP to tell it that the composition is based upon a wt fraction
        % not atom like it is for the fuel.

        [ ZAIDdens ] = AdenCalc( ZAIDdens, matdbdenin, ChOn, Volume);

        % The Output data can now be rounded to the appropriate number of
        % significant decimal places.

        ZAIDdens(4:end,3)=num2cell(cellfun(@(x) RoundM(x,MatSigFig,'S'), ZAIDdens(4:end,3)));

        % For a composition, the numbers need to be negative for mcnp.

        matdbout=matdbintemp;
        matdbout(2,:)=num2cell(cellfun(@(x) x.*(-1),matdbout(2,:)));

    end

end

%% Burnup Calculation

if FuelMat || BlanketMat


    FPsDat=cell2mat(FPsAds(2,:));
    colsLFPs=cellfind(FPsAds,'LFP',2,'NoMatch');
    colsLa=cellfind(FPsAds,'57139',2,'NoMatch');
    colsNd=cellfind(FPsAds,'60148',2,'NoMatch');

    LFPsad=FPsDat(1,colsLFPs);
    LaNum=FPsDat(1,colsLa);
    NdNum=FPsDat(1,colsNd);


    NumFis=0.5*(sum(LFPsad)+LaNum+NdNum);

    InitialAtoms=NumFis+UraniumSum;
```

```
    BurnUp=(InitialAtoms-ActinidesSum)/InitialAtoms;


end

%% Output Checks

% A check has to be performed to make sure that the % composition is
% normalized, if it isnt there could be a problem with the data.

if ~DimLoad

    CompTot=ZAIDdens{3,3};

    if RoundM(CompTot,4,'S')~=RoundM(1,4,'S')

        dispPrint('There is a problem with the Composition Calc.')
        dispPrint('The composition does not add to 1.')
        dispPrint(' ')
        dispPrint('Check the command window to troubleshoot.')
        keyboard

    end

end

%% Output material and density information

% ZAIDdens has the following format DATA is a cell containing the actual
% data

%                   ZAID numbers | DATA | Totals
%       Isotope Molecular Weight | DATA | 'N/A'
%             Atom/Weight % Comp | DATA | 'Should equal 1'
%       Initial/Calculated Adens | DATA | 'Total Aden'
%                Calculated Mass | DATA | 'Total Mass'  **ONLY FOR F/B**

if ~DimLoad

    ZAIDdens{1,2}=cellfun(@(x) Num2StrM(x, '%.8g'),num2cell(ZAIDdens{1,2}), 'UniformOutput', false);

    ISOAden=ZAIDdens;

    denout=ZAIDdens{4,3};

end

FirstRun=false;

end
```

B.20. FPsDataRemove.m

```
function [ FPsAds, matdbout, NaAds ] = FPsDataRemove( matdbin )

%% *************************** 4th circle ******************************

% This function reads in any data and then separates the FPs from the fuel
% data.


matdbout=matdbin;

LFPcols=cellfind(matdbout(1,:),'LFP',2,'NoMatch');
FIScols=cellfind(matdbout(1,:),'Fissium',2,'Match');
LA139cols=cellfind(matdbout(1,:),'57139',2,'Match');
ND148cols=cellfind(matdbout(1,:),'60148',2,'Match');
Nacols=cellfind(matdbout(1,:),'11023',2,'Match');

if FIScols==0

    FPscols=[LFPcols; LA139cols; ND148cols];

else

    FPscols=[LFPcols; FIScols; LA139cols; ND148cols];

end

% Attempt to grab the FPs ZAIDS

try
    FPsTitles=matdbout(1,FPscols);
catch ME

    ME.identifier
    disp('Trouble in FPsDataRemove.m')
    disp(' ')
    disp('This error can happen if La139 or Nd148 are listed in the database without being in ZAID format.')
    disp('They should be listed as 57139 and 60148')
    disp(' ')
    disp('Keyboard command for further troubleshooting')
```

```
    keyboard

end

% Grab the FPs adens
FPsNums=str2double(matdbout(2,FPscols));

%All fission products need to be added together

FPsaden=sum(FPsNums);
FPsAds=[[FPsTitles, 'Totals']; [num2cell(FPsNums), num2cell(FPsaden)]];

% Grab the Na ZAID and num
if Nacols~=0;

    NaTitles=matdbout(1,Nacols);
    NaNum=str2double(matdbout(2,Nacols));
    NaAds=[NaTitles; num2cell(NaNum)];

    %Replace the LFPs La Nd and Fissium with empty cells.
    matdbout(:,[FPscols; Nacols])=[];

else

    NaAds=['0';num2cell(0)];

    %Replace the LFPs La Nd and Fissium with empty cells.
    matdbout(:,FPscols)=[];

end


end
```

B.21. AdenCalc.m

```
function [ FPsAds, matdbout, NaAds ] = FPsDataRemove( matdbin )

%% ************************** 4th circle ********************************

% This function reads in any data and then separates the FPs from the fuel
% data.


matdbout=matdbin;

LFPcols=cellfind(matdbout(1,:),'LFP',2,'NoMatch');
FIScols=cellfind(matdbout(1,:),'Fissium',2,'Match');
LA139cols=cellfind(matdbout(1,:),'57139',2,'Match');
ND148cols=cellfind(matdbout(1,:),'60148',2,'Match');
Nacols=cellfind(matdbout(1,:),'11023',2,'Match');

if FIScols==0

    FPscols=[LFPcols; LA139cols; ND148cols];

else

    FPscols=[LFPcols; FIScols; LA139cols; ND148cols];

end

% Attempt to grab the FPs ZAIDS

try
    FPsTitles=matdbout(1,FPscols);
catch ME

    ME.identifier
    disp('Trouble in FPsDataRemove.m')
    disp(' ')
    disp('This error can happen if La139 or Nd148 are listed in the database without being in ZAID format.')
    disp('They should be listed as 57139 and 60148')
    disp(' ')
    disp('Keyboard command for further troubleshooting')
    keyboard

end

% Grab the FPs adens
FPsNums=str2double(matdbout(2,FPscols));

%All fission products need to be added together

FPsaden=sum(FPsNums);
FPsAds=[[FPsTitles, 'Totals']; [num2cell(FPsNums), num2cell(FPsaden)]];

% Grab the Na ZAID and num
if Nacols~=0;

    NaTitles=matdbout(1,Nacols);
    NaNum=str2double(matdbout(2,Nacols));
    NaAds=[NaTitles; num2cell(NaNum)];
```

```
    %Replace the LFPs La Nd and Fissium with empty cells.
    matdbout(:,[FPscols; Nacols])=[];

else

    NaAds=['0';num2cell(0)];

    %Replace the LFPs La Nd and Fissium with empty cells.
    matdbout(:,FPscols)=[];

end


end
```

B.22. HWCRB4CLoader.m

```
function [ B4CDatOut,ISOaden ] = HWCRB4CLoader( ChON )

global KeyboardStop
global BoronPerturb
global OtherMatsPath

% This function loads the specific burnups for the B4C in the HWCRs.

% HWCR import

DPerturb=false;

if iscell(BoronPerturb)

   Pert=BoronPerturb{4};

   ScaleF=BoronPerturb{5};

   BoronPerturb=false;

   DPerturb=true;

end

if BoronPerturb

    B4CDatIn=importdata(OtherMatsPath);

else

    B4CDatIn=importdata('Specs\Materials\RealDb\Control Mats.xlsx');

end

B4CDatNumIn=num2cell(B4CDatIn.data);
B4CDatTextIn=B4CDatIn.textdata;

B4CZaids=B4CDatNumIn(1,2:4);

B4CDatOut=B4CDatTextIn(3:end,1);

for ii=1:size(B4CDatOut,1)

    currB4C=[B4CZaids;B4CDatNumIn(ii+1,2:4)];
    curdB4C=B4CDatNumIn{ii+1,1};

    if DPerturb

        curdB4C=curdB4C+(Pert*ScaleF);

    end

    [ISOaden{ii,2},~,adB4C,mB4C]=AtomDenCalc(currB4C,curdB4C,ChON,'Ctrl');
    ISOaden(ii,1)=B4CDatTextIn(ii+2,1);

    B4CDatOut{ii,2}=adB4C;

    B4CDatOut{ii,3}=mB4C;

end

global BadBoronMat

if BadBoronMat

    % This introduces an old error in the boron material composition

    B4CDatOut{1,3}=[{5010},{5011},{6000};...
        {0.14424056},{0.63835944},{0.2174}];

end

end
```

B.23. DenTempAdj.m

```
function [ OtherMaterials, ISOaden ] = DenTempAdj(OtherMatsRows, OtherMaterialsNoAdj, ISOaden , deltak )

global MatSigFig RoomTemp

% this function adjusts the mass densities of the materials.

% This only performs a correction on the SS and the Na

% This code will need to be expanded to perform an actual adjustment.

%% Need to decompile the information.

OtherMaterials=OtherMaterialsNoAdj;

%% Gather the variables into one variable

% Atom Densities

adSodium=   OtherMaterials{OtherMatsRows(1)-3,2};
adArgon=    OtherMaterials{OtherMatsRows(2)-3,2};
adHelium=   OtherMaterials{OtherMatsRows(3)-3,2};
adPgas=     OtherMaterials{OtherMatsRows(4)-3,2};
adSS304=    OtherMaterials{OtherMatsRows(5)-3,2};
adB4Cwwr=   OtherMaterials{OtherMatsRows(6)-3,2};
adSS316=    OtherMaterials{OtherMatsRows(7)-3,2};
adFPwwr=    OtherMaterials{OtherMatsRows(8)-3,2};
adB4C=      OtherMaterials{OtherMatsRows(9)-3,2};
adSUP=      OtherMaterials{OtherMatsRows(10)-3,2};
adSLO=      OtherMaterials{OtherMatsRows(11)-3,2};
adSLC=      OtherMaterials{OtherMatsRows(12)-3,2};
adSIHDLC=   OtherMaterials{OtherMatsRows(13)-3,2};
adSafetySUP=OtherMaterials{OtherMatsRows(14)-3,2};
adB4CSUP=   OtherMaterials{OtherMatsRows(15)-3,2};
adRefSS304s=OtherMaterials{OtherMatsRows(16)-3,2};
adRefSS316s=OtherMaterials{OtherMatsRows(17)-3,2};
adHEU=      OtherMaterials{OtherMatsRows(18)-3,2};

% Mass densities

dSodium=   ISOaden{OtherMatsRows(1)-3,2}{end,end};
dArgon=    ISOaden{OtherMatsRows(2)-3,2}{end,end};
dHelium=   ISOaden{OtherMatsRows(3)-3,2}{end,end};
dPgas=     ISOaden{OtherMatsRows(4)-3,2}{end,end};
dSS304=    ISOaden{OtherMatsRows(5)-3,2}{end,end};
dB4Cwwr=   ISOaden{OtherMatsRows(6)-3,2}{end,end};
dSS316=    ISOaden{OtherMatsRows(7)-3,2}{end,end};
dFPwwr=    ISOaden{OtherMatsRows(8)-3,2}{end,end};
dB4C=      ISOaden{OtherMatsRows(9)-3,2}{end,end};
dSUP=      ISOaden{OtherMatsRows(10)-3,2}{end,end};
dSLO=      ISOaden{OtherMatsRows(11)-3,2}{end,end};
dSLC=      ISOaden{OtherMatsRows(12)-3,2}{end,end};
dSIHDLC=   ISOaden{OtherMatsRows(13)-3,2}{end,end};
dSafetySUP=ISOaden{OtherMatsRows(14)-3,2}{end,end};
dB4CSUP=   ISOaden{OtherMatsRows(15)-3,2}{end,end};
dRefSS304s=ISOaden{OtherMatsRows(16)-3,2}{end,end};
dRefSS316s=ISOaden{OtherMatsRows(17)-3,2}{end,end};
dHEU=      ISOaden{OtherMatsRows(18)-3,2}{end,end};

% Material Comps

mSodium=   OtherMaterials{OtherMatsRows(1)-3,3};
mArgon=    OtherMaterials{OtherMatsRows(2)-3,3};
mHelium=   OtherMaterials{OtherMatsRows(3)-3,3};
mPgas=     OtherMaterials{OtherMatsRows(4)-3,3};
mSS304=    OtherMaterials{OtherMatsRows(5)-3,3};
mB4Cwwr=   OtherMaterials{OtherMatsRows(6)-3,3};
mSS316=    OtherMaterials{OtherMatsRows(7)-3,3};
mFPwwr=    OtherMaterials{OtherMatsRows(8)-3,3};
mB4C=      OtherMaterials{OtherMatsRows(9)-3,3};
mSUP=      OtherMaterials{OtherMatsRows(10)-3,3};
mSLO=      OtherMaterials{OtherMatsRows(11)-3,3};
mSLC=      OtherMaterials{OtherMatsRows(12)-3,3};
mSIHDLC=   OtherMaterials{OtherMatsRows(13)-3,3};
mSafetySUP=OtherMaterials{OtherMatsRows(14)-3,3};
mB4CSUP=   OtherMaterials{OtherMatsRows(15)-3,3};
mRefSS304s=OtherMaterials{OtherMatsRows(16)-3,3};
mRefSS316s=OtherMaterials{OtherMatsRows(17)-3,3};
mHEU=      OtherMaterials{OtherMatsRows(18)-3,3};

%% Alpha values

% Linear expansion coefficient from Matweb
ss304alpha=SS304AlphaRng(deltak);

% Since the neutron shield uses the same d
Shieldalpha=ss304alpha;

% Linear expansion coefficientfrom engineeringtoolbox.com
ss316alpha=SS316AlphaRng(deltak);

%% Beta values

%  Beta  for  sodium  =  2.75E-4  from  https://syeilendrapramuditya.wordpress.com/2010/11/30/sodium-volumetric-thermal-
expansion-coefficient/
Sodiumbeta=NaBetaRng(deltak);
```

```
%% SS304

% Estimated beta=alpha*3;
ss304beta=3*ss304alpha;

% New SS304 Density
SS304DenAdj=(1+(ss304beta*deltak));
adSS304=RoundM(adSS304/SS304DenAdj,MatSigFig,'S');
dSS304=RoundM(dSS304/SS304DenAdj,MatSigFig,'S');

% New B4Cwwr Density
adB4Cwwr=RoundM(adB4Cwwr/SS304DenAdj,MatSigFig,'S');
dB4Cwwr=RoundM(dB4Cwwr/SS304DenAdj,MatSigFig,'S');

%% Stainless Steel 304 Reflector Smear

% Grab the sodium ratio
Ref304Narat=mRefSS304s{2,2};

% Linear expansion for SLO and SUP are weight averaged for their
% coefficients
Ref304beta=(abs(Ref304Narat-1)*(ss304beta)+(Ref304Narat)*(Sodiumbeta));

% New SUP Density
Ref304DenAdj=(1+(Ref304beta*deltak));
adRefSS304s=RoundM(adRefSS304s/Ref304DenAdj,MatSigFig,'S');
dRefSS304s=RoundM(dRefSS304s/Ref304DenAdj,MatSigFig,'S');

%% Stainless Steel 316 Reflector Smear

% Grab the sodium ratio
Ref316Narat=mRefSS316s{2,2};

% Linear expansion for SLO and SUP are weight averaged for their
% coefficients
Ref316beta=(abs(Ref316Narat-1)*(ss304beta)+(Ref316Narat)*(Sodiumbeta));

% New SUP Density
Ref316DenAdj=(1+(Ref316beta*deltak));
adRefSS316s=RoundM(adRefSS316s/Ref316DenAdj,MatSigFig,'S');
dRefSS316s=RoundM(dRefSS316s/Ref316DenAdj,MatSigFig,'S');

%% SS316

% Estimated beta=alpha*3;
ss316beta=3*ss316alpha;

% New SS316 Density
SS316DenAdj=(1+(ss316beta*deltak));
adSS316=RoundM(adSS316/SS316DenAdj,MatSigFig,'S');
dSS316=RoundM(dSS316/SS316DenAdj,MatSigFig,'S');

% New wire wrap SS316 Density
adFPwwr=RoundM(adFPwwr/SS316DenAdj,MatSigFig,'S');
dFPwwr=RoundM(dFPwwr/SS316DenAdj,MatSigFig,'S');

%% Sodium

% New Sodium Density
SodiumDenAdj=(1+(Sodiumbeta*deltak));
adSodium=RoundM(adSodium/SodiumDenAdj,MatSigFig,'S');
dSodium=RoundM(dSodium/SodiumDenAdj,MatSigFig,'S');

%% Smeared upper extension

% Grab the sodium ratio
SUPNarat=mSUP{2,2};

% Linear expansion for SLO and SUP are weight averaged for their
% coefficients
SUPbeta=(abs(SUPNarat-1)*(ss304beta)+(SUPNarat)*(Sodiumbeta));

% New SUP Density
SupDenAdj=(1+(SUPbeta*deltak));
adSUP=RoundM(adSUP/SupDenAdj,MatSigFig,'S');
dSUP=RoundM(dSUP/SupDenAdj,MatSigFig,'S');

%% Smeared Lower Extension

% Grab the sodium ratio
SLONarat=mSLO{2,2};

% SLObeta
SLObeta=(abs(SLONarat-1)*(ss304beta)+(SLONarat)*(Sodiumbeta));

% New SUP Density
SLODenAdj=(1+(SLObeta*deltak));
adSLO=RoundM(adSLO/SLODenAdj,MatSigFig,'S');
dSLO=RoundM(dSLO/SLODenAdj,MatSigFig,'S');

%% Smeared Lower Cylinder

% Grab the sodium ratio
SLCNarat=mSLC{2,2};
```

```
% Linear expansion for SLO and SUP are weight averaged for their
% coefficients
SUPbeta=(abs(SLCNarat-1)*(ss304beta)+(SLCNarat)*(Sodiumbeta));

% New SUP Density
SLCDenAdj=(1+(SUPbeta*deltak));
adSLC=RoundM(adSLC/SLCDenAdj,MatSigFig,'S');
dSLC=RoundM(dSLC/SLCDenAdj,MatSigFig,'S');

%% Smeared Inner Hex Duct Lower Cylinder

% Grab the sodium ratio
SIHDLCNarat=mSIHDLC{2,2};

% SLObeta
SIHDLCbeta=(abs(SIHDLCNarat-1)*(ss304beta)+(SIHDLCNarat)*(Sodiumbeta));

% New SUP Density
SIHDLCDenAdj=(1+(SIHDLCbeta*deltak));
adSIHDLC=RoundM(adSIHDLC/SIHDLCDenAdj,MatSigFig,'S');
dSIHDLC=RoundM(dSIHDLC/SIHDLCDenAdj,MatSigFig,'S');

%% Smeared Safety Upper Shield

% Grab the sodium ratio
SafetySUPNarat=mSafetySUP{2,2};

% SLObeta
SafetySUPbeta=(abs(SafetySUPNarat-1)*(ss304beta)+(SafetySUPNarat)*(Sodiumbeta));

% New SUP Density
SafetySUPDenAdj=(1+(SafetySUPbeta*deltak));
adSafetySUP=RoundM(adSafetySUP/SafetySUPDenAdj,MatSigFig,'S');
dSafetySUP=RoundM(dSafetySUP/SafetySUPDenAdj,MatSigFig,'S');

%% Smeared B4c Shield

% Grab the sodium ratio
B4CSUPNarat=mB4CSUP{2,2};

% SLObeta
B4CSUPbeta=(abs(B4CSUPNarat-1)*(ss304beta)+(B4CSUPNarat)*(Sodiumbeta));

% New SUP Density
B4CSUPDenAdj=(1+(B4CSUPbeta*deltak));
adB4CSUP=RoundM(adB4CSUP/B4CSUPDenAdj,MatSigFig,'S');
dB4CSUP=RoundM(dB4CSUP/B4CSUPDenAdj,MatSigFig,'S');

%% Recompile information

OtherMaterials{OtherMatsRows(1)-3,2}=adSodium;
OtherMaterials{OtherMatsRows(2)-3,2}=adArgon;
OtherMaterials{OtherMatsRows(3)-3,2}=adHelium;
OtherMaterials{OtherMatsRows(4)-3,2}=adPgas;
OtherMaterials{OtherMatsRows(5)-3,2}=adSS304;
OtherMaterials{OtherMatsRows(6)-3,2}=adB4Cwwr;
OtherMaterials{OtherMatsRows(7)-3,2}=adSS316;
OtherMaterials{OtherMatsRows(8)-3,2}=adFPwwr;
OtherMaterials{OtherMatsRows(9)-3,2}=adB4C;
OtherMaterials{OtherMatsRows(10)-3,2}=adSUP;
OtherMaterials{OtherMatsRows(11)-3,2}=adSLO;
OtherMaterials{OtherMatsRows(12)-3,2}=adSLC;
OtherMaterials{OtherMatsRows(13)-3,2}=adSIHDLC;
OtherMaterials{OtherMatsRows(14)-3,2}=adSafetySUP;
OtherMaterials{OtherMatsRows(15)-3,2}=adB4CSUP;
OtherMaterials{OtherMatsRows(16)-3,2}=adRefSS304s;
OtherMaterials{OtherMatsRows(17)-3,2}=adRefSS316s;
OtherMaterials{OtherMatsRows(18)-3,2}=adHEU;

% Mass densities

ISOaden{OtherMatsRows(1)-3,2}{end,end}= dSodium;
ISOaden{OtherMatsRows(2)-3,2}{end,end}= dArgon;
ISOaden{OtherMatsRows(3)-3,2}{end,end}= dHelium;
ISOaden{OtherMatsRows(4)-3,2}{end,end}= dPgas;
ISOaden{OtherMatsRows(5)-3,2}{end,end}= dSS304;
ISOaden{OtherMatsRows(6)-3,2}{end,end}= dB4Cwwr;
ISOaden{OtherMatsRows(7)-3,2}{end,end}= dSS316;
ISOaden{OtherMatsRows(8)-3,2}{end,end}= dFPwwr;
ISOaden{OtherMatsRows(9)-3,2}{end,end}= dB4C;
ISOaden{OtherMatsRows(10)-3,2}{end,end}=dSUP;
ISOaden{OtherMatsRows(11)-3,2}{end,end}=dSLO;
ISOaden{OtherMatsRows(12)-3,2}{end,end}=dSLC;
ISOaden{OtherMatsRows(13)-3,2}{end,end}=dSIHDLC;
ISOaden{OtherMatsRows(14)-3,2}{end,end}=dSafetySUP;
ISOaden{OtherMatsRows(15)-3,2}{end,end}=dB4CSUP;
ISOaden{OtherMatsRows(16)-3,2}{end,end}=dRefSS304s;
ISOaden{OtherMatsRows(17)-3,2}{end,end}=dRefSS316s;
ISOaden{OtherMatsRows(18)-3,2}{end,end}=dHEU;


end

function Alpha=SS304AlphaRng(deltak)
```

```
global RoomTemp
global CtoKconv

Temp=deltak+RoomTemp;

AlphaRng=[100,17.3e-6;315,17.8e-6;650,18.7e-6];

[row,~]=find(AlphaRng(:,1)>=Temp);

if row(1)~=1

    TempH=AlphaRng(row,1);
    TempL=AlphaRng(row-1,1);

    TDiff=TempH-TempL;

    Tfrac=(Temp-TempL)/TDiff;

    AlphH=AlphaRng(row,2);
    AlphL=AlphaRng(row-1,2);

    ADiff=AlphH-AlphL;

    Alpha=AlphaRng(row-1,2)+(Tfrac*ADiff);

end

end

function Alpha=SS316AlphaRng(deltak)

global RoomTemp
global CtoKconv

Temp=deltak+RoomTemp;

AlphaRng=[100,16e-6;315,16.2e-6;540,17.5e-6];

[row,~]=find(AlphaRng(:,1)>=Temp);

if row(1)~=1

    TempH=AlphaRng(row,1);
    TempL=AlphaRng(row-1,1);

    TDiff=TempH-TempL;

    Tfrac=(Temp-TempL)/TDiff;

    AlphH=AlphaRng(row,2);
    AlphL=AlphaRng(row-1,2);

    ADiff=AlphH-AlphL;

    Alpha=AlphaRng(row-1,2)+(Tfrac*ADiff);

end

end

function Beta=NaBetaRng(deltak)

global RoomTemp
global CtoKconv

Temp=deltak+RoomTemp+CtoKconv;

BetaRng=[400,2.41e-4;...
          500,2.5e-4;...
          600,2.6e-4;...
          700,2.71e-4;...
          800,2.82e-4;...
          900,2.95e-4];

[row,~]=find(BetaRng(:,1)>=Temp);

if row(1)~=1

    TempH=BetaRng(row(1),1);
    TempL=BetaRng(row(1)-1,1);

    TDiff=TempH-TempL;

    Tfrac=(Temp-TempL)/TDiff;

    BetaH=BetaRng(row(1),2);
    BetaL=BetaRng(row(1)-1,2);

    BDiff=BetaH-BetaL;

    Beta=BetaRng(row(1)-1,2)+(Tfrac*BDiff);

end
```

```
end
```

B.24. BurnCalc.m

```
function [ AvgBurn ] = BurnCalc( CurMat )

% Seperate Fissions products from non

for ii=1:3

    CurMatTemp=[CurMat(1,:);CurMat(ii+1,:)];

    [ FPsAds, Actinides ] = FPsDataRemove( CurMatTemp );

    ActinideZaids=Actinides(1,:); ActinideAdens=Actinides(2,:);

    % Actinide sum

    ActAdensNums=cell2mat(delNaN( ActinideAdens, 'Convert All' , 'num' ));
    ActZaidsNums=cell2mat(delNaN( ActinideZaids, 'Convert All' , 'num' ));

    ActinidesSum=sum(ActAdensNums,2);

    % Uranium sum

    [~, Ucol]=find(ActZaidsNums>92000 & ActZaidsNums<93000);
    UraniumSum=sum(ActAdensNums(1, Ucol));

    % Calculate burnup.

    FPsDat=cell2mat(FPsAds(2,:));
    colsLFPs=cellfind(FPsAds,'LFP',2,'NoMatch');
    colsLa=cellfind(FPsAds,'57139',2,'NoMatch');
    colsNd=cellfind(FPsAds,'60148',2,'NoMatch');

    LFPsad=FPsDat(1,colsLFPs);
    LaNum=FPsDat(1,colsLa);
    NdNum=FPsDat(1,colsNd);

    NumFis=0.5*(sum(LFPsad)+LaNum+NdNum);

    InitialAtoms=NumFis+UraniumSum;

    BurnUp(ii,1)=100*(InitialAtoms-ActinidesSum)/InitialAtoms;

end

% We must average the burnup to apply a uniform swell.

AvgBurn=mean(BurnUp);

end
```

B.25. ElementSwell.m

```
function [DimMap, SS304ad, SS316ad, BigSwell,feod] = ElementSwell(DimMap, SATy, ReactMapPOSind, AvgBurn, OLDSS304ad,
OLDSS316ad )
% This function will swell Fuel elements

%% Variable prep

global MNumC MSATyC SAPosC SANomen NomenTyC SmearSATYs
global hdidC hdwtC ihdidC ihdwtC crhC fehC feidC feodC fewtC
global fepC depC feoxC

if AvgBurn>=3.03; BigSwell=true; AvgBurn=3.03; else; BigSwell=false; end

% Grab the element height

OLDfeh=DimMap{ReactMapPOSind+1,fehC};

% Grab the Outer Clad Radius

OLDfeod=DimMap{ReactMapPOSind+1,feodC};

% Grab the Inner Clad Radius

OLDfeid=DimMap{ReactMapPOSind+1,feidC};

% Grab the wall thickness

OLDfewt=DimMap{ReactMapPOSind+1,fewtC};

% Grab the duct id

hdid=DimMap{ReactMapPOSind+1,hdidC};

if any(SATy==[3,4,5])

    hdid=DimMap{ReactMapPOSind+1,ihdidC};
```

```
end

% Core region height

crh=DimMap{ReactMapPOSind+1,crhC};

% Grab the hexduct wall thickness

hdwt=DimMap{ReactMapPOSind+1,hdwtC};

if any(SATy==[3,4,5])

    hdwt=DimMap{ReactMapPOSind+1,ihdwtC};

end

% Grab the offset

OLDfeox=DimMap{ReactMapPOSind+1,feoxC};

SS316=false;
SS304=false;

% Number of pins their origin and material

switch SATy
    case 1
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,fepC};
    case 2
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,fepC};
    case 3
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,fepC};
    case 4
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,fepC};
    case 5
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,fepC};
    case 6
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,depC};
    case 8
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,depC};
    case 10
        EXPID=DimMap{ReactMapPOSind+1,SANomen};
        SS316=true;
        Pitch=DimMap{ReactMapPOSind+1,fepC};

end


%% Define the functions that will swell the dimensions. These equations were
% derived from regressions of the plots from Ezinger

sd304Swell=@(x) (0.0006*(x)^4) - (0.0051*(x)^3) + (0.0736*(x)^2) - (0.0757*x);

sh304Swell=@(x) (0.00021*(x)^4) - (0.0205*(x)^3) + (0.1791*(x)^2) - (0.0071*x);

sd316Swell=@(x) (-8e-5*(x)^4) + (0.0053*(x)^3) - (0.0104*(x)^2) + (0.0101*x);

sh316Swell=@(x) (-8e-4*(x)^4) + (0.011*(x)^3) + (0.0054*(x)^2) + (0.031*x);

%% Split apart the two materials for the pin

if SS304

    % Diameter

    feod=OLDfeod*(1+sd304Swell(AvgBurn));
    fewt=OLDfewt*(1+sd304Swell(AvgBurn));
    feid=feod-2*fewt;

    % length

    feh=OLDfeh+sh304Swell(AvgBurn)/10;

end

if SS316

    % Diameter

    feod=OLDfeod*(1+sd316Swell(AvgBurn));
    fewt=OLDfewt*(1+sd316Swell(AvgBurn));
    feid=feod-2*fewt;

    % length

    feh=OLDfeh+sh316Swell(AvgBurn)/10;

end
```

```
%% Old Volume

OLDOuterVol=pi*((OLDfeod/2)^2)*OLDfeh;
OLDInnerVol=pi*((OLDfeid/2)^2)*(OLDfeh-2*OLDfewt);

OldVol=OLDOuterVol-OLDInnerVol;

%% Before the new volume is calculated, a check needs to be performed to see
%  if the pin is bigger than the lattice element

% Create a circle of points based upon feod
feor=feod/2;

% Need to also calculate the shift of the pin this will be the difference
% in radii between the old and the new element

feox=feor-(OLDfeod/2)+OLDfeox;

Xcirc=linspace(-feor,0,100)'-feox;

Ycirc=sqrt((feor.^2)-((Xcirc+feox).^2));

% Create a line based upon the north west facet

EleWinS=tand(30)*Pitch;

y1=sind(60)*EleWinS;
x1=-cosd(60)*EleWinS;

y2=0;
x2=2*x1;

% find an equation for the line of that facet
try
coefln = polyfit([x1, x2], [y1, y2], 1);
catch;keyboard;end
yln=coefln(1).*Xcirc+coefln(2);

% Now compare the y values, if any of the y values from the circle are greater
% than the values from the line, then we have violated our model

modelviolation=false;

if sum(Ycirc>=yln)>=1; modelviolation=true; keyboard; end

%% New volume

OuterVol=pi*((feod/2)^2)*feh;
InnerVol=pi*((feid/2)^2)*(feh-2*fewt);

NewVol=OuterVol-InnerVol;

% Calculate the Volume ratio

VolRat=NewVol/OldVol;

% Change the atom densities

if SS304

    SS304ad=OLDSS304ad/VolRat;
    SS316ad=0;

end

if SS316

    SS304ad=0;
    SS316ad=OLDSS316ad/VolRat;

end

DimMap{ReactMapPOSind+1,fehC}=feh;
DimMap{ReactMapPOSind+1,feodC}=feod;
DimMap{ReactMapPOSind+1,feidC}=feid;
DimMap{ReactMapPOSind+1,fewtC}=fewt;
DimMap{ReactMapPOSind+1,feoxC}=feox;

if BigSwell; DimMap{ReactMapPOSind+1,feoxC+1}='Element Swell exceeded lattice window! average burn reset to 3.317'; end

end
```

B.26. SlugSwell.m

```
function [ nsd, osd, sh, VolChng, sdViolate, MaxBurn ] = SlugSwell( AvgBurn,OLDsd,OLDsh,feid)
% This function calculates the burnup and swell of the current materials comosition.

sdViolate=false;

% Old volume

OLDvol=OLDsh*pi*(OLDsd/2)^2;
```

```
% The two constants used are

% The Einziger Seidel paper listed a range of 5.4 to 6.8% radial growth
% The benchmark will use 6.1

% Radial Direction
RadBurn=0.061; % Units are % swell per % at burnup
AxBurn=0.007; % Units are % swell per % at burnup

nsd=OLDsd*(1+(AvgBurn*RadBurn));
sh=OLDsh*(1+(AvgBurn*AxBurn));

% Self check has to be performed to make sure the slug diameter does not
% exceed feid, if it does, then the radius is just changed to a little less
% than feid.

osd=nsd;

if nsd>=feid

    % If a violation occurs, 99% of feid is used for the new id and burnup

    sdViolate=true;

    % find the new burnup
    MaxBurn=((feid*0.99)-OLDsd)/RadBurn;

    nsd=OLDsd*(1+(MaxBurn*RadBurn));
    sh=OLDsh*(1+(MaxBurn*AxBurn));

else

    MaxBurn=AvgBurn;

end

% new volume

Vol=sh*pi*(nsd/2)^2;

VolChng=Vol-OLDvol;
VolChngFrac=Vol/OLDvol;

if (VolChngFrac-1) <= 0.3; VolChng=0; end

end
```

B.27. dispPrint.m

```
function [ ] = dispPrint( InText )

% To use dispPrint as a progress bar, make the input a 2 by 1 cell array
% with the progress bar text in row one and the progress counters in row
% two with the first number being the progress and the second being the
% total.

persistent numCharH
global f1
global dax
global Title
persistent CallCnt
persistent numCharW
global MatLabOld
persistent dispText
persistent ProgCnt
persistent FracCntOld
persistent C1
persistent C2

IsProg=false;
FracCntEnd=false;
PrintNewLn=true;
UpdateCnt=true;

if size(InText,1)==2 && size(InText,2)==1

    Cnts=InText{2,1};
    CntsProg=Cnts(1);
    CntsTot=Cnts(2);
    InText=InText{1,1};

    IsProg=true;

    PNC=50;

    C2=CntsProg;

    if isempty(ProgCnt); ProgCnt=1; C1=0; FracCntOld=0; end

    CntsDiff=abs(C2-C1);

    if (CntsProg+CntsDiff)>=CntsTot
```

```
        FracCntEnd=true;
        FracCntOld=0;

    else

        % Progress Ratio
        CntsRat=CntsProg/CntsTot;

        FracCnt=round(CntsRat*PNC);

        if FracCnt==FracCntOld

            UpdateCnt=false;

        else

            FracCntOld=FracCnt;

        end

    end

    C1=CntsProg;

end

if UpdateCnt

    if MatLabOld

        if IsProg

            clc
            disp([InText ': ' int2str(RoundM(Cnts(1)/CntsDiff,0)) ' of ' int2str(RoundM(Cnts(2)/CntsDiff,0))])

        else

            disp(InText)

        end

    else

        if isempty(CallCnt)

            CallCnt=1;

            if isempty(get(groot,'CurrentFigure'));

                f1=gcf;

                f1.Color = [0 0.5 0.5];
                f1.ToolBar = 'none';
                f1.Units='pixels';
                f1.Position=[0 0 1260 700];
                f1.Units='characters';
                f1.Name=Title;
                f1.NumberTitle='off';

            else

                f1=gcf;
                f1.Name=Title;

            end

            FigSize=f1.Position;

            numCharW=round(FigSize(3)-0.2*FigSize(3));
            numCharH=round(FigSize(4)/1.7);

            dax=subplot(2,2,1,'Parent',f1);

            hold(dax,'on');

            axis(dax,[0  numCharW 0 numCharH],'off');

            dax.Clipping='off';

            text(-0.2*numCharW,numCharH,InText,'Parent',dax,'VerticalAlignment','top','FontSize',12)

            title(dax,{'Status Messages'; ' '})

            dispText={InText};

        else

            OldDispTextLastLn=dispText(end,:);

            if CallCnt==numCharH

               dispText=InText;
               CallCnt=1;
```

```
            if ~FracCntEnd
                if IsProg

                    SpacCnt=PNC-FracCnt;

                    InText=['[', repmat('*',1,FracCnt),repmat(' ',1,SpacCnt), ']'];

                    dispText=[dispText; {InText}];

                end
            else

                    InText=['[', repmat('*',1,PNC*0.5-4),'Complete!',repmat('*',1,PNC*0.5-5), ']'];

                    dispText=[dispText; {InText}];

                    clear ProgCnt

            end
        else

            if IsProg

                if ~FracCntEnd

                    SpacCnt=PNC-FracCnt;

                    progTitle=InText;

                    InText=['[', repmat('*',1,FracCnt),repmat(' ',1,SpacCnt), ']'];

                    if ProgCnt==1

                        dispText=[dispText; {progTitle}; {InText}];

                        ProgCnt=ProgCnt+1;

                        CallCnt=CallCnt+1;

                    else

                        dispText=[dispText(1:end-1,:); {InText}];

                    end

                    if strcmp(OldDispTextLastLn,dispText{end,:}) && IsProg

                        PrintNewLn=false;

                    end

                else

                    InText=['[', repmat('*',1,PNC*0.5-4),'Complete!',repmat('*',1,PNC*0.5-5), ']'];

                    dispText=[dispText(1:end-1); {InText}];

                    clear ProgCnt

                    CallCnt=CallCnt+1;

                end


            else

                dispText=[dispText; {InText}];
                CallCnt=CallCnt+1 ;

            end



        end

        if PrintNewLn

            cla(dax)
            text(-0.2*numCharW,numCharH,dispText,'Parent',dax,'VerticalAlignment','top',...
                'FontSize',12,'FontWeight','bold','FontName','Courier New');

        end

    end

    drawnow limitrate

end

end

end
```

B.28. HexOrg.m

```
function [ hdorg,OrgMap ] = HexOrg( nsa,hdp,ReactMap )

%%
global debugMICKA
global OrgOut

%*********************************Debug****************************************

%if debugMICKA; hdp=10; end

OrgOut=false;
OrgAdj=false;

%*****************************Preload Variables*******************************
OrgMap=0;
% Set New OrgMap var

if OrgOut

    OrgMapTitles={'x','y','z','Other'};

    OrgMap=[ReactMap, [OrgMapTitles; cell(nsa,length(OrgMapTitles))]];

end

%hdorg stands for "Hex duct origins"
%This will be an array which contains the following array information
%hdorg={SubAssm number,xh,yh,zh}
%xh,yh,zh are the coordinates for the origin of the hex duct.

hdorg=zeros([nsa,4]);

%%
%*****************Calculation of the Origins of the ducts**********************

%This section calculates the center of each of the hexagonal ducts



%The number of rings needs to be determined so that the origin
%functions can be written. Since the number of assemblies per side
%matches that of the number of rings, the equation to determine number
%of assemblies will be rewritten to solve for number of rings.

nr=ceil((3+sqrt(12*nsa-3))/6);

%an Array must be created that lists the number of assemblies in each
%ring.

for ii=1:nr

    %The following calculates the number of elements in each ring plus the
    %total number of elements for the ith ring plus all previous rings

    nar(ii,:)=[ii,6*(ii-1),3*(((ii-1)^2)+ii)-2];

    %This if statement checks to see if it it the first element in the
    %problem.
    if ii==1

        nar(1,:)=[1,1,1];

    end

    %This if statement checks to see if there can only be a partial ring
    %and sets the number of the partial ring.
    if nar(ii,3)>nsa
        nar(ii,:)=[ii,nsa-sum(nar(1:ii-1,2)),nsa];
    end
end


%The change in x and the change in y from one assembly to another is a
%constant, therefore they only need to be calculated once and then just
%added or subtracted.

%Calculation of the x dimension.
%xpt stands for "x pitch" ypt stands for "y pitch" zpt stands for "z pitch"
    xpt=hdp*cosd(60);
    ypt=hdp*sind(60);
    zpt=0;


%Dummy counting variable to keep track of which assembly the code is
%calculating the origin.
    nsatemp=1;


%The following for loop has if statements. The first if statement sets the
%center hex duct at 0,0,0. The second if statement contains a for loop
%which builds the hect duct for each ring.

for ii=1:nr

    %Three more variables are needed as temporary variables to keep track of
```

```
%where the travel has taken place.
    xpttemp=0;
    ypttemp=0;
    zpttemp=0;

%This if statement sets the origin of the first hex duct which is
%located at 0,0,0

if ii==1

    % The first sub assembly is at the origin 0,0,0
    %Assign the sub assm number and origin.
    hdorg(ii,:)=[ii,xpttemp,ypttemp,zpttemp];
    if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end

end


if ii>=2

%Vertex 1
    nsatemp=nsatemp+1;
    xpttemp=(ii-1)*xpt;
    ypttemp=(ii-1)*ypt;
    hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
    if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
    if nsatemp==nsa
        break;
    end

%Vertex 2
    for j=1:ii-1
        nsatemp=nsatemp+1;
        xpttemp=xpttemp+xpt;
        ypttemp=ypttemp-ypt;
        hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
        if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
        if nsatemp==nsa
            break;
        end
    end

    if nsatemp==nsa
        break;
    end

 %Vertex 3
    for j=1:ii-1
        nsatemp=nsatemp+1;
        xpttemp=xpttemp-xpt;
        ypttemp=ypttemp-ypt;
        hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
        if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
        if nsatemp==nsa
          break;
        end
    end

    if nsatemp==nsa
        break;
    end

 %Vertex 4
    for j=1:ii-1
        nsatemp=nsatemp+1;
        xpttemp=xpttemp-2*xpt;
        hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
        if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
        if nsatemp==nsa
            break;
        end
    end

    if nsatemp==nsa
        break;
    end

 %Vertex 5
    for j=1:ii-1
        nsatemp=nsatemp+1;
        xpttemp=xpttemp-xpt;
        ypttemp=ypttemp+ypt;
        hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
        if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
        if nsatemp==nsa
            break;
        end
    end

    if nsatemp==nsa
        break;
    end

 %Vertex 6
    for j=1:ii-1
```

```
            nsatemp=nsatemp+1;
            xpttemp=xpttemp+xpt;
            ypttemp=ypttemp+ypt;
            hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
            if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
            break;
        end

      %Vertex 7
        for j=1:ii-2
            nsatemp=nsatemp+1;
            xpttemp=xpttemp+2*xpt;
            hdorg(nsatemp,:)=[nsatemp,xpttemp,ypttemp,zpttemp];
            if OrgOut; OrgMap(nsatemp+1,6:9)=[{xpttemp},{ypttemp},{zpttemp},{[]}]; end
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
            break;
        end

    end


end

% Performs Special origin calculation

 if OrgAdj; SaOrgXYO( OrgMap ); end

global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotHexOrgDat.mat')
    dispPrint('     TotHexOrgDat saved....');

end

if OrgOut

    xlswrite('OrgMap.xlsx',OrgMap)

end

end
```

B.29. SaOrgXYO.m

```
function [  ] = SaOrgXYO( OrgMap )
% This function takes in the origin of all of the subassemblies and then
% choses the unique types. It then takes the origin of each type and
% calculations the origin of the other subaseemblies of the same type.

global MNumC MSATyC SAPosC SANomen NomenTyC

SATycol=1;
Ncol=2;
EPOScol=3;
Mncol=4;

OriginalSAs={'SATy','Nomen','Ebr POsition', 'Micka Number';...
             1,'Driver','03C01',8;...
             2,'Half Worth','01A01',1;...
             3,'Safety','03D01',10;...
             4,'HWCR','05C03',40;...
             5,'Control','05D01',42;...
             6,'Dummy','02D01',3;...
             7,'Reflector','07C02',93;...
             8,'Blanket','11C01',272;...
             10,'Experiment','04C02',21};

OrigMnum=cell2mat(OriginalSAs(2:end,Mncol));
OrigSATy= cell2mat(OriginalSAs(2:end,SATycol));

% create arrays for the origins

DrOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{2,:},5.88772,10.1978301807394,0,[]};

DrHFWOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{3,:},0,0,0,[]};

SafeOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
```

```
       OriginalSAs{4,:},11.77544,0,0,[]};

HWCROrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{5,:},17.66316,10.1978301807394,0,[]};

ContOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{6,:},23.55088,0,0,[]};

DumOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{7,:},5.88772,0,0,[]};

RefOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{8,:},20.60702,25.4945754518486,0,[]};

BlankOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{9,:},29.4386,50.9891509036972,0,[]};

ExpOrig={'SATy','Nomen','Ebr POsition', 'Micka Number','x','y','z','Other';...
       OriginalSAs{10,:},11.77544,10.1978301807394,0,[]};

EXPnuml=[21,1; 24,6; 38,0; 44,0; 52,0; 67,1; 89,0]; % Mnum col 1 is like SAtype col 2, 0 means unique

NSAOrigAR={DrOrig;DrHFWOrig;SafeOrig;HWCROrig;ContOrig;DumOrig;RefOrig;BlankOrig;[];ExpOrig};
NSAOrig=cell(9,4);

KKl=size(OrgMap,1);

OrgColMap=6:8;
OrgColAr=5:7;

for kk=2:KKl

    curSATy=OrgMap{kk,MSATyC};
    curMnum=OrgMap{kk,MNumC};
    curOrg=OrgMap(kk,OrgColMap);
    curPOS=OrgMap(kk,SAPosC);

    if any(curSATy==OrigSATy)

       curNSAorig=NSAOrigAR{curSATy};
       curSATyorig=cell2mat(curNSAorig(2,OrgColAr));

       if ~any(curMnum==OrigMnum)

           NumcurOrg=cell2mat(curOrg(1:3));

           if curSATy==10

               [EXProw,~]=find(curMnum==EXPnuml(:,1)) ;

               if EXPnuml(EXProw,2)==0

                  curNSAorig(end+1,1:7)=[{curSATy},curNSAorig(2,2),curPOS,{curMnum},num2cell(NumcurOrg)];

                  NSAOrigAR{curSATy}=curNSAorig;

               else

                   curNSAorig2=NSAOrigAR{EXPnuml(EXProw,2)};
                   curSATyorig2=cell2mat(curNSAorig2(2,OrgColAr));

                   NewOrg=round(NumcurOrg-curSATyorig2,4);

                   curNSAorig(end+1,1:7)=[{curSATy},curNSAorig(2,2),curPOS,{curMnum},num2cell(NewOrg)];

                   NSAOrigAR{curSATy}=curNSAorig;

               end

           else

               NewOrg=round(NumcurOrg-curSATyorig,4);

               curNSAorig(end+1,1:7)=[{curSATy},curNSAorig(2,2),curPOS,{curMnum},num2cell(NewOrg)];

               NSAOrigAR{curSATy}=curNSAorig;

           end

       end

      % if curSATy==1; keyboard; end

    end


end

% Plot the new Origins

for ii=1:size(NSAOrigAR,1)

    if ii~=9

        curNSAorig=NSAOrigAR{ii};
```

```
        curName=curNSAorig{2,Ncol};

        curFig=figure('Name',curName);

        curAx=axes(curFig);

        hold(curAx,'on');

        for jj=2:size(curNSAorig,1)

            curOrg=cell2mat(curNSAorig(jj,OrgColAr(1:2)));

            scatter(curAx,curOrg(1),curOrg(2));

        end

        xlswrite([curName '.xlsx'],curNSAorig);

    end

end

keyboard

end
```

B.30. PlaneMaker.m

```
function [ surfc ] = PlaneMaker( NumSlice,DimMap)
%This function creates planes for both the fuel pins and the dividers for
%the katana effect.

global debugMICKA
global KatanaPls
global surfnum
global planesurfheight
global hexsurfnum
global samove
global fbPl
global ebPl
global MKIIvsMKIIA

%BorD = true (Bench) or not (Dissertation).

global BorD

 %% Reactor Map Vars Breakout

global MSATyC

 %% Dim Vars breakout
% This section breaks out all of the variables into seperate variables.

global sazoC saatC sazmC fshC fslasC feozC eshC eslasC deoC

%fgb means fluid gas barrier in pins
%esb means experimental shield barrier

dispPrint('Creating Cutplanes')
dispPrint(' ')

%% Calcuates and sorts unique fluid gas barriers
%First extract sodium levels

%Assembly type
Type=DimMap(2:end,MSATyC);

% Subassembly movements
sazo=DimMap(2:end,sazoC);
saat=DimMap(2:end,saatC);
sazm=DimMap(2:end,sazmC);

% Blanket pin specs
esh=DimMap(2:end,eshC);
eslas=DimMap(2:end,eslasC);
deo=DimMap(2:end,deoC);

%Replaces empty cells with 0

esh(cellfun(@isempty,esh))={0};
eslas(cellfun(@isempty,eslas))={0};
sazm(cellfun(@isempty,sazm))={0};
sazo(cellfun(@isempty,sazo))={0};
saat(cellfun(@isempty,saat))={0};

% Initialize variables

ebarriers=zeros(size(esh,1),1);

dispPrint('Loading Data...  ')

%% Experiement shield barrier
```

```
for kk=1:size(esh,1)

    esht=(esh{kk}+eslas{kk}+deo{kk})-(sazo{kk});

    if isequal(esht,0) || isempty(esht)

        ebarriers(kk,1)=0;
        ebarriers(kk,2)=0;

    else

        ebarriers(kk,1)=Type{kk};
        ebarriers(kk,2)=esht;

        %The blankets have the same sections as the fuel rods.
        ebarriers(kk,3)=esh{kk}/3;
        ebarriers(kk,4)=eslas{kk};

    end

end

%% Find the unique barriers

dispPrint(' ')
dispPrint('Finding Unique Barriers...')

[eUnqlvls,eMICKAnums,~]=unique(ebarriers,'rows','stable');

samove=[eMICKAnums,eUnqlvls];

llcnt=size(samove,1);

dispPrint([num2str(llcnt) ' barriers found.'])

gg=0;

for ll=1:llcnt

    if samove((ll-gg),2)==0 && samove((ll-gg),3)==0

        if ll==1

            samovet1=samove(ll+1:end,:);
            samove=samovet1;

        end

        if ll<llcnt && ll>1

            samovet1=samove(1:(ll-gg)-1,:);
            samovet2=samove((ll-gg)+1:end,:);
            samove=[samovet1;samovet2];

        end

        if ll==llcnt

            samovet1=samove(1:end-1,:);
            samove=samovet1;

        end

        gg=gg+1;

    end

end

%% Total height of sodium in MKIIA/AI fuel elements and blankets


if size(samove,1)>0

    egbh=samove(samove(:,2)==8,3);
    essh=samove(samove(:,2)==8,4);
    neslas=samove(samove(:,2)==8,5);

else

    egbh=0;
    essh=0;
    neslas=0;

end

if isempty(egbh)

    CreateEgbh=false;

else

    CreateEgbh=true;

end
```

```
CreateFgbh=false;

surfname=NameCard(' Fuel Barriers Needed, SAs will create their own.','Divider');

%%  Benchmark Plane creation
if BorD

    % Blanket Sections

    if CreateEgbh

        % Gas barrier

        surfctempb1=CharChecker({surfnum,'    PZ',egbh,...
                        '$ Blanket Na/He Boundary'});

        ebPl{1,1}=surfnum;
        ebPl{1,2}=egbh;

        surfnum=surfnum+1;

        % Section  1/2 barrier

        surfctempb2=CharChecker({surfnum,'    PZ',egbh-neslas-essh,...
                        '$ Blanket Section Plane Sec 1/2'});

        ebPl{2,1}=surfnum;
        ebPl{2,2}=egbh-neslas-essh;

        surfnum=surfnum+1;

        % Section  2/3 barrier

        surfctempb3=CharChecker({surfnum,'    PZ',egbh-neslas-(2*essh),...
                        '$ Blanket Section Plane Sec 2/3'});

        ebPl{3,1}=surfnum;
        ebPl{3,2}=egbh-neslas-(2*essh);

        surfnum=surfnum+1;

    end

    % Append everything together


    if CreateEgbh

        surfctemp=[surfctempb1;surfctempb2;surfctempb3];

    end

else

 %% Dissertation  Plane creation

    % Find the truely unique barriers. There are 2 slices done right off
    % the bat since we section our fuel into 3 sections.

    % Shared planes needs to be turned back on

    CreateFgbh=true;
    CreateEgbh=false;

    %% First create the planes which seperate the slug sections

    % The slug heights need to be acquired.
    % In order for this to work a driver or halfworth drivers needs to be
    % written.

    DrHFWSATy=2;

    SATysC=cell2mat(DimMap(2:end-1,MSATyC));

    [rowSATy,~]=find(SATysC==DrHFWSATy,1,'first');

    fsh=DimMap{rowSATy+1,fshC};
    fslas=DimMap{rowSATy+1,fslasC};

    fgbh=fslas+fsh;

    fshSec=fsh/3;

    if CreateFgbh

        % Gas barrier

        surfctempf1=CharChecker({surfnum,'    PZ',fgbh,...
                        '$ Driver Na/He Boundary'});

        fbPl{1,1}=surfnum;
        fbPl{1,2}=fgbh;

        surfnum=surfnum+1;
```

```
    % Fuel slug sections

    % With the gas barrier found and assigned, the two planes for the fuel
    % sections need to be calculated and added. This is done by taking the
    % element length, dividing by 3 and then subtracting 1 section length
    % sodium level above fuel. Two section length will give the other.

    % Section  1/2 barrier

     surfctempf2=CharChecker({surfnum,'    PZ',fgbh-fslas-fshSec,...
                    '$ Fuel Section Plane Sec 1/2'});

    fbPl{2,1}=surfnum;
    fbPl{2,2}=fgbh-fslas-fshSec;

    surfnum=surfnum+1;

    % Section  2/3 barrier

     surfctempf3=CharChecker({surfnum,'    PZ',fgbh-fslas-(2*fshSec),...
                    '$ Fuel Section Plane Sec 2/3'});

    fbPl{3,1}=surfnum;
    fbPl{3,2}=fgbh-fslas-(2*fshSec);

    surfnum=surfnum+1;

end

% Blanket Sections

if CreateEgbh

    % Gas barrier

    surfctempb1=CharChecker({surfnum,'    PZ',egbh,...
                    '$ Blanket Na/He Boundary'});

    ebPl{1,1}=surfnum;
    ebPl{1,2}=egbh;

    surfnum=surfnum+1;

    % Section  1/2 barrier

    surfctempb2=CharChecker({surfnum,'    PZ',egbh-neslas-essh,...
                    '$ Blanket Section Plane Sec 1/2'});

    ebPl{2,1}=surfnum;
    ebPl{2,2}=egbh-neslas-essh;

    surfnum=surfnum+1;

    % Section  2/3 barrier

    surfctempb3=CharChecker({surfnum,'    PZ',egbh-neslas-(2*essh),...
                    '$ Blanket Section Plane Sec 2/3'});

    ebPl{3,1}=surfnum;
    ebPl{3,2}=egbh-neslas-(2*essh);

    surfnum=surfnum+1;

end

if  CreateFgbh

    surfctemp=[surfctempf1;surfctempf2;surfctempf3];

end

if CreateEgbh

    surfctempb=[surfctempb1;surfctempb2;surfctempb3];
    surfctemp=[surfctemp; surfctempb];

end


%% Additonal Katana Slices

KatanaPls=[{'Gas Barrier';'Sec 12';'Sec 23'},fbPl];

KPlsS=size(KatanaPls,1);

if NumSlice>4

    % ***NEED TO WRITE THIS CODE***
    % When additional planes are created and added, they need to be in
    % decending order otherwise ANSYSloader will throw an error
    keyboard

    for ii=1:size(KatanaPls,1)
```

```
            %This is the first surface written, so planesurfnum has
            %already been intialized to 1 in main.m

            surfctemp(ii,:)=CharChecker({surfnum,'    PZ',KatanaPls(ii,3),...
                strjoin({'$ MICKA plane #',Num2StrM(ii)})});
            surfnum=surfnum+1;

            KatanaPls(ii,4)=surfnum;

        end

    end

    % The comment card for the cut plane name

    surfname=NameCard('MICKA Slices Planes','Title');

    %This for loop creates the cut plane surfaces


end

surfc=[surfname; surfctemp];

global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotPlaneDat.mat')
    dispPrint('     TotPlaneDat saved....');

end

end
```

B.31. NameCard.m

```
function [ outstr ] = NameCard( instr,isTitle )
%This function takes in a string title and outputs it into an even MCNP
%format.

global comout

%The input arguement are either 'Title' for title or 'Divider' for divider

if strcmp(comout,'Y')

    astlen=ceil((75-numel(instr))/2);

    surfnametemp=strjoin({repmat('*',1,astlen), instr ,repmat('*',1,astlen)});

    if numel(surfnametemp)>77

        surfname=surfnametemp(1:end-1);
        aststr=repmat('*',1,numel(surfname));

    else

        surfname=surfnametemp;
        aststr=repmat('*',1,numel(surfname));

    end

    if strcmp(isTitle,'Title')

        strtemp1={' ';aststr; surfname;aststr; ' '};

        c(1:5,1)={'c'};

        outstr=[c, strtemp1];

    end

    if strcmp(isTitle,'Divider')

        strtemp1={' '; surfname; ' '};

        c(1:3,1)={'c'};

        outstr=[c, strtemp1];

    end

else

        outstr=[];

end

end
```

B.32. CharChecker.m

```
function [ cardo ] = CharChecker( cardi )
%This program checks all of the lines of the cards and checks to see if
%they go over 80 chars. If they do, the line is split and put onto the next
%line. The program also changes all of the input cards into long strings.

global comout
global SmallValCutOff
global CharLen

cs=size(cardi);

% These two nested for loops first changes all but the first column to a
% string and then creates a new array which has two columns, the surface
% number or cell number then the subsequent card information.

for ii=1:cs(1,1)

    for jj=2:cs(1,2)

        if any(abs(cardi{ii,jj})<1E-10) && SmallValCutOff
            cardi{ii,jj}=0;
        end

       cardi{ii,jj}=Num2StrM(cardi{ii,jj},'%.8g');

    end

    carditemp(ii,:)=[cardi(ii,1), strjoin(cardi(ii,2:end))];

end

for ii=1:cs(1,1)

%spcind is a variable that contains the location of all of the spaces
%in the card

    spcindtemp=cell2mat(strfind(carditemp(ii,2), ' '));
    comind=cell2mat(strfind(carditemp(ii,2), '$'));
    comlen=0;
    spcind=spcindtemp;

    if strcmp(comout,'N') && ~isempty(comind)

         cardtemp=carditemp{ii,2};
         carditemp{ii,2}=cardtemp(1:comind-1);

    end

    %This for loops finds the comment and seperates it from the card

    for ll=1:length(spcindtemp)

        if spcindtemp(ll)>comind

            com=carditemp{ii,2};
            spcind=spcindtemp(1:ll-1);
            comlen=numel(com(comind:end));

            break
        end

    end

    kk=1;
    begrow=0;

    if sum(cellfun(@numel,carditemp(ii,:)))<=(CharLen+comlen)
        cardotemp=carditemp(ii,:);
    end

    for jj=1:length(spcind)

        if kk==1

            if spcind(jj)-begrow>=(CharLen-length(Num2StrM(carditemp{ii,1},'%.8g')))
                kk=kk+1;
                strtemp=carditemp{ii,2};
                strtemp1=strtemp(1:spcind(jj-1));
                begrow=spcind(jj-1);
                cardotemp={carditemp{ii,1}, strtemp1};

            end

        end

        if kk>1
            if spcind(jj)-begrow>=74


                strtemp3={'    ',strtemp(begrow:spcind(jj-1))};
                begrow=spcind(jj-1);

                cardotemp=[cardotemp; strtemp3];

            end
```

```
            if jj==length(spcind)

                strtemp3={'    ',strtemp(begrow:end)};
                cardotemp=[cardotemp; strtemp3];

            end
        end



    end

    NumCardL=length(Num2StrM(cardotemp{end,1}));
    NCardL=NumCardL+numel(cardotemp{end,2});

    if NCardL>80  &&  ~strcmp(cardotemp{1,1},'c')

        if isempty(cell2mat(strfind(cardotemp(end,2), '$')));

            dispPrint(' ')
            dispPrint('Current Card: ');
            dispPrint(cardotemp{1,2});
            dispPrint('Card Length >80 chars, toubleshoot.');
            keyboard;

        end

    end

    % This moves the comment card to position 79

    NComInd=strfind(cardotemp{1,2}, '$');

    if ~isempty(NComInd)

       CardStrT=cardotemp{1,2};

       CardStr=CardStrT(1,1:NComInd-1);

       ComStr=CardStrT(1,NComInd:end);

       FillStr=repmat(' ',1,CharLen-NumCardL-length(CardStr)-1);

       NCard=[CardStr,FillStr,ComStr];

       cardotemp{1,2}=NCard;

    end

    if ii==1

        cardo=cardotemp;

    end

    if ii>1

        cardo=[cardo; cardotemp];

    end

end

if ~exist('cardo','var') ; cardo=[]; end

end
```

B.33. ThermalXHex.m

```
function [ hdsorg ] = ThermalXHex( hdh,hdorg,nsa )
%This program takes in the hex duct origin and combines the origin with
%with the plane section heights and produces a new array of hex duct
%section origins. This will be used eventually to determine the bend of the
%duct.

global BorD
global KatanaPls
global NumSlice
global UseKatanaExDat
global KatanaExagPlot

persistent FirstRun
persistent SliceOrg

% First Run check

if isempty(FirstRun)

    LoadANSYSDat=true;

else
```

```
    LoadANSYSDat=false;

end

% To make sure there is always an output

hdsorg=0;

%% Benchmark code

if BorD

    % All this code does is reformat hdorg so that it can be used by the
    % maker.

    % Need a switch to prevent an error with katanas use of react map.

    if length(hdh)>1; hdh=0; end

    hdsorg=[hdh ,hdorg(2),hdorg(3), 0];

else if ~LoadANSYSDat

        %% This is the dissertation code.
        % the way it will work is by taking in an unadjusted origin of a particular
        % slice, then comparing that to the generated list of movements as
        % calculated by ansys and then read in. It will do a straight comparison,
        % meaning in will taking in a number of the format {MICKA num}{Slice Number}
        % e.g 400001 would be the first slice from the bottom. It will take this
        % number in compare it to the list and find the same slice number has been
        % moved by +x,+y and then given back to the maker.

        Mnum=hdorg(1);

        % Find the assembly in SliceOrg

        RowInd=cellfind(SliceOrg(:,1),Mnum,1,'Match');

        % Clear hdsorg

        clear hdsorg

        if RowInd==0;

            disp([Num2StrM(hdorg(1)) ' SA does not have Def data!'])

            for dd=1:size(SliceOrg(1,3:end),2)

                hdsorg(dd,:)=[dd,hdorg(2:end)];

            end

        else

            for dd=1:size(SliceOrg(RowInd,3:end),2)

                hdsorg(dd,:)=round([dd,SliceOrg{RowInd,dd+2}],5);

            end

        end

    end

end

if LoadANSYSDat && ~BorD

    % The import code does not exist yet, nor a seperate function. For now
    % we are going to generate a dummy variable

    % For loading, hdsorg needs to be assigned to comehting otherwise an
    % error is thrown.
    hdsorg=0;

    SliceOrg=cell(nsa+1,NumSlice+2);

    % Fill the variable with the origins

    for hh=2:size(SliceOrg,1)

        if hh==2;

           SliceOrg{1,1}='Micka Number';
           SliceOrg{1,2}='SA Org';

           for gg=1:size(SliceOrg,2)-2;

               SliceOrg{1,gg+2}=['Slice ' Num2StrM(gg)];

           end

        end

        SliceOrg{hh,1}=hdorg(hh-1,1);
```

```
        SliceOrg{hh,2}=hdorg(hh-1,2:end);

    end

    %% This is where the program diverges into ANSYS loading versus example data loading.

    ReactMap=hdh;

    % ANSYS data

    [ SliceOrg ] = ANSYSLoader( SliceOrg, KatanaPls, ReactMap );

    %% The ANSYSLoader needs to be fixed for the new format of Katanapls

    % An updated hdorg needs to be created to place the assemblies in
    % their new origins.

    SliceOrgT=SliceOrg(:,[1:2,end]);

    for rr=2:size(SliceOrg,1)

       curorg=SliceOrgT{rr,end};
       curMnum=SliceOrgT{rr,1};

       [hdorgR,~]=find(hdorg(:,1)==curMnum);

       if ~isempty(hdorgR); hdorg(hdorgR,2:end)=curorg; end

    end

end

FirstRun=false;

end
```

B.34. ANSYSLoader.m

```
function [ SliceOrg ] = ANSYSLoader( SliceOrg, KatanaPls, ReactMap )
% This funnction reads in the ANSYS data

%% Create file IDs for the files required.

% Path to the ANSYS data folder in MICKA

AnsysPath='Main\Katana\ANSYS\ANSYS Data';
AnsysPathNAS='R:\ANSYS\Thesis\EBRII\Input Files';
AnsysResPath='R:\ANSYS\Thesis\EBRII\Def Results';

%% Check for Old File

if exist([AnsysPath '\StructInp.mat'], 'file') == 2

    load([AnsysPath '\StructInp.mat']);

else

    % Open the 'Input' file

    [InputfileID, InpErrMSG]=fopen([AnsysPathNAS '\Struct.inp']);

    % The way this code works in by grabbing the node numbers associated with
    % which assembly, then adding the deformed origins.

    %% Create the catches to find the appropriate lines

    NameSelectFindSTR='CMBLOCK,C_';
    NameSelectFindSTR2='FACES';
    NSLen=length(NameSelectFindSTR);

    LnCnt=0;
    Cnt1E5=0;

    NodeNums=cell(1,2);
    NameSelectNums=0;

    FirstSubWrite=true;
    FirstWrite=true;
    NormRead=true;

    QuitLp=false;

    %% Read in all the node numbers

    while ~feof(InputfileID) && ~QuitLp

        if NormRead

            curLn=fgetl(InputfileID);

        end

        NormRead=true;
```

```
StrFInd=strfind(curLn,NameSelectFindSTR);
StrFInd2=strfind(curLn,NameSelectFindSTR2);

if ~isempty(StrFInd) && isempty(StrFInd2)

   % Grab the EBRpos

   curPOS=curLn(NSLen+1:NSLen+5);

   % Turn off outer while loop reading

   NormRead=false;

   % Grab two lines to get the node numbers

   fgetl(InputfileID);
   curLn=fgetl(InputfileID);

   if ~isempty(curLn)

       QuitWhLp=false;

       while ~QuitWhLp

           if FirstSubWrite

               curNodeLnNum={curLn};
               FirstSubWrite=false;

           else

              curNodeLnNum=[curNodeLnNum; {curLn}];

           end

            curLn=fgetl(InputfileID);

            LnCnt=LnCnt+1;

            if strcmp(curLn,'/golist')

                QuitLp=true;

            end

            if isempty(curLn) QuitWhLp=true; curLn=[0,0,0]; end

            if ~strcmp(curLn(1:3),'   ') QuitWhLp=true; end

       end

    end

   if FirstWrite

       NodeNums{1,1}=curPOS;
       NodeNums{1,2}=curNodeLnNum;
       FirstWrite=false;

       NameSelectNums=NameSelectNums+1;

   else

       NodeNums{end+1,1}=curPOS;
       NodeNums{end,2}=curNodeLnNum;

       NameSelectNums=NameSelectNums+1;

   end

   clc
   disp([Num2StrM(Cnt1E5) 'e5 Lines Read']);
   disp([Num2StrM(NameSelectNums) ' Assemblies Found']);

   FirstSubWrite=true;

   clear StrFInd
   clear curNodeLnNum

end

%% Internal Loop counter and display.

LnCnt=LnCnt+1;

if  mod(LnCnt,100000)==0

   Cnt1E5=Cnt1E5+1;

   clc
   disp([Num2StrM(Cnt1E5) 'e5 Lines Read']);
   disp([Num2StrM(NameSelectNums) ' Assemblies Found']);
```

```
    end

    if strcmp(curLn,'/golist')

        QuitLp=true;

    end

end


%% Match positions to MICKA numbers

EBRMnum=cell(size(NodeNums,1),1);

for cc=1:size(NodeNums,1)

   curPOS=NodeNums{cc,1};

   MnumRow=cellfind(ReactMap(:,3),curPOS,1,'Match');

   EBRPOS(cc,1)=ReactMap(MnumRow,1);

end

NodeNums=[EBRPOS,NodeNums];

%% process the node numbers

% convert the cell arrays into double arrays

for gg=1:size(NodeNums,1)

   clc
   disp([Num2StrM(Cnt1E5) 'e5 Lines Read']);
   disp([Num2StrM(NameSelectNums) ' Assemblies Found']);
   disp([Num2StrM(gg) ' Assemblies Processed']);

   curNodeNums=NodeNums{gg,3};

   % Number of columns

   curNodeRows=size(curNodeNums,1);

   for jj=1:curNodeRows

       % Current row

       curRow=curNodeNums{jj,1};

       % Need to replace the spaces with an asterixs

       for rr=7:-1:1

           curRow=strrep(curRow,repmat(' ',1,rr),'_');

       end

       curSpcInds=strfind(curRow,'_');

       curSpcIndsLen=length(curSpcInds);

       for uu=1:curSpcIndsLen

          if uu==curSpcIndsLen

              Num=str2double(curRow(curSpcInds(uu)+1:end));

              curNodeDoubleAr(jj,uu)=Num;

              if isnan(Num)

                  disp('NaN found, reading error')
                  keyboard;

              end

          else

             Num=str2double(curRow(curSpcInds(uu)+1:curSpcInds(uu+1)-1));

             curNodeDoubleAr(jj,uu)=Num;

             if isnan(Num)

                 disp('NaN found, reading error')
                 keyboard;

             end

          end

          clear Num

       end
```

```
        end

        % Place it into a one column format

        for ee=1:size(curNodeDoubleAr,2)

            if ee==1

               cur1ColAr=curNodeDoubleAr(:,1);

            else

                cur1ColAr=[cur1ColAr;curNodeDoubleAr(:,ee)];

            end

        end

        % Make sure no zero nodes are written

        NodeNums{gg,3}=cur1ColAr(cur1ColAr~=0);

        clear curNodeDoubleAr
        clear cur1ColAr

    end

    % Create a master linear array

    for vv=1:size(NodeNums,1)

        curNums=NodeNums{vv,3};

        if vv==1

            MasterAr=curNums;

        else

            MasterAr=[MasterAr; curNums];

        end

    end

end

% Line to save a new struct input file.
save([AnsysPath '\StructInp.mat']);

%% Select the deflection file to use

[ filenameX, filepathX ] = FileSelect( AnsysResPath );
[ filenameY, filepathY ] = FileSelect( AnsysResPath );

% Grab the time

UndInd=strfind(filenameX,'_');

ResTime=filenameX(1:UndInd-1);

if exist(['Main\Katana\ANSYS\ANSYS Data\' ResTime '_SliceOrg.mat'], 'file') == 2

        load(['Main\Katana\ANSYS\ANSYS Data\' ResTime '_SliceOrg.mat'])

else

    %% Open the HD deform files

    [DefXInputFileID, DefXInpErrMSG]=fopen([filepathX '\' filenameX]);
    [DefYInputFileID, DefYInpErrMSG]=fopen([filepathY '\' filenameY]);

    %% Read in Deflections

    fgetl(DefXInputFileID);
    fgetl(DefYInputFileID);

    disp('Reading X Deformations')

    DefXin=cell2mat(textscan(DefXInputFileID,'%f64 %f64 %f64 %f64 %f64','Delimiter','\t'));

    disp('Reading Y Deformations')

    DefYin=cell2mat(textscan(DefYInputFileID,'%f64 %f64 %f64 %f64 %f64','Delimiter','\t'));

    %% Apply deflections to nodes

    % The array will be structured [Node Num, Non Def Org, Def Org] NumNodes x 7

    OldMasterAr=MasterAr;

    % Reformat Master Ar
```

```
MasterAr=[OldMasterAr,zeros(size(OldMasterAr,1),6)];

% Create sparse matrices

XDefSpar=sparse(DefXin(:,1),1,1:length(DefXin(:,1)));
YDefSpar=sparse(DefYin(:,1),1,1:length(DefYin(:,1)));

for ss=1:size(MasterAr,1)

    if  mod(ss,50000)==0

        clc
        disp([Num2StrM(Cnt1E5) 'e5 Lines Read']);
        disp([Num2StrM(NameSelectNums) ' Assemblies Found']);
        disp([Num2StrM(gg) ' Assemblies Processed']);
        disp('Reading X Deformations')
        disp('Reading Y Deformations')
        disp([Num2StrM(ss) ' of ' Num2StrM(size(MasterAr,1)) ' Deflections Applied']);

    end

   curnode=MasterAr(ss,1);

   %% X Deform

   XDefinrow=XDefSpar(curnode);

   curXdef=DefXin(XDefinrow,5);

   MasterAr(ss,2:4)=DefXin(XDefinrow,2:4);

   curXorg=DefXin(XDefinrow,2);

   MasterAr(ss,5)=curXdef+curXorg;

   %% Y Deform

   YDefinrow=YDefSpar(curnode);

   curYdef=DefYin(YDefinrow,5);

   curYorg=DefYin(YDefinrow,3);

   MasterAr(ss,6)=curYdef+curYorg;

   %% Z Deform

   curZorg=DefYin(YDefinrow,4);

   MasterAr(ss,7)=curZorg;

end

MastSpar=sparse(MasterAr(:,1),1,1:length(MasterAr(:,1)));

%% Attach the nodes deforms to the sub assemblies

% Create a sparse array of the master list

for qq=1:size(NodeNums,1)

    CurSA=NodeNums{qq,3};

    NCurSA=[CurSA, zeros(length(CurSA),6)];

    for jj=1:length(CurSA)

        curNode=CurSA(jj);

        MastArRow=MastSpar(curNode);

        NCurSA(jj,2:end)=MasterAr(MastArRow,2:end);

    end

    NodeNums{qq,3}=NCurSA;

end

%% Using the katana planes and , calculate the average shift of each SA section

KatLvls=KatanaPls(1:end,3)';
NumCut=size(KatLvls,2);
NumSlc=size(KatanaPls,1)+1;

% Create a new variable which lists the displacement distance not origin
% this can be used later for error checking

SliceDef=SliceOrg;

for qq=1:NumSlc

   %% Set the cut Levels
   % A Correctio needs to be made to the planes since the origin of the
   % mcnp geometry is different than ANSYS.
```

```
        LEHHDWT=61.455;

        if qq==1 % Highest Section

            HiPl=10000; % Upper Extension and everything down to the first section
            LoPl=KatLvls{1,qq}+LEHHDWT;

        end

        if qq>1 && qq<NumSlc % Middle Sections

            HiPl=KatLvls{1,qq-1}+LEHHDWT;
            LoPl=KatLvls{1,qq}+LEHHDWT;

        end

        if qq==NumSlc % Lowest Section

            HiPl=KatLvls{1,qq-1}+LEHHDWT;
            LoPl=-10000;

        end


        %% Read through each assembly that has node numbers

        for rr=1:size(NodeNums,1)

            curPOS=NodeNums{rr,2};

            curNodeNums=NodeNums{rr,3};

            curMnum=NodeNums{rr,1};

            SOrgRow=cellfind(SliceOrg(:,1), curMnum,1,'Match');

            curSAOrg=SliceOrg{SOrgRow,2};

            %% Find the the ndoes that belong between two levels

            curNodeSecs=curNodeNums(curNodeNums(:,4)<=HiPl & curNodeNums(:,4)>=LoPl,:);

            % Two notes, first columns 2 and 3 are the original position
            %             5 and 6 are the deformed postions
            %             It should be Deformed - Original to get def
            %             The leads to Original + def to get deformed

            curAvgXDef=mean(curNodeSecs(:,5)-curNodeSecs(:,2));
            curAvgYDef=mean(curNodeSecs(:,6)-curNodeSecs(:,3));

            % Uncertainty

            NodeMax=max(curNodeSecs(:,2)-curNodeSecs(:,5));
            NodeMin=min(curNodeSecs(:,2)-curNodeSecs(:,5));
            NodeRang=NodeMax-NodeMin;

            AvgUnc=NodeRang/(2*sqrt(size(curNodeSecs,1)));

            NewOrg=[curSAOrg(1)+curAvgXDef,curSAOrg(2)+curAvgYDef, curSAOrg(3)];
            NewDef=[curAvgXDef,curAvgYDef, 0];

            SliceOrg{SOrgRow,qq+2}=NewOrg;
            SliceDef{SOrgRow,qq+2}=NewDef;

        end


    end

    % Trim SliceOrg

    SliceOrgFull=SliceOrg;

    SliceOrg=SliceOrgFull(1:SOrgRow,:);

    %% Save Slice Org

    nameVar=[ResTime '_SliceOrg.mat'];

    save(['Main\Katana\ANSYS\ANSYS Data\' nameVar],'SliceOrg');

end

plotSA=false;


%% As a self check we are going to plot 1 SA

if plotSA

    OrgSAax=subplot(1,2,1);
    DefSAax=subplot(1,2,2);

    ExagFact=3;
```

```
    ExagFact2=ExagFact*10;

    scatter3(OrgSAax,NCurSA(:,2)*ExagFact,NCurSA(:,3)*ExagFact,NCurSA(:,4))
    scatter3(DefSAax,((NCurSA(:,2)*ExagFact)-NCurSA(:,5))*ExagFact2+(NCurSA(:,2)*ExagFact),...
                     ((NCurSA(:,3)*ExagFact)-NCurSA(:,6))*ExagFact2+(NCurSA(:,3)*ExagFact),...
                     NCurSA(:,7))

    axis(OrgSAax,'equal')
    axis(DefSAax,'equal')


    view(OrgSAax,[-90,0])
    view(DefSAax,[-90,0])

end


end
```

B.35. GarbageCollect.m

```
function [ MatMap , MatChngMap ] = GarbageCollect(MatMap)

global MNumC MSATyC NomenTyC FmC SmearSATYs

% This function removes the duplicate material numbers

dispPrint('Peforming Cleanup')

%Grab the generic material line

GenMat=MatMap(end,:);

%Measure the length of all the materials

gmdat=zeros(1,size(GenMat,2));
MatChngMap=[MatMap(:,MNumC:NomenTyC), cell(size(MatMap,1),size(MatMap(1,FmC:end),2))];

%% Columns not to check will be the blanket

for kk=6:3:size(GenMat,2)

    temp1=GenMat{1,kk+1};
    temp2=size(GenMat{1,kk+2},2);

    if isempty(temp1); temp1=0; end

    gmdat(1,kk)=temp1+temp2;

end

dispPrint(' ')

for mm=2:size(MatMap,1)-1

    mdattemp=zeros(1,size(MatMap,2));

    SATy=MatMap{mm,MSATyC};
    Mnum=MatMap{mm,MNumC};

    if isempty(SmearSATYs(Mnum==SmearSATYs(1:end,1)))

        for ii=9:3:size(MatMap,2)

            temp1=MatMap{mm,ii+1};
            temp2=size(MatMap{mm,ii+2},2);

            if isempty(temp1); temp1=0; end

            mdattemp(1,ii)=temp1+temp2;

        end

        [~,mdai,gmi]=...
            intersect(mdattemp,gmdat,'sorted');

        %Change the MatMap material numbers

        MatMap(mm,mdai(2:end))=GenMat(1,gmi(2:end));
        MatMap(mm,mdai(2:end)+2)={[]};
        MatChngMap(mm,mdai(2:end))=MatMap(1,mdai(2:end));

    end

    dispPrint(['First Collection Pass...';{[mm,size(MatMap,1)-1]}]);

end

dispPrint(' ')

for mm=2:size(MatMap,1)-1

    mdattemp=zeros(1,size(MatMap,2));
```

```
    SATy=MatMap{mm,MSATyC};
    Mnum=MatMap{mm,MNumC};

    if isempty(SmearSATYs(Mnum==SmearSATYs(1:end,1)))

        for ii=9:3:size(MatMap,2)

            temp1=MatMap{mm,ii+1};
            temp2=size(MatMap{mm,ii+2},2);

            if isempty(temp1); temp1=0; end

            mdattemp(1,ii)=temp1+temp2;

        end

        [~,mdai,gmi]=...
            intersect(mdattemp,gmdat,'sorted');

        %Change the MatMap material numbers

        MatMap(mm,mdai(2:end))=GenMat(1,gmi(2:end));
        MatMap(mm,mdai(2:end)+2)={[]};
        MatChngMap(mm,mdai(2:end))=MatMap(1,mdai(2:end));

    end

    dispPrint(['Second Collection Pass...';{[mm,size(MatMap,1)-1]}]);

end

dispPrint(' ')

for mm=2:size(MatMap,1)-1

    mdattemp=zeros(1,size(MatMap,2));

    SATy=MatMap{mm,MSATyC};
    Mnum=MatMap{mm,MNumC};

    if isempty(SmearSATYs(Mnum==SmearSATYs(1:end,1)))

        for ii=9:3:size(MatMap,2)

            temp1=MatMap{mm,ii+1};
            temp2=size(MatMap{mm,ii+2},2);

            if isempty(temp1); temp1=0; end

            mdattemp(1,ii)=temp1+temp2;

        end

        [~,mdai,gmi]=...
            intersect(mdattemp,gmdat,'sorted');

        %Change the MatMap material numbers

        MatMap(mm,mdai(2:end))=GenMat(1,gmi(2:end));
        MatMap(mm,mdai(2:end)+2)={[]};
        MatChngMap(mm,mdai(2:end))=MatMap(1,mdai(2:end));

    end

    dispPrint(['Third Collection Pass...';{[mm,size(MatMap,1)-1]}]);

end

dispPrint(' ')

global SaveDebugMatVars

if SaveDebugMatVars

    dispPrint('Saving MatLab Variable MatChngMap');
    dispPrint(' ')

    save('Maps\MatMap.mat','MatChngMap')

    dispPrint(' ')

    dispPrint('Saving MatLab New Material Map');
    dispPrint(' ')

    save('Maps\NewMatMap.mat','MatMap')

end


global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
```

```
    save('Debug\TotGarbageColDat.mat')
    dispPrint('     TotGarbageColDat saved....');

end

end
```

B.36. DataMaker.m

```
function [sourcenum , datac ] = DataMaker(MCNPpara,hdorg,ReactMap,MatMap,DimMap)
%This function creates the data cards.
%Since the material cards are extensive, a seperate function was needed
%just to keep the main.m clean looking. It takes one input arguement and
%exports the data cards as a large array.

global planesurfnum
global BorD
global SAlphaBetas
global SABcorr
global DelTemp
global FindOldLib
global CtoKconv
global RoomTemp
global SmearSATYs
persistent firstrun
persistent SATYsab

%% Column Globals

global MNumC MSATyC SAPosC SANomen NomenTyC crhC fshC fepC

%% Other Vars

% This variable disables the smaller variable cutoff, the limit is
% nominally 1e-10 but the material cards contain concentrations smaller
% then that.

global SmallValCutOff

%BorD = true (Bench) or not (Dissertation).

datac1=CharChecker({'KCODE     ',...
    MCNPpara{2},MCNPpara{3},MCNPpara{4},MCNPpara{5}});

% This writes the source
sourcenum=0;

datac2temp={'KSRC     '};

%% s(a,b)

if isempty(firstrun) && SABcorr;

    % s(a,b) additions

    SATYsab=cell2mat(SAlphaBetas(:,2));

end

sabon=false;

%%

% Write the source using only drivers and halfworth drivers

crhDr=DimMap{2,crhC};
fshDr=DimMap{2,fshC};
fs1hDr=(fshDr/3);
fshpos=RoundM([fs1hDr/2,fs1hDr+fs1hDr/2,2*fs1hDr+fs1hDr/2],3,'significant');

for ii=1:length(hdorg(:,1))

    hdsorg=ThermalXHex(crhDr,hdorg(ii,:),0);
    SATy=DimMap{ii+1,MSATyC};
    fep=DimMap{ii+1,fepC};
    xfep=fep*cosd(30);
    yfep=fep*sind(30);

    if SATy==1;

        sourcenum=sourcenum+1;

        for hh=1:size(hdsorg,1)-1

            datac2temp=[datac2temp, {round(hdsorg(hh,2),3),round(hdsorg(hh,3),3),hdorg(hh,4)+fshpos(hh)}];

        end

    end

    % The half worth drivers need to have the source moved into a fueled
    % pin and not a dummy pin. That is what the feps are for.
```

```
    if SATy==2;

        sourcenum=sourcenum+1;

        for gg=1:size(hdsorg,1)-1

            datac2temp=[datac2temp, {round(hdsorg(gg,2)+xfep,3),round(hdsorg(gg,3)+yfep,3),hdsorg(gg,4)+fshpos(gg)}];

        end

    end

    if size(datac2temp,2)>280; break; end

end

%The following statement is only used incase no drivers are written.

try

    datac2temp{2};

catch

    datac2temp=[datac2temp 0 0 15];

end

datac2=CharChecker(datac2temp);

%% Material Loader

dispPrintmat={'Fuel/Blanket Slug';...
              'Fuel Sodium';...
              'Fuel Plenum Gas';...
              'Fuel Cladding';...
              'Poison Slug';...
              'Fuel Pin Wirewrap';...
              'Poison Plenum Gas';...
              'Poison Cladding';...
              'Dummy Pin';...
              'Hex Duct';...
              'Duct Sodium';...
              'Smeared Upper Ext';...
              'Smeared Lower Ext';...
              'Smeared Lower Adapter';...
              'Smeared Inner Hext Duct Lower Adapter';...
              'Other Pins SS';...
              'Pin Shield';...
              'Spare';...
              'Spare'};


dispPrintgenmat={'Sodium';...
                 'Argon';...
                 'Helium';...
                 'Plenum Gas';...
                 'SS304';...
                 'B4C Wire Wrap';...
                 'SS316';...
                 'Fuel Pin Wirewrap';...
                 'BOL B4C';...
                 'Spare';...
                 'Smeared Drivers Upper Ext';...
                 'Smeared Lower Ext';...
                 'Smeared Lower Adapter';
                 'Smeared Inner Hext Duct Lower Adapter';...
                 'Smeared Safety Upper Ext';...
                 'Smeared B4C Pin Shield';...
                 'Stainless Steel 304 Reflector Smear';...
                 'Stainless Steel 316 Reflector Smear';...
                 'Highly Enriched Uranium'};...

% The small value cut off will now be disabled while the materials are
% being written.

%Write the generic mat Cards first

SmallValCutOff=false;

for jj=6:3:size(MatMap(1,:),2)

    gendatatitletemp1=NameCard('Generic Materials','Divider');
    gendatatitletemp2=NameCard(dispPrintgenmat{(jj/3)-1},'Divider');
    gendatactemp=CharChecker(MICKAtoMCNPmat(MatMap{end,jj},MatMap{end,jj+2}));

    if any(30==SATYsab) && SABcorr;

        SATy=30;
        [ sabc ] = SAlphaBeta( MatMap{end,jj}, MatMap{end,jj+2}, SATy);

    else sabc=0;
    end

    if ~iscell(sabc);
```

```
        if jj==6

            gendatac3temp=[gendatatitletemp1; gendatatitletemp2; gendatactemp];

        else

            gendatac3temp=[gendatac3temp; gendatatitletemp2; gendatactemp];

        end

    else

        if jj==6

            gendatac3temp=[gendatatitletemp1; gendatatitletemp2; gendatactemp; sabc];

        else

            gendatac3temp=[gendatac3temp; gendatatitletemp2; gendatactemp; sabc];

        end

    end

end

%Write actual materials

for ii=2:size(MatMap,MNumC)-1

    SATy=MatMap{ii,MSATyC};
    Mnum=MatMap{ii,MNumC};

    % The following statement adds sab corrections to the xsec

    if any(SATy==SATYsab) && SABcorr; sabon=true; else sabc=0; end

    % This condition matchs to see if the subassembly is not a smeared one.
    % Smeared SAs require a new data card to be created inside of the
    % maker, subsequently the following lines are not written.

    if ~any(Mnum==SmearSATYs(1:end,1)) ||  SATy==0

        %Sa title and Region

        datatitletemp1=NameCard(['MICKA #: ' num2str(MatMap{ii,MNumC}) ' ID: ' ...
            MatMap{ii,SANomen} ' POS: ' MatMap{ii,SAPosC}],'Title');

        for kk=6:3:size(MatMap,2)

            if isempty(MatMap{ii,kk})

                datactemp=NameCard(['No ' dispPrintmat{(kk/3)-1}],'Divider');

                if kk==6

                    datac3temp=datactemp;

                else

                    datac3temp=[datac3temp; datactemp];

                end

            else

                MatNum=MatMap{ii,kk};
                MatDb=MatMap{ii,kk+2};

               % This part will be for when multiple sections are ready

                if size(MatDb,1)>2

                    datatitletemp2=NameCard(dispPrintmat{(kk/3)-1},'Divider');

                    for hh=1:length(MatNum)

                        datatitletemp3=NameCard(['Pin Sec: ' num2str(hh)],'Divider');

                        datactemp=CharChecker(MICKAtoMCNPmat(MatNum(hh),...
                            [MatDb(1,:); MatDb(hh+1,:)]));

                         if sabon; [ sabc ] = SAlphaBeta( MatNum(hh), MatDb, SATy); end

                         if kk==6 && hh==1

                            datac3temp=[datatitletemp2; datatitletemp3; datactemp];

                         else

                            datac3temp=[datac3temp; datatitletemp3; datactemp];

                         end
```

```
                    end

                else

                    datatitletemp2=NameCard(dispPrintmat{(kk/3)-1},'Divider');
                    datactemp=CharChecker(MICKAtoMCNPmat(MatNum,MatDb));

                     if sabon; [ sabc ] = SAlphaBeta( MatNum, MatDb, SATy); end

                     if ~iscell(sabc)

                         if kk==6

                            datac3temp=[datatitletemp2; datactemp];

                         else

                            datac3temp=[datac3temp; datatitletemp2; datactemp];

                         end

                     else

                         if kk==6

                            datac3temp=[datatitletemp2; datactemp; sabc];

                         else

                            datac3temp=[datac3temp; datatitletemp2; datactemp; sabc];

                         end

                     end

                end

            end

        end

    end

    if ii==2

        datac3=gendatac3temp;

    end

    if ~any(Mnum==SmearSATYs(1:end,1)) ||  SATy==0

       datac3=[datac3; datatitletemp1; datac3temp];

    end

   dispPrint(['Creating Data Cards: ';{[ii,size(MatMap,1)-1]}]);

   sabon=false;

end

% Reengage cutoff

SmallValCutOff=true;

% Flux mesh parameters
% Middle of the core
FM0org=[-85,-85,16];
FM0IMESH=85;     FM0iints=200;
FM0JMESH=85;     FM0jints=200;
FM0KMESH=17;      FM0kints=1;

% Lower Extension
FM1org=[-85,-85,-17];
FM1IMESH=85;     FM1iints=200;
FM1JMESH=85;     FM1jints=200;
FM1KMESH=-16;     FM1kints=1;

% Upper Extension
FM2org=[-85,-85,47];
FM2IMESH=85;     FM2iints=200;
FM2JMESH=85;     FM2jints=200;
FM2KMESH=48;      FM2kints=1;

% XZ slice
FM3org=[-210,0,-210];
FM3IMESH=210;     FM3iints=200;
FM3JMESH=1;       FM3jints=1;
FM3KMESH=210;      FM3kints=200;

if BorD
%     Sets the dumping values benchmark

    ndm=floor(MCNPpara{5}*0.05);
    ndp=MCNPpara{2}*ndm; %npg*ng for histories per dump
```

```
    mct=-2;
    ndmp=1;
    dmmp=ndm;

    %                           Events to be written to LOG Def:600
    %                             |  Dist between Surfs to be concidered coincident
    %                             |     |    Insist on threading 0=Disable
    %                             |     |           | Normal Input Checking 0=enable
    %                             |     |           | | Print of VOV info 1=Disbale
    %                             |     |           | | |           LJA Array
    %                             |     |           | | |            |
    datac4=CharChecker({'DBCN','4J 20 3J 1E-6 38J' ,'0 0 1','18J',   'J';...
           'c TMP',Num2StrM(DelTemp+CtoKconv+RoomTemp),' ','NUMCELLS','R';...
           'KOPTS','BLOCKSIZE=20','KINETICS=YES','PRECURSOR=YES','';...
           'RAND','GEN=2','STRIDE=500000','','';...
           'PRDMP',ndp,ndm,mct,[num2str(ndmp) ' ' num2str(dmmp)];...
           'PRINT', '-40 -50 -70',...
                 '-72 -90 -98',...
                 '-110 -120 -130',...
                 '-140 -175 -178';...
           'TALNP',' ', ' ', ' ', ' ';...
           % Mesh tally cards.
           'FMESH4:N     ','ORIGIN=',FM0org(1),FM0org(2),FM0org(3);...
           '     ','IMESH=',FM0IMESH,'    iints=',FM0iints;...
           '     ','JMESH=',FM0JMESH,'    jints=',FM0jints;...
             '      KMESH=',FM0KMESH,'    kints=',FM0kints,'    out=col';...
           'FMESH14:N    ','ORIGIN=',FM1org(1),FM1org(2),FM1org(3);...
           '     ','IMESH=',FM1IMESH,'    iints=',FM1iints;...
           '     ','JMESH=',FM1JMESH,'    jints=',FM1jints;...
             '      KMESH=',FM1KMESH,'    kints=',FM1kints,'    out=col';...
           'FMESH24:N    ','ORIGIN=',FM2org(1),FM2org(2),FM2org(3);...
           '     ','IMESH=',FM2IMESH,'    iints=',FM2iints;...
           '     ','JMESH=',FM2JMESH,'    jints=',FM2jints;...
             '      KMESH=',FM2KMESH,'    kints=',FM2kints,'    out=col';...
           'FMESH34:N    ','ORIGIN=',FM3org(1),FM3org(2),FM3org(3);...
           '     ','IMESH=',FM3IMESH,'    iints=',FM3iints;...
           '     ','JMESH=',FM3JMESH,'    jints=',FM3jints;...
             '      KMESH=',FM3KMESH,'    kints=',FM3kints,'    out=col';});

else

    ndm=floor(MCNPpara{5}*0.05);
    ndp=MCNPpara{2}*ndm; %npg*ng for histories per dump
    mct=-2;
    ndmp=1;
    dmmp=ndm;

    %                           Events to be written to LOG Def:600
    %                             |  Dist between Surfs to be concidered coincident
    %                             |     |    Insist on threading 0=Disable
    %                             |     |           | Normal Input Checking 0=enable
    %                             |     |           | | Print of VOV info 1=Disbale
    %                             |     |           | | |           LJA Array
    %                             |     |           | | |            |
    datac4=CharChecker({'DBCN','4J 20 3J 1E-6 38J' ,'0 0 1','18J',   'J';...
           'c TMP',Num2StrM(DelTemp+CtoKconv+RoomTemp),' ','NUMCELLS','R';...
           'RAND','GEN=2','STRIDE=500000','','';...
           'PRDMP',ndp,ndm,mct,[num2str(ndmp) ' ' num2str(dmmp)];...
           'PRINT', '-40 -50 -70',...
                 '-72 -90 -98',...
                 '-110 -120 -130',...
                 '-140 -175 -178';...
           'TALNP',' ', ' ', ' ', ' ';});

end

datac=[datac1; datac2; datac4; datac3];

dispPrint(' ')

global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotDataMakDat.mat')
    dispPrint('     TotDataMakDat saved....');

end

end
```

B.37. Plotters.m

```
function [ ] = Plotters( MatMap, BurnMap, MassMap, MoveMap, hdorg)
% This function is an overall code to start all of the plotters

%% Initialize variables

global  nsaBplot PrintForPub

%% ********************Create a burnup map figure*************************
```

```
MaxMinBurn=[max(cell2mat(BurnMap(2:nsaBplot,end)));...
            min(cell2mat(BurnMap(2:nsaBplot,end)))];

for ll=1:nsaBplot

    dispPrint(['Create Burnup Map...';{[ll,nsaBplot]}])
    BurnPlot(hdorg(ll,:),MatMap(ll+1,:),MaxMinBurn,BurnMap{ll+1,end},PrintForPub);

    % Export the Picture

    if ll==nsaBplot

        print('-opengl','Maps\MaterialMaps\BurnMapFig.pdf','-dpdf','-r600');

        print('Maps\MaterialMaps\BurnMapFig.eps','-depsc','-loose');

    end

end


dispPrint(' ');

%% ********************Create a mass map figure**************************

MaxMinMass=[max(cell2mat(MassMap(2:nsaBplot,end)));...
            min(cell2mat(MassMap(2:nsaBplot,end)))];

for ll=1:nsaBplot

    dispPrint(['Create Mass Map...';{[ll,nsaBplot]}])
    MassPlot(hdorg(ll,:),MatMap(ll+1,:),MaxMinMass,MassMap{ll+1,end},PrintForPub);

    % Export the Picture

    if ll==nsaBplot

        print('-opengl','Maps\MaterialMaps\MassMapFig.pdf','-dpdf','-r600');

        print('Maps\MaterialMaps\MassMapFig.eps','-depsc','-loose');

    end

end

dispPrint(' ');

%% ********************Create a move map figure**************************

MaxMinMove=[max(cell2mat(MoveMap(2:nsaBplot,end)));...
            min(cell2mat(MoveMap(2:nsaBplot,end)))];

for ll=1:nsaBplot

    dispPrint(['Create Move Map...';{[ll,nsaBplot]}])

     MovePlot(hdorg(ll,:),MoveMap(ll+1,:),MaxMinMove,MoveMap{ll+1,end});

    % Export the Picture

    if ll==nsaBplot

        print('-opengl','Maps\MaterialMaps\MoveMapFig.pdf','-dpdf','-r600');

        print('Maps\MaterialMaps\MoveMapFig.eps','-depsc','-loose');

        savefig(gcf,'Maps\MaterialMaps\MoveMapFig.fig','compact')

    end

end

end
```

B.38. BurnPlot.m

```
function [ ] = BurnPlot( hdorg,MatMap,MaxMinBurn,Burnup, PrintForPub)


persistent bax
persistent PatchHands
persistent BurnColor
persistent BClen
persistent KKprev
persistent LegendStr
persistent BurnVals
persistent BurnRange
global MatLabOld
global debugMICKA
global SANomen

BurnupPrint=true;
```

```
if Burnup==0; BurnupPrint=false; end
if isempty(Burnup); BurnupPrint=false; end

if MatLabOld; return; end

% Sets the rounding on the numbers
Rform='%.3f';
DLoc='top';

if debugMICKA || PrintForPub;

    FontSizeM=0.025;

    if PrintForPub

        FontSizeM=0.035;
        Rform='%.1f';
        DLoc='middle';

    end

else

    FontSizeM=0.02;

end

if isempty(Burnup); Burnup=0; end


if hdorg(1)==1

    % Initialize Variables

    Incr=0.5; % Percent burnup between colors

    MinR=mod(MaxMinBurn(2,1),Incr);
    MaxR=mod(MaxMinBurn(1,1),Incr);

    Min=MaxMinBurn(2,1)-MinR;
    Max=MaxMinBurn(1,1)-MaxR;

    if MinR>Incr; MinR=Incr; end
    if MinR<=Incr; MinR=0; end

    if MaxR>=Incr; MaxR=1; end
    if MaxR<Incr; MaxR=Incr; end

    Min=Min+MinR;
    Max=Max+MaxR;

    while mod(((Max-Min)/Incr),3)~=0

        Max=Max+Incr;

    end

%     n=((Max-Min)/(3*Incr)); % Number of points between colors.
    BurnRange=(Min:Incr:Max)';

    n=length(BurnRange)-1;
    RngAdj=n*(1/3);

%     blue=[zeros(n-RngAdj,1), linspace(0,1,n-RngAdj)', linspace(1,0,n-RngAdj)'];
%     green=[linspace(0,1,n)', ones(n,1), zeros(n,1)];
%     yellow=[ones(n+RngAdj,1), linspace(1,0,n+RngAdj)', zeros(n+RngAdj,1)];
%
%     BurnColor=[blue; green(2:end,:); yellow(2:end,:)];
    BurnColorT=colormap(bone(n+RngAdj));
    BurnColor=BurnColorT(RngAdj+1:end,:);
%      BurnColor=flipud(colormap(hot(n)));

    BClen=length(BurnColor);

    BurnVals=zeros(BClen,1);

    LegendStr=cell(BClen,1);

    PatchHands=gobjects(BClen,1);

end

for ll=1:length(BurnRange)

    if Burnup < BurnRange(ll,1)

        BurnColorInd=ll;

        break

    end

    if ll==length(BurnRange) && Burnup == BurnRange(ll,1)

        BurnColorInd=ll;
```

```
    end

end

hdp=5.89;

hdr=(hdp/2);
hdvr=hdr/cosd(30);
hdry=hdp*sind(60);

hdx=zeros(6,1);
hdy=zeros(6,1);
hdc=zeros(1,1,3);

SAID=MatMap{SANomen};
orgx=hdorg(2);
orgy=hdorg(3);
orgz=hdorg(4);

%Vert1

vt1x=orgx;
vt1y=orgy+hdvr;

%Vert2

vt2x=orgx+hdr;
vt2y=orgy+(hdvr/2);

%Vert3

vt3x=orgx+hdr;
vt3y=orgy-(hdvr/2);

%Vert4

vt4x=orgx;
vt4y=orgy-hdvr;

%Vert5

vt5x=orgx-hdr;
vt5y=orgy-(hdvr/2);

%Vert6

vt6x=orgx-hdr;
vt6y=orgy+(hdvr/2);

hdx(:,1)=[vt1x;vt2x;vt3x;vt4x;vt5x;vt6x];
hdy(:,1)=[vt1y;vt2y;vt3y;vt4y;vt5y;vt6y];

% SEt the color
try
hdc(1,:)=BurnColor(BurnColorInd,:);
hdc(1,1,1)=BurnColor(BurnColorInd,1);
hdc(1,1,2)=BurnColor(BurnColorInd,2);
hdc(1,1,3)=BurnColor(BurnColorInd,3);
catch
    keyboard
end


if hdorg(1)==1

    f2=figure;

    % Make the color White
    f2.Color='w';

    bax=axes;

    f2.Units='pixels';

    FigPOS=f2.OuterPosition;

    l=FigPOS(1);
    b=FigPOS(2);
    w=FigPOS(3);
    h=FigPOS(4);

    f2.OuterPosition=[l-w b w h];

    title(bax,{'Reactor Burnup (%at) Map'; ' '},'FontUnits','normalized','FontSize',0.05)

    axis(bax,'off');

    hold(bax,'on');

    p1=patch(hdx,hdy,hdc,'Parent',bax);

    KKprev=0;

    if BurnupPrint;
```

```
        text(hdorg(2),hdorg(3),num2str(Burnup,Rform),'Parent',bax,...
            'VerticalAlignment',DLoc,'HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

    if ~PrintForPub

        text(hdorg(2),hdorg(3),SAID,'Parent',bax,...
            'VerticalAlignment','bottom','HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');
    end

    axis(bax,'equal')

else

    p1=patch(hdx,hdy,hdc,'Parent',bax);

    if BurnupPrint;

        text(hdorg(2),hdorg(3),num2str(Burnup,Rform),'Parent',bax,...
            'VerticalAlignment',DLoc,'HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

    if ~PrintForPub

        text(hdorg(2),hdorg(3),SAID,'Parent',bax,...
            'VerticalAlignment','bottom','HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

end

if PrintForPub; title(bax,''); end

%% Legend code

    kk=1;

    for ll=1:length(BurnRange)

        if Burnup < BurnRange(ll,1)

            if Burnup==0

                BurnRangeText=' 0 at%';

            else

                BurnRangeText=['<' Num2StrM(BurnRange(ll,1),'%.4g') ' at%'];

            end


            BurnVals(ll,1)=1;

            break

        end

        if ll==length(BurnRange) && Burnup == BurnRange(ll,1)

            BurnRangeText=['<' Num2StrM(BurnRange(ll,1),'%.4g') ' at%'];
            BurnVals(ll,1)=1;

        end

    end

    kk=length(find(BurnVals>0));

    if kk~=KKprev

        LegendStr(BurnColorInd,1)={BurnRangeText};
        PatchHands(BurnColorInd,1)=p1;

        qq=1;

        for nn=1:size(LegendStr)

           if ishandle(PatchHands(nn,1))

               if qq==1

                   PatchHandsT=PatchHands(nn,1);
                   LegendStrT=LegendStr(nn,1);
```

```
                qq=qq+1;

            else

                PatchHandsT=[PatchHandsT; PatchHands(nn,1)];
                LegendStrT=[LegendStrT; LegendStr(nn,1)];

            end

        end

    end


        KKprev=kk;

        legend(PatchHandsT,LegendStrT,'FontSize',12,...
            'FontWeight','bold','Location','northeastoutside');

end

end
```

B.39. MassPlot.m

```
function [ ] = MassPlot( hdorg,MatMap,MaxMinMass,Mass,PrintForPub)


persistent bax
persistent PatchHands
persistent BurnColor
persistent BClen
persistent KKprev
persistent LegendStr
persistent BurnVals
persistent BurnRange
global MatLabOld
global debugMICKA
global SANomen

BurnupPrint=true;

if Mass==0; BurnupPrint=false; end
if isempty(Mass); BurnupPrint=false; end

if MatLabOld; return; end

% Sets the rounding on the numbers
Rform='%.3f';
DLoc='top';

if debugMICKA || PrintForPub;

    FontSizeM=0.025;

    if PrintForPub

        FontSizeM=0.035;
        Rform='%.1f';
        DLoc='middle';

    end

else

    FontSizeM=0.02;

end

if isempty(Mass); Mass=0; end


if hdorg(1)==1

    % Initialize Variables

    Incr=0.5; % Percent mass between colors

    MinR=mod(MaxMinMass(2,1),Incr);
    MaxR=mod(MaxMinMass(1,1),Incr);

    Min=MaxMinMass(2,1)-MinR;
    Max=MaxMinMass(1,1)-MaxR;

    if MinR>Incr; MinR=Incr; end
    if MinR<=Incr; MinR=0; end

    if MaxR>=Incr; MaxR=1; end
    if MaxR<Incr; MaxR=Incr; end

    Min=Min+MinR;
    Max=Max+MaxR;
```

```
    while mod(((Max-Min)/Incr),3)~=0

        Max=Max+Incr;

    end

%     n=((Max-Min)/(3*Incr)); % Number of points between colors.
    BurnRange=(Min:Incr:Max)';

    n=length(BurnRange)-1;
    RngAdj=n*(1/3);

    blue=[zeros(n-RngAdj,1), linspace(0,1,n-RngAdj)', linspace(1,0,n-RngAdj)'];
    green=[linspace(0,1,n)', ones(n,1), zeros(n,1)];
    yellow=[ones(n+RngAdj,1), linspace(1,0,n+RngAdj)', zeros(n+RngAdj,1)];
    BurnColor=[blue; green(2:end,:); yellow(2:end,:)];

%     BurnColorT=colormap(bone(n+RngAdj));
%     BurnColor=BurnColorT(RngAdj+1:end,:);
%      BurnColor=flipud(colormap(hot(n)));

    BClen=length(BurnColor);

    BurnVals=zeros(BClen,1);

    LegendStr=cell(BClen,1);

    PatchHands=gobjects(BClen,1);

end

for ll=1:length(BurnRange)

    if Mass < BurnRange(ll,1)

        BurnColorInd=ll;

        break

    end

    if ll==length(BurnRange) && Mass == BurnRange(ll,1)

        BurnColorInd=ll;

    end

end

hdp=5.89;

hdr=(hdp/2);
hdvr=hdr/cosd(30);
hdry=hdp*sind(60);

hdx=zeros(6,1);
hdy=zeros(6,1);
hdc=zeros(1,1,3);

SAID=MatMap{SANomen};
orgx=hdorg(2);
orgy=hdorg(3);
orgz=hdorg(4);

%Vert1

vt1x=orgx;
vt1y=orgy+hdvr;

%Vert2

vt2x=orgx+hdr;
vt2y=orgy+(hdvr/2);

%Vert3

vt3x=orgx+hdr;
vt3y=orgy-(hdvr/2);

%Vert4

vt4x=orgx;
vt4y=orgy-hdvr;

%Vert5

vt5x=orgx-hdr;
vt5y=orgy-(hdvr/2);

%Vert6

vt6x=orgx-hdr;
vt6y=orgy+(hdvr/2);
```

```
hdx(:,1)=[vt1x;vt2x;vt3x;vt4x;vt5x;vt6x];
hdy(:,1)=[vt1y;vt2y;vt3y;vt4y;vt5y;vt6y];

% SEt the color
try
hdc(1,:)=BurnColor(BurnColorInd,:);
hdc(1,1,1)=BurnColor(BurnColorInd,1);
hdc(1,1,2)=BurnColor(BurnColorInd,2);
hdc(1,1,3)=BurnColor(BurnColorInd,3);
catch
    keyboard
end


if hdorg(1)==1

    f2=figure;

    % Make the color White
    f2.Color='w';

    bax=axes;

    f2.Units='pixels';

    FigPOS=f2.OuterPosition;

    l=FigPOS(1);
    b=FigPOS(2);
    w=FigPOS(3);
    h=FigPOS(4);

    f2.OuterPosition=[l-w b-h w h];

    title(bax,{'Reactor Mass (gms) Map'; ' '},'FontUnits','normalized','FontSize',0.05)

    axis(bax,'off');

    hold(bax,'on');

    p1=patch(hdx,hdy,hdc,'Parent',bax);

    KKprev=0;

    if BurnupPrint;

        text(hdorg(2),hdorg(3),num2str(Mass,Rform),'Parent',bax,...
            'VerticalAlignment',DLoc,'HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

    if ~PrintForPub

        text(hdorg(2),hdorg(3),SAID,'Parent',bax,...
            'VerticalAlignment','bottom','HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

    axis(bax,'equal')

else

    p1=patch(hdx,hdy,hdc,'Parent',bax);

    if BurnupPrint;

        text(hdorg(2),hdorg(3),num2str(Mass,Rform),'Parent',bax,...
            'VerticalAlignment',DLoc,'HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

    if ~PrintForPub

        text(hdorg(2),hdorg(3),SAID,'Parent',bax,...
            'VerticalAlignment','bottom','HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end

end


if PrintForPub; title(bax,''); end

%% Legend code

    kk=1;
```

```
    for ll=1:length(BurnRange)

        if Mass < BurnRange(ll,1)

            if Mass==0

                BurnRangeText=' 0g';

            else

                BurnRangeText=['<' Num2StrM(BurnRange(ll,1),'%.4g') 'g'];

            end


            BurnVals(ll,1)=1;

            break

        end

        if ll==length(BurnRange) && Mass == BurnRange(ll,1)

            BurnRangeText=['<' Num2StrM(BurnRange(ll,1),'%.4g') 'g'];
            BurnVals(ll,1)=1;

        end

    end

    kk=length(find(BurnVals>0));

    if kk~=KKprev

        LegendStr(BurnColorInd,1)={BurnRangeText};
        PatchHands(BurnColorInd,1)=p1;

        qq=1;

        for nn=1:size(LegendStr)

           if ishandle(PatchHands(nn,1))

               if qq==1

                   PatchHandsT=PatchHands(nn,1);
                   LegendStrT=LegendStr(nn,1);
                    qq=qq+1;

               else

                   PatchHandsT=[PatchHandsT; PatchHands(nn,1)];
                   LegendStrT=[LegendStrT; LegendStr(nn,1)];

               end

           end

        end


            KKprev=kk;

            legend(PatchHandsT,LegendStrT,'FontSize',12,...
                'FontWeight','bold','Location','northeastoutside');

    end


end
```

B.40. MovePlot.m

```
function [ ] = MovePlot( hdorg,MatMap,MaxMinMove,Move)

persistent Blank
persistent Dr
persistent DrHFW
persistent Safety
persistent HWCR
persistent Cont
persistent SST
persistent Refl
persistent Blanket
persistent EXP
persistent Wall
persistent BClen
persistent Histnum
persistent bax
persistent PatchHands
persistent BurnColor
persistent KKprev
persistent LegendStr
```

```
persistent BurnVals
persistent BurnRange
global MatLabOld
global debugMICKA
global SAtocol;

if MatLabOld; return; end

if debugMICKA; FontSizeM=0.025; else FontSizeM=0.015; end

if isempty(Move); Move=0; UnMove=0; end

printText=false;

if hdorg(1)==1

Blank=0;
Dr=0;
DrHFW=0;
Safety=0;
HWCR=0;
Cont=0;
SST=0;
Refl=0;
Blanket=0;
EXP=0;
Wall=0;

end

hdp=5.89;

hdr=(hdp/2);
hdvr=hdr/cosd(30);
hdry=hdp*sind(60);

hdx=zeros(6,1);
hdy=zeros(6,1);
hdz=zeros(6,1);
hdc=zeros(1,1,3);

SATy=MatMap{2};
orgx=hdorg(2);
orgy=hdorg(3);
orgz=hdorg(4);

if SATy==10; Move=0; UnMove=0; end

if any(SATy==[3,4,5]);

    printText=true;

    UnMove=Move-14;

end

%Vert1

vt1x=orgx;
vt1y=orgy+hdvr;

%Vert2

vt2x=orgx+hdr;
vt2y=orgy+(hdvr/2);

%Vert3

vt3x=orgx+hdr;
vt3y=orgy-(hdvr/2);

%Vert4

vt4x=orgx;
vt4y=orgy-hdvr;

%Vert5

vt5x=orgx-hdr;
vt5y=orgy-(hdvr/2);

%Vert6

vt6x=orgx-hdr;
vt6y=orgy+(hdvr/2);

hdx(:,1)=[vt1x;vt2x;vt3x;vt4x;vt5x;vt6x];
hdy(:,1)=[vt1y;vt2y;vt3y;vt4y;vt5y;vt6y];
hdz(:,1)=[UnMove;UnMove;UnMove;UnMove;UnMove;UnMove];

hdz2(:,1)=[Move;Move;Move;Move;Move;Move];

Fc1x=[vt1x;vt1x;vt2x;vt2x];
Fc1y=[vt1y;vt1y;vt2y;vt2y];
Fc1z=[UnMove;Move;Move;UnMove];
```

```
Fc2x=[vt2x;vt2x;vt3x;vt3x];
Fc2y=[vt2y;vt2y;vt3y;vt3y];
Fc2z=[UnMove;Move;Move;UnMove];

Fc3x=[vt3x;vt3x;vt4x;vt4x];
Fc3y=[vt3y;vt3y;vt4y;vt4y];
Fc3z=[UnMove;Move;Move;UnMove];

Fc4x=[vt4x;vt4x;vt5x;vt5x];
Fc4y=[vt4y;vt4y;vt5y;vt5y];
Fc4z=[UnMove;Move;Move;UnMove];

Fc5x=[vt5x;vt5x;vt6x;vt6x];
Fc5y=[vt5y;vt5y;vt6y;vt6y];
Fc5z=[UnMove;Move;Move;UnMove];

Fc6x=[vt6x;vt6x;vt1x;vt1x];
Fc6y=[vt6y;vt6y;vt1y;vt1y];
Fc6z=[UnMove;Move;Move;UnMove];

if SATy==0

    hdc(1,1,1)=0;
    hdc(1,1,2)=0;
    hdc(1,1,3)=0;

else

    hdc(1,:)=SAtocol(SATy,2:end);
    hdc(1,1,1)=SAtocol(SATy,2);
    hdc(1,1,2)=SAtocol(SATy,3);
    hdc(1,1,3)=SAtocol(SATy,4);

end

switch SATy

    case 0
        Blank=Blank+1;
    case 1
        Dr=Dr+1;
    case 2
        DrHFW=DrHFW+1;
    case 3
        Safety=Safety+1;
    case 4
        HWCR=HWCR+1;
    case 5
        Cont=Cont+1;
    case 6
        SST=SST+1;
    case 7
        Refl=Refl+1;
    case 8
        Blanket=Blanket+1;
    case 9

    case 10
        EXP=EXP+1;
    case 11
        Wall=Wall+1;

end

count=true;

if strcmp(MatMap{2},'Y')

    count=false;

end

if count

    if hdorg(1)==1

    Histnum=SATy;

    else

        Histnum=[Histnum, SATy];

    end

end

CatVal=[1 2 3 4 5 6 7 8 10 11 0];
CatNames={'Driver','Driver HFW','Safety','HWCR','Control','Dummy',...
    'Reflector','Blanket','EXPerimental','Wall','Blank'};

Cata=categorical(Histnum,CatVal,CatNames);

if hdorg(1)==1
```

```
    f2=figure;

    bax=axes;

    f2.Units='pixels';
    FigPOS=f2.OuterPosition;

    l=FigPOS(1);
    b=FigPOS(2);
    w=FigPOS(3);
    h=FigPOS(4);

    f2.OuterPosition=[l b-h w h];

    title(bax,{'Move Map'; ' '},'FontUnits','normalized','FontSize',0.05)

    axis(bax,'off');

    hold(bax,'on');

    view(bax,3)

    p1= [patch(hdx,hdy,hdz,hdc,'Parent',bax);    % Bottom
         patch(hdx,hdy,hdz2,hdc,'Parent',bax);   % Top
         patch(Fc1x,Fc1y,Fc1z,hdc,'Parent',bax); % FC1
         patch(Fc2x,Fc2y,Fc2z,hdc,'Parent',bax); % FC2
         patch(Fc3x,Fc3y,Fc3z,hdc,'Parent',bax); % FC3
         patch(Fc4x,Fc4y,Fc4z,hdc,'Parent',bax); %
         patch(Fc5x,Fc5y,Fc5z,hdc,'Parent',bax);
         patch(Fc6x,Fc6y,Fc6z,hdc,'Parent',bax)];

    PatchHands=p1(1);

    KKprev=0;


    view([33, 33])

    if printText

        text(hdorg(2),hdorg(3),Move+1,[num2str(Move,'%.3f') '"'],'Parent',bax,...
            'VerticalAlignment','top','HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end


    axis(bax,'equal')

else

    p1= [patch(hdx,hdy,hdz,hdc,'Parent',bax);    % Bottom
         patch(hdx,hdy,hdz2,hdc,'Parent',bax);   % Top
         patch(Fc1x,Fc1y,Fc1z,hdc,'Parent',bax); % FC1
         patch(Fc2x,Fc2y,Fc2z,hdc,'Parent',bax); % FC2
         patch(Fc3x,Fc3y,Fc3z,hdc,'Parent',bax); % FC3
         patch(Fc4x,Fc4y,Fc4z,hdc,'Parent',bax); %
         patch(Fc5x,Fc5y,Fc5z,hdc,'Parent',bax);
         patch(Fc6x,Fc6y,Fc6z,hdc,'Parent',bax)];


    if printText

        text(hdorg(2),hdorg(3),Move+1,[num2str(Move,'%.3f') '"'],'Parent',bax,...
            'VerticalAlignment','top','HorizontalAlignment','center',...
            'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
            'FontSmoothing','on');

    end



end

%% Legend code

%     kk=1;
%
%     for ll=1:length(BurnRange)
%
%         if Move < BurnRange(ll,1)
%
%             if Move==0
%
%                 BurnRangeText='No Movement';
%
%             else
%
%                 BurnRangeText=[Num2StrM(BurnRange(ll,1),'%.4g') ''''];
%
%             end
%
%
%             BurnVals(ll,1)=1;
```

```
%
%             break
%
%         end
%
%         if ll==length(BurnRange) && Move == BurnRange(ll,1)
%
%             BurnRangeText=['<' Num2StrM(BurnRange(ll,1),'%.4g') ''''];
%             BurnVals(ll,1)=1;
%
%         end
%
%     end
%
%     kk=length(find(BurnVals>0));
%
%     if kk~=KKprev
%
%         LegendStr(BurnColorInd,1)={BurnRangeText};
%         PatchHands(BurnColorInd,1)=p1(1);
%
%         qq=1;
%
%         for nn=1:size(LegendStr)
%
%            if ishandle(PatchHands(nn,1))
%
%                if qq==1
%
%                    PatchHandsT=PatchHands(nn,1);
%                    LegendStrT=LegendStr(nn,1);
%                     qq=qq+1;
%
%                else
%
%                    PatchHandsT=[PatchHandsT; PatchHands(nn,1)];
%                    LegendStrT=[LegendStrT; LegendStr(nn,1)];
%
%                end
%
%            end
%
%         end
%
%
%             KKprev=kk;
%
%             legend(PatchHandsT,LegendStrT,'FontSize',12,...
%                 'FontWeight','bold','Location','northeastoutside');
%
%     end

kk=0;

if DrHFW>0;  kk=kk+1; end
if Dr>0;  kk=kk+1; end
if Safety>0;  kk=kk+1; end
if HWCR>0;  kk=kk+1; end
if Cont>0;  kk=kk+1; end
if SST>0;  kk=kk+1; end
if Refl>0;  kk=kk+1; end
if Blanket>0;  kk=kk+1; end
if EXP>0;  kk=kk+1; end
if Blank>0;  kk=kk+1; end

if SATy==10; CatNamInd=9; end
if SATy==0; CatNamInd=11; end
if ~any(SATy==[0,10]); CatNamInd=SATy; end

if kk~=KKprev

    if kk==1;

        LegendStr=CatNames(CatNamInd);

    else

        LegendStr=[LegendStr, CatNames(CatNamInd)];
        PatchHands=[PatchHands p1(1)];

    end

    KKprev=kk;

    legend(PatchHands,LegendStr,'Location','northeastoutside');

end

end
```

B.41. AssemblyMaker.m

```
function [ cellc, surfc, datac ] = AssemblyMaker( hdp, hdorg, ReactMap, DimMap, MatMap, nsa, pmsurfc, datac, DescC  )
```

```
% Assembly Maker is used to call hexmaker and intiate the code surrounding
% HexMaker

global SaveSAMakerVars MSATyC MNumC mcnpfix SmearSATYs

dispPrint(' ');

% The following variable is used to keep track of which cells and surfaces are
% assign to which assembly. This has no bearing on the output file, it is more
% for information purposes.

NumMapTitles={'Starting Cell Numbers','Ending Cell Numbers','Cell Numbers',...
    'Starting Surface Numbers','Ending Surface Numbers','Surface Numbers',...
    'Duct Surface Number'};

NumMap=[ReactMap, [NumMapTitles; cell(nsa,size(NumMapTitles,2))]];

for ii=1:nsa

   if ~SaveSAMakerVars; dispPrint(['Creating MCNP Cards...';{[ii,nsa]}]); end

   Mnum=MatMap{ii+1,MNumC};

   [cellctemp,surfctemp,Smeardatac,NumMap]=HexMaker(hdorg(ii,:),mcnpfix,NumMap,...
       MatMap(ii+1,:),DimMap(ii+1,:));

    if ~isempty(SmearSATYs(Mnum==SmearSATYs(1:end,1)))

        datac=[datac;Smeardatac];

    end

    if ii==1
        cellc=[DescC ;cellctemp];
        surfc=[pmsurfc; surfctemp];
    end

    if ii>1

        surfc=[surfc; surfctemp];
        cellc=[cellc; cellctemp];

    end

    hold on

    if ii==1;

        nr=ceil((3+sqrt(12*nsa-3))/6);
        nnsa=3*((nr^2)+(nr+1))-2;

    end

    if ii==nsa

        PlotSA(hdorg(ii,:),MatMap(ii+1,:),hdp,nnsa);
        [hdorgPlot,OrgMap]=HexOrg(nnsa,hdp,ReactMap);

        dispPrint(' ');

        for jj=1:(nnsa-nsa)

            if ~SaveSAMakerVars;

                dispPrint(['Creating Blank Assemblies....';{[jj,nnsa-nsa]}]);

            end

            PlotSA(hdorgPlot(ii+jj,:),[{'Y'},{11}],hdp,nnsa);

        end

        dispPrint(' ')

        %     Create a .mat file for the mats for reading later.
        dispPrint('Saving MatLab Variable NumMap');
        dispPrint(' ')

        save('Debug\NumMap.mat','NumMap')

    else

        PlotSA(hdorg(ii,:),MatMap(ii+1,:),hdp,nnsa);

    end

end
```

B.42. HexMaker.m

```
function [ cellc,surfc,datac,NumMap ] = HexMaker(hdorg,mcnpfix,NumMap,MatMap,DimMap )
```

```
%This function imports the driver data and then writes the  assembly.

%%
%Each case will be separate assembly type to be written. It will call a
%reader function for the dimensional data for that particular MICKA type.
%The case will load and pass the variables to the particular assm type

%%
% For information gathering purposes, this section will associate cell and
% surface numbers with assembly ids and MICKA numbers

% Initialize variables
global OldControlSA
global BorD
global surfnum
global cellnum
global SAsurfnum
global MSATyC MNumC
global SmearSATYs

SATy=MatMap{1,MSATyC};
Mnum=MatMap{1,MNumC};

NewDataC=false;

% Check for smeared assemblies

if ~isempty(SmearSATYs(Mnum==SmearSATYs(1:end,1)))

    [Mnrow,~]=find(Mnum==SmearSATYs(1:end,1));

    SATy=SmearSATYs(Mnrow,3);

end

bend=0; % Old variable but needed

%% Activate MICKA bending

if ~BorD && ~SATy==0

    SATy=SATy+10;

end

switch SATy

%% Benchmark Makers

    case 0
    % Used as an empty assm
        [ cellc,surfc ] = BlankMaker(hdorg);

    case 1

        [ cellc,surfc ] = DriverMakerMKII(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 2

        [ cellc,surfc ] = DriverMakerMKIIHFW(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 3

        [ cellc,surfc ] = SafetyMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 4

        [ cellc,surfc ] = HWCRMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 5

        [ cellc,surfc ] = ControlMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 6

        [ cellc,surfc ] = DummyMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 7

        [ cellc,surfc ] = ReflectorMaker(hdorg,mcnpfix,bend,MatMap,DimMap );

    case 8

        [ cellc,surfc ] = BlanketMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 9

        [ cellc,surfc ] = WallMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 10

        [ cellc,surfc ] = CustomMaker(hdorg,mcnpfix,bend,MatMap,DimMap); % Custom assemblies

        %% Katana Asssemblies
```

```
    case 11

        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 12

        [ cellc,surfc ] = DriverMakerMKIIHFWBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 13

        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 14 % HWCR

        % Only HWCRs that were at full insertion were modeled as a driver
        % with 61 pins.
        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 15 % Control

        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 16 % Dummy

        [ cellc,surfc ] = DummyMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 17 % Reflector

        [ cellc,surfc ] = ReflectorMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap );

    case 18 % Blanket

        [ cellc,surfc ] = BlanketMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 19

        [ cellc,surfc ] = WallMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 20

        [ cellc,surfc ] = CustomMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap); % Custom assemblies

%% Smeared Makers by Ryan

    case 22

        [ cellc,surfc,MatMap ] = DriverHWMakerS(hdorg,mcnpfix,bend,MatMap,DimMap);
        NewDataC=true;

    case 26

        [ cellc,surfc,MatMap ] = DummyMakerS(hdorg,mcnpfix,bend,MatMap,DimMap );
        NewDataC=true;

    case 27

        [ cellc,surfc,MatMap ] = ReflectorMakerS(hdorg,mcnpfix,bend,MatMap,DimMap );
        NewDataC=true;

    case 28

        [ cellc,surfc,MatMap ] = BlanketMakerSP1(hdorg,mcnpfix,bend,MatMap,DimMap);
        NewDataC=true;

end

%% Need to reformat the data output from the smeared materials

if NewDataC

    datac=MatMap2DataC(MatMap);

else

    datac=[];

end

Oldsurfnum=surfnum;
Oldcellnum=cellnum;
OldSAsurfnum=SAsurfnum;
RowNum=hdorg(1)+1;

Newsurfnum=surfnum-1;
Newcellnum=cellnum-1;

NumMap{RowNum,6}=Oldcellnum;
NumMap{RowNum,7}=Newcellnum;
NumMap{RowNum,8}=linspace(Oldcellnum,Newcellnum,Newcellnum-Oldcellnum+1);
NumMap{RowNum,9}=Oldsurfnum;
NumMap{RowNum,10}=Newsurfnum;
NumMap{RowNum,11}=linspace(Oldsurfnum,Newsurfnum,Newsurfnum-Oldsurfnum+1);
NumMap{RowNum,12}=OldSAsurfnum;

end
```

```
function datac=MatMap2DataC(MatMap)

global     MNumC SAPosC SANomen SAlphaBetas MSATyC SABcorr SmallValCutOff FdC
persistent firstrun
persistent SATYsab

if isempty(firstrun) && SABcorr;

    % s(a,b) additions

    SATYsab=cell2mat(SAlphaBetas(:,2));

end

MatMap{FdC+1}(1,:)=delNaN(MatMap{FdC+1}(1,:),'Convert all','num');

sabon=false;
SmallValCutOff=false;

dispPrintmat={'Smeared Material 1';...
              'Smeared Material 2';...
              'Smeared Material 3';...
              'Smeared Material 4';...
              'Smeared Material 5';...
              'Smeared Material 6';...
              'Smeared Material 7';...
              'Smeared Material 8';...
              'Smeared Material 9';...
              'Smeared Material 10';...
              'Smeared Material 11';...
              'Smeared Material 12';...
              'Smeared Material 13';...
              'Smeared Material 14';...
              'Smeared Material 15';...
              'Smeared Material 16';...
              'Smeared Material 17';...
              'Smeared Material 18';...
              'Smeared Material 19';};

datatitletemp1=NameCard(['MICKA #: ' num2str(MatMap{MNumC}) ' ID: ' ...
    MatMap{SANomen} ' POS: ' MatMap{SAPosC}],'Title');

SATy=MatMap{MSATyC};

% The following statement adds sab corrections to the xsec

if any(SATy==SATYsab) && SABcorr; sabon=true; else sabc=0; end

for kk=6:3:size(MatMap,2)

    if isempty(MatMap{kk})

            datactemp=NameCard(['No ' dispPrintmat{(kk/3)-1}],'Divider');

            if kk==6

                datac=[datatitletemp1; datactemp];

            else

                datac=[datac; datactemp];

            end

        else

            MatNum=MatMap{kk};
            MatDb=MatMap{kk+2};

            if size(MatDb,1)>2

                datatitletemp2=NameCard(dispPrintmat{(kk/3)-1},'Divider');

                for hh=1:length(MatNum)

                    datatitletemp3=NameCard(['Pin Sec: ' num2str(hh)],'Divider');

                    datactemp=CharChecker(MICKAtoMCNPmat(MatNum(hh),...
                        [MatDb(1,:); MatDb(hh+1,:)]));

                     if sabon; [ sabc ] = SAlphaBeta( MatNum(hh), MatDb, SATy); end

                     if kk==6 && hh==1

                        datac=[datatitletemp1; datatitletemp2; datatitletemp3; datactemp];

                     else

                        datac=[datac; datatitletemp3; datactemp];

                     end

                end
```

```
            else

                datatitletemp2=NameCard(dispPrintmat{(kk/3)-1},'Divider');
                datactemp=CharChecker(MICKAtoMCNPmat(MatNum,MatDb));

                 if sabon; [ sabc ] = SAlphaBeta( MatNum, MatDb, SATy); end

                 if ~iscell(sabc)

                     if kk==6

                        datac=[datatitletemp1; datatitletemp2; datactemp];

                     else

                        datac=[datac; datatitletemp2; datactemp];

                     end

                 else

                     if kk==6

                        datac=[datatitletemp1; datatitletemp2; datactemp; sabc];

                     else

                        datac=[datac; datatitletemp2; datactemp; sabc];

                     end

                 end

            end

    end

end

SmallValCutOff=true;
```

B.43. PlotSA.m

```
function [ ] = PlotSA( hdorg,MatMap,hdp,TotalCls)

persistent Blank
persistent Dr
persistent DrHFW
persistent Safety
persistent HWCR
persistent Cont
persistent SST
persistent Refl
persistent Blanket
persistent EXP
persistent Wall
persistent Histnum
persistent PatchHands
persistent PatchHands2
persistent KKprev
persistent LegendStr
persistent ClCnt

global pax
global pax2
global hax
global f1
global MatLabOld
global SAtocol;
global f2
global debugMICKA
global MNumC MSATyC SANomen

if MatLabOld
    return
end

MatLabOldInternal=false;
CreateRGBCore=false;
RGBtext=false;
RGBMnum=false;

% Sets the rounding on the numbers
Rform='%.0f';
DLoc1='top';
DLoc2='bottom';

if debugMICKA;

    CreateRGBCore=true;
```

```
    FontSizeM=0.025;
    Rform='%.0f';
    DLoc1='top';
    DLoc2='bottom';

else

    FontSizeM=0.015;

end

if verLessThan('matlab','8.6'); MatLabOldInternal=true; end

if hdorg(1)==1

    Blank=0;
    Dr=0;
    DrHFW=0;
    Safety=0;
    HWCR=0;
    Cont=0;
    SST=0;
    Refl=0;
    Blanket=0;
    EXP=0;
    Wall=0;
    ClCnt=0;

end

ClCnt=ClCnt+1;

hdr=(hdp/2);
hdvr=hdr/cosd(30);
hdry=hdp*sind(60);

hdx=zeros(6,1);
hdy=zeros(6,1);
hdc=zeros(1,1,3);

SATy=MatMap{MSATyC};

orgx=hdorg(2);
orgy=hdorg(3);
orgz=hdorg(4);

count=true;

if strcmp(MatMap{MNumC},'Y')

    count=false;

    SAID='';

else

    SAID=MatMap{SANomen};

end

%Vert1

vt1x=orgx;
vt1y=orgy+hdvr;

%Vert2

vt2x=orgx+hdr;
vt2y=orgy+(hdvr/2);

%Vert3

vt3x=orgx+hdr;
vt3y=orgy-(hdvr/2);

%Vert4

vt4x=orgx;
vt4y=orgy-hdvr;

%Vert5

vt5x=orgx-hdr;
vt5y=orgy-(hdvr/2);

%Vert6

vt6x=orgx-hdr;
vt6y=orgy+(hdvr/2);

hdx(:,1)=[vt1x;vt2x;vt3x;vt4x;vt5x;vt6x];
hdy(:,1)=[vt1y;vt2y;vt3y;vt4y;vt5y;vt6y];

if SATy==0
```

```
    hdc(1,1,1)=0;
    hdc(1,1,2)=0;
    hdc(1,1,3)=0;

else

    hdc(1,:)=SAtocol(SATy,2:end);

    hdc(1,1,1)=SAtocol(SATy,2);
    hdc(1,1,2)=SAtocol(SATy,3);
    hdc(1,1,3)=SAtocol(SATy,4);

end

switch SATy

    case 0
        Blank=Blank+1;
    case 1
        Dr=Dr+1;
    case 2
        DrHFW=DrHFW+1;
    case 3
        Safety=Safety+1;
    case 4
        HWCR=HWCR+1;
    case 5
        Cont=Cont+1;
    case 6
        SST=SST+1;
    case 7
        Refl=Refl+1;
    case 8
        Blanket=Blanket+1;
    case 9

    case 10
        EXP=EXP+1;
    case 11
        Wall=Wall+1;

end

if count

    if hdorg(1)==1

    Histnum=SATy;

    else

        Histnum=[Histnum, SATy];

    end

end

CatVal=[1 2 3 4 5 6 7 8 10 11 0];
CatNames={'Driver','Driver HFW','Safety','HWCR','Control','Dummy',...
    'Reflector','Blanket','EXPerimental','Wall','Blank'};

Cata=categorical(Histnum,CatVal,CatNames);

if hdorg(1)==1

    pax=subplot(2,2,2,'Parent',f1);

    cla(pax);

    title(pax,{'Reactor Build Map'; ' '})
    axis(pax,'off');

    hold(pax,'on');

    hax=subplot(2,2,4,'Parent',f1);
    title(hax,{'Histogram of Assembly Types'; ' '})
    hold(hax,'off');

    p1=patch(hdx,hdy,hdc,'Parent',pax);

    if CreateRGBCore

        f2=figure;

        pax2=axes('Parent',f2);
        axis(pax2,'off');

        p2=patch(hdx,hdy,hdc,'Parent',pax2);

        PatchHands2=p2;

        axis(pax2,'equal')

        if RGBtext
```

```
            text(hdorg(2),hdorg(3),SAID,'Parent',pax2,...
                'VerticalAlignment',DLoc1,'HorizontalAlignment','center',...
                'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
                'FontSmoothing','on');

            text(hdorg(2),hdorg(3),Num2StrM(SATy,Rform),'Parent',pax2,...
                'VerticalAlignment',DLoc2,'HorizontalAlignment','center',...
                'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
                'FontSmoothing','on');

        end

        if RGBMnum

            text(hdorg(2),hdorg(3),Num2StrM(ClCnt),'Parent',pax2,...
                'VerticalAlignment','middle','HorizontalAlignment','center',...
                'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
                'FontSmoothing','on');

        end

    end

    KKprev=0;
    PatchHands=p1;

    axis(pax,'equal')

    if MatLabOldInternal

        hist(hax,Cata);

    else

        if ClCnt==TotalCls

            histogram(hax,Cata,'BarWidth',0.5);

        end

    end

    binX=linspace(1,size(CatNames,2),size(CatNames,2));
    binY=[Dr,DrHFW,Safety,HWCR,Cont,SST,Refl,Blanket,EXP,Wall,Blank];

    for ii=1:length(binY)

        binYstr{ii}=num2str(binY(ii));

    end

    text(binX,binY,binYstr,'Parent',hax,'VerticalAlignment','bottom');

else

    p1=patch(hdx,hdy,hdc,'Parent',pax);

    if CreateRGBCore;

        p2=patch(hdx,hdy,hdc,'Parent',pax2);

        if RGBtext

            text(hdorg(2),hdorg(3),SAID,'Parent',pax2,...
                'VerticalAlignment',DLoc1,'HorizontalAlignment','center',...
                'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
                'FontSmoothing','on');

            text(hdorg(2),hdorg(3),Num2StrM(SATy,Rform),'Parent',pax2,...
                'VerticalAlignment',DLoc2,'HorizontalAlignment','center',...
                'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
                'FontSmoothing','on');

        end

        if RGBMnum

            text(hdorg(2),hdorg(3),Num2StrM(ClCnt),'Parent',pax2,...
                'VerticalAlignment','middle','HorizontalAlignment','center',...
                'FontUnits','normalized','FontSize',FontSizeM,'FontWeight','bold',...
                'FontSmoothing','on');

        end

    end

    if ClCnt==TotalCls

        cla(hax)

        if MatLabOldInternal

            hist(hax,Cata);

        else
```

```
           histogram(hax,Cata,'BarWidth',0.5);

        end

        binX=linspace(1,size(CatNames,2),size(CatNames,2));
        binY=[Dr,DrHFW,Safety,HWCR,Cont,SST,Refl,Blanket,EXP,Wall,Blank];

        for ii=1:length(binY)

            binYstr{ii}=num2str(binY(ii));

        end

        text(binX,binY,binYstr,'Parent',hax,'VerticalAlignment','bottom');

    end

end

kk=0;

if DrHFW>0;  kk=kk+1; end
if Dr>0;  kk=kk+1; end
if Safety>0;  kk=kk+1; end
if HWCR>0;  kk=kk+1; end
if Cont>0;  kk=kk+1; end
if SST>0;  kk=kk+1; end
if Refl>0;  kk=kk+1; end
if Blanket>0;  kk=kk+1; end
if EXP>0;  kk=kk+1; end
if Blank>0;  kk=kk+1; end

if SATy==10; CatNamInd=9; end
if SATy==0; CatNamInd=11; end
if ~any(SATy==[0,10]); CatNamInd=SATy; end

if kk~=KKprev

    if kk==1;

        LegendStr=CatNames(CatNamInd);

    else

        LegendStr=[LegendStr, CatNames(CatNamInd)];
        PatchHands=[PatchHands p1];
        if CreateRGBCore;

            PatchHands2=[PatchHands2 p2];

            legend(PatchHands2,LegendStr,'Location','northeastoutside');

        end

    end

    KKprev=kk;

    legend(PatchHands,LegendStr,'Location','northeastoutside');

end
```

B.44. HexMaker.m

```
function [ cellc,surfc,datac,NumMap ] = HexMaker(hdorg,mcnpfix,NumMap,MatMap,DimMap )

%This function imports the driver data and then writes the  assembly.

%%
%Each case will be separate assembly type to be written. It will call a
%reader function for the dimensional data for that particular MICKA type.
%The case will load and pass the variables to the particular assm type

%%
% For information gathering purposes, this section will associate cell and
% surface numbers with assembly ids and MICKA numbers

% Initialize variables
global OldControlSA
global BorD
global surfnum
global cellnum
global SAsurfnum
global MSATyC MNumC
global SmearSATYs

SATy=MatMap{1,MSATyC};
Mnum=MatMap{1,MNumC};

NewDataC=false;
```

```
% Check for smeared assemblies

if ~isempty(SmearSATYs(Mnum==SmearSATYs(1:end,1)))

    [Mnrow,~]=find(Mnum==SmearSATYs(1:end,1));

    SATy=SmearSATYs(Mnrow,3);

end

bend=0; % Old variable but needed

%% Activate MICKA bending

if ~BorD && ~SATy==0

    SATy=SATy+10;

end

switch SATy

%% Benchmark Makers

    case 0
    % Used as an empty assm
        [ cellc,surfc ] = BlankMaker(hdorg);

    case 1

        [ cellc,surfc ] = DriverMakerMKII(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 2

        [ cellc,surfc ] = DriverMakerMKIIHFW(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 3

        [ cellc,surfc ] = SafetyMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 4

        [ cellc,surfc ] = HWCRMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 5

        [ cellc,surfc ] = ControlMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 6

        [ cellc,surfc ] = DummyMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 7

        [ cellc,surfc ] = ReflectorMaker(hdorg,mcnpfix,bend,MatMap,DimMap );

    case 8

        [ cellc,surfc ] = BlanketMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 9

        [ cellc,surfc ] = WallMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 10

        [ cellc,surfc ] = CustomMaker(hdorg,mcnpfix,bend,MatMap,DimMap); % Custom assemblies

        %% Katana Asssemblies

    case 11

        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 12

        [ cellc,surfc ] = DriverMakerMKIIHFWBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 13

        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 14 % HWCR

        % Only HWCRs that were at full insertion were modeled as a driver
        % with 61 pins.
        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 15 % Control

        [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 16 % Dummy
```

```
        [ cellc,surfc ] = DummyMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 17 % Reflector

        [ cellc,surfc ] = ReflectorMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap );

    case 18 % Blanket

        [ cellc,surfc ] = BlanketMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 19

        [ cellc,surfc ] = WallMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 20

        [ cellc,surfc ] = CustomMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap); % Custom assemblies

%% Smeared Makers by Ryan

    case 22

        [ cellc,surfc,MatMap ] = DriverHWMakerS(hdorg,mcnpfix,bend,MatMap,DimMap);
        NewDataC=true;

    case 26

        [ cellc,surfc,MatMap ] = DummyMakerS(hdorg,mcnpfix,bend,MatMap,DimMap );
        NewDataC=true;

    case 27

        [ cellc,surfc,MatMap ] = ReflectorMakerS(hdorg,mcnpfix,bend,MatMap,DimMap );
        NewDataC=true;

    case 28

        [ cellc,surfc,MatMap ] = BlanketMakerSP1(hdorg,mcnpfix,bend,MatMap,DimMap);
        NewDataC=true;

end

%% Need to reformat the data output from the smeared materials

if NewDataC

    datac=MatMap2DataC(MatMap);

else

    datac=[];

end

Oldsurfnum=surfnum;
Oldcellnum=cellnum;
OldSAsurfnum=SAsurfnum;
RowNum=hdorg(1)+1;

Newsurfnum=surfnum-1;
Newcellnum=cellnum-1;

NumMap{RowNum,6}=Oldcellnum;
NumMap{RowNum,7}=Newcellnum;
NumMap{RowNum,8}=linspace(Oldcellnum,Newcellnum,Newcellnum-Oldcellnum+1);
NumMap{RowNum,9}=Oldsurfnum;
NumMap{RowNum,10}=Newsurfnum;
NumMap{RowNum,11}=linspace(Oldsurfnum,Newsurfnum,Newsurfnum-Oldsurfnum+1);
NumMap{RowNum,12}=OldSAsurfnum;

end

function datac=MatMap2DataC(MatMap)

global     MNumC SAPosC SANomen SAlphaBetas MSATyC SABcorr SmallValCutOff FdC
persistent firstrun
persistent SATYsab

if isempty(firstrun) && SABcorr;

    % s(a,b) additions

    SATYsab=cell2mat(SAlphaBetas(:,2));

end

MatMap{FdC+1}(1,:)=delNaN(MatMap{FdC+1}(1,:),'Convert all','num');

sabon=false;
SmallValCutOff=false;

dispPrintmat={'Smeared Material 1';...
              'Smeared Material 2';...
              'Smeared Material 3';...
              'Smeared Material 4';...
```

```
              'Smeared Material 5';...
              'Smeared Material 6';...
              'Smeared Material 7';...
              'Smeared Material 8';...
              'Smeared Material 9';...
              'Smeared Material 10';...
              'Smeared Material 11';...
              'Smeared Material 12';...
              'Smeared Material 13';...
              'Smeared Material 14';...
              'Smeared Material 15';...
              'Smeared Material 16';...
              'Smeared Material 17';...
              'Smeared Material 18';...
              'Smeared Material 19';};

datatitletemp1=NameCard(['MICKA #: ' num2str(MatMap{MNumC}) ' ID: ' ...
    MatMap{SANomen} ' POS: ' MatMap{SAPosC}],'Title');

SATy=MatMap{MSATyC};

% The following statement adds sab corrections to the xsec

if any(SATy==SATYsab) && SABcorr; sabon=true; else sabc=0; end

for kk=6:3:size(MatMap,2)

    if isempty(MatMap{kk})

            datactemp=NameCard(['No ' dispPrintmat{(kk/3)-1}],'Divider');

            if kk==6

                datac=[datatitletemp1; datactemp];

            else

                datac=[datac; datactemp];

            end

        else

            MatNum=MatMap{kk};
            MatDb=MatMap{kk+2};

            if size(MatDb,1)>2

                datatitletemp2=NameCard(dispPrintmat{(kk/3)-1},'Divider');

                for hh=1:length(MatNum)

                    datatitletemp3=NameCard(['Pin Sec: ' num2str(hh)],'Divider');

                    datactemp=CharChecker(MICKAtoMCNPmat(MatNum(hh),...
                        [MatDb(1,:); MatDb(hh+1,:)]));

                     if sabon; [ sabc ] = SAlphaBeta( MatNum(hh), MatDb, SATy); end

                     if kk==6 && hh==1

                        datac=[datatitletemp1; datatitletemp2; datatitletemp3; datactemp];

                     else

                        datac=[datac; datatitletemp3; datactemp];

                     end

                end

            else

                datatitletemp2=NameCard(dispPrintmat{(kk/3)-1},'Divider');
                datactemp=CharChecker(MICKAtoMCNPmat(MatNum,MatDb));

                 if sabon; [ sabc ] = SAlphaBeta( MatNum, MatDb, SATy); end

                 if ~iscell(sabc)

                     if kk==6

                        datac=[datatitletemp1; datatitletemp2; datactemp];

                     else

                        datac=[datac; datatitletemp2; datactemp];

                     end

                 else

                     if kk==6

                        datac=[datatitletemp1; datatitletemp2; datactemp; sabc];
```

```
                    else

                        datac=[datac; datatitletemp2; datactemp; sabc];

                    end

                end

            end

    end

end

SmallValCutOff=true;


end
```

B.45. DriverMakerMKII.m

```
function [ cellc,surfc ] = DriverMakerMKII(hdorg,mcnpfix,bend,MatMap,DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugDrNFE
global surfnum cellnum
global hexsurfnum planesurfnum fbPl SAsurfnum createplane
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC feoxC fewwdC nfeC fepC fetphC fesphC

%% Material Vars

global FmC FdC FNamC FNadC FPGmC FPGdC FCladmC FCladdC FPwwrmC FPwwdC
global DuctmC DuctdC DnamC DnadC SUPmC SUPdC SLOmC SLOdC LAmC LadC

%**************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%**************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt); bh=DimMap{bhC};
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC}+mcnpfix;
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Fuel slug section heights
fssh=fsh/3;

%Convert The diameters in radi
hdor=hdod/2;
hdir=hdid/2;
feor=feod/2;
lcr=lcd/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;

% Set the number of lattice elements if applicable
if LATp; nle=nfe; end

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugDrNFE;  end

%**************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');
```

```
    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LUN=hdorg(1)*1000+1;

    LUNu=strjoin({'U=' Num2StrM(LUN)},'');
    LUNf=strjoin({'fill=' Num2StrM(LUN)},'');

    LUNfp=LUN+1;

    % Since this is a driver, the pins are all the same universe number.
    Latnums=repmat(LUNfp,1,nfe);

    % Debugging line to pad the universemap with zeros.
    if debugMICKA; Latnums=[repmat(LUNfp,1,nfe), zeros(1,nle-nfe)]; end

    % Lattice Window
    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatEleZo=-feh/2;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice boundings
    LATr=hdir-(hdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;
    LATzo=mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;

end

%*********************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=fslas+fsh+feoz;

if createplane

    [ fbPl, surfPlane ] = PinPlaneMaker( nfslas, fslas, fssh );

end

pinparm={hdorg(1),fbPl,nfslas,UNICfp,mcnpfix};

%*********************************************************************
% Duct Materials

% Duct and Cylinder
```

```
Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%**********************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap
FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

%%
%**********************************************************************

%Assigns dimensions and materials for the pins.
pind=[nfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];

%**********************************************************************

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.
hdsorg=ThermalXHex(crh,hdorg,bend);

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

% If the assembly is experimental, then the comment line will reflect that

if MatMap{MSATyC}==10

    surfname=NameCard(['EXP Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    surfname=NameCard(['Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end

%%
% The following for loop creates the core region sections.


%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
ODOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)-leh-lch-hdwt,0,0,crh+ueh+leh+lch+2*hdwt,...
hdor, '$ Outer Wall of Hex Duct'});

%Assign the surface number to be used later in the cell creation
```

```
surf1=SAsurfnum;

%Update the surnum number
SAsurfnum=SAsurfnum+1;

if LAT || LATp

    LATNas=CharChecker({surfnum,'    SO',2*hdh,...
    '$ Surr Na For Lat'});

    surf15=surfnum;

    surfnum=surfnum+1;

end

%Creates the inner duct for the upper extension.

% Inner Duct Upper Extention

UEs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)+crh,0,0,ueh,hdir,...
'$ Upper Extent Inner Hex'});

%Assign the surface number to be used later in the cell creation
UEInum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Inner Duct Core Region
CRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4),0,0,crh,hdir,...
'$ Inner Wall Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf2=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Need to add a plane

if createplane

    CRs=[CRs; surfPlane];

end

if (LAT || LATp); surfctemp=[ODOWs; LATNas; UEs; CRs ];
else    surfctemp=[ODOWs; UEs; CRs ]; end

%%
% Lower Extension Surface Cards

% Inner Duct
LEs=CharChecker({surfnum,'    RHP',hdorg(2),...
    hdorg(3),hdorg(4),0,0,-leh,hdir,' $ Lower Extent Inner Hex'});

%Assign the surface number to be used later in the cell creation
LEInum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

ODLPls=CharChecker({surfnum,'    PZ',-leh-hdwt,...
                '$ Plane Separation Duct to Cylinder'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Lower Cylinder Extension

ODCyls=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-leh-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

%Assign the surface number to be used later in the cell creation
ODCylnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Lattice surfaces.

if LATp

    %Naming card
    LATname=NameCard('Pin Lattice Cards for Elements','Divider');

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LATzo ...
        ,0,0,LATh,LATr,'$ Element Lattice Bounding Surface'});
```

```
    LATpArnum=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatEleZo,...
        0,0,LatEleh,0,LatEler,0,'$ Lattice Window'});

    LatWinElenum=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfc=[LATname; LATArs; LatWinEle];

end

% Append the surfaces to the surface card.
surfctemp=[surfctemp; LEs;  ODLPls;  ODCyls];

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={surf1};

else

    hexsurfnum=[hexsurfnum, {surf1}];

end

%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.

porg=PinOrg(nfe,fep,hdsorg(2),hdsorg(3),hdorg(4)+feoz);

% The pin origin needs to be moved slightly
if LATp; porg(:,2)=hdsorg(2)-LATpino; porg(:,3)=hdsorg(3); end

for ii=1:nfe

%Like the main program, the origins of the pins need to be determined so
%that they can be fed into a program to write the cards. This is what the
%pinorg program does.

%ocws = Oyter Cladding Wall Surface
%wws = wire wrap surface

    [ pincellctemp, pinsurfctemp, ocws , wws ]=...
        PinMaker(porg(ii,:),pind,pinm,pinparm);

    pinsurfnum(ii)=ocws;
    wirewrapsurfnum(ii)=wws;

    if (ii==1)

        FPs=pinsurfctemp;
        FPc=pincellctemp;

    end

    if (ii>1)

        FPs=[FPs; pinsurfctemp];
        FPc=[FPc; pincellctemp];

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
```

```
%assemblies

if MatMap{MSATyC}==10

    cellname=NameCard(['EXP Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    cellname=NameCard(['Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end

%% %Hex Duct Cell cards

%**************************** Benchmark *********************************

% Outter Duct Wall

ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
    surf2(1),UEInum,LEInum,ODLPlnum,UNIC,'$ Hex Duct'});

cellnum=cellnum+1;

% Inside Duct Upper Extent Homog
UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -UEInum,...
    UNIC,'$ Homog Upper Ext'});

cellnum=cellnum+1;

%Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

%Inside duct Lower Extension Homog

LEc=CharChecker({cellnum,'     ',SLOm,SLOd, -LEInum,...
    UNIC,'$ Homog Lower Ext'});

cellnum=cellnum+1;

%Sodium around the lower extension cylinder

CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1,...
    -ODLPlnum,ODCylnum,UNIC,'$ Na Surr Lower Ext'});

cellnum=cellnum+1;

% Lower Cylinder

Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
    UNIC,'$ Lower Cyn Homog'});

cellnum=cellnum+1;

if LAT==1

    LATNac=CharChecker({cellnum,'     ',Dnam,...
        Dnad,surf1,-surf15,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

end

if LATp

    %name the area

    LATpcellname=NameCard('Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum,'Lat=2',LUNu,...
    'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN, nle ,nar, Latnums );

    LATpcLns=CharChecker({'     ' Num2StrM(LATparln)});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum,LUNf,...
    UNIC,' $ Pin Lattice'});

    cellnum=cellnum+1;
```

```
    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

end

%***************************** Benchmark *********************************

if LATp

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1),...
        wirewrapsurfnum(1),UNICfp, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

    CRc=CharChecker([CRc,LATpArnum,UNIC,...
        '$ Sodium coolant']);

else

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm)];

    end

    %Taks on the importance and comment

    CRc=CharChecker([CRc,UNIC,'$ Sodium coolant']);

end

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LATp

    if LAT

        cellctemp=[ODc; LATNac; UEc; CRc; FPc; FPNac; LATpc; LEc;...
            CylNac; Cylc];

    else

        cellctemp=[ODc; UEc; CRc; FPc; FPNac; LATpc; LEc;...
            CylNac; Cylc];

    end

else

    if LAT

        cellctemp=[ODc; LATNac; UEc; CRc; FPc; LEc;...
            CylNac; Cylc];

    else

        cellctemp=[ODc; UEc; CRc; FPc; LEc;...
            CylNac; Cylc];

    end

end

%***************************** Benchmark *********************************

%% Appending the cards together

if LATp

    surfc=[surfname; surfctemp; FPs; LATsurfc];
    cellc=[cellname; cellctemp];

else

    surfc=[surfname; surfctemp; FPs];
    cellc=[cellname; cellctemp];
```

```
end

%%

global SaveSAMakerVars

if SaveSAMakerVars

    if MatMap{MSATyC}==10

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_EXP_' MatMap{SANomen} '_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    else

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Dr_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    end

end

end
```

B.46. PinOrg.m

```
function [ porg ] = PinOrg( np,pp,xor,yor,zor )

%%
%*************************Preload Variables*******************************


%porg stands for "Pin origins"
%This will be an array which contains the following array information
%porg={Pin number,xp,yp,zp}
%xh,yh,zh are the coordinates for the origin of the hex duct.
porg=zeros([np,4]);

%%
%***************Calculation of the Origins of the ducts********************

%This section calculates the center of each of the hexagonal ducts



%The number of rings needs to be determined so that the origin
%functions can be written. Since the number of pins per side
%matches that of the number of rings, the equation to determine number
%of pin will be rewritten to solve for number of rings.

nr=ceil((3+sqrt(12*np-3))/6);

%an Array must be created that lists the number of assemblies in each
%ring.

for i=1:nr

    %The following calculates the number of elements in each ring plus the
    %total number of elements for the ith ring plus all previous rings

    npr(i,:)=[i,6*(i-1),3*(((i-1)^2)+i)-2];

    %This if statement checks to see if it it the first element in the
    %problem.
    if i==1

        npr(1,:)=[1,1,1];

    end

    %This if statement checks to see if there can only be a partial ring
    %and sets the number of the partial ring.
    if npr(i,3)>np
        npr(i,:)=[i,np-sum(npr(1:i-1,2)),np];
    end
end


%The change in x and the change in y from one assembly to another is a
%constant, therefore they only need to be calculated once and then just
%added or subtracted.

%Calculation of the x dimension.
%xpt stands for "x pitch" ypt stands for "y pitch" zpt stands for "z pitch"
%The zpt will be distance from the bottom of the outter hex to the bottom
%for the pin.
    xpt=pp*cosd(30);
    ypt=pp*0.5;
    zpt=0.02;
```

```
%Dummy counting variable to keep track of which element the code is
%calculating the origin.
    nptemp=1;


%The following for loop has if statements. The first if statement sets the
%center hex duct at 0,0,0. The second if statement contains a for loop
%which builds the hect duct for each ring.

for i=1:nr

    %Three more variables are needed as temporary variables to keep track of
    %where the travel has taken place.
        xpttemp=xor;
        ypttemp=yor;
        zpttemp=zor;

    %This if statement sets the origin of the first pin which is
    %located at hdorg

    if i==1

        % The first sub assembly is at the origin 0,0,0
        %Assign the sub assm number and origin.
        porg(i,:)=[i,xpttemp,ypttemp,zpttemp];

    end


    if i>=2

    %Vertex 1
        nptemp=nptemp+1;
        ypttemp=ypttemp+(i-1)*2*ypt;
        porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
        if nptemp==np
            break;
        end

    %Vertex 2
        for j=1:i-1
            nptemp=nptemp+1;
            xpttemp=xpttemp+xpt;
            ypttemp=ypttemp-ypt;
            porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
            if nptemp==np
                break;
            end
        end

        if nptemp==np
            break;
        end

     %Vertex 3
        for j=1:i-1
            nptemp=nptemp+1;
            ypttemp=ypttemp-2*ypt;
            porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
            if nptemp==np
              break;
            end
        end

        if nptemp==np
            break;
        end

     %Vertex 4
        for j=1:i-1
            nptemp=nptemp+1;
            xpttemp=xpttemp-xpt;
            ypttemp=ypttemp-ypt;
            porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
            if nptemp==np
                break;
            end
        end

        if nptemp==np
            break;
        end

     %Vertex 5
        for j=1:i-1
            nptemp=nptemp+1;
            xpttemp=xpttemp-xpt;
            ypttemp=ypttemp+ypt;
            porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
            if nptemp==np
                break;
            end
        end

        if nptemp==np
```

```
            break;
        end

     %Vertex 6
        for j=1:i-1
            nptemp=nptemp+1;
            ypttemp=ypttemp+2*ypt;
            porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
            if nptemp==np
                break;
            end
        end

        if nptemp==np
            break;
        end

      %Vertex 7
        for j=1:i-2
            nptemp=nptemp+1;
            xpttemp=xpttemp+xpt;
            ypttemp=ypttemp+ypt;
            porg(nptemp,:)=[nptemp,xpttemp,ypttemp,zpttemp];
            if nptemp==np
                break;
            end
        end

        if nptemp==np
            break;
        end

    end


end


end
```

B.47. PinMaker.m

```
function [ cellc, surfc , ocws , wr] = PinMaker(porg,pind,pinm,pinparm)

%This function creates the pins in the same manner as the hexmaker.
% MKIIA pin

%This function Designs a pin as follows

% A slug immersed in fluid
% Fluid Plenum above the immersed slug
% Cladding around both fluids
% Wire Wrap around the cladding

global surfnum
global cellnum

global BorD

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

% These lines set the plane surfnums.

if size(pinm,1)>1

    % These lines set the plane surfnums.

    PinPlanes=pinparm{2};

    gasbarsurfnum=PinPlanes{1,1};
    sec12surfnum=PinPlanes{2,1};
    sec23surfnum=PinPlanes{3,1};

else

    PinPlanes=pinparm{2};

    gasbarsurfnum=PinPlanes{1,1};

end

%Sets the MICKA num
hdnum=pinparm{1};

%Fuel sodium level
fslas=pinparm{3};
```

```
%Sets the universe number
UNIC=pinparm{4};

%Mcnpfix
mcnpfix=100*pinparm{end};

%%
%****************Define the pin material parameters**********************

%slug material and density

if BorD && size(pinm,1)>1

    pinmln=3;

    slugs1m=pinm(pinmln-2,1);
    slugs1d=pinm(pinmln-2,2);
    slugs2m=pinm(pinmln-1,1);
    slugs2d=pinm(pinmln-1,2);
    slugs3m=pinm(pinmln,1);
    slugs3d=pinm(pinmln,2);

else

    pinmln=1;
    slugs1m=pinm(pinmln,1);
    slugs1d=pinm(pinmln,2);

end

%cladding material and density
cladm=pinm(pinmln+1,1);
cladd=pinm(pinmln+1,2);

%wire wrap material and density
wwrm=pinm(pinmln+2,1);
wwrd=pinm(pinmln+2,2);

%Coolant material and density
nam=pinm(pinmln+3,1);
nad=pinm(pinmln+3,2);

%plenum gas material and density

PGm=pinm(pinmln+4,1);
PGd=pinm(pinmln+4,2);

%%
%*********************************Pin Dimensions***************

% Spade height This will change the pin height to accomodate a spade

%Height of fuel from the bottom of the spade to bottom of fuel slug

%Fuel slug height
fsh=pind(2);

%Fuel Slug Radius
fsr=pind(3);

%cladding wall thickness
fewt=pind(4);

%Cladding inner radius
feir=pind(5);

%Pin height
feh=pind(6);

%Pin radius
feor=pind(7);

%Wire Wrap radius
fewwr=pind(8);

%Wire Wrap Length
fewwh=pind(9);

% Fuel Plug Top Length
fetph=pind(10);

% Spade Plug length
fesph=pind(11);

%%
%*******************************Surface Cards*****************************

%Since the cards for the pin and the subsequent pin sections will become
%huge, I will put a title in the cards for each pin now.

surfname=NameCard(['Pin: ' num2str(porg(1)) ...
                   ' of SA: ' num2str(hdnum)],'Divider');

%Outer Diameter Cladding
```

```
surfctemp=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)...
    ,0,0,feh,feor,'$ Pin:',porg(1),'Outer Cladding wall'});

%Assign the surface number to be used later in the cell creation
ocws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Inner Diameter Cladding

surfctemp2=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+fesph...
    ,0,0,feh-(fetph+fesph),feir,...
    '$ Pin:',porg(1),'Inner cladding wall'});

%Assign the surface number to be used later in the cell creation
icws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Fuel slug cylinder

surfctemp3=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+fesph+mcnpfix...
    ,0,0,fsh,fsr,'$ Pin:',porg(1),'Fuel slug boundary'});

%Assign the surface number to be used later in the cell creation
fs=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%WireWrap
%This will have to call another function to calculate and generate the
%wirewrap positions. For right now we are just going to do one cylinder
%running up the side of the outter pin cladding. The cylinder will be
%parrallel to the pin with a 90 degree offset in the xy plane.

surfctemp4=CharChecker({surfnum,'    RCC',porg(2)+feor+fewwr+mcnpfix,...
    porg(3),porg(4),0,0,fewwh,fewwr,'$ Pin:',porg(1),'Wire Wrap'});

%Since the wrap is on the outside of the pin, it also needs to give its
%number to pinsurf so that hex maker can know the sodium is on the outside.

wr=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%%
%*****************************Cell Cards*******************************
% Pin number title to seperate the pins

cellname=NameCard(['Pin: ' num2str(porg(1)) ...
    ' of SA: ' num2str(hdnum)],'Divider');

% These if statements are to check if this is the top of the hex duct section.
% the impact of this is that the top plane does not need to be used.

% Top part above the top of the element and above the sodium height.

%***************************** Benchmark *********************************
%Creates a fuel element

% Cladding

cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
    icws,UNIC,'$ Pin:',porg(1),' Cladding'});

cellnum=cellnum+1;

% Helium
try
cellctemp1=CharChecker({cellnum,'     ',PGm,PGd, gasbarsurfnum,...
    -icws,UNIC,'$ Pin:',porg(1),' Plenum Gas'});
catch;keyboard;end
cellnum=cellnum+1;

%Fuel sec1

cellctemp2=CharChecker({cellnum,'     ',slugs1m,slugs1d, -fs,sec12surfnum,...
      UNIC,'$ Pin:',porg(1),' Fuel Slug Sec 1'});

 cellnum=cellnum+1;

if pinmln==3

    %Fuel sec2

    cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',slugs2m,slugs2d,...
        -fs,-sec12surfnum, sec23surfnum,...
        UNIC,'$ Pin:',porg(1),' Fuel Slug Sec 2'})];

     cellnum=cellnum+1;
```

```
    %Fuel sec2

    cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',slugs3m,slugs3d, -fs,...
          -sec23surfnum, UNIC,'$ Pin:',porg(1),' Fuel Slug Sec 3'})];

     cellnum=cellnum+1;

end

% Sodium

cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',nam,nad,...
     -gasbarsurfnum, -icws,fs,UNIC,'$ Pin:',porg(1),' Sodium'})];

cellnum=cellnum+1;

%Wire Wrap

cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',wwrm,wwrd,...
    -wr,UNIC,'$ Pin:',porg(1),' Wire Wrap'})];

cellnum=cellnum+1;

%***************************** Benchmark *********************************

%Append the cell cards

cellc=[cellname; cellctemp; cellctemp1; cellctemp2];

%%
%Assign the cards to be returned to hexmaker.
surfc=[surfname; surfctemp; surfctemp2; surfctemp3; surfctemp4];

%%

global SaveSAMakerVars
persistent FirstRun

FirstRun=true;

if SaveSAMakerVars && FirstRun

    % Save TotDimDat
    save('Debug\SAmakerVar\Tot_PoisonPin_Dat.mat')
    dispPrint('     TotDat_PoisonPin_saved....');

    FirstRun=false;

end

end
```

B.48. MCNPPOSMaker.m

```
function [ MCNPlatln,MCNPlatTemp ] = MCNPPOSMaker( LUN,nsa,nar,Latnums )

%%
%**************************Preload Variables*******************************

%hdorg stands for "Hex duct origins"
%This will be an array which contains the following array information
%hdorg={SubAssm number,xh,yh,zh}
%xh,yh,zh are the coordinates for the origin of the hex duct.


%%
%***************Calculation of the Origins of the ducts*******************

%This section calculates the center of each of the hexagonal ducts

%The number of rings needs to be determined so that the origin
%functions can be written. Since the number of assemblies per side
%matches that of the number of rings, the equation to determine number
%of assemblies will be rewritten to solve for number of rings.

nr=ceil((3+sqrt(12*nsa-3))/6)+nar;

%an Array must be created that lists the number of assemblies in each
%ring.

for kk=1:nr


    %This if statement checks to see if it it the first element in the
    %problem.
    if kk==1

        nsar(1,:)=[1,1,1];

    else

    %The following calculates the number of elements in each ring plus the
    %total number of elements for the ith ring plus all previous rings
```

```
       nsar(kk,:)=[kk,6*(kk-1),3*(((kk-1)^2)+kk)-2];

    end

    %This if statement checks to see if there can only be a partial ring
    %and sets the number of the partial ring.

    if nsar(kk,3)>nsa
        nsar(kk,:)=[kk,nsa-sum(nsar(1:kk-1,2)),nsa];
    end

end


%The change in x and the change in y from one assembly to another is a
%constant, therefore they only need to be calculated once and then just
%added or subtracted.

%Calculation of the x dimension.
%xpt stands for "x pitch" ypt stands for "y pitch" zpt stands for "z pitch"


%Dummy counting variable to keep track of which assembly the code is
%calculating the origin.
    nsatemp=1;


%The following for loop has if statements. The first if statement sets the
%center hex duct at 0,0,0. The second if statement contains a for loop
%which builds the hect duct for each ring.
    MCNPlatTemp=zeros(nr*2-1,nr*2-1);

%Three more variables are needed as temporary variables to keep track of
%where the travel has taken place.
    xpt=nr;
    ypt=nr;

for kk=1:nr

    %This if statement sets the origin of the first hex duct which is
    %located at 0,0,0

    if kk==1

    % The first sub assembly is at the origin 0,0,0
    %Assign the sub assm number and origin.
    MCNPlatTemp(xpt,xpt)=Latnums(nsatemp);

    end

    if kk>=2

        if kk==2

       %Vertex 1
            nsatemp=nsatemp+1;
            %xpt=xpt;
            ypt=ypt+1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);

            if nsatemp==nsa
                break;
            end

        else

       %Vertex 1
            nsatemp=nsatemp+1;
            xpt=nr;
            ypt=nr+kk-1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);

            if nsatemp==nsa
                break;
            end

        end

    %Vertex 2
        for j=1:kk-1
            nsatemp=nsatemp+1;
            xpt=xpt+1;
            ypt=ypt-1;
            try
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);
            catch
                keyboard;
            end
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
```

```
            break;
        end

     %Vertex 3
        for j=1:kk-1

            nsatemp=nsatemp+1;
            ypt=ypt-1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);

            if nsatemp==nsa
              break;
            end

        end

        if nsatemp==nsa
            break;
        end

     %Vertex 4
        for j=1:kk-1
            nsatemp=nsatemp+1;
            xpt=xpt-1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
            break;
        end

     %Vertex 5
        for j=1:kk-1
            nsatemp=nsatemp+1;
            xpt=xpt-1;
            ypt=ypt+1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
            break;
        end

     %Vertex 6
        for j=1:kk-1
            nsatemp=nsatemp+1;
            ypt=ypt+1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
            break;
        end

      %Vertex 7
        for j=1:kk-2
            nsatemp=nsatemp+1;
            xpt=xpt+1;
            MCNPlatTemp(ypt,xpt)=Latnums(nsatemp);
            if nsatemp==nsa
                break;
            end
        end

        if nsatemp==nsa
            break;
        end

    end


end

MCNPlatTemp(MCNPlatTemp==0)=LUN;

% Take the array and place it into a one dimensional array

for ii=1:size(MCNPlatTemp,1)

   MLTln=MCNPlatTemp(ii,:);

   if ii==1

       MCNPlatln=MLTln;

   else
```

```
       MCNPlatln=[MCNPlatln, MLTln];

   end


end


end
```

B.49. DriverMakerMKIIHFW.m

```
function [ cellc,surfc ] = DriverMakerMKIIHFW(hdorg,mcnpfix,bend, MatMap, DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugDrHFWNFE DebugDrHFWNDE
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum fbPl createplane
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC fewwdC nfeC fepC ndeC fetphC fesphC
global feoxC

%% Material Vars

global FmC FdC FNamC FNadC FPGmC FPGdC  FCladmC FCladdC FPwwrmC FPwwdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC LAmC LadC DmC DdC

DebugStruct=false;

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt); bh=DimMap{bhC};
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC}+mcnpfix;
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Dummy Elements, they are the same as fuel elements but solid ss
deod=feod; deid=feid; deh=feh; dewt=fewt; dewwd=fewwd;...
    dewwh=fewwh; nde=DimMap{ndeC};

% Fuel slug section heights
fssh=fsh/3;

%Convert The diameters in radi
hdor=hdod/2;
hdir=hdid/2;
feor=feod/2;
lcr=lcd/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;
deor=deod/2;
deir=deid/2;
dewwr=dewwd/2;

% Set the number of lattice elements if applicable
if LATp; nle=nfe+nde; end


% Override pin count if debug micka is true
if debugMICKA;  nfe=DebugDrHFWNFE; nde=DebugDrHFWNDE; end


%Wire Wrap Switch 1 = on 0 = off.
```

```
dewws=1;

% Solid dummy element 1=solid 0=hollow;
dess=1;

%**************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;


else

    UNIC=' IMP:N=1';

end

if LATp==1

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LUN=hdorg(1)*1000+1;

    LUNu=strjoin({'U=' Num2StrM(LUN)},'');
    LUNf=strjoin({'fill=' Num2StrM(LUN)},'');

    LUNfp=LUN+1;
    LUNdp=LUNfp+1;

    % Since this is a driver, the pins are all the same universe number.
    % Latnums=repmat(LUNfp,1,nfe);
    % The previous line is commented out because a HFW driver has its own pin
    % map that it needs to follow.

    % Debugging line to pad the universemap with zeros.
        if debugMICKA; LatnumsT=...
                [repmat(LUNdp,1,nde), repmat(LUNfp,1,nfe), zeros(1,nle-nfe-nde)]; end

    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatElezo=-feh/2;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice array size

    LATr=hdir-(hdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;
    LATzo=mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins
    nde=1;

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');
    UNICdp=strjoin({' U=' Num2StrM(LUNdp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;
    UNICdp=UNIC;

end
```

```
%*********************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=fslas+fsh+feoz;

if createplane

    [ fbPl, surfPlane ] = PinPlaneMaker( nfslas, fslas, fssh );

end

pinparm={hdorg(1),fbPl,nfslas,UNICfp,mcnpfix};
dpinparm={hdorg(1),fbPl,nfslas,UNICdp,mcnpfix};

%*********************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%*********************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap
FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

% Dummy Pins

% Dummy Fuel Slug
Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Dummy Fuel Slug
Deim=Dm;
Deid=Dd;

% Wire Wrap
Dewwrm=FWwrm;
Dewwrd=FWwrd;

%*********************************************************************
%Assigns dimensions and materials for the pins.
pind=[nfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
dpind=[nfslas;NaN;NaN;dewt;deir;deh;deor;dewwr;dewwh;dewws;dess];

%Passed to the pins
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];
dpinm=[Dm,Dd;Deim,Deid;Dewwrm,Dewwrd];
```

```
%*********************************************************************
%The following is a halfworth driver map to map where the dummy pins go.

pinmap=[1;1;2;2;1;2;2;2;2;1;1;2;1;2;2;1;1;2;1;2;1;2;1;1;2;2;1;2;1;1;2;2;1;...
    1;2;1;1;1;2;2;1;1;2;2;1;2;1;2;1;1;2;2;1;1;2;2;1;2;1;2;1;2;2;1;1;2;2;1;...
    1;2;2;1;2;1;2;1;2;1;2;2;1;1;2;2;1;1;2;2;1;2;1;2;1];

hwdmap=[linspace(1,nfe+nde,nfe+nde)' pinmap(1:nfe+nde)];

% Changes the pin map and hwdmap to a lattice.

if LATp

    hwdmap=[1,1;1,2];
    Latnums=pinmap';

    % Create the lattice map from the pinmap by replacing 1 and 2's with
    % universe numbers.

    Latnums(Latnums==1)=LUNdp;
    Latnums(Latnums==2)=LUNfp;

end

% Debug
if debugMICKA && LATp; Latnums=LatnumsT; end

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);

%%
%Debug only
%
% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Half Worth Driver MKIIA M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%%
% The following for loop creates the core region sections.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
ODOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)-leh-lch-hdwt,0,0,crh+ueh+leh+lch+2*hdwt,...
hdor, '$ Outer Wall of Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf1=SAsurfnum;

%Update the surnum number
SAsurfnum=SAsurfnum+1;

if (LAT || LATp)

    LATNas=CharChecker({surfnum,'    SO',2*hdh,...
        '$ Surr Na For Lat'});

    surf15=surfnum;

    surfnum=surfnum+1;

end

%Creates the inner duct for the upper extension.

% Inner Duct Upper Extention

UEs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)+crh,0,0,ueh,hdir,...
'$ Upper Extent Inner Hex'});

%Assign the surface number to be used later in the cell creation
UEInum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Inner Duct Core Region
CRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4),0,0,crh,hdir,...
'$ Inner Wall Hex Duct'});
```

```
%Assign the surface number to be used later in the cell creation
surf2=surfnum;

%Update the surnum number
surfnum=surfnum+1;

 if createplane

     CRs=[CRs; surfPlane];

 end

if LAT || LATp; surfctemp=[ODOWs; LATNas; UEs; CRs ];
 else    surfctemp=[ODOWs; UEs; CRs ]; end

%%
% Lower Extension Surface Cards

% Inner Duct
LEs=CharChecker({surfnum,'    RHP',hdorg(2),...
hdorg(3),hdorg(4),0,0,-leh,hdir,' $ Lower Extent Inner Hex'});

%Assign the surface number to be used later in the cell creation
LEInum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

ODLPls=CharChecker({surfnum,'    PZ',-leh-hdwt,...
    '$ Plane Separation Duct to Cylinder'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Lower Cylinder Extension

ODCyls=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
hdorg(4)-leh-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

%Assign the surface number to be used later in the cell creation
ODCylnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Lattice surfaces.

if LATp

    %Naming card
    LATname=NameCard('Lattice Cards for Elements','Divider');

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LATzo ...
        ,0,0,LATh,LATr,'$ Element Lattice Bounding Surface'});

    LATpArnum=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezo...
        ,0,0,LatEleh,0,LatEler,0,'$ Lattice Window'});

    LatWinElenum=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfc=[LATname; LATArs; LatWinEle];

end

% Append the surfaces to the surface card.
surfctemp=[surfctemp; LEs;  ODLPls;  ODCyls];

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={surf1};

else

    hexsurfnum=[hexsurfnum, {surf1}];

end

%% Pin Creator
```

```
%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe+nde,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe+nde,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.

porg=PinOrg((nfe+nde),fep,hdsorg(2),hdsorg(3),hdorg(4)+feoz);

% The pin origin needs to be moved slightly
if LATp; porg(:,2)=hdsorg(2)-LATpino; porg(:,3)=hdsorg(3); end

for ii=1:(nfe+nde)

    %Like the main program, the origins of the pins need to be determined so
    %that they can be fed into a program to write the cards. This is what the
    %pinorg program does.

    %ocws = Oyter Cladding Wall Surface
    %wws = wire wrap surface

    %Dummy Pins

    if hwdmap(ii,2)==1

        [ pincellctemp, pinsurfctemp, ocws , wws]=...
            DummyPinMaker(porg(ii,:),dpind,dpinm,dpinparm);

    end

    %Fuel Pins

    if hwdmap(ii,2)==2

        [ pincellctemp, pinsurfctemp, ocws , wws ]=...
            PinMaker(porg(ii,:),pind,pinm,pinparm);

    end

        pinsurfnum(ii)=ocws;
        wirewrapsurfnum(ii)=wws;

        if (ii==1)

            FDPs=pinsurfctemp;
            FDPc=pincellctemp;

        end

        if (ii>1)

            FDPs=[FDPs; pinsurfctemp];
            FDPc=[FDPc; pincellctemp];

        end
end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['Half Worth Driver MKIIA M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%% %Hex Duct Cell cards

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Driver Half Worth #' num2str(hdorg(1))],'Divider');

%***************************** Benchmark *********************************

% Outter Duct Wall

ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
    surf2,UEInum,LEInum,ODLPlnum,...
    UNIC,'$ Hex Duct'});
```

```
cellnum=cellnum+1;


% Inside Duct Upper Extent Homog

UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -UEInum...
    UNIC ,'$ Homog Upper Ext'});

cellnum=cellnum+1;

%Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

%Inside duct Lower Extension Homog

LEc=CharChecker({cellnum,'     ',SLOm,SLOd, -LEInum...
    UNIC ,'$ Homog Lower Ext'});

cellnum=cellnum+1;

%Sodium around the lower extension cylinder

CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1,...
    -ODLPlnum,ODCylnum,UNIC ,'$ Na Surr Lower Ext'});

cellnum=cellnum+1;

% Lower Cylinder

Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
    UNIC ,'$ Lower Cyn Homog'});

cellnum=cellnum+1;

if LAT

     LATNac=CharChecker({cellnum,'     ',Dnam,...
         Dnad,surf1,-surf15,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

end

if LATp

    %name the area

    LATpcellname=NameCard('Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum,'Lat=2',LUNu,...
        'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN, nle ,nar, Latnums );

    LATpcLns=CharChecker({'     ' num2str(LATparln)});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum,LUNf,...
        UNIC,' $ Pin Lattice'});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

end

%***************************** Benchmark *********************************

if LATp

    PNaname=NameCard('Surrounding sodium Cell for Pins','Divider');

    % This defines the surrounding sodium around the dummy pin.

    DPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1),...
        wirewrapsurfnum(1),UNICdp, '$ Fuel Dummy Pin Cell'});

    cellnum=cellnum+1;

    % This defines the surrounding sodium around the fuel pin.
```

```
    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(2),...
        wirewrapsurfnum(2),UNICfp, '$ Fuel Dummy Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnum,UNIC,...
            '$ Sodium coolant']);


else

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc,UNIC,...
            '$ Sodium coolant']);

end

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LATp

    if LAT

         cellctemp=[ODc; LATNac; UEc; CRc; FDPc; PNaname; DPNac;...
            FPNac; LATpc; LEc; CylNac; Cylc];

    else

         cellctemp=[ODc; UEc; CRc; FDPc; PNaname; DPNac;...
            FPNac; LATpc; LEc; CylNac; Cylc];

    end

else

    if LAT

        cellctemp=[ODc; LATNac; UEc; CRc; FDPc; LEc;...
            CylNac; Cylc];

    else

        cellctemp=[ODc; UEc; CRc; FDPc; LEc;...
            CylNac; Cylc];

    end

end


%***************************** Benchmark *********************************

%% Appending the cards together

if LATp

    surfc=[surfname; surfctemp; FDPs; LATsurfc];
    cellc=[cellname; cellname2; cellctemp];

else

    surfc=[surfname; surfctemp; FDPs];
    cellc=[cellname; cellname2; cellctemp];

end

%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_DrHFW_Dat.mat'])
```

```
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.50. SafetyMaker.m

```
function [ cellc,surfc ] = SafetyMaker(hdorg,mcnpfix,bend,MatMap,DimMap)
%%This function writes the Safety Rods

%declare the global variables
global debugMICKA DebugSafeNFE
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC fewwdC nfeC fepC fetphC fesphC feoxC
global ihdhC ihdodC ihdwtC ihdlcdC ihdlchC sazoC saatC sazmC

%% Material Vars

global FmC FdC FNamC FNadC FPGmC FPGdC  FCladmC FCladdC FPwwrmC FPwwdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC IHDLAmC IHDLAdC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
bh=DimMap{bhC}; crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC};
lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Inner Hex Duct
ihdh=DimMap{ihdhC}; ihdod=DimMap{ihdodC}; ihdwt=DimMap{ihdwtC};
ihdid=ihdod-(2*ihdwt); ihdlcd=DimMap{ihdlcdC}; ihdlch=DimMap{ihdlchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC};
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Fuel slug section heights
fssh=fsh/3;

%Convert The diameters in radi
% Hex Ducts         %Fuel Elements
ihdor=ihdod/2;       feor=feod/2;
ihdir=ihdid/2;       fsr=fsd/2;
ihdlcr=ihdlcd/2;     feir=feid/2;
hdor=hdod/2;         fewwr=fewwd/2;
hdir=hdid/2;
lcr=lcd/2;

% Set the number of lattice elements if applicable
if LATp; nle=nfe; end

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugSafeNFE; end

%*************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end
```

```
if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LUN=hdorg(1)*1000+1;

    LUNu=strjoin({'U=' Num2StrM(LUN)},'');
    LUNf=strjoin({'fill=' Num2StrM(LUN)},'');

    LUNfp=LUN+1;

    % Since this is a driver, the pins are all the same universe number.
    Latnums=repmat(LUNfp,1,nfe);

    % Debugging line to pad the universemap with zeros.
    if debugMICKA; Latnums=[repmat(LUNfp,1,nfe), zeros(1,nle-nfe)]; end

    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatElezo=mcnpfix;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice array size

    LATr=ihdir-(ihdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;

end

%*************************************************************************
%Subassembly movement

%Since the control rod in the full insertion position is 8.255cm from the
%origin of the core 0,0,0 we need to lower the origin of the assembly by
%that much. hdorg(4) is the z reference. This should not be changeD!
%s.a.z.o (sub assembly z offset)

sazo=DimMap{sazoC};

%Total allowable travel, saat sub assembly allowbale travel

saat=DimMap{saatC};

%This following variable is the induced movement due to drive mechanism.
%Movement is cm of insertion. This means full insertion = 35.56. Full out =
%0 cm. sazm, sub assembly movement

sazm=DimMap{sazmC};

%*************************************************************************
%This code sets the barrier between gas/shield and sodium

%nfslas now becomes the height of the sodium block from the region. This is
%done because the planes are defined independant of the duct.

mfslas=fslas+fsh+feoz-sazo-saat+sazm;
pinparm={hdorg(1),[],mfslas,UNICfp,mcnpfix};
newpl=true;
```

```
%**************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SIHDLCm=MatMap{IHDLAmC};
SIHDLCd=MatMap{IHDLAdC};

%**************************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap

FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

%%
%**************************************************************************

%Assigns dimensions and materials for the pins.
pind=[mfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];

%**************************************************************************
%%

%This section is the control rod movement code. This will enable us to move
%the control subassembly up and down. hdorg(4) refers to the 0" z height.
%The rest of the assembly is built from this reference position including
%the KAtana effect.

% Copy HDorg so that it can be used on the outer sheth
Ohdorg=[hdorg(1:3), hdorg(4)-leh-sazo];

%Offest plus movement
hdorg(4)=hdorg(4)-sazo-saat+sazm;

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

Ohdsorg=ThermalXHex(hdh,Ohdorg,bend);
hdsorg=ThermalXHex(crh,hdorg,bend);

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Safety Rod MKIIC M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%%
% The following for loop creates the core region sections.
```

```
%A plane needs to be created that the pins use as the barrier between
% above the fluid level and below. It has to be defined per assembly
% because some assemblies move.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
IDOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)-ihdwt,0,0,crh+ueh+2*ihdwt,...
ihdor, '$ Outer Wall of Inner Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf1=SAsurfnum;

%Update the surnum number
SAsurfnum=SAsurfnum+1;

if (LAT || LATp)

 IDOWs=[IDOWs; CharChecker({surfnum,'    SO',2*hdh,...
     '$ Surr Na For Lat'})];

 surf15=surfnum;

 surfnum=surfnum+1;

end

% This plane caps off the outer duct.

ODUPls=CharChecker({surfnum,'    PZ', Ohdsorg(4)+hdh,...
            '$ Plane to cap outter hex duct'});

%I need this facet for later.
ODUPnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Outter Wall Outer Duct

ODOWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
Ohdsorg(3),Ohdorg(4)-hdwt,0,0,hdh+2*hdwt,hdor,...
'$ Duct Outer Tube Outer Wall'});

%Assign the surface number to be used later in the cell creation
ODOWnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Outter Wall Inner Duct

ODIWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
Ohdsorg(3),Ohdorg(4),0,0,hdh+hdwt,hdir,...
'$ Duct Outer Tube Inner Wall'});

%Assign the surface number to be used later in the cell creation
ODIWnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Creates the inner duct for the upper extension.
%Also calls the Poison Pins to be made

% Inner Duct Upper Extention

IDSUPs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)+crh,0,0,ueh,ihdir,...
'$ Upper Ext Inner Hex'});

%Assign the surface number to be used later in the cell creation
IDUEnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Inner Duct Core Region
IDCRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4),0,0,crh,ihdir,...
'$ Inner Wall Inner Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf2=surfnum;

%Update the surnum number
surfnum=surfnum+1;

if newpl

% Barrier between fluid and gas

 barrierpl=CharChecker({surfnum,'    PZ',mfslas,...
        '$ Sodium Gas Boundary'});
```

```
 bPl{1,1}=surfnum;
 bPl{1,2}=mfslas;

surfnum=surfnum+1;

if size(Fm,1)>1

    % Section  1/2 barrier

    barrierpl=[barrierpl; CharChecker({surfnum,'    PZ',mfslas-fslas-fssh,...
                    '$ Fuel Section Plane Sec 1/2'})];

    bPl{2,1}=surfnum;
    bPl{2,2}=mfslas-fslas-fssh;

    surfnum=surfnum+1;

    % Section  2/3 barrier

     barrierpl=[barrierpl; CharChecker({surfnum,'    PZ',mfslas-fslas-(2*fssh),...
                    '$ Fuel Section Plane Sec 2/3'})];

    bPl{3,1}=surfnum;
    bPl{3,2}=mfslas-fslas-(2*fssh);

    surfnum=surfnum+1;

end

pinparm(1,2)={bPl};
IDCRs=[IDCRs; barrierpl];

newpl=false;

end

surfcUP=[ surfname; ODUPls;...
     ODOWs; ODIWs; IDOWs; IDSUPs; IDCRs ];

%%
% Lower Extension Surface Cards

% This plane caps off the lower tube

ODLPc=CharChecker({surfnum,'    PZ',Ohdorg(4),...
    '$ Plane to cap lower cylinder tube.'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Outter wall Cylinder Extension

ODOCyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch-hdwt,lcr,' $ Lower Extension OW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODOCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Wall Cylinder Extension

ODICyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch,lcr-hdwt,' $ Outer Duct IW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODICylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Cylinder Extension

IDCylc=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-mcnpfix,0,0,-ihdlch-hdwt,ihdlcr,' $ Inner Duct Cylinder'});

%Assign the surface number to be used later in the cell creation
IDCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

 if LATp

    %Naming card
    LATname=NameCard('Lattice Cards for Elements','Divider');

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezo ...
        ,0,0,LATh,LATr,'$ Element Lattice Bounding Surface'});
```

```
    LATpArnum=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4),0,0,LatEleh,0,LatEler,...
        0,'$ Lattice Window'});

    LatWinElenum=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfc=[LATname; LATArs; LatWinEle];

 end

% Append the surfaces to the surface card.
surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc];

%%


%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.
porg=PinOrg(nfe,fep,hdsorg(2),hdsorg(3),hdorg(4)+feoz);

 % The pin origin needs to be moved slightly
 if LATp; porg(:,2)=hdsorg(2)-LATpino; porg(:,3)=hdsorg(3); end

for mm=1:nfe

%Like the main program, the origins of the pins need to be determined so
%that they can be fed into a program to write the cards. This is what the
%pinorg program does.

%ocws = Oyter Cladding Wall Surface
%wws = wire wrap surface

[ pincellctemp, pinsurfctemp, ocws , wws ]=...
        PinMaker(porg(mm,:),pind,pinm,pinparm);

pinsurfnum(mm)=ocws;
wirewrapsurfnum(mm)=wws;

    if (mm==1)

        FPs=pinsurfctemp;
        FPc=pincellctemp;

    end

    if (mm>1)

        FPs=[FPs; pinsurfctemp];
        FPc=[FPc; pincellctemp];

    end
end



%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

SAcellname=NameCard(['SAFETY Rod M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');
ODcname=NameCard(['Outer Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
IDcname=NameCard(['Inner Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
UEcname=NameCard(['Upper Extension Sub Assembly #' num2str(hdorg(1))],'Divider');
LATcname=NameCard(['Infinite sodium for  Sub Assembly Lattice  #' num2str(hdorg(1))],'Divider');
CRcname=NameCard(['Core Region Sub Assembly #' num2str(hdorg(1))],'Divider');
```

```
%% %Hex Duct Cell cards


%***************************** Benchmark *********************************

% Outter duct outter wall.

ODc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODUPnum, -ODOWnum,...
     ODIWnum,ODOCylnum, UNIC,'$ Outer Hex Duct'});

ODcnum=cellnum;

cellnum=cellnum+1;

% Outer Duct Lower cylinder

ODCylc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODLPlnum,...
    -ODOCylnum,ODICylnum, UNIC,'$ Lower Cyn OD'});

OWLCcnum=cellnum;

cellnum=cellnum+1;

% Wall Inner Duct

IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
     surf2,IDUEnum,UNIC,'$ Inner Duct Safety '});

cellnum=cellnum+1;

% Lower Cylinder

IDCylc=CharChecker({cellnum,'     ',SIHDLCm,SIHDLCd, surf1,...
    -IDCylnum,UNIC,'$ Lower Cyn ID'});

cellnum=cellnum+1;

% Inside Duct Upper Ext Homog Region

UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -IDUEnum,...
     UNIC,'$ Up Ext Smear'});

cellnum=cellnum+1;

% Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

 if LAT

     LATNac=CharChecker({cellnum,'     ',Dnam,...
         Dnad,['#' Num2StrM(ODcnum)],surf1,...
         ['#' Num2StrM(OWLCcnum)],IDCylnum,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

    ODOWnums=[{surf1}, {IDCylnum}];

 else

     % this code adds a sign into hexsurfnum to let micka know
     % that the following surf is actually a not cell.

    ODnums=[ ODcnum; OWLCcnum];

    ODOWnums=[{surf1}, {IDCylnum}, {ODnums}];

 end

 % Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

if LATp

    %name the area

    LATpcellname=NameCard('Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum,'Lat=2',LUNu,...
        'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;
```

```
    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN, nle ,nar, Latnums );

    LATpcLns=CharChecker({'     ' Num2StrM(LATparln)});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum,LUNf,...
        UNIC,' $ Pin Lattice'});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

end

%***************************** Benchmark *********************************


if LATp

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1,1),...
        wirewrapsurfnum(1),UNICfp, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnum,UNIC,...
            '$ Sodium coolant']);


else

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc, UNIC,...
            '$ Sodium coolant']);

end

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LATp

    if LAT

        cellctemp=[ODc; ODCylc; IDcname; IDc; IDCylc; LATcname;...
            LATNac; UEc; CRcname; CRc; FPc; FPNac; LATpc];

    else

        cellctemp=[ODc; ODCylc; IDcname; IDc; IDCylc; LATcname;...
            UEc; CRcname; CRc; FPc; FPNac; LATpc];

    end

else

    if LAT

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; LATNac; UEcname; UEc; CRcname; CRc; FPc];

    else

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
```

```
            LATcname; UEcname; UEc; CRcname; CRc; FPc];

    end
end

%***************************** Benchmark *********************************


%% Appending the cards together

if LATp

    surfc=[surfcUP; FPs; LATsurfc];
    cellc=[SAcellname; cellctemp];

else

    surfc=[surfcUP; FPs];
    cellc=[SAcellname; cellctemp];

end

%%

global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Safe_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.51. HWCRMaker.m

```
function [ cellc,surfc ] = HWCRMaker(hdorg,mcnpfix,bend,MatMap,DimMap)
%This function writes the High Worth Control Rods

%declare the global variables
global debugMICKA DebugHWCRNFE DebugHWCRNPE
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC fewwdC nfeC fepC ihdhC ihdodC ihdwtC
global ihdlcdC ihdlchC pshC psdC pszoC pslasC pehC peodC pewtC peoC pewwhC
global pewwdC npeC pepC sazoC saatC sazmC fetphC fesphC pesbhC pebphC feoxC
global Dnpe

%% Material Vars

global FmC FdC  FNamC FNadC FPGmC FPGdC FCladmC FCladdC
global PmC PdC FPwwrmC FPwwdC PPGmC PPGdC PCladmC PCladdC DuctmC DuctdC
global DnamC DnadC SLOmC SLOdC IHDLAmC IHDLAdC  ShieldPmC ShieldPdC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
bh=DimMap{bhC}; crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC};
lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Inner Hex Duct
ihdh=DimMap{ihdhC}; ihdod=DimMap{ihdodC}; ihdwt=DimMap{ihdwtC}; ihdid=ihdod-(2*ihdwt);
ihdlcd=DimMap{ihdlcdC}; ihdlch=DimMap{ihdlchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC};
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

%Poison Elements
```

```
psh=DimMap{pshC}; psd=DimMap{psdC}; pszo=DimMap{pszoC}; pslas=DimMap{pslasC};
peh=DimMap{pehC}; peod=DimMap{peodC}; pewt=DimMap{pewtC}; peid=peod-(2*pewt);
peo=DimMap{peoC}; pewwh=DimMap{pewwhC}; pewwd=DimMap{pewwdC}; npe=DimMap{npeC};
pep=DimMap{pepC}; pesbh=DimMap{pesbhC}; pebph=DimMap{pebphC}; peox=pewwd/3;

% Fuel slug section heights
fssh=fsh/3;

%Convert The diameters in radi
% Hex Ducts         %Fuel Elements     %Poison Elements
ihdor=ihdod/2;       feor=feod/2;       peor=peod/2;
ihdir=ihdid/2;       fsr=fsd/2;         psr=psd/2;
ihdlcr=ihdlcd/2;     feir=feid/2;       peir=peid/2;
hdor=hdod/2;         fewwr=fewwd/2;     pewwr=pewwd/2;
hdir=hdid/2;
lcr=lcd/2;

% Set the number of lattice elements if applicable
if LATp; nlef=nfe; nlep=npe; end

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugHWCRNFE; npe=DebugHWCRNPE; end

%**********************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment
    LUNf=hdorg(1)*1000+1;
    LUNp=LUNf+1;

    LUNfp=LUNp+1;
    LUNpp=LUNfp+1;

    LUNuf=strjoin({'U=' Num2StrM(LUNf)},'');
    LUNfl=strjoin({'fill=' Num2StrM(LUNf)},'');

    LUNup=strjoin({'U=' Num2StrM(LUNp)},'');
    LUNpl=strjoin({'fill=' Num2StrM(LUNp)},'');

    % Since this is a driver, the pins are all the same universe number.
    Latnumsf=repmat(LUNfp,1,nfe);
    Latnumsp=repmat(LUNpp,1,npe);

    % Debugging line to pad the universemap with zeros.
    if debugMICKA; LatnumsfT=...
            [repmat(LUNfp,1,nfe), repmat(LUNfp,1,nfe), zeros(1,nlef-nfe)]; end

    if debugMICKA; LatnumspT=...
            [repmat(LUNpp,1,npe), repmat(LUNpp,1,npe), zeros(1,nlep-npe)]; end

    LatElerf=fep/2;
    LatElehf=2*feh; % This height is increased to not give a perfect fit
    LatElezof=mcnpfix;

    LatElerp=pep/2;
    LatElehp=2*peh; % This height is increased to not give a perfect fit
    LatElezop=pszo;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpinof=feox;
    LATpinop=peox;

    %Geometry calculations

    %Number of hex duct rings.

    nrf=ceil((3+sqrt(12*nlef-3))/6)-1;
    nrp=ceil((3+sqrt(12*nlep-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;
```

```
    fillcf=strjoin({num2str(-(nrf+nar)) ':' num2str(nrf+nar)},'');
    fillcp=strjoin({num2str(-(nrp+nar)) ':' num2str(nrp+nar)},'');

    % Lattice array size

    if feh>=fewwh; FLatH=feh; else; FLatH=fewwh; end

    LAThf=FLatH+2*mcnpfix;
    LATrf=ihdir-(ihdir/100);

    if peh>=pewwh; PLatH=peh; else; PLatH=pewwh; end

    LAThp=PLatH+mcnpfix;
    LATrp=ihdir-(ihdir/100);

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins
    npe=1;

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');
    UNICpp=strjoin({' U=' Num2StrM(LUNpp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;
    UNICpp=UNIC;

end

%***************************************************************************
%Subassembly movement

%Since the control rod in the full insertion position is 8.255cm from the
%origin of the core 0,0,0 we need to lower the origin of the assembly by
%that much. hdorg(4) is the z reference. This should not be changeD!
%s.a.z.o (sub assembly z offset)

sazo=DimMap{sazoC};

%Total allowable travel, saat sub assembly allowbale travel

saat=DimMap{saatC};

%This following variable is the induced movement due to drive mechanism.
%Movement is cm of insertion. This means full insertion = 35.56. Full out =
%0 cm. sazm, sub assembly movement

sazm=DimMap{sazmC};

%*************************************************************************
%This code sets the barrier between gas/shield and sodium

%nfslas now becomes the height of the sodium block from the region. This is
%done because the planes are defined independant of the duct.

mfslas=fslas+fsh+feoz-sazo-saat+sazm;
mpslas=pslas+psh+peo+pszo-sazo-saat+sazm;
pinparm={hdorg(1),[],mfslas,UNICfp,mcnpfix};
ppinparm={hdorg(1),[],[],mpslas,UNICpp,mcnpfix};
newpl=true;

%*************************************************************************
%Assigns dimensions and materials for the pins.

pind=[mfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
ppind=[mpslas;psh;psr;pewt;peir;peh;peor;pewwr;pewwh;pebph];

%*************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Duct Fluid

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SIHDLCm=MatMap{IHDLAmC};
SIHDLCd=MatMap{IHDLAdC};

%*************************************************************************
```

```
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap
FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Poison Slug

Pm=MatMap{PmC};
Pd=MatMap{PdC};

% Poison Fluid Bond
% Not sure if this is needed but keeping as a place holder to prevent error

PNam=0;
PNad=0;

% Poison Plenum Gas

PPGm=MatMap{PPGmC};
PPGd=MatMap{PPGdC};

% Poison Cladding

PCladm=MatMap{PCladmC};
PCladd=MatMap{PCladdC};

% Poison Wire Wrap
PWwrm=MatMap{FPwwrmC};
PWwrd=MatMap{FPwwdC};

% Poison Shield

PShieldm=MatMap{ShieldPmC};
PShieldd=MatMap{ShieldPdC};

%Passed to the pins
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];
ppinm=[Pm,Pd;PCladm,PCladd;PWwrm,PWwrd;PShieldm,PShieldd;PNam,PNad;PPGm,PPGd];

% Debug
if debugMICKA && LATp; Latnumsf=LatnumsfT; Latnumsp=LatnumspT; end

%************************************************************************
%%

%This section is the control rod movement code. This will enable us to move
%the control subassembly up and down. hdorg(4) refers to the 0" z height.
%The rest of the assembly is built from this reference position including
%the KAtana effect.

% Copy HDorg so that it can be used on the outer sheth
Ohdorg=[hdorg(1:3), hdorg(4)-leh];

%Offest plus movement
hdorg(4)=hdorg(4)-sazo-saat+sazm;

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

Ohdsorg=ThermalXHex(hdh,Ohdorg,bend);
hdsorg=ThermalXHex(crh,hdorg,bend);

%This next line performs the expansion. It moves the pins slightly outward
%from the center of the core. This will be replaced by pin section movement
%later on in PinSectionMaker but for not it will reside here as a proof of
%concept.

%porg=ThermalX(porgtemp);


%The surfaces must be written first for the duct and all the pins, then the
%cell cards.

%%
```

```
%Debug only

% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['High Worth Control Rod MKIIS M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

ppinsurfnum=zeros(npe,size(hdsorg,1));
pwirewrapsurfnum=zeros(npe,size(hdsorg,1));

%%
% The following for loop creates the core region sections.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
IDOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)-ihdwt,0,0,crh+ueh+2*ihdwt,...
ihdor, '$ Outer Wall of Inner Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf1=SAsurfnum;

%Update the surnum number
SAsurfnum=SAsurfnum+1;


if (LAT || LATp)

 LATNas=CharChecker({surfnum,'    SO',2*hdh,...
     '$ Surr Na For Lat'});

 surf15=surfnum;

 surfnum=surfnum+1;

end

% This plane caps off the outer duct.

ODUPls=CharChecker({surfnum,'    PZ', Ohdsorg(4)+hdh,...
               '$ Plane to cap outter hex duct'});

%I need this facet for later.
ODUPnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Outter Wall Outer Duct

 ODOWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
 Ohdsorg(3),Ohdorg(4)-hdwt,0,0,hdh+2*hdwt,hdor,...
 '$ Duct Outer Tube Outer Wall'});

%Assign the surface number to be used later in the cell creation
ODOWnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Outter Wall Inner Duct

 ODIWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
 Ohdsorg(3),Ohdorg(4),0,0,hdh+hdwt,hdir,...
 '$ Duct Outer Tube Inner Wall'});

%Assign the surface number to be used later in the cell creation
ODIWnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

%Creates the inner duct for the upper extension.
%Also calls the Poison Pins to be made

% Inner Duct Upper Extention

 IDPRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)+crh,0,0,ueh,ihdir,...
 '$ Poison Region Inner Hex'});

%Assign the surface number to be used later in the cell creation
IDPRnum=surfnum;
```

```
%Update the surnum number
 surfnum=surfnum+1;

 if newpl

     % Barrier between fluid and gas Poison

     pbarrierpl=CharChecker({surfnum,'    PZ',mpslas,...
            '$ Poison Gas Boundary'});

     ppinparm(1,2)={surfnum};

     surfnum=surfnum+1;

     % Barrier Between Shield and Gas

     pShieldGasBar=CharChecker({surfnum,'    PZ',mpslas+pesbh,...
            '$ Poison Shield Gas Boundary'});

     ppinparm(1,3)={surfnum};

     surfnum=surfnum+1;

    % Barrier between fluid and gas for fuel

     barrierpl=CharChecker({surfnum,'    PZ',mfslas,...
            '$ Sodium Gas Boundary'});

     bPl{1,1}=surfnum;
     bPl{1,2}=mfslas;

     surfnum=surfnum+1;

     if size(Fm,1)>1

        % Section  1/2 barrier

        barrierpl=[barrierpl; CharChecker({surfnum,'    PZ',mfslas-fslas-fssh,...
                        '$ Fuel Section Plane Sec 1/2'})];

        bPl{2,1}=surfnum;
        bPl{2,2}=mfslas-fslas-fssh;

        surfnum=surfnum+1;

        % Section  2/3 barrier

         barrierpl=[barrierpl; CharChecker({surfnum,'    PZ',mfslas-fslas-(2*fssh),...
                        '$ Fuel Section Plane Sec 2/3'})];

        bPl{3,1}=surfnum;
        bPl{3,2}=mfslas-fslas-(2*fssh);

        surfnum=surfnum+1;

     end

     pinparm(1,2)={bPl};
     IDPRs=[IDPRs; pShieldGasBar; pbarrierpl; barrierpl];

     newpl=false;

 end

 %Calculates the origin of the poison pins

 porg=PinOrg(npe,pep,hdsorg(2),hdsorg(3),hdorg(4)+pszo);

 % The following line will remain, but the need to change the origin is
 % not needed because the wire wrap is soo small in comparison to the
 % pitch that it tucks nicely into the hex vertex.
 % if LATp; porg(:,2)=-LATpinop; porg(:,3)=0; end

% The pin origin needs to be moved slightly

if LATp; porg(:,2)=hdsorg(2)-LATpinop; porg(:,3)=hdsorg(3); end

     for mm=1:npe

        %Like the main program, the origins of the pins need to be determined so
        %that they can be fed into a program to write the cards. This is what the
        %pinorg program does.

        %ocws = Oyter Cladding Wall Surface
        %wws = wire wrap surface

            [ ppincellctemp, ppinsurfctemp, ocws , wws ]=...
                PoisonPinMaker(porg(mm,:),ppind,ppinm,ppinparm);

            ppinsurfnum(mm,1)=ocws;
            pwirewrapsurfnum(mm,1)=wws;

            if (mm==1)
```

```
                PPs=ppinsurfctemp;
                PPc=ppincellctemp;

            end

            if (mm>1)

                PPs=[PPs; ppinsurfctemp];
                PPc=[PPc; ppincellctemp];

            end
     end

% Inner Duct Core Region
IDCRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4),0,0,crh,ihdir,...
'$ Inner Wall Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf2=surfnum;

%Update the surnum number
surfnum=surfnum+1;

if (LAT || LATp); surfcUP=[surfname; ODUPls; ODOWs; ODIWs; IDOWs; LATNas; IDPRs; IDCRs ];
else    surfcUP=[surfname; ODUPls; ODOWs; ODIWs; IDOWs; IDPRs; IDCRs]; end

%%

% Lower Extension Surface Cards

% This plane caps off the lower tube

ODLPc=CharChecker({surfnum,'    PZ',Ohdorg(4),...
                '$ Plane to cap lower cylinder tube.'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Outter wall Cylinder Extension

ODOCyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch-hdwt,lcr,' $ Lower Extension OW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODOCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Wall Cylinder Extension

ODICyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch,lcr-hdwt,' $ Outer Duct IW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODICylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Cylinder Extension

IDCylc=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-mcnpfix,0,0,-ihdlch-hdwt,ihdlcr,' $ Inner Duct Cylinder'});

%Assign the surface number to be used later in the cell creation
IDCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

 % Lattice surfaces.

if LATp

     % Poison Surface Cards

    %Naming card
    LATnamep=NameCard('Lattice Cards for Poison Elements','Divider');

    % Lattice array surface.
    LATArsp=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezop ...
        ,0,0,LAThp,LATrp,'$ Element Lattice Bounding Surface'});

    LATpArnump=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinElep=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4),0,0,LatElehp,0,LatElerp,...
        0,'$ Lattice Window'});
```

```
    LatWinElenump=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfcp=[LATnamep; LATArsp; LatWinElep];

    % Fuel Pins

    %Naming card
    LATnamef=NameCard('Lattice Cards for Elements','Divider');

    % Lattice array surface.
    LATArsf=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezof ...
        ,0,0,LAThf,LATrf,'$ Element Lattice Bounding Surface'});

    LATpArnumf=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinElef=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4),0,0,LatElehf,0,LatElerf,...
        0,'$ Lattice Window'});

    LatWinElenumf=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfcf=[LATnamef; LATArsf; LatWinElef];

    % Append the surfaces to the surface card.
    surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc; PPs; LATsurfcp];

else

    % Append the surfaces to the surface card.
    surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc; PPs];

end


%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.

porg=PinOrg(nfe,fep,hdsorg(2),hdsorg(3),hdorg(4)+feoz);

% The pin origin needs to be moved slightly

if LATp; porg(:,2)=hdsorg(2)-LATpinof; porg(:,3)=hdsorg(3); end

for mm=1:nfe

    %Like the main program, the origins of the pins need to be determined so
    %that they can be fed into a program to write the cards. This is what the
    %pinorg program does.

    %ocws = Oyter Cladding Wall Surface
    %wws = wire wrap surface

    [ pincellctemp, pinsurfctemp, ocws , wws ]=...
        PinMaker(porg(mm,:),pind,pinm,pinparm);

    pinsurfnum(mm)=ocws;
    wirewrapsurfnum(mm)=wws;

    if (mm==1)

        FPs=pinsurfctemp;
        FPc=pincellctemp;

    end

    if (mm>1)
```

```
        FPs=[FPs; pinsurfctemp];
        FPc=[FPc; pincellctemp];

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

SAcellname=NameCard(['High Worth Control Rod M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');
ODcname=NameCard(['Outer Duct Sub Assembly: ' num2str(hdorg(1))],'Divider');
IDcname=NameCard(['Inner Duct Sub Assembly: ' num2str(hdorg(1))],'Divider');
LATcname=NameCard(['Inf Na for Lattice Sub Assembly: ' num2str(hdorg(1))],'Divider');
PRcname=NameCard(['Poison Region Sub Assembly #' num2str(hdorg(1))],'Divider');
CRcname=NameCard(['Core Region Sub Assembly #' num2str(hdorg(1))],'Divider');

%% %Hex Duct Cell cards

%***************************** Benchmark *********************************

% Outter duct outter wall.

ODc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODUPnum, -ODOWnum,...
     ODIWnum,ODOCylnum, UNIC,'$ Outer HWCR Hex Duct'});

ODcnum=cellnum;

cellnum=cellnum+1;

% Outter Wall Lower Cylinder

ODCylc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODLPlnum,...
    -ODOCylnum,ODICylnum, UNIC,'$ Lower Cyn OD'});

OWLCcnum=cellnum;

cellnum=cellnum+1;

% Inner  Duct Wall

IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
     surf2,IDPRnum, UNIC,'$ Inner HWCR Hex Duct'});

cellnum=cellnum+1;

% Inside Poison pin Region

PRc={cellnum,'     ',Dnam,Dnad, -IDPRnum};

cellnum=cellnum+1;

% Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

% Lower Cylinder

IDCylc=CharChecker({cellnum,'     ',SIHDLCm,SIHDLCd,surf1,...
    -IDCylnum, UNIC,'$ Lower Cyn ID'});

cellnum=cellnum+1;

% Infinite sodium sphere for the lattice.

 if LAT==1

     LATNac=CharChecker({cellnum,'     ',Dnam,...
         Dnad,['#' Num2StrM(ODcnum)],surf1,...
         ['#' Num2StrM(OWLCcnum)],IDCylnum,-surf15,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

    ODOWnums=[{surf1}, {IDCylnum}];

 else

     % this code adds a sign into hexsurfnum to let micka know
     % that the following surf is actually a not cell.

    ODnums=[ ODcnum; OWLCcnum];

    ODOWnums=[{surf1}, {IDCylnum}, {ODnums}];

 end

  % Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})
```

```
    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

if LATp

    % POison pins
    %name the area

    LATpcellnamep=NameCard('Poison Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArcp=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenump,'Lat=2',LUNup,...
        'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;

    LATpFlcp=CharChecker({'      ','fill ',fillcp,fillcp,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparlnp,LATarp]=MCNPPOSMaker( LUNp, nlep ,nar, Latnumsp );

    LATpcLnsp=CharChecker({'     ' num2str(LATparlnp)});

    % Fills the lattice into the core.

    LATpcellp=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnump,LUNpl,...
        UNIC,' $ Poison Pin Lattice'});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpcp=[LATpcellnamep; LATpArcp; LATpFlcp; LATpcLnsp; LATpcellp];

    % Fuel Pins
    %name the area

    LATpcellnamef=NameCard('Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArcf=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenumf,'Lat=2',LUNuf,...
        'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;

    LATpFlcf=CharChecker({'      ','fill ',fillcf,fillcf,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparlnf,LATarf]=MCNPPOSMaker( LUNf, nlef ,nar, Latnumsf );

    LATpcLnsf=CharChecker({'     ' num2str(LATparlnf)});

    % Fills the lattice into the core.

    LATpcellf=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnumf,LUNfl,...
        UNIC,' $ Pin Lattice'});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpcf=[LATpcellnamef; LATpArcf; LATpFlcf; LATpcLnsf; LATpcellf];

end

%***************************** Benchmark *********************************

if LATp

    % Poison Pins

    PNanamep=NameCard('Sodium Cell Surrounding Poison Pins','Divider');

    % This defines the surrounding sodium around the fuel pin.

    FPNacp=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,ppinsurfnum(1),...
        pwirewrapsurfnum(1),UNICpp, '$ Poison Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        PRc=CharChecker([PRc,LATpArnump,UNIC,...
```

```
            '$ Sodium coolant']);

    % Fuel Pins

    PNanamef=NameCard('Sodium Cell Surrounding Fuel Pins','Divider');

    % This defines the surrounding sodium around the fuel pin.

    FPNacf=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1),...
        wirewrapsurfnum(1),UNICfp, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnumf,UNIC,...
            '$ Sodium coolant']);


else

    % Inside Duct Upper Poison Region

    PRc={cellnum,'     ',Dnam,Dnad, -IDPRnum};

    cellnum=cellnum+1;

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(ppinsurfnum,1)

        PRc=[PRc ppinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(pwirewrapsurfnum,1)

        PRc=[PRc pwirewrapsurfnum(mm)];

    end

    PRc=CharChecker([PRc,UNIC,...
    '$ Sodium coolant Poison Region']);

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc,UNIC,...
            '$ Sodium coolant']);
end

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LATp

    if LAT

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; LATNac; PRcname; PRc; PPc; PNanamep; FPNacp; LATpcp;...
            CRcname; CRc; FPc; PNanamef; FPNacf; LATpcf];

    else

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; PRcname; PRc; PPc; PNanamep; FPNacp; LATpcp;...
            CRcname; CRc; FPc; PNanamef; FPNacf; LATpcf];

    end

else

    if LAT

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; LATNac; PRcname; PRc; PPc; CRcname; CRc; FPc];
```

```
    else

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; PRcname; PRc; PPc; CRcname; CRc; FPc];


    end

end

%***************************** Benchmark *********************************


%% Appending the cards together



if LATp

    surfc=[surfcUP; FPs; LATsurfcf];
    cellc=[SAcellname; cellctemp];

else

    surfc=[surfcUP; FPs];
    cellc=[SAcellname; cellctemp];

end

%% Save variables for Debug

global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_HWCR_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end


end
```

B.52. ControlMaker.m

```
function [ cellc,surfc ] = ControlMaker(hdorg,mcnpfix,bend,MatMap,DimMap)

%%This function writes the Control Rod

%declare the global variables
global debugMICKA DebugContNFE
global surfnum cellnum
global hexsurfnum planesurfnum fbPl SAsurfnum createplane
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC feoxC fewwdC nfeC fepC usbhC
global ihdhC ihdodC ihdwtC ihdlcdC ihdlchC sazoC saatC sazmC fetphC fesphC

%% Material Vars

global FmC FdC FNamC FNadC FPGmC FPGdC  FCladmC FCladdC FPwwrmC FPwwdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC IHDLAmC IHDLAdC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC};
lcd=DimMap{lcdC}; lch=DimMap{lchC}; usbh=DimMap{usbhC}; ush=ueh-usbh;

%Inner Hex Duct
ihdh=DimMap{ihdhC}; ihdod=DimMap{ihdodC}; ihdwt=DimMap{ihdwtC}; ihdid=ihdod-(2*ihdwt);
ihdlcd=DimMap{ihdlcdC}; ihdlch=DimMap{ihdlchC};
```

```
%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC};
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Fuel slug section heights
fssh=fsh/3;

%Convert The diameters in radi
ihdor=ihdod/2;
ihdir=ihdid/2;
ihdlcr=ihdlcd/2;
hdor=hdod/2;
hdir=hdid/2;
lcr=lcd/2;
feor=feod/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;

% Set the number of lattice elements if applicable
if LATp; nle=nfe; end

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugContNFE; end

%**************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LUN=hdorg(1)*1000+1;

    LUNu=strjoin({'U=' Num2StrM(LUN)},'');
    LUNf=strjoin({'fill=' Num2StrM(LUN)},'');

    LUNfp=LUN+1;

    % Since this is a driver, the pins are all the same universe number.
    Latnums=repmat(LUNfp,1,nfe);

    % Debugging line to pad the universemap with zeros.
    if debugMICKA; Latnums=[repmat(LUNfp,1,nfe), zeros(1,nle-nfe)]; end

    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatElezo=mcnpfix;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice array size

    LATr=ihdir-(ihdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
```

```
    % function is only called once.

    nfe=1;   % New number of pins

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;

end

%*************************************************************************
%Subassembly movement

%Since the control rod in the full insertion position is 8.255cm from the
%origin of the core 0,0,0 we need to lower the origin of the assembly by
%that much. hdorg(4) is the z reference. This should not be changeD!
%s.a.z.o (sub assembly z offset)

sazo=DimMap{sazoC};

%Total allowable travel, saat sub assembly allowbale travel

saat=DimMap{saatC};

%This following variable is the induced movement due to drive mechanism.
%Movement is cm of insertion. This means full insertion = 35.56. Full out =
%0 cm. sazm, sub assembly movement

sazm=DimMap{sazmC};

%*************************************************************************
%This code sets the barrier between gas/shield and sodium

%nfslas now becomes the height of the sodium block from the region. This is
%done because the planes are defined independant of the duct.

mfslas=fslas+fsh+feoz-sazo-saat+sazm;
pinparm={hdorg(1),[],mfslas,UNICfp,mcnpfix};
newpl=true;

%*************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

Slom=MatMap{SLOmC};
Slod=MatMap{SLOdC};

% Smeared Lower Adapter

SIHDLCm=MatMap{IHDLAmC};
SIHDLCd=MatMap{IHDLAdC};

%*************************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap

FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};
```

```
% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

%%
%**********************************************************************
%Assigns dimensions and materials for the pins.

pind=[mfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];

%%
%**********************************************************************


%This section is the control rod movement code. This will enable us to move
%the control subassembly up and down. hdorg(4) refers to the 0" z height.
%The rest of the assembly is built from this reference position including
%the KAtana effect.

% Copy HDorg so that it can be used on the outer sheth
Ohdorg=[hdorg(1:3), hdorg(4)-leh];

%Offest plus movement
hdorg(4)=hdorg(4)-sazo-saat+sazm;

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

Ohdsorg=ThermalXHex(hdh,Ohdorg,bend);
hdsorg=ThermalXHex(crh,hdorg,bend);

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Control Rod MKIIC M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%%
% The following for loop creates the core region sections.

%A plane needs to be created that the pins use as the barrier between
% above the fluid level and below. It has to be defined per assembly
% because some assemblies move.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
 IDOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)-ihdwt,0,0,crh+ueh+2*ihdwt,...
 ihdor, '$ Outer Wall of Inner Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf1=SAsurfnum;

%Update the surnum number
 SAsurfnum=SAsurfnum+1;

if (LAT || LATp)

     IDOWs=[IDOWs; CharChecker({surfnum,'    SO',2*hdh,...
         '$ Surr Na For Lat'})];

     surf15=surfnum;

     surfnum=surfnum+1;

end

% This plane caps off the outer duct.

ODUPls=CharChecker({surfnum,'    PZ', Ohdsorg(4)+hdh,...
                '$ Plane to cap outter hex duct'});

%I need this facet for later.
ODUPnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Outter Wall Outer Duct

 ODOWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
 Ohdsorg(3),Ohdorg(4)-hdwt,0,0,hdh+2*hdwt,hdor,...
 '$ Duct Outer Tube Outer Wall'});

%Assign the surface number to be used later in the cell creation
ODOWnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;
```

```
% Outter Wall Inner Duct

 ODIWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
 Ohdsorg(3),Ohdorg(4),0,0,hdh+hdwt,hdir,...
 '$ Duct Outer Tube Inner Wall'});

%Assign the surface number to be used later in the cell creation
ODIWnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

%Creates the inner duct for the upper extension.
%Also calls the Poison Pins to be made

% Inner Duct Upper Extention

 IDSUPs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)+crh+usbh,0,0,ush,ihdir,...
 '$ Upper Ext Inner Hex'});

%Assign the surface number to be used later in the cell creation
IDUEnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Inner Duct Shield Block

 IDSBs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)+crh,0,0,usbh,ihdir,...
 '$ Shield Block Top'});

%Assign the surface number to be used later in the cell creation
IDSBnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Inner Duct Core Region
 IDCRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4),0,0,crh,ihdir,...
 '$ Inner Wall Inner Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf2=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

 if newpl

     % Barrier between fluid and gas

     barrierpl=CharChecker({surfnum,'    PZ',mfslas,...
            '$ Sodium Gas Boundary'});

     bPl{1,1}=surfnum;
     bPl{1,2}=mfslas;

     surfnum=surfnum+1;

     if size(Fm,1)>1

        % Section  1/2 barrier

        barrierpl=[barrierpl; CharChecker({surfnum,'    PZ',mfslas-fslas-fssh,...
                        '$ Fuel Section Plane Sec 1/2'})];

        bPl{2,1}=surfnum;
        bPl{2,2}=mfslas-fslas-fssh;

        surfnum=surfnum+1;

        % Section  2/3 barrier

         barrierpl=[barrierpl; CharChecker({surfnum,'    PZ',mfslas-fslas-(2*fssh),...
                        '$ Fuel Section Plane Sec 2/3'})];

        bPl{3,1}=surfnum;
        bPl{3,2}=mfslas-fslas-(2*fssh);

        surfnum=surfnum+1;

     end

     pinparm(1,2)={bPl};
     IDCRs=[IDCRs; barrierpl];

     newpl=false;

 end

 surfcUP=[ surfname; ODUPls;...
     ODOWs; ODIWs; IDOWs; IDSUPs; IDSBs; IDCRs ];
```

```
%%
% Lower Extension Surface Cards

% This plane caps off the lower tube

ODLPc=CharChecker({surfnum,'    PZ',Ohdorg(4),...
    '$ Plane to cap lower cylinder tube.'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Outter wall Cylinder Extension

ODOCyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch-hdwt,lcr,' $ Lower Extension OW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODOCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Wall Cylinder Extension

ODICyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch,lcr-hdwt,' $ Outer Duct IW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODICylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Cylinder Extension

IDCylc=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-mcnpfix,0,0,-ihdlch,ihdlcr,' $ Inner Duct Cylinder'});

%Assign the surface number to be used later in the cell creation
IDCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

 if LATp

    %Naming card
    LATname=NameCard('Pin Lattice Cards','Divider');

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezo ...
        ,0,0,LATh,LATr,'$ Element Lattice Bounding Surface'});

    LATpArnum=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4),0,0,LatEleh,0,LatEler,...
        0,'$ Lattice Window'});

    LatWinElenum=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfc=[LATname; LATArs; LatWinEle];

 end

% Append the surfaces to the surface card.
surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc];

%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
```

```
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.
porg=PinOrg(nfe,fep,hdsorg(2),hdsorg(3),hdorg(4)+feoz);

% The origin needs to be moved slightly in the -x direction so that the
% wirewrap fits in the cell
% The pin origin needs to be moved slightly
if LATp; porg(:,2)=hdsorg(2)-LATpino; porg(:,3)=hdsorg(3); end

for mm=1:nfe

    %Like the main program, the origins of the pins need to be determined so
    %that they can be fed into a program to write the cards. This is what the
    %pinorg program does.

    %ocws = Oyter Cladding Wall Surface
    %wws = wire wrap surface


    [ pincellctemp, pinsurfctemp, ocws , wws ]=...
        PinMaker(porg(mm,:),pind,pinm,pinparm);

    pinsurfnum(mm)=ocws;
    wirewrapsurfnum(mm)=wws;

    if (mm==1)

        FPs=pinsurfctemp;
        FPc=pincellctemp;

    end

    if (mm>1)

        FPs=[FPs; pinsurfctemp];
        FPc=[FPc; pincellctemp];

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

SAcellname=NameCard(['CONTROL rod M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');
ODcname=NameCard(['Outer Duct Sub Assembly  #' num2str(hdorg(1))],'Divider');
IDcname=NameCard(['Inner Duct Sub Assembly  #' num2str(hdorg(1))],'Divider');
UEcname=NameCard(['Upper Extension Sub Assembly  #' num2str(hdorg(1))],'Divider');
LATcname=NameCard(['Infinite sodium for Lattice Sub Assembly  #' num2str(hdorg(1))],'Divider');
CRcname=NameCard(['Core Region Sub Assembly  #' num2str(hdorg(1))],'Divider');

%% %Hex Duct Cell cards

%***************************** Benchmark *********************************

% Outter duct outter wall.

ODc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODUPnum, -ODOWnum,...
     ODIWnum,ODOCylnum, UNIC,'$ Outer Duct CONTROL '});

ODcnum=cellnum;

cellnum=cellnum+1;

% Outer Duct Lower cylinder

ODCylc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODLPlnum,...
    -ODOCylnum,ODICylnum, UNIC,'$ Lower Cyn OD'});

OWLCcnum=cellnum;

cellnum=cellnum+1;

% Wall Inner Duct

IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
     surf2,IDUEnum,IDSBnum,UNIC,'$ Inner Duct CONTROL '});

cellnum=cellnum+1;

% Lower Cylinder

IDCylc=CharChecker({cellnum,'     ',SIHDLCm,SIHDLCd, surf1,...
    -IDCylnum,UNIC,'$ Lower Cyn ID'});

cellnum=cellnum+1;

% Inside Duct Upper Ext Homog Region

UEc=CharChecker({cellnum,'     ',Dnam,Dnad, -IDUEnum,...
```

```
     UNIC,'$ Up Ext Smear'});

cellnum=cellnum+1;

% Inside Duct Upper Ext Homog Region

SBc=CharChecker({cellnum,'     ',SUPm,SUPd, -IDSBnum,...
     UNIC,'$ Shield Block'});

cellnum=cellnum+1;

% Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

 if LAT

     LATNac=CharChecker({cellnum,'     ',Dnam,...
         Dnad,['#' Num2StrM(ODcnum)],surf1,...
         ['#' Num2StrM(OWLCcnum)],IDCylnum,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

    ODOWnums=[{surf1}, {IDCylnum}];

 else

     % this code adds a sign into hexsurfnum to let micka know
     % that the following surf is actually a not cell.

    ODnums=[ ODcnum; OWLCcnum];

    ODOWnums=[{surf1}, {IDCylnum}, {ODnums}];

 end

  % Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

if LATp

    %name the area

    LATpcellname=NameCard('Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum,'Lat=2',LUNu,...
        'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN, nle ,nar, Latnums );

    LATpcLns=CharChecker({'     ' Num2StrM(LATparln)});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum,LUNf,...
        UNIC,' $ Pin Lattice'});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

end

%***************************** Benchmark *********************************

if LATp

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1,1),...
        wirewrapsurfnum(1),UNICfp, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;
```

```
    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnum,UNIC,...
            '$ Sodium coolant']);


else

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc, UNIC,...
            '$ Sodium coolant']);

end

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LATp

    if LAT

        cellctemp=[ODc; ODCylc; IDcname; IDc; IDCylc; LATcname;...
            LATNac; UEc; SBc; CRcname; CRc; FPc; FPNac; LATpc];

    else

        cellctemp=[ODc; ODCylc; IDcname; IDc; IDCylc; LATcname;...
            UEc; SBc; CRcname; CRc; FPc; FPNac; LATpc];

    end

else

    if LAT

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; LATNac; UEcname; UEc; SBc; CRcname; CRc; FPc];

    else

        cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
            LATcname; UEcname; UEc; SBc; CRcname; CRc; FPc];

    end
end

%***************************** Benchmark *********************************


%% Appending the cards together

if LATp

    surfc=[surfcUP; FPs; LATsurfc];
    cellc=[SAcellname; cellctemp];

else

    surfc=[surfcUP; FPs];
    cellc=[SAcellname; cellctemp];

end
%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Control_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.53. PoisonPinMaker.m

```
function [ cellc, surfc ,ocw ,wr] = PoisonPinMaker(porg,ppind,ppinm,ppinparm)
%This function creates the pins in the same manner as the hexmaker.

%This function Designs a pin as follows

% A slug immersed in fluid
% Fluid Plenum above the immersed slug
% Cladding around both fluids
% Wire Wrap around the cladding

global surfnum
global cellnum
global pinsurfnum
global planesurfnum
global BorD

%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

%Sets the MICKA num
hdnum=ppinparm{1};

%Shield Barrier Surface number
HePl=ppinparm{2};

%Shield Gas Barrier Surface number
HeSPl=ppinparm{3};

%Fuel sodium level
pslas=ppinparm{4};

%Universse importance
UNIC=ppinparm{5};

%Mcnpfix
mcnpfix=ppinparm{end};

%%
%***************Define the pin material parameters**********************

%slug material and density
slugm=ppinm(1,1);
slugd=ppinm(1,2);

% %cladding material and density
cladm=ppinm(2,1);
cladd=ppinm(2,2);

% %DEBUG!!!
% cladm=0;
% cladd=' ';

%wire wrap material and density
wwrm=ppinm(3,1);
wwrd=ppinm(3,2);

shieldm=ppinm(4,1);
shielddd=ppinm(4,2);

%Coolant material and density
nam=ppinm(5,1);
nad=ppinm(5,2);

%plenum gas material and density
PGm=ppinm(6,1);
PGd=ppinm(6,2);

%%
%*********************************Additional Pin Dimensions***************

%Fuel slug length
psh=ppind(2);

%Fuel Slug Radius
psr=ppind(3);

%cladding wall thickness
pewt=ppind(4);

%Cladding inner radius
peir=ppind(5);

%Pin height
peh=ppind(6);

%Pin radius
peor=ppind(7);

%Wire Wrap radius
pewwr=ppind(8);

%Wire Wrap Length
pewwh=ppind(9);
```

```
% Poison Element Bottom Plug he
pebph=ppind(10);

%%
%*****************************Surface Cards*****************************

%Since the cards for the pin and the subsequent pin sections will become
%huge, I will put a title in the cards for each pin now.

surfname=NameCard(['Poison Pin: ' num2str(porg(1)) ' of SA: '...
    num2str(hdnum)],'Divider');

%Outer Diameter Cladding

surfctemp=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)...
    ,0,0,peh,peor,'$ Pin:',porg(1),'Outer Cladding wall'});

%Assign the surface number to be used later in the cell creation
ocw=surfnum;

%Write that surface number to the pinsurfarray
%This is to tell hex maker which is the outter surface to be placed in the
%cell definition for the sodium inside of the duct.
pinsurfnum=[pinsurfnum; surfnum];

%Update the surnum number
surfnum=surfnum+1;

%Inner Diameter Cladding

surfctemp2=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+pebph...
    ,0,0,peh-(pebph),peir,'$ Pin:',porg(1),'Inner cladding wall'});

%Assign the surface number to be used later in the cell creation
icw=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Fuel slug cylinder

surfctemp3=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+pebph+mcnpfix...
    ,0,0,psh,psr,'$ Pin:',porg(1),'Poison slug boundary'});

%Assign the surface number to be used later in the cell creation
ps=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%WireWrap
%This will have to call another function to calculate and generate the
%wirewrap positions. For right now we are just going to do one cylinder
%running up the side of the outter pin cladding. The cylinder will be
%parrallel to the pin with a 90 degree offset in the xy plane.

surfctemp4=CharChecker({surfnum,'    RCC',porg(2)+peor+pewwr+mcnpfix,...
    porg(3),porg(4),0,0,pewwh,pewwr,'$ Pin:',porg(1),'Wrap'});

%Since the wirewrap is on the outside of the pin, it also needs to give its
%number to pinsurf so that hex maker can know the sodium is on the outside.

pinsurfnum=[pinsurfnum; surfnum];

%Assign the surface number to be used later in the cell creation
wr=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%%
%*****************************Cell Cards*********************************
% Pin number title to seperate the pins

cellname=NameCard(['Poison Pin: ' num2str(porg(1)) ' of SA: '...
    num2str(hdnum)],'Divider');

%cladding

cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocw,...
    icw,UNIC,'$ Pin:',porg(1),'Cladding'});

cellnum=cellnum+1;

%Helium

cellctemp2=CharChecker({cellnum,'     ',PGm,PGd, -HePl,...
    -icw,ps,UNIC,...
    '$ Pin:',porg(1),'Helium'});

cellnum=cellnum+1;

% Poison
```

```
cellctemp3=CharChecker({cellnum,'     ',slugm,slugd, -ps,...
    UNIC,'$ Pin:',porg(1),'Poison Slug'});

cellnum=cellnum+1;

% SS Shield

cellctemp4=CharChecker({cellnum,'     ',shieldm,shielddd,-HeSPl, HePl,...
    -icw,UNIC,'$ Pin:',porg(1),'SS Shield'});

cellnum=cellnum+1;

% Gas Above Shield

cellctemp5=CharChecker({cellnum,'     ',PGm,PGd,HeSPl,...
    -icw,UNIC,'$ Pin:',porg(1),'SS Shield Gas'});

cellnum=cellnum+1;

%Wire Wrap

cellctemp6=CharChecker({cellnum,'     ',wwrm,wwrd,...
    -wr,UNIC,'$ Pin:',porg(1),'Wire Wrap'});

cellnum=cellnum+1;


%%
%Assign the cards to be returned to hexmaker.
cellc=[cellname; cellctemp; cellctemp2; cellctemp3; cellctemp4; cellctemp5; cellctemp6];
surfc=[surfname; surfctemp; surfctemp2; surfctemp3; surfctemp4];

%%

global SaveSAMakerVars
persistent FirstRun

FirstRun=true;

if SaveSAMakerVars && FirstRun

    % Save TotDimDat
    save('Debug\SAmakerVar\Tot_PoisonPin_Dat.mat')
    dispPrint('     TotDat_PoisonPin_saved....');

    FirstRun=false;

end

end
```

B.54. DummyMaker.m

```
function [ cellc,surfc ] = DummyMaker(hdorg,mcnpfix,bend,MatMap,DimMap)
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables

global debugMICKA DebugDumNDE
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum fgb
global LAT
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC crhC lcdC lchC sazoC dehC deodC dewtC
global deoC ndeC depC

%% Material Vars

global PPGmC PPGdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC LAmC LadC DmC DdC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).


%*************************************************************************
%Sets the universe num

if LAT==1

    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;
```

```
else

    UNIC=' IMP:N=1';

end

%************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
crh=DimMap{crhC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

% Dummy Elements, they are the same as fuel elements but solid ss
deh=DimMap{dehC}; deod=DimMap{deodC}; dewt=DimMap{dewtC}; deid=deod-(2*dewt);  ...
    deo=DimMap{deoC}+mcnpfix; nde=DimMap{ndeC}; dep=DimMap{depC}; dslas=NaN; dsh=NaN;

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
lcr=lcd/2;
deor=deod/2;
deir=deid/2;

%Wire Wrap Switch 1 = on 0 = off.
dewws=0;

% Solid dummy element 1=solid 0=hollow;
dess=1;

% Override pin count if debug micka is true
if debugMICKA; nde=DebugDumNDE; end

%************************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

ndslas=dslas+dsh+deo;

dpinparm={hdorg(1),fgb,ndslas,UNIC,mcnpfix};

%************************************************************************
%Subassembly movement

%Since the dummy is just one long core region, I decided to adjust the
%origin of the whole assembly like a control rod.

sazo=DimMap{sazoC};

%************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Duct Fluid

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

%************************************************************************
% Pin Materials

% Dummy Element

Dm=MatMap{DmC};
Dd=MatMap{DdC};


% Dummy Fuel Slug
Deim=MatMap{PPGmC};
Deid=MatMap{PPGdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%%
%************************************************************************

%Assigns dimensions and materials for the pins.
dpinm=[Dm,Dd;Deim,Deid;NaN,NaN];
```

```
dpind=[ndslas;NaN;NaN;dewt;deir;deh;deor;NaN;NaN;dewws;dess];

%**********************************************************************
%%

%This section is the dummy origin adjustment.

%Offest plus movement
hdorg(4)=hdorg(4)-sazo;

%**********************************************************************
%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);

%This next line performs the expansion. It moves the pins slightly outward
%from the center of the core. This will be replaced by pin section movement
%later on in PinSectionMaker but for not it will reside here as a proof of
%concept.

%porg=ThermalX(porgtemp);


%The surfaces must be written first for the duct and all the pins, then the
%cell cards.

%%
%Debug only

% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')


%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

% If the assembly is experimental, then the comment line will reflect that

if MatMap{MSATyC}==10

    surfname=NameCard(['EXP Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    surfname=NameCard(['Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end


%%
% The following for loop creates the core region sections.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
 surfctemp1=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)-lch-hdwt,0,0,crh+lch+2*hdwt,...
 hdor, '$ Outer Wall of Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf1=SAsurfnum;

%Update the surnum number
 SAsurfnum=SAsurfnum+1;

 if LAT==1

     surfctemp1=[surfctemp1; CharChecker({surfnum,'    SO',2*hdh,...
         '$ Surr Na For Lat'})];

     surf15=surfnum;

     surfnum=surfnum+1;

 end

% Inner Duct Core Region
 surfctemp2=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4),0,0,crh,hdir,...
 '$ Inner Wall Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf2=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

surfctemp=[surfctemp1; surfctemp2];
```

```
%%
% Lower Extension Surface Cards

surfctemplo3=CharChecker({surfnum,'    PZ', hdorg(4)-hdwt,...
                '$ Plane Separation Duct to Cylinder'});

%I need this facet for later.
LEPl=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Cylinder Extension

surfctemplo4=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

%Assign the surface number to be used later in the cell creation
LECyl=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Append the surfaces to the surface card.
surfctemp=[surfctemp; surfctemplo3;  surfctemplo4];

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={surf1};

else

    hexsurfnum=[hexsurfnum, {surf1}];

end

%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nde,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.

porg=PinOrg(nde,dep,hdsorg(2),hdsorg(3),hdorg(4)+deo);

for ii=1:nde

%Like the main program, the origins of the pins need to be determined so
%that they can be fed into a program to write the cards. This is what the
%pinorg program does.

%ocws = Outter Cladding Wall Surface
%wws = wire wrap surface

    [ pincellctemp, pinsurfctemp, ocws , ~ ]=...
        DummyPinMaker(porg(ii,:),dpind,dpinm,dpinparm);

    pinsurfnum(ii)=ocws;

    if (ii==1)

        pinsurfc=pinsurfctemp;
        pincellc=pincellctemp;

    end

    if (ii>1)

        pinsurfc=[pinsurfc; pinsurfctemp];
        pincellc=[pincellc; pincellctemp];

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
```

```
%assemblies

cellname=NameCard(['Dummy M#' num2str(hdorg(1)) ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

if MatMap{MSATyC}==10

    cellname=NameCard(['EXP Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    cellname=NameCard(['Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end

%% %Hex Duct Cell cards

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Dummy: ' num2str(hdorg(1))],'Divider');

%***************************** Benchmark *********************************

% Outter Duct Wall

cellctemp1=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
     surf2,LEPl,UNIC,'$ Hex Duct Assm'});

cellnum=cellnum+1;

 if LAT

     cellctemp1=[cellctemp1; CharChecker({cellnum,'     ',Dnam,...
         Dnad,surf1,-surf15,UNIC,'$ INF Na for Lat'})];

    cellnum=cellnum+1;

 end

%Inside duct Core Region

cellctemp2={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

%Sodium around the lower extension cylinder

cellctemplo1=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1,...
    -LEPl,LECyl,UNIC,'$ Na Surr Lower Ext'});

cellnum=cellnum+1;

% Lower Cylinder

cellctemplo2=CharChecker({cellnum,'     ',SLCm,SLCd, -LECyl,...
    UNIC,'$ Lower Cyn Homog'});

cellnum=cellnum+1;

%***************************** Benchmark *********************************

%Adds the pin surface numbers to the inside duct definition

for mm=1:size(pinsurfnum,1)

    cellctemp2=[cellctemp2 pinsurfnum(mm)];

end

%Taks on the importance and comment

cellctemp2=CharChecker([cellctemp2,UNIC,...
    '$ Sodium coolant']);

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

    cellctemp=[cellctemp1; cellctemp2; cellctemplo1; cellctemplo2];

%***************************** Benchmark *********************************

%% Appending the cards together

surfc=[surfname; surfctemp; pinsurfc];
cellc=[cellname; cellname2; cellctemp; pincellc;];

%%

global SaveSAMakerVars

if SaveSAMakerVars
```

```
    if MatMap{MSATyC}==10

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_EXP_' MatMap{SANomen} '_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    else

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Dummy_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    end

end

end
```

B.55. DummyPinMaker.m

```
function [ cellc, surfc , ocws , wr ] = DummyPinMaker(porg,dpind,dpinm,dpinparm,...
                                                hdsorg,secnum)
%This function creates the pins in the same manner as the hexmaker.

global surfnum
global cellnum
global planesurfnum
global BorD

%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

%Sets the MICKA num
hdnum=dpinparm{1};

%Universe / IMP
UNIC=dpinparm{4};

%Mcnpfix
mcnpfix=100*dpinparm{end};

% Initialize wire wrap surface number
wr=0;

%%
%****************Define the pin material parameters**********************

%cladding material and density
cladm=dpinm(1,1);
cladd=dpinm(1,2);

%Inner material
deim=dpinm(2,1);
deid=dpinm(2,2);

%wirewrap material
dewwrm=dpinm(3,1);
dewwrd=dpinm(3,2);

%%
%*********************************Pin Dimensions**************

%Dummy element wall thickness.
dewt=dpind(4);

%Pin Inner radius
deir=dpind(5);

%Pin height
deh=dpind(6);

%Pin outter radius
deor=dpind(7);

%Wire Wrap radius
dewwr=dpind(8);

% Dummy Element Wire Wrap height
dewwh=dpind(9);

%Wire Wrap Switch
% This switch controls if a wire wrap is created or not.
dewws=dpind(10);

% Dummy Element Solid switch determines if a solid dummy pin is being made
% or a hollow one.
dess=dpind(11);

%%
%*****************************Surface Cards*****************************
```

```
%Since the cards for the pin and the subsequent pin sections will become
%huge, I will put a title in the cards for each pin now.

surfname=NameCard(['Pin: ' num2str(porg(1)) ' of SA: ' num2str(hdnum)],'Divider');

%Outer Diameter Cladding

surfctemp=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)...
    ,0,0,deh,deor,'$ Pin:',porg(1),'Dummy Pin Outter Clad'});

%Assign the surface number to be used later in the cell creation
ocws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

if dess==0

    %Inner Diameter Cladding

    surfctemp=[surfctemp; CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+dewt...
        ,0,0,deh-2*dewt,deir,'$ Pin:',porg(1),'Dummy Pin Inner Clad'})];

    icws=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

end

if dewws==1

    %WireWrap
    %This will have to call another function to calculate and generate the
    %wirewrap positions. For right now we are just going to do one cylinder
    %running up the side of the outter pin cladding. The cylinder will be
    %parrallel to the pin with a 90 degree offset in the xy plane.

    surfctemp2=CharChecker({surfnum,'    RCC',porg(2)+deor+dewwr+mcnpfix,...
        porg(3),porg(4),0,0,dewwh,dewwr,'$ Pin:',porg(1),'Wire Wrap'});

    %Since the wirewrap is on the outside of the pin, it also needs to give its
    %number to pinsurf so that hex maker can know the sodium is on the outside.

    wr=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

end

%%
%*****************************Cell Cards*********************************
% Pin number title to seperate the pins

cellname=NameCard(['Pin: ' num2str(porg(1)) ' Sec: ' ...
                   ' of SA: ' num2str(hdnum)],'Divider');

% These if statements are to check if this is the top of the hex duct section.
% the impact of this is that the top plane does not need to be used.

% Top part above the top of the element and above the sodium height.


%***************************** Benchmark **********************************
%Creates a dummy pin element

        if dess==0

            % Outter Clad

            cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
                icws,UNIC,'$ Pin:',porg(1),'Dummy Pin OClad'});

            cellnum=cellnum+1;

            % Inner Clad

            cellctemp=[cellctemp; CharChecker({cellnum,'     ',deim,deid, -icws,...
                UNIC,'$ Pin:',porg(1),'Dummy Pin Inside Region'})];

            % Update the cell number
             cellnum=cellnum+1;

        end

        if dess==1

            % Outter Clad

            cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
                UNIC,'$ Pin:',porg(1),'Dummy Pin'});

            % Update the cell number
```

```
             cellnum=cellnum+1;

        end

        if dewws==1

            %Wire Wrap

            cellctemp2= CharChecker({cellnum,'     ',dewwrm,dewwrd,...
                    -wr,UNIC,'$ Pin:',porg(1),' Wire Wrap'});

            % Update the cell number
             cellnum=cellnum+1;

        end

%***************************** Benchmark *********************************


%%
%Assign the cards to be returned to hexmaker.

if dewws==0

    % No Wire Wrap
    surfc=[surfname; surfctemp];
    cellc=[cellname; cellctemp];

end

if dewws==1

    % Wire Wrap
    surfc=[surfname; surfctemp; surfctemp2];
    cellc=[cellname; cellctemp; cellctemp2];


end

end
```

B.56. ReflectorMaker.m

```
function [ cellc,surfc ] = ReflectorMaker(hdorg,mcnpfix,bend,MatMap,DimMap)
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global surfnum
global cellnum
global hexsurfnum
global planesurfnum
global SAsurfnum
global LAT
global BorD

 %% Reactor Map Vars Breakout

global SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC crhC lcdC lchC sazoC

%% Material Vars

global DuctmC DuctdC DnamC DnadC  SLOmC SLOdC LAmC LadC DmC DdC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).


%*************************************************************************
%Sets the universe num

if LAT==1

    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

%*************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
```

```
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
crh=DimMap{crhC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
lcr=lcd/2;

%*************************************************************************
%Subassembly movement

%Since the dummy is just one long core region, I decided to adjust the
%origin of the whole assembly like a control rod.

sazo=DimMap{sazoC};

%************************************************************************
%This section is the reflector origin adjustment.

%Offest plus movement
hdorg(4)=hdorg(4)-sazo;

%************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Duct Fluid

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% HexBlocks

Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%************************************************************************

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);

%This next line performs the expansion. It moves the pins slightly outward
%from the center of the core. This will be replaced by pin section movement
%later on in PinSectionMaker but for not it will reside here as a proof of
%concept.

%porg=ThermalX(porgtemp);

%The surfaces must be written first for the duct and all the pins, then the
%cell cards.

%%
%Debug only

% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Reflector M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%%
% The following for loop creates the core region sections.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
 surfctemp1=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
```

```
 hdsorg(3),hdorg(4)-lch-hdwt,0,0,crh+lch+2*hdwt,...
 hdor, '$ Outer Wall of Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf1=SAsurfnum;

%Update the surnum number
 SAsurfnum=SAsurfnum+1;

 if LAT==1

     surfctemp1=[surfctemp1; CharChecker({surfnum,'    SO',2*hdh,...
         '$ Surr Na For Lat'})];

     surf15=surfnum;

     surfnum=surfnum+1;

 end

 % Inner Duct Core Region
 surfctemp2=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4),0,0,crh,hdir,...
 '$ Inner Wall Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf2=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

surfctemp=[surfctemp1; surfctemp2];

%%
% Lower Extension Surface Cards

surfctemplo3=CharChecker({surfnum,'    PZ', hdorg(4)-hdwt,...
                '$ Plane Separation Duct to Cylinder'});

%I need this facet for later.
LEPl=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Cylinder Extension

surfctemplo4=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

%Assign the surface number to be used later in the cell creation
LECyl=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Append the surfaces to the surface card.
surfctemp=[surfctemp; surfctemplo3;  surfctemplo4];

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={surf1};

else

    hexsurfnum=[hexsurfnum, {surf1}];

end


%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['Reflector M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%% %Hex Duct Cell cards

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Reflector: ' num2str(hdorg(1))],'Divider');

%**************************** Benchmark *********************************

% Outter Duct Wall

cellctemp1=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
    surf2,LEPl,UNIC,'$ Hex Duct Reflector'});
```

```
 cellnum=cellnum+1;

 if LAT==1

     cellctemp1=[cellctemp1; CharChecker({cellnum,'     ',Dnam,...
         Dnad,surf1,-surf15,UNIC,'$ INF Na for Lat'})];

    cellnum=cellnum+1;

 end

% Hex Blocks

cellctemp2=CharChecker({cellnum,'     ',Dm,Dd, -surf2,...
  UNIC,'$ Hex Block'});

cellnum=cellnum+1;

%Sodium around the lower extension cylinder

cellctemplo1=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1,...
    -LEPl,LECyl,UNIC,'$ Na Surr Lower Ext'});

cellnum=cellnum+1;

% Lower Cylinder

cellctemplo2=CharChecker({cellnum,'     ',SLCm,SLCd, -LECyl,...
    UNIC,'$ Lower Cyn Homog'});

cellnum=cellnum+1;

%***************************** Benchmark *********************************


%***************************** Benchmark *********************************

        %Appends the upper lower extensions and Core Region cards

            cellctemp=[cellctemp1; cellctemp2; cellctemplo1; cellctemplo2];

%***************************** Benchmark *********************************


%% Appending the cards together

surfc=[surfname; surfctemp];
cellc=[cellname; cellname2; cellctemp];

%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Reflector_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.57. BlanketMaker.m

```
function [ cellc,surfc  ] = BlanketMaker( hdorg,mcnpfix,bend,MatMap,DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugBlankNDE
global surfnum cellnum
global hexsurfnum planesurfnum ebPl SAsurfnum createplane
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC lcdC lchC
global ndeC deodC dewtC dehC deoC depC eshC esdC eslasC sazoC

%% Material Vars

global FmC FdC FNamC FNadC FCladmC FCladdC
global DuctmC DuctdC DnamC DnadC SLOmC SLOdC LAmC LadC

%*************************************************************************
%Sets whether this is benchmark or Dissertation
```

```
%BorD = true (Bench) or not (Dissert).

%*********************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
crh=DimMap{crhC}; lcd=DimMap{lcdC}; lch=DimMap{lchC}; bh=DimMap{bhC};

% Dummy Elements, they are the same as fuel elements
deh=DimMap{dehC}; deod=DimMap{deodC}; dewt=DimMap{dewtC}; deid=deod-(2*dewt);...
    deo=DimMap{deoC}+mcnpfix; nde=DimMap{ndeC}; dep=DimMap{depC};

% Blanket Slugs, they are assigned as experimental.
esh=DimMap{eshC}; esd=DimMap{esdC}; eslas=DimMap{eslasC};

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
lcr=lcd/2;
deor=deod/2;
deir=deid/2;
esr=esd/2;

%Wire Wrap Switch 1 = on 0 = off.
dewws=0;

% Set the number of lattice elements if applicable
if LATp; nle=nde; end

% Override pin count if debug micka is true
if debugMICKA; nde=DebugBlankNDE; end

%*********************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LUN=hdorg(1)*1000+1;

    LUNu=strjoin({'U=' Num2StrM(LUN)},'');
    LUNf=strjoin({'fill=' Num2StrM(LUN)},'');

    LUNfp=LUN+1;

    % Since this is a driver, the pins are all the same universe number.
    Latnums=repmat(LUNfp,1,nde);

    % Debugging line to pad the universemap with zeros.
    if debugMICKA; Latnums=[repmat(LUNfp,1,nde), zeros(1,nle-nde)]; end

    LatEler=dep/2;
    LatEleh=2*deh; % This height is increased to not give a perfect fit
    LatElezo=-deh/2;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    % LATpino=bh;
     LATpino=0;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;
```

```
    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice array size

    LATr=hdir-(hdir/1000);
    LATh=deh+2*mcnpfix;
    LATzo=mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nde=1;   % New number of pins

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;

end

%**********************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

neslas=eslas+esh+deo;

bpinparm={hdorg(1),ebPl,neslas,UNICfp,mcnpfix};

%**********************************************************************
%Subassembly movement

%Since the dummy is just one long core region, I decided to adjust the
%origin of the whole assembly like a control rod.

sazo=DimMap{sazoC};

%**********************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Duct Fluid

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

%**********************************************************************
% Pin Materials

% Blanket Slug

BPm=MatMap{FmC}';
BPd=MatMap{FdC};

% Cladding

BPCladm=MatMap{FCladmC};
BPCladd=MatMap{FCladdC};

% Wire Wrap

BPWwrm=MatMap{FCladmC};
BPWwrd=MatMap{FCladdC};

% Blanket Fluid Bond

BPnam=MatMap{FNamC};
BPnad=MatMap{FNadC};

% Blanket plenum appears to be steel. But keeping gas as placeholder
BPPm=BPCladm;
BPPd=BPCladd;

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%%
%**********************************************************************

%Assigns dimensions and materials for the pins.
```

```
bpinm=[BPm,BPd;BPCladm,BPCladd;BPWwrm,BPWwrd;BPnam,BPnad;BPPm,BPPd];
bpind=[NaN;esh;esr;dewt;deir;deh;deor;NaN;NaN];

%**********************************************************************
%%

%This section is the dummy origin adjustment.

%Offest plus movement
hdorg(4)=hdorg(4)-sazo;

%**********************************************************************
%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);

%This next line performs the expansion. It moves the pins slightly outward
%from the center of the core. This will be replaced by pin section movement
%later on in PinSectionMaker but for not it will reside here as a proof of
%concept.

%porg=ThermalX(porgtemp);


%The surfaces must be written first for the duct and all the pins, then the
%cell cards.

%%
%Debug only

% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')


%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Blanket M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%%
% The following for loop creates the core region sections.

for ii=1:size(hdsorg,1)

        %This for loop creates the surface cards for the hex duct sections

        % Outer Duct Assembly
         ODOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(ii,2),...
         hdsorg(ii,3),hdorg(4)-lch-hdwt,0,0,hdh+lch+2*hdwt,...
         hdor, '$ Outer Wall of Hex Duct'});

         %Assign the surface number to be used later in the cell creation
         surf1(ii)=SAsurfnum;

        %Update the surnum number
         SAsurfnum=SAsurfnum+1;

         if (LAT || LATp) && ii==1

             LATNas=CharChecker({surfnum,'    SO',2*hdh,...
                 '$ Surr Na For Lat'});

             surf15=surfnum;

             surfnum=surfnum+1;

         end

        % Inner Duct Core Region
         CRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
         hdsorg(ii,3),hdorg(4),0,0,hdh,hdir,...
         '$ Inner Wall Hex Duct'});

         %Assign the surface number to be used later in the cell creation
         surf2(ii)=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

     if ii==1

         if (LAT || LATp); surfctemp=[ODOWs; LATNas; CRs ];
         else    surfctemp=[ODOWs; CRs ]; end

     else

         if (LAT || LATp); surfctemp=[surfctemp; ODOWs; LATNas; CRs ];
         else    surfctemp=[surfctemp; ODOWs; CRs ]; end
```

```
     end

end


%%
% Lower Extension Surface Cards


ODLPls=CharChecker({surfnum,'    PZ', hdorg(4)-hdwt,...
                '$ Plane Separation Duct to Cylinder'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Cylinder Extension

ODCyls=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

%Assign the surface number to be used later in the cell creation
ODCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

if LATp

    %Naming card
    LATname=NameCard('Pin Lattice Cards for Elements','Divider');

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LATzo ...
        ,0,0,LATh,LATr,'$ Element Lattice Bounding Surface'});

    LATpArnum=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezo,0,0,LatEleh,0,LatEler,...
        0,'$ Lattice Window'});

    LatWinElenum=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfc=[LATname; LATArs; LatWinEle];

 end

% Append the surfaces to the surface card.
surfctemp=[surfctemp; ODLPls;  ODCyls];

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={surf1};

else

    hexsurfnum=[hexsurfnum, {surf1}];

end

%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nde,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

for kk=1:size(hdsorg,1)

    %The new HD section origin must be changed for porg to work

    %Pin maker has to be adjust just like hex maker to account for the
    %slices. This part must be done in a series of if statements so that
    %the slicing does not cut half sections.

    %Porg calculates the origin of all the pins in a particular section.
```

```
    porg=PinOrg(nde,dep,hdsorg(kk,2),hdsorg(kk,3),hdorg(4)+deo);

     % The pin origin needs to be moved slightly
     if LATp; porg(:,2)=hdsorg(kk,2)-LATpino; porg(:,3)=hdsorg(kk,3); end

    for ii=1:nde

    %Like the main program, the origins of the pins need to be determined so
    %that they can be fed into a program to write the cards. This is what the
    %pinorg program does.

    %ocws = Outter Cladding Wall Surface
    %wws = wire wrap surface

try
        [ pincellctemp, pinsurfctemp, ocws ]=...
            BlanketPinMaker(porg(ii,:),bpind,bpinm,bpinparm,hdsorg,kk);
catch;keyboard;end
        pinsurfnum(ii,kk)=ocws;

        if (ii==1)&&(kk==1)

            FPs=pinsurfctemp;
            FPc=pincellctemp;

        end

        if (ii>1)||(kk>1)

            FPs=[FPs; pinsurfctemp];
            FPc=[FPc; pincellctemp];

        end
    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['Blanket M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%% %Hex Duct Cell cards

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Blanket: ' num2str(hdorg(1))],'Divider');

for ii=1:size(hdsorg,1)

    %Duct wall

    if ii==1

           if BorD==0;

            % Outter Duct Wall

            ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
                 surf2(ii), planesurfnum(ii),...
                'IMP:N=1','$ Hex Duct Upper Ext'});

            cellnum=cellnum+1;

            %Inside duct Core Region

            CRc={cellnum,'     ',Dnam,Dnad, -surf2(ii),planesurfnum(ii)};

            cellnum=cellnum+1;

          else
%**************************** Benchmark *********************************

            % Outter Duct Wall

            ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(1),...
                 surf2(1),ODLPlnum,UNIC,'$ Hex Duct Assm'});

            cellnum=cellnum+1;

            %Inside duct Core Region

            CRc={cellnum,'     ',Dnam,Dnad, -surf2(ii)};

            cellnum=cellnum+1;

            %Sodium around the lower extension cylinder

            CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1(ii),...
```

```
                -ODLPlnum,ODCylnum,UNIC,'$ Na Surr Lower Ext'});

            cellnum=cellnum+1;

            % Lower Cylinder

            Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
                UNIC,'$ Lower Cyn Homog'});

            cellnum=cellnum+1;

             if LAT

                 LATNac=CharChecker({cellnum,'     ',Dnam,...
                     Dnad,surf1(1),-surf15,UNIC,'$ INF Na for Lat'});

                cellnum=cellnum+1;

             end

            if LATp

                %name the area

                LATpcellname=NameCard('Pin Lattice Cards','Divider');

                %Create the lattice fill card

                LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum,'Lat=2',LUNu,...
                    'IMP:N=1 $ Element Lattice'});

                cellnum=cellnum+1;

                LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

                %Creates the lattice line, fills in the outside zeros with the lat universe
                %aka the wall

                [LATparln,LATar]=MCNPPOSMaker( LUN, nle ,nar, Latnums );

                LATpcLns=CharChecker({'     ' Num2StrM(LATparln)});

                % Fills the lattice into the core.

                LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum,LUNf,...
                    UNIC,' $ Pin Lattice'});

                cellnum=cellnum+1;

                % Append the cards together.

                LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

             end

%***************************** Benchmark *********************************

           end

    end

    if ii~=1&&ii~=size(hdsorg,1)

        ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
            surf2(ii),-planesurfnum(ii-1),planesurfnum(ii),...
            'IMP:N=1','$ Hex Duct'});

        cellnum=cellnum+1;

        CRc={cellnum,'     ',Dnam,Dnad, -surf2(ii),...
            -planesurfnum(ii-1),planesurfnum(ii)};

        cellnum=cellnum+1;

    end

    if ii ~= 1 && ii==size(hdsorg,1)

        ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
            surf2(ii),ODLPlnum,-planesurfnum(ii-1),...
            'IMP:N=1','$ Hex Duct'});

        cellnum=cellnum+1;

        %Inside duct Core Region

        CRc={cellnum,'     ',Dnam,Dnad, -surf2(ii),-planesurfnum(ii-1)};

        cellnum=cellnum+1;

        %Sodium around the lower extension cylinder

        CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1(ii),...
            -ODLPlnum,ODCylnum,'IMP:N=1 $ Na Surr Lower Ext'});
```

```
        cellnum=cellnum+1;

        % Lower Cylinder

        Cylc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODCylnum,...
            'IMP:N=1 $ Lower Cyn Homog'});

        cellnum=cellnum+1;

    end

    if LATp

        % This defines the surrounding sodium around the fuel pin.

        FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1),...
            UNICfp, '$ Blanket Pin Cell'});

        cellnum=cellnum+1;

        %Taks on the importance and comment

            CRc=CharChecker([CRc,LATpArnum,UNIC,...
                '$ Sodium coolant']);


    else

        %Adds the pin surface numbers to the inside duct definition

        for mm=1:size(pinsurfnum,1)

            CRc=[CRc pinsurfnum(mm,ii)];

        end

        %Taks on the importance and comment

            CRc=CharChecker([CRc,UNIC,...
                '$ Sodium coolant']);

    end

    if ii==1

        %Appends the upper extensions cards

        if BorD==0

        %Appends the upper extensions cards

             cellctemp=[ODc; CRc];

        else

%***************************** Benchmark *********************************

        %Appends the upper lower extensions and Core Region cards


        if LATp

            if LAT

                cellctemp=[ODc; LATNac; CRc; FPc; FPNac; LATpc; ...
                    CylNac; Cylc];

            else

                cellctemp=[ODc; CRc; FPc; FPNac; LATpc; ...
                    CylNac; Cylc];

            end

        else

            if LAT

                cellctemp=[ODc; LATNac; CRc; FPc; ...
                    CylNac; Cylc];
            else

                cellctemp=[ODc; CRc; FPc; ...
                    CylNac; Cylc];

            end

        end

%***************************** Benchmark *********************************

        end

    end
```

```
    if ii~=1 && ii~=size(hdsorg,1)

       % Appends the mid section cards together

       cellctemp=[cellctemp; ODc; CRc];

    end


    if  ii~=1 && ii==size(hdsorg,1)

       %Appends the lower extension cards together

       cellctemp=[cellctemp; ODc; CRc;...
           CylNac; Cylc; ];

    end


end


%% Appending the cards together

if LATp

    surfc=[surfname; surfctemp; FPs; LATsurfc];
    cellc=[cellname; cellname2; cellctemp];

else

    surfc=[surfname; surfctemp; FPs];
    cellc=[cellname; cellname2; cellctemp];

end

%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Blanket_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.58. BlanketPinMaker.m

```
function [ cellc, surfc , ocws ] = BlanketPinMaker( porg,bpind,bpinm,bpinparm,...
                                                    hdsorg,secnum )
%This function creates the pins in the same manner as the hexmaker.
% MKIIA pin

%This function Designs a pin as follows

% A slug immersed in fluid
% Fluid Plenum above the immersed slug
% Cladding around both fluids
% Wire Wrap around the cladding

global surfnum
global cellnum
global planesurfnum
global BorD

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

if BorD

    if size(bpinm,1)>1

    % These lines set the plane surfnums.

    PinPlanes=bpinparm{2};

    gasbarsurfnum=PinPlanes{1,1};
    sec12surfnum=PinPlanes{2,1};
    sec23surfnum=PinPlanes{3,1};

    else

        PinPlanes=bpinparm{2};

        gasbarsurfnum=PinPlanes{1,1};

    end
```

```
end

%Sets the MICKA num
hdnum=bpinparm{1};

%Fuel sodium level
eslas=bpinparm{3};

%Universe plus impoartance
UNIC=bpinparm{4};

%Mcnpfix
mcnpfix=100*bpinparm{end};

%%
%****************Define the pin material parameters**********************

if BorD && size(bpinm,1)>1

    pinmln=3;

    slugs1m=bpinm(pinmln-2,1);
    slugs1d=bpinm(pinmln-2,2);
    slugs2m=bpinm(pinmln-1,1);
    slugs2d=bpinm(pinmln-1,2);
    slugs3m=bpinm(pinmln,1);
    slugs3d=bpinm(pinmln,2);

else

    pinmln=1;
    slugs1m=bpinm(pinmln,1);
    slugs1d=bpinm(pinmln,2);

end

%cladding material and density
cladm=bpinm(pinmln+1,1);
cladd=bpinm(pinmln+1,2);

% wire wrap material and density
wwrm=bpinm(pinmln+2,1);
wwrd=bpinm(pinmln+2,2);

%Coolant material and density
nam=bpinm(pinmln+3,1);
nad=bpinm(pinmln+3,2);

%plenum area material and density
PGm=bpinm(pinmln+4,1);
PGd=bpinm(pinmln+4,2);

%%
%*********************************Pin Dimensions**************

%Fuel slug height
bsh=bpind(2);

%Fuel Slug Radius
bsr=bpind(3);

%cladding wall thickness
bewt=bpind(4);

%Cladding inner radius
beir=bpind(5);

%Pin height
beh=bpind(6);

%Pin radius
beor=bpind(7);


%%
%*******************************Surface Cards*****************************

%Since the cards for the pin and the subsequent pin sections will become
%huge, I will put a title in the cards for each pin now.

surfname=NameCard(['Pin: ' num2str(porg(1)) 'SA: ' num2str(hdnum)],'Divider');

%Outer Diameter Cladding

surfctemp=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)...
    ,0,0,beh,beor,'$ Pin:',porg(1),'Outer Cladding wall'});

%Assign the surface number to be used later in the cell creation
ocws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Inner Diameter Cladding
```

```
surfctemp2=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+bewt...
    ,0,0,beh-2*bewt,beir,...
    '$ Pin:',porg(1),'Inner cladding wall'});

%Assign the surface number to be used later in the cell creation
icws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Fuel slug cylinder

surfctemp3=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+bewt+mcnpfix...
    ,0,0,bsh,bsr,'$ Pin:',porg(1),'Blanket slug boundary'});

%Assign the surface number to be used later in the cell creation
fs=surfnum;

%Update the surnum number
surfnum=surfnum+1;


%%
%*****************************Cell Cards*********************************
% Pin number title to seperate the pins

cellname=NameCard(['Pin: ' num2str(porg(1)) ' Sec: ' num2str(secnum)...
                   ' of SA: ' num2str(hdnum)],'Divider');

% These if statements are to check if this is the top of the hex duct section.
% the impact of this is that the top plane does not need to be used.

% Top part above the top of the element and above the sodium height.

if secnum==1

    if ~BorD;

    %Dissertation

    % Cladding

    cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
        icws,planesurfnum(secnum),'IMP:N=1','$ Pin:',porg(1),...
        'Cladding'});

    cellnum=cellnum+1;

    % Helium

    cellctemp1=CharChecker({cellnum,'     ',PGm,PGd, planesurfnum(secnum),...
        -icws,'IMP:N=1','$ Pin:',porg(1),'Plenum Gas'});

    cellnum=cellnum+1;

    else

%***************************** Benchmark *********************************
%Creates a fuel element

        % Cladding

        cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
            icws,UNIC,'$ Pin:',porg(1),'Cladding'});

        cellnum=cellnum+1;

        % Helium

        cellctemp1=CharChecker({cellnum,'     ',PGm,PGd, gasbarsurfnum,...
            -icws,UNIC,'$ Pin:',porg(1),'Plenum Gas'});

        cellnum=cellnum+1;

        % Blanket Slug sec1

        cellctemp2=CharChecker({cellnum,'     ',slugs1m,slugs1d, -fs,sec12surfnum,...
              UNIC,'$ Pin:',porg(1),'Blanket Slug Sec 1'});

         cellnum=cellnum+1;

         if pinmln==3

            %Fuel sec2

            cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',slugs2m,slugs2d,...
                -fs,-sec12surfnum, sec23surfnum,...
                UNIC,'$ Pin:',porg(1),'Blanket Slug Sec 2'})];

             cellnum=cellnum+1;

            %Fuel sec2

            cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',slugs3m,slugs3d, -fs,...
```

```
                -sec23surfnum, UNIC,'$ Pin:',porg(1),'Blanket Slug Sec 3'})];

            cellnum=cellnum+1;

         end

        % Sodium

        cellctemp3=CharChecker({cellnum,'     ',nam,nad,...
             -gasbarsurfnum, -icws,fs,UNIC,'$ Pin:',porg(1),'Sodium'});

        cellnum=cellnum+1;

        cellctemp1=[cellctemp1; cellctemp2; cellctemp3];

%***************************** Benchmark *********************************

    end

    %Append the cell cards

    cellc=[cellname; cellctemp; cellctemp1];

end

% Below the top of element and above the sodium height.

if secnum > 1 && eslas <= hdsorg(secnum,4)

    cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
        icws,-planesurfnum(secnum-1),planesurfnum(secnum),...
        'IMP:N=1','$Pin:',porg(1),'Sec # ',num2str(secnum),'Cladding'});

    cellnum=cellnum+1;

    % Argon

    cellctemp1=CharChecker({cellnum,'     ',PGm,PGd, -planesurfnum(secnum-1),...
        planesurfnum(secnum),-icws,'IMP:N=1',...
        '$Pin:',porg(1),'Sec # ',num2str(secnum),'Plenum Gas'});

    cellnum=cellnum+1;

    %Append the cell cards

    cellc=[cellname; cellctemp; cellctemp1];

end

%Below the sodium

if secnum > 1 && eslas > hdsorg(secnum,4) && secnum~=size(hdsorg,1)

    % cladding

    cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
        icws,-planesurfnum(secnum-1),planesurfnum(secnum),...
        'IMP:N=1','$Pin:',porg(1),'Sec # ',num2str(secnum),'Cladding'});

    cellnum=cellnum+1;

    % Sodium

    cellctemp1=CharChecker({cellnum,'     ',nam,nad, -planesurfnum(secnum-1),...
        planesurfnum(secnum),-icws,fs,'IMP:N=1',...
        '$Pin:',porg(1),'Sec # ',num2str(secnum),'Sodium'});

    cellnum=cellnum+1;

    %Fuel

    cellctemp2=CharChecker({cellnum,'     ',slugs1m,slugs1d, -fs,...
        -planesurfnum(secnum-1),planesurfnum(secnum),'IMP:N=1',...
        '$Pin:',porg(1),'Sec # ',num2str(secnum),'Fuel Slug'});

    cellnum=cellnum+1;

    %Append the cell cards

    cellc=[cellname; cellctemp; cellctemp1; cellctemp2];

end

%Bottom of the element.

if secnum ~= 1 && secnum==size(hdsorg,1)

    %cladding

    cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
        -planesurfnum(secnum-1),icws,...
        'IMP:N=1','$Pin:',porg(1),'Sec # ',num2str(secnum),'Cladding'});

    cellnum=cellnum+1;
```

```
    % Sodium

    cellctemp1=CharChecker({cellnum,'     ',nam,nad, -planesurfnum(secnum-1),...
        -icws,fs,'IMP:N=1',...
        '$Pin:',porg(1),'Sec # ',num2str(secnum),'Sodium'});

    cellnum=cellnum+1;

    %Fuel

    cellctemp2=CharChecker({cellnum,'     ',slugs1m,slugs1d, -fs,...
        -planesurfnum(secnum-1),'IMP:N=1',...
        '$Pin:',porg(1),'Sec # ',num2str(secnum),'Fuel Slug'});

    cellnum=cellnum+1;

    %Append the cell cards

    cellc=[cellname; cellctemp; cellctemp1; cellctemp2];

end

%%
%Assign the cards to be returned to hexmaker.
surfc=[surfname; surfctemp; surfctemp2; surfctemp3];


end
```

B.59. CustomMaker.m

```
function [ cellc,surfc ] = CustomMaker( hdorg,mcnpfix,bend,MatMap,DimMap )

%This function pairs out any custom assemblies that need to be created.
% The ebr-ii list is as follows

%{

MICKA ID: | MICKA Type | EBR-II Position | ID

    21           10            04C02        C2776A
    24           10            04D02        X320C
    38           10            05C01        XX10
    44           10            05D03        XX09
    52           10            05F03        XY-16
    67           10            06D01        X412
    89           10            06B03        X402A

%}


% Grab the identifier

EXPID=MatMap{1,4};

switch EXPID

    case 'C2776A'

    % Since this assembly is just a tagged driver it will be called as just
    % a regular driver with no custom options.

        [ cellc,surfc ] = DriverMakerMKII(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 'X320C'

    % This assembly is just being modeled as a dummy so the regular dummy
    % assembly will be called.

        [ cellc,surfc ] = DummyMaker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 'XX10'

    % This is the instrumented assembly and is completely custom. It is a
    % 19 pin assembly that has similar dimensions to a driver but the pins
    % are a larger diameter. It has 18 solid ss316 pins and 1 hollow pin.
    % To model this assembly, it will require the use of the half worth
    % driver with custom modifications to accomodate the unique geometry.

        [ cellc,surfc ] = XX10Maker(hdorg,mcnpfix,bend, MatMap, DimMap );

    case 'XX09'

    % This is the instrumented assembly and is completely custom. It is a
    % 61 pin assembly that has similar dimensions to a driver but is the
    % size of a CONTROL rod. It has 59 fuel pins and 2 hollow pins.
    % To model this assembly, it will require the use of the half worth
    % driver with custom modifications to accomodate the unique geometry.

       [ cellc,surfc ] = XX09Maker(hdorg,mcnpfix,bend,MatMap,DimMap);
```

```
    case 'XY-16'

    % This is just a dummy control rod, so it has the same dimensions
    % and materials just solid pins.

       [ cellc,surfc ] = XY16Maker(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 'X412'

        % This is just a driver but it appears to be a MKIIB driver,
        % whatever that means.

       [ cellc,surfc ] = DriverMakerMKII(hdorg,mcnpfix,bend,MatMap,DimMap);

    case 'X402A'

        % This is just a driver with the center pin replaced with a dummy.

       [ cellc,surfc ] = X402AMaker(hdorg,mcnpfix,bend,MatMap,DimMap);


end


end
```

B.60. XX10Maker.m

```
function [ cellc,surfc ] = XX10Maker(hdorg,mcnpfix,bend, MatMap, DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugXX10NFE DebugXX10NDE
global surfnum cellnum
global hexsurfnum planesurfnum fbPl SAsurfnum createplane
global LAT
global BorD


 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fewwdC
global ndeC ihdhC ihdodC ihdwtC ihdlcdC ihdlchC deodC dewtC dehC deoC depC
global nfeC

%% Material Vars

global DuctmC DuctdC DnamC DnadC SUPmC SUPdC SLOmC SLOdC IHDLAdC IHDLAmC
global DmC DdC PPGmC PPGdC OPdC OPmC

% This function creates the XX10 subassembly. The code has been copied from
% a half worth driver and a CONTROL rod and is modified to support the create of solid dummy
% pins and hollow ones.

% This is achieved by setting the solid pins to be the "fuel" and the
% hollow pin to be the dummy. Any time a variable with the "f" suffix is
% used, it means solid pin. "d" suffix means hollow;


%***************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%***************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

%***************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
bh=DimMap{bhC}; crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC};
```

```
lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Inner Hex Duct
ihdh=DimMap{ihdhC}; ihdod=DimMap{ihdodC}; ihdwt=DimMap{ihdwtC}; ihdid=ihdod-(2*ihdwt);
ihdlcd=DimMap{ihdlcdC}; ihdlch=DimMap{ihdlchC};

% Dummy Elements, these are the hollow ones!
deod=DimMap{deodC}; dewt=DimMap{dewtC}; deid=deod-(2*dewt); deh=DimMap{dehC};
deo=DimMap{deoC}; dewwd=DimMap{fewwdC}; dewwh=deh; dep=DimMap{depC}; nde=DimMap{nfeC};

%Fuel Elements, these are the solid ones!
feh=deh; feod=deod; feo=deo; fewt=dewt; fewwh=feh; fewwd=DimMap{fewwdC};
nfe=DimMap{ndeC}-nde; fep=dep;

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
ihdor=ihdod/2;
ihdir=ihdid/2;
ihdlcr=ihdlcd/2;
feor=feod/2;
lcr=lcd/2;
fewwr=fewwd/2;
deor=deod/2;
deir=deid/2;
dewwr=dewwd/2;

%Wire Wrap Switch 1 = on 0 = off.
dewws=1;
fewws=1;

% Solid dummy element 1=solid 0=hollow;
dess=0;
fess=1;

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugXX10NFE; nde=DebugXX10NDE; end

%**********************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=NaN+NaN+feo;

pinparm={hdorg(1),fbPl,nfslas,UNIC,mcnpfix};
dpinparm={hdorg(1),fbPl,nfslas,UNIC,mcnpfix};

%**********************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SIHDLCm=MatMap{IHDLAmC};
SIHDLCd=MatMap{IHDLAdC};

%**********************************************************************
% Pin Materials

% Dummy Pins

% Dummy Clading material
Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Dummy Fuel Slug
Deim=MatMap{PPGmC};
Deid=MatMap{PPGdC};

% Dummy Element Wirewrap
Dewwrm=MatMap{OPmC};
Dewwrd=MatMap{OPdC};

%**********************************************************************
```

```
%Assigns dimensions and materials for the pins.
pind=[nfslas;NaN;NaN;fewt;NaN;feh;feor;fewwr;fewwh;fewws;fess];
dpind=[nfslas;NaN;NaN;dewt;deir;deh;deor;dewwr;dewwh;dewws;dess];

%Passed to the pins
pinm=[Dm,Dd;NaN,NaN;Dewwrm,Dewwrd];
dpinm=[Dm,Dd;Deim,Deid;Dewwrm,Dewwrd];

%*********************************************************************
%The following is a halfworth driver map to map where the dummy pins go.

pinmap=[2;2;2;2;2;2;2;2;2;2;2;2;2;1;2;2;2;2;2];

hwdmap=[linspace(1,nfe+nde,nfe+nde)' pinmap(1:nfe+nde)];

% Copy HDorg so that it can be used on the outer sheth
Ohdorg=[hdorg(1:3), hdorg(4)-leh];

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

Ohdsorg=ThermalXHex(hdh,Ohdorg,bend);
hdsorg=ThermalXHex(crh,hdorg,bend);

%%
%Debug only
%
% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['EXP INSAT M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%% The following for loop creates the core region sections.

for ii=1:size(hdsorg,1)

    %This for loop creates the surface cards for the hex duct sections

    % Outer Duct Assembly
     IDOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4)-ihdlch-ihdwt,0,0,crh+ueh+ihdlch+2*ihdwt,...
     ihdor, '$ Outer Wall of Inner Hex Duct Sec',num2str(ii)});

     %Assign the surface number to be used later in the cell creation
     surf1(ii)=SAsurfnum;

    %Update the surnum number
     SAsurfnum=SAsurfnum+1;

     if LAT  && ii==1

         IDOWs=[IDOWs; CharChecker({surfnum,'    SO',2*hdh,...
             '$ Surr Na For Lat'})];

         surf15=surfnum;

         surfnum=surfnum+1;

     end

    if ii==1

        % This plane caps off the outer duct.

        ODUPls=CharChecker({surfnum,'    PZ', Ohdsorg(4)+hdh,...
                        '$ Plane to cap outter hex duct'});

        %I need this facet for later.
        ODUPnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        % Outter Wall Outer Duct

         ODOWs=CharChecker({surfnum,'    RHP',Ohdsorg(ii,2),...
         Ohdsorg(ii,3),Ohdorg(4)-hdwt,0,0,hdh+2*hdwt,hdor,...
         '$ Duct Outer Tube Outer Wall'});

        %Assign the surface number to be used later in the cell creation
        ODOWnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        % Outter Wall Inner Duct
```

```
         ODIWs=CharChecker({surfnum,'    RHP',Ohdsorg(ii,2),...
         Ohdsorg(ii,3),Ohdorg(4),0,0,hdh+hdwt,hdir,...
         '$ Duct Outer Tube Inner Wall'});

        %Assign the surface number to be used later in the cell creation
        ODIWnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        %Creates the inner duct for the upper extension.
        %Also calls the Poison Pins to be made

        % Inner Duct Upper Extention

         IDSUPs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
         hdsorg(ii,3),hdorg(4)+crh,0,0,ueh,ihdir,...
         '$ Upper Ext Inner Hex'});

        %Assign the surface number to be used later in the cell creation
        IDUEnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

    end

    % Inner Duct Core Region
     IDCRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4),0,0,crh,ihdir,...
     '$ Inner Wall Inner Hex Duct Sec',num2str(ii)});

     %Assign the surface number to be used later in the cell creation
     surf2(ii)=surfnum;

    %Update the surnum number
     surfnum=surfnum+1;

     % Need to add a plane

     if ii==1

         surfcUP=[ surfname; ODUPls;...
             ODOWs; ODIWs; IDOWs; IDSUPs; IDCRs ];

     else

         surfcUP=[surfcUP; IDOWs; IDCRs];

     end

end


%%
% Lower Extension Surface Cards

% This plane caps off the lower tube

ODLPc=CharChecker({surfnum,'    PZ',Ohdorg(4),...
                '$ Plane to cap lower cylinder tube.'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Outter wall Cylinder Extension

ODOCyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch-hdwt,lcr,' $ Lower Extension OW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODOCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Wall Cylinder Extension

ODICyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch,lcr-hdwt,' $ Outer Duct IW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODICylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Cylinder Extension

IDCylc=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-mcnpfix,0,0,-ihdlch-hdwt,ihdlcr,' $ Inner Duct Cylinder'});
```

```
%Assign the surface number to be used later in the cell creation
IDCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Append the surfaces to the surface card.
surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc];


%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

for kk=1:size(hdsorg,1)

    %The new HD section origin must be changed for porg to work

    %Pin maker has to be adjust just like hex maker to account for the
    %slices. This part must be done in a series of if statements so that
    %the slicing does not cut half sections.

    %Porg calculates the origin of all the pins in a particular section.

      porg=PinOrg((nfe+nde),fep,hdsorg(kk,2),hdsorg(kk,3),hdorg(4)+feo);

    for ii=1:(nfe+nde)

        %Like the main program, the origins of the pins need to be determined so
        %that they can be fed into a program to write the cards. This is what the
        %pinorg program does.

        %ocws = Oyter Cladding Wall Surface
        %wws = wire wrap surface

        %Dummy Pins

        if hwdmap(ii,2)==1

            [ pincellctemp, pinsurfctemp, ocws , wws]=...
                DummyPinMaker(porg(ii,:),dpind,dpinm,dpinparm,hdsorg,kk);

        end

        %Fuel Pins

        if hwdmap(ii,2)==2

            [ pincellctemp, pinsurfctemp, ocws , wws ]=...
                DummyPinMaker(porg(ii,:),pind,pinm,pinparm,hdsorg,kk);

        end

            pinsurfnum(ii,kk)=ocws;
            wirewrapsurfnum(ii,kk)=wws;

            if (ii==1)&&(kk==1)

                DPs=pinsurfctemp;
                DPc=pincellctemp;

            end

            if (ii>1)||(kk>1)

                DPs=[DPs; pinsurfctemp];
                DPc=[DPc; pincellctemp];

            end
     end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

SAcellname=NameCard(['EXP INSAT M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');
ODcname=NameCard(['Outer Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
IDcname=NameCard(['Inner Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
UEcname=NameCard(['Upper Extension Sub Assembly #' num2str(hdorg(1))],'Divider');
LATcname=NameCard(['Infinite sodium for Lattice Sub Assembly #' num2str(hdorg(1))],'Divider');
CRcname=NameCard(['Core Region Sub Assembly #' num2str(hdorg(1))],'Divider');
```

```
%% %Hex Duct Cell cards

for ii=1:size(hdsorg,1)

    %Duct wall

    if ii==1

        if ~BorD;

            % Outter Duct Wall

            cellctemp1=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
                 surf2(ii),UEIsurfnum, planesurfnum(ii),...
                'IMP:N=1','$ Sec #',num2str(ii), 'Hex Duct Upper Ext'});

            cellnum=cellnum+1;

            cellctempup1=CharChecker([cellnum,'     ',SUPm,SUPd, -UEIsurfnum...
                , ' IMP:N=1','$ Up Ext Homg']);

            %Inside duct Core Region

            DPc={cellnum,'     ',Dnam,Dnad, -surf2(ii),planesurfnum(ii)};

            cellnum=cellnum+1;

        else

%**************************** Benchmark *********************************

            % Outter duct outter wall.

            ODc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODUPnum, -ODOWnum,...
                 ODIWnum,ODOCylnum, UNIC,'$ Outer Duct XX10 '});

            ODcnum=cellnum;

            cellnum=cellnum+1;

            % Outer Duct Lower cylinder

            ODCylc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODLPlnum,...
                -ODOCylnum,ODICylnum, UNIC,'$ Lower Cyn OD'});

            OWLCcnum=cellnum;

            cellnum=cellnum+1;

            % Wall Inner Duct

            IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
                 surf2(ii),IDUEnum,UNIC,'$ Inner Duct XX10 '});

            cellnum=cellnum+1;

            % Lower Cylinder

            IDCylc=CharChecker({cellnum,'     ',SIHDLCm,SIHDLCd, surf1(1),...
                -IDCylnum,UNIC,'$ Lower Cyn ID'});

            cellnum=cellnum+1;

            % Inside Duct Upper Ext Homog Region

            UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -IDUEnum,...
                 UNIC,'$ Up Ext Smear'});

            cellnum=cellnum+1;

            % Inside duct Core Region

            CRc={cellnum,'     ',Dnam,Dnad, -surf2(ii)};

            cellnum=cellnum+1;

             if LAT==1

                 LATNac=CharChecker({cellnum,'     ',Dnam,...
                     Dnad,['#' Num2StrM(ODcnum)],surf1(1),...
                     ['#' Num2StrM(OWLCcnum)],IDCylnum,UNIC,'$ INF Na for Lat'});

                cellnum=cellnum+1;

                ODOWnums=[{surf1}, {IDCylnum}];

             else

                 % this code adds a sign into hexsurfnum to let micka know
                 % that the following surf is actually a not cell.

                ODnums=[ ODcnum; OWLCcnum];

                ODOWnums=[{surf1}, {IDCylnum}, {ODnums}];
```

```
            end

              % Appending the Outerduct surface numbers to the hexsurfnum variable

            if isempty(hexsurfnum{1,1})

                hexsurfnum={ODOWnums};

            else

                hexsurfnum=[hexsurfnum, {ODOWnums}];

            end

%***************************** Benchmark *********************************

        end

    end

    if ii~=1&&ii~=size(hdsorg,1)

        cellctemp1=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
            surf2(ii),-planesurfnum(ii-1),planesurfnum(ii),...
            'IMP:N=1','$ Sec #',num2str(ii),'Hex Duct'});

        cellnum=cellnum+1;

        DPc={cellnum,'     ',Dnam,Dnad, -surf2(ii),...
            -planesurfnum(ii-1),planesurfnum(ii)};

        cellnum=cellnum+1;

    end

    if ii ~= 1 && ii==size(hdsorg,1)

        cellctemp1=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(ii),...
            surf2(ii),LEIsurfnum,LEPl,-planesurfnum(ii-1),...
            'IMP:N=1','$ Sec #',num2str(ii),'Hex Duct'});

        cellnum=cellnum+1;

        %Inside duct Core Region

        DPc={cellnum,'     ',Dnam,Dnad, -surf2(ii),-planesurfnum(ii-1)};

        cellnum=cellnum+1;

        %Sodium around the lower extension cylinder

        cellctemplo1=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1(ii),...
            -LEPl,LECyl,'IMP:N=1 $ Na Surr Lower Ext'});

        cellnum=cellnum+1;

        % Lower Cylinder

        cellctemplo2=CharChecker({cellnum,'     ',Ductm,Ductd, -LECyl,...
            'IMP:N=1 $ Lower Cyn Homog'});

        cellnum=cellnum+1;


    end

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm,ii)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm,ii)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc, UNIC,...
            '$ Sec #',num2str(ii), 'Sodium coolant']);

    if ii==1

        if ~BorD

        %Appends the upper extensions cards

        cellctemp=[cellctemp1; cellctempup1; CRc;];
```

```
        else
%***************************** Benchmark *********************************

        %Appends the upper lower extensions and Core Region cards

        if LAT

            cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
                LATcname; LATNac; UEcname; UEc; CRcname; CRc; DPc];

        else

            cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
                LATcname; UEcname; UEc; CRcname; CRc; DPc];

        end

%***************************** Benchmark *********************************
        end

    end

    if ii~=1 && ii~=size(hdsorg,1)

       % Appends the mid section cards together

       cellctemp=[cellctemp; cellctemp1; DPc];

    end


    if  ii~=1 && ii==size(hdsorg,1)

       %Appends the lower extension cards together

       cellctemp=[cellctemp; cellctemp1; DPc; cellctemplo;...
           cellctemplo1; cellctemplo2; ];

    end


end


%% Appending the cards together

surfc=[surfcUP; DPs];
cellc=[SAcellname; cellctemp];

%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_XX10_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.61. XX09Maker.m

```
function [ cellc,surfc ] = XX09Maker(hdorg,mcnpfix,bend, MatMap, DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugXX09NFE DebugXX09NDE
global surfnum cellnum
global hexsurfnum planesurfnum fbPl SAsurfnum createplane
global LAT
global BorD

 %% Reactor Map Vars Breakout

global SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fewwdC
global nfeC ihdhC ihdodC ihdwtC ihdlcdC ihdlchC fetphC fesphC
global fshC fsdC fslasC fehC feodC fewtC feozC fepC ndeC

%% Material Vars

global DuctmC DuctdC DnamC DnadC SUPmC SUPdC SLOmC SLOdC IHDLAdC IHDLAmC
global DmC DdC FNamC FNadC FmC FdC FCladmC FCladdC FPwwrmC FPwwdC
global FPGmC FPGdC

% This function creates the XX10 subassembly. The code has been copied from
```

```
% a half worth driver and a CONTROL rod and is modified to support the create of solid dummy
% pins and hollow ones.

% This is achieved by setting the solid pins to be the "fuel" and the
% hollow pin to be the dummy. Any time a variable with the "f" suffix is
% used, it means solid pin. "d" suffix means hollow;


%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

%************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
bh=DimMap{bhC}; crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC};
lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Inner Hex Duct
ihdh=DimMap{ihdhC}; ihdod=DimMap{ihdodC}; ihdwt=DimMap{ihdwtC}; ihdid=ihdod-(2*ihdwt);
ihdlcd=DimMap{ihdlcdC}; ihdlch=DimMap{ihdlchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC};
fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Fuel slug section heights
fssh=fsh/3;

% Dummy Elements, they are the same as fuel elements but solid ss
deod=feod; deid=feid; deh=feh; dewt=fewt; deoz=feoz; dewwd=fewwd;...
    dewwh=fewwh; nde=DimMap{ndeC};

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
lcr=lcd/2;
ihdor=ihdod/2;
ihdir=ihdid/2;
ihdlcr=ihdlcd/2;
feor=feod/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;
deor=deod/2;
deir=deid/2;
dewwr=dewwd/2;

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugXX09NFE; nde=DebugXX09NDE; end

% Add the pin numbers together

np=nfe+nde;

%Wire Wrap Switch 1 = on 0 = off.
dewws=1;

% Solid dummy element 1=solid 0=hollow;
dess=0;

%************************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=fslas+fsh+feoz;

if createplane

    [ fbPl, surfPlane ] = PinPlaneMaker( nfslas, fslas, fssh );
```

```
end

pinparm={hdorg(1),fbPl,nfslas,UNIC,mcnpfix};
dpinparm={hdorg(1),fbPl,nfslas,UNIC,mcnpfix};

%**********************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SIHDLCm=MatMap{IHDLAmC};
SIHDLCd=MatMap{IHDLAdC};
%**********************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap

FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

% Dummy Pins

% Dummy Fuel Slug
Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Dummy Fuel Slug
Deim=Dm;
Deid=Dd;

% Wire Wrap
Dewwrm=FWwrm;
Dewwrd=FWwrd;

%**********************************************************************
%Assigns dimensions and materials for the pins.
pind=[nfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
dpind=[nfslas;NaN;NaN;dewt;deir;deh;deor;dewwr;dewwh;dewws;dess];

%Passed to the pins
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];
dpinm=[Dm,Dd;Deim,Deid;Dewwrm,Dewwrd];

%**********************************************************************
%The following is a halfworth driver map to map where the dummy pins go.

pinmap=[2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;...
    2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;1;1;2;2;2;2;2;2;2;2;2;2;];

hwdmap=[linspace(1,np,np)' pinmap(1:np)];


% Copy HDorg so that it can be used on the outer sheth
Ohdorg=[hdorg(1:3), hdorg(4)-leh];
```

```
%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

Ohdsorg=ThermalXHex(hdh,Ohdorg,bend);
hdsorg=ThermalXHex(crh,hdorg,bend);

%%
%Debug only
%
% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['EXP INSAT M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%% The following for loop creates the core region sections.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
IDOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4)-ihdlch-ihdwt,0,0,crh+ueh+ihdlch+2*ihdwt,...
ihdor, '$ Outer Wall of Inner Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf1=SAsurfnum;

%Update the surnum number
SAsurfnum=SAsurfnum+1;

if LAT==1

    IDOWs=[IDOWs; CharChecker({surfnum,'    SO',2*hdh,'$ Surr Na For Lat'})];

   surf15=surfnum;

   surfnum=surfnum+1;

end

% This plane caps off the outer duct.

ODUPls=CharChecker({surfnum,'    PZ', Ohdsorg(4)+hdh,...
                '$ Plane to cap outter hex duct'});

%I need this facet for later.
ODUPnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Outter Wall Outer Duct

 ODOWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
 Ohdsorg(3),Ohdorg(4)-hdwt,0,0,hdh+2*hdwt,hdor,...
 '$ Duct Outer Tube Outer Wall'});

%Assign the surface number to be used later in the cell creation
ODOWnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Outter Wall Inner Duct

 ODIWs=CharChecker({surfnum,'    RHP',Ohdsorg(2),...
 Ohdsorg(3),Ohdorg(4),0,0,hdh+hdwt,hdir,...
 '$ Duct Outer Tube Inner Wall'});

%Assign the surface number to be used later in the cell creation
ODIWnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

%Creates the inner duct for the upper extension.
%Also calls the Poison Pins to be made

% Inner Duct Upper Extention

 IDSUPs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)+crh,0,0,ueh,ihdir,...
 '$ Upper Ext Inner Hex'});

%Assign the surface number to be used later in the cell creation
IDUEnum=surfnum;

%Update the surnum number
```

```
 surfnum=surfnum+1;

% Inner Duct Core Region
IDCRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
hdsorg(3),hdorg(4),0,0,crh,ihdir,'$ Inner Wall Inner Hex Duct'});

%Assign the surface number to be used later in the cell creation
surf2=surfnum;

%Update the surnum number
surfnum=surfnum+1;


if createplane

    IDCRs=[IDCRs; surfPlane];

end

surfcUP=[ surfname; ODUPls;...
    ODOWs; ODIWs; IDOWs; IDSUPs; IDCRs ];

%%
% Lower Extension Surface Cards

% This plane caps off the lower tube

ODLPc=CharChecker({surfnum,'    PZ',Ohdorg(4),...
                '$ Plane to cap lower cylinder tube.'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Outter wall Cylinder Extension

ODOCyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch-hdwt,lcr,' $ Lower Extension OW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODOCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Wall Cylinder Extension

ODICyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch,lcr-hdwt,' $ Outer Duct IW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODICylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Cylinder Extension

IDCylc=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-mcnpfix,0,0,-ihdlch-hdwt,ihdlcr,' $ Inner Duct Cylinder'});

%Assign the surface number to be used later in the cell creation
IDCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Append the surfaces to the surface card.
surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc];

%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros((np),size(hdsorg,1));
wirewrapsurfnum=zeros((np),size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.

porg=PinOrg(np,fep,hdsorg(2),hdsorg(3),hdorg(4)+feoz);
```

```
for ii=1:(np)

    %Like the main program, the origins of the pins need to be determined so
    %that they can be fed into a program to write the cards. This is what the
    %pinorg program does.

    %ocws = Oyter Cladding Wall Surface
    %wws = wire wrap surface

    %Dummy Pins

    if hwdmap(ii,2)==1

        [ pincellctemp, pinsurfctemp, ocws , wws]=...
            DummyPinMaker(porg(ii,:),dpind,dpinm,dpinparm);

    end

    %Fuel Pins

    if hwdmap(ii,2)==2

        [ pincellctemp, pinsurfctemp, ocws , wws ]=...
            PinMaker(porg(ii,:),pind,pinm,pinparm);

    end

        pinsurfnum(ii)=ocws;
        wirewrapsurfnum(ii)=wws;

        if (ii==1)

            EPs=pinsurfctemp;
            EPc=pincellctemp;

        end

        if (ii>1)

            EPs=[EPs; pinsurfctemp];
            EPc=[EPc; pincellctemp];

        end
end


%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

SAcellname=NameCard(['EXP INSAT M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');
ODcname=NameCard(['Outer Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
IDcname=NameCard(['Inner Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
UEcname=NameCard(['Upper Extension Sub Assembly #' num2str(hdorg(1))],'Divider');
LATcname=NameCard(['Inifinite sodium for Lattice Sub Assembly #' num2str(hdorg(1))],'Divider');
CRcname=NameCard(['Core Region Sub Assembly #' num2str(hdorg(1))],'Divider');

%% %Hex Duct Cell cards

%***************************** Benchmark *********************************

% Outter duct outter wall.

ODc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODUPnum, -ODOWnum,...
     ODIWnum,ODOCylnum, UNIC,'$ Outer Duct XX09 '});

ODcnum=cellnum;

cellnum=cellnum+1;

% Outer Duct Lower cylinder

ODCylc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODLPlnum,...
    -ODOCylnum,ODICylnum, UNIC,'$ Lower Cyn OD'});

OWLCcnum=cellnum;

cellnum=cellnum+1;

% Wall Inner Duct

IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
     surf2,IDUEnum,UNIC,'$ Inner Duct XX09 '});

cellnum=cellnum+1;

% Lower Cylinder

IDCylc=CharChecker({cellnum,'     ',SIHDLCm,SIHDLCd, surf1,...
    -IDCylnum,UNIC,'$ Lower Cyn ID'});

cellnum=cellnum+1;
```

```
% Inside Duct Upper Ext Homog Region

UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -IDUEnum,...
     UNIC,'$ Up Ext Smear'});

cellnum=cellnum+1;

% Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

 if LAT==1

     LATNac=CharChecker({cellnum,'     ',Dnam,...
         Dnad,['#' Num2StrM(ODcnum)],surf1,...
         ['#' Num2StrM(OWLCcnum)],IDCylnum,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

    ODOWnums=[{surf1}, {IDCylnum}];

 else

     % this code adds a sign into hexsurfnum to let micka know
     % that the following surf is actually a not cell.

    ODnums=[ ODcnum; OWLCcnum];

    ODOWnums=[{surf1}, {IDCylnum}, {ODnums}];

 end

  % Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

%***************************** Benchmark *********************************

%Adds the pin surface numbers to the inside duct definition

for mm=1:size(pinsurfnum,1)

    CRc=[CRc pinsurfnum(mm)];

end

%Adds the wirewrap surface numbers to the inside duct definition.

for mm=1:size(wirewrapsurfnum,1)

    CRc=[CRc wirewrapsurfnum(mm)];

end

%Taks on the importance and comment

CRc=CharChecker([CRc, UNIC,'$ Sodium coolant']);

%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LAT

    cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
        LATcname; LATNac; UEcname; UEc; CRcname; CRc; EPc];

else

    cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
        LATcname; UEcname; UEc; CRcname; CRc; EPc];


end

%***************************** Benchmark *********************************


%% Appending the cards together

surfc=[surfcUP; EPs];
cellc=[SAcellname; cellctemp];

%%
global SaveSAMakerVars
```

```
if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_XX09_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.62. XY16Maker.m

```
function [ cellc,surfc ] = XY16Maker(hdorg,mcnpfix,bend, MatMap, DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugXY16NDE
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum fbPl
global LAT
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fewwdC
global ihdhC ihdodC ihdwtC ihdlcdC ihdlchC sazoC saatC sazmC
global ndeC deodC dewtC dehC deoC depC

%% Material Vars

global DmC DdC PPGmC PPGdC OPmC OPdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC IHDLAmC IHDLAdC

% This function creates the XX10 subassembly. The code has been copied from
% a half worth driver and a CONTROL rod and is modified to support the create of solid dummy
% pins and hollow ones.

% This is achieved by setting the solid pins to be the "fuel" and the
% hollow pin to be the dummy. Any time a variable with the "f" suffix is
% used, it means solid pin. "d" suffix means hollow;


%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%************************************************************************
%Sets the universe num

if LAT==1

    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

%************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt); bh=DimMap{bhC};
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Inner Hex Duct
ihdh=DimMap{ihdhC}; ihdod=DimMap{ihdodC}; ihdwt=DimMap{ihdwtC}; ihdid=ihdod-(2*ihdwt);
ihdlcd=DimMap{ihdlcdC}; ihdlch=DimMap{ihdlchC};

% Dummy Elements, these are the hollow ones!
deod=DimMap{deodC}; dewt=DimMap{dewtC}; deid=deod-(2*dewt); deh=DimMap{dehC};
deo=DimMap{deoC}; dewwd=DimMap{fewwdC}; dewwh=deh; dep=DimMap{depC}; nde=DimMap{ndeC};;

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
ihdor=ihdod/2;
ihdir=ihdid/2;
ihdlcr=ihdlcd/2;
lcr=lcd/2;
```

```
deor=deod/2;
deir=deid/2;
dewwr=dewwd/2;

%Wire Wrap Switch 1 = on 0 = off.
dewws=1;

% Solid dummy element 1=solid 0=hollow;
dess=1;

% Override pin count if debug micka is true
if debugMICKA; nde=DebugXY16NDE; end

%**************************************************************************
%Subassembly movement

%Since the control rod in the full insertion position is 8.255cm from the
%origin of the core 0,0,0 we need to lower the origin of the assembly by
%that much. hdorg(4) is the z reference. This should not be changeD!
%s.a.z.o (sub assembly z offset)

sazo=DimMap{sazoC};

%Total allowable travel, saat sub assembly allowbale travel

saat=DimMap{saatC};

%This following variable is the induced movement due to drive mechanism.
%Movement is cm of insertion. This means full insertion = 35.56. Full out =
%0 cm. sazm, sub assembly movement

sazm=DimMap{sazmC};

%**************************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=NaN+NaN+deo;

dpinparm={hdorg(1),fbPl,nfslas,UNIC,mcnpfix};

%**************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SIHDLCm=MatMap{IHDLAmC};
SIHDLCd=MatMap{IHDLAdC};

%**************************************************************************
% Pin Materials

% Dummy Pins

% Dummy Clading material
Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Dummy Fuel Slug
Deim=MatMap{PPGmC};
Deid=MatMap{PPGdC};

% Dummy Element Wirewrap
Dewwrm=MatMap{OPmC};
Dewwrd=MatMap{OPdC};

%**************************************************************************
%Assigns dimensions and materials for the pins.

dpind=[nfslas;NaN;NaN;dewt;deir;deh;deor;dewwr;dewwh;dewws;dess];
dpinm=[Dm,Dd;Deim,Deid;Dewwrm,Dewwrd];


%%
```

```
%************************************************************************

%This section is the control rod movement code. This will enable us to move
%the control subassembly up and down. hdorg(4) refers to the 0" z height.
%The rest of the assembly is built from this reference position including
%the KAtana effect.

% Copy HDorg so that it can be used on the outer sheth
Ohdorg=[hdorg(1:3), hdorg(4)-leh];

%Offest plus movement
hdorg(4)=hdorg(4)-sazo-saat+sazm;

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

Ohdsorg=ThermalXHex(hdh,Ohdorg,bend);
hdsorg=ThermalXHex(crh,hdorg,bend);


%%
%Debug only
%
% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['EXP Control Dummy M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%% The following for loop creates the core region sections.

for ii=1:size(hdsorg,1)

    %This for loop creates the surface cards for the hex duct sections

    % Outer Duct Assembly
     IDOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4)-ihdwt,0,0,crh+ueh+2*ihdwt,...
     ihdor, '$ Outer Wall of Inner Hex Duct Sec',num2str(ii)});

     %Assign the surface number to be used later in the cell creation
     surf1(ii)=SAsurfnum;

    %Update the surnum number
     SAsurfnum=SAsurfnum+1;

    if LAT==1 && ii==1

         IDOWs=[IDOWs; CharChecker({surfnum,'    SO',2*hdh,...
             '$ Surr Na For Lat'})];

         surf15=surfnum;

         surfnum=surfnum+1;

    end

    if ii==1

        % This plane caps off the outer duct.

        ODUPls=CharChecker({surfnum,'    PZ', Ohdsorg(4)+hdh,...
                        '$ Plane to cap outter hex duct'});

        %I need this facet for later.
        ODUPnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        % Outter Wall Outer Duct

         ODOWs=CharChecker({surfnum,'    RHP',Ohdsorg(ii,2),...
         Ohdsorg(ii,3),Ohdorg(4)-hdwt,0,0,hdh+2*hdwt,hdor,...
         '$ Duct Outer Tube Outer Wall'});

        %Assign the surface number to be used later in the cell creation
        ODOWnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        % Outter Wall Inner Duct

         ODIWs=CharChecker({surfnum,'    RHP',Ohdsorg(ii,2),...
         Ohdsorg(ii,3),Ohdorg(4),0,0,hdh+hdwt,hdir,...
         '$ Duct Outer Tube Inner Wall'});
```

```
        %Assign the surface number to be used later in the cell creation
        ODIWnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        %Creates the inner duct for the upper extension.
        %Also calls the Poison Pins to be made

        % Inner Duct Upper Extention

         IDSUPs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
         hdsorg(ii,3),hdorg(4)+crh,0,0,ueh,ihdir,...
         '$ Upper Ext Inner Hex'});

        %Assign the surface number to be used later in the cell creation
        IDUEnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

    end

    % Inner Duct Core Region
     IDCRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4),0,0,crh,ihdir,...
     '$ Inner Wall Inner Hex Duct Sec',num2str(ii)});

     %Assign the surface number to be used later in the cell creation
     surf2(ii)=surfnum;

    %Update the surnum number
     surfnum=surfnum+1;

     if ii==1

         surfcUP=[ surfname; ODUPls;...
             ODOWs; ODIWs; IDOWs; IDSUPs; IDCRs ];

     else

         surfcUP=[surfcUP; IDOWs; IDCRs];

     end

end


%%
% Lower Extension Surface Cards

% This plane caps off the lower tube

ODLPc=CharChecker({surfnum,'    PZ',Ohdorg(4),...
               '$ Plane to cap lower cylinder tube.'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Outter wall Cylinder Extension

ODOCyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch-hdwt,lcr,' $ Lower Extension OW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODOCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Wall Cylinder Extension

ODICyls=CharChecker({surfnum,'    RCC',Ohdsorg(2),Ohdsorg(3),...
    Ohdorg(4)+mcnpfix,0,0,-lch,lcr-hdwt,' $ Outer Duct IW Cylinder'});

%Assign the surface number to be used later in the cell creation
ODICylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Inner Cylinder Extension

IDCylc=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-mcnpfix,0,0,-ihdlch-hdwt,ihdlcr,' $ Inner Duct Cylinder'});

%Assign the surface number to be used later in the cell creation
IDCylnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;
```

```
% Append the surfaces to the surface card.
surfcUP=[surfcUP;  ODLPc; ODOCyls; ODICyls; IDCylc];


%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nde,size(hdsorg,1));
wirewrapsurfnum=zeros(nde,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

for kk=1:size(hdsorg,1)

    %The new HD section origin must be changed for porg to work

    %Pin maker has to be adjust just like hex maker to account for the
    %slices. This part must be done in a series of if statements so that
    %the slicing does not cut half sections.

    %Porg calculates the origin of all the pins in a particular section.

      porg=PinOrg(nde,dep,hdsorg(kk,2),hdsorg(kk,3),hdorg(4)+deo);

    for mm=1:nde

        %Like the main program, the origins of the pins need to be determined so
        %that they can be fed into a program to write the cards. This is what the
        %pinorg program does.

        %ocws = Oyter Cladding Wall Surface
        %wws = wire wrap surface

        %Dummy Pins

        [ pincellctemp, pinsurfctemp, ocws , wws]=...
            DummyPinMaker(porg(mm,:),dpind,dpinm,dpinparm,hdsorg,kk);

        pinsurfnum(mm,kk)=ocws;
        wirewrapsurfnum(mm,kk)=wws;

        if (mm==1)&&(kk==1)

            DPs=pinsurfctemp;
            DPc=pincellctemp;

        end

        if (mm>1)||(kk>1)

            DPs=[DPs; pinsurfctemp];
            DPc=[DPc; pincellctemp];

        end
    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

SAcellname=NameCard(['EXP Dummy Control M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');
ODcname=NameCard(['Outer Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
IDcname=NameCard(['Inner Duct Sub Assembly #' num2str(hdorg(1))],'Divider');
UEcname=NameCard(['Upper Extension Sub Assembly #' num2str(hdorg(1))],'Divider');
LATcname=NameCard(['Inf Na for LAT Sub Assembly #' num2str(hdorg(1))],'Divider');
CRcname=NameCard(['Core Region Sub Assembly #' num2str(hdorg(1))],'Divider');

%% %Hex Duct Cell cards

for mm=1:size(hdsorg,1)

    %Duct wall

    if mm==1

        if ~BorD;

            % Outter Duct Wall

            IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(mm),...
                 surf2(mm),IDUEnum, planesurfnum(mm),...
                'IMP:N=1','$ Sec #',num2str(mm), 'Hex Duct Upper Ext'});

            cellnum=cellnum+1;
```

```
            UEc=CharChecker([cellnum,'     ',SUPm,SUPd, -IDUEnum...
                , ' IMP:N=1','$ Up Ext Homg']);

            %Inside duct Core Region

            CRc={cellnum,'     ',Dnam,Dnad, -surf2(mm),planesurfnum(mm)};

            cellnum=cellnum+1;

        else

%***************************** Benchmark *********************************

            % Outter duct outter wall.

            ODc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODUPnum, -ODOWnum,...
                 ODIWnum,ODOCylnum, UNIC,'$ Outer Duct CONTROL '});

            ODcnum=cellnum;

            cellnum=cellnum+1;

            % Outer Duct Lower cylinder

            ODCylc=CharChecker({cellnum,'     ',Ductm,Ductd,-ODLPlnum,...
                -ODOCylnum,ODICylnum, UNIC,'$ Lower Cyn OD'});

            OWLCcnum=cellnum;

            cellnum=cellnum+1;

            % Wall Inner Duct

            IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(mm),...
                 surf2(mm),IDUEnum,UNIC,'$ Inner Duct CONTROL '});

            cellnum=cellnum+1;

            % Lower Cylinder

            IDCylc=CharChecker({cellnum,'     ',SIHDLCm,SIHDLCd, surf1(1),...
                -IDCylnum,UNIC,'$ Lower Cyn ID'});

            cellnum=cellnum+1;

            % Inside Duct Upper Ext Homog Region

            UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -IDUEnum,...
                 UNIC,'$ Up Ext Smear'});

            cellnum=cellnum+1;

            % Inside duct Core Region

            CRc={cellnum,'     ',Dnam,Dnad, -surf2(mm)};

            cellnum=cellnum+1;

             if LAT==1

                 LATNac=CharChecker({cellnum,'     ',Dnam,...
                     Dnad,['#' Num2StrM(ODcnum)],surf1(1),...
                     ['#' Num2StrM(OWLCcnum)],IDCylnum,UNIC,'$ INF Na for Lat'});

                cellnum=cellnum+1;

                ODOWnums=[{surf1}, {IDCylnum}];

             else

                 % this code adds a sign into hexsurfnum to let micka know
                 % that the following surf is actually a not cell.

                ODnums=[ ODcnum; OWLCcnum];

                ODOWnums=[{surf1}, {IDCylnum}, {ODnums}];

             end

              % Appending the Outerduct surface numbers to the hexsurfnum variable

            if isempty(hexsurfnum{1,1})

                hexsurfnum={ODOWnums};

            else

                hexsurfnum=[hexsurfnum, {ODOWnums}];

            end

%***************************** Benchmark *********************************

        end
```

```
    end

    if mm~=1&&mm~=size(hdsorg,1)

        IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(mm),...
            surf2(mm),-planesurfnum(mm-1),planesurfnum(mm),...
            'IMP:N=1','$ Sec #',num2str(mm),'Hex Duct'});

        cellnum=cellnum+1;

        CRc={cellnum,'     ',Dnam,Dnad, -surf2(mm),...
            -planesurfnum(mm-1),planesurfnum(mm)};

        cellnum=cellnum+1;

    end

    if mm ~= 1 && mm==size(hdsorg,1)

        IDc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1(mm),...
            surf2(mm),LEIsurfnum,LEPl,-planesurfnum(mm-1),...
            'IMP:N=1','$ Sec #',num2str(mm),'Hex Duct'});

        cellnum=cellnum+1;

        %Inside duct Core Region

        CRc={cellnum,'     ',Dnam,Dnad, -surf2(mm),-planesurfnum(mm-1)};

        cellnum=cellnum+1;

        %Sodium around the lower extension cylinder

        cellctemplo1=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1(mm),...
            -LEPl,LECyl,'IMP:N=1 $ Na Surr Lower Ext'});

        cellnum=cellnum+1;

        % Lower Cylinder

        IDCylc=CharChecker({cellnum,'     ',Ductm,Ductd, -LECyl,...
            'IMP:N=1 $ Lower Cyn Homog'});

        cellnum=cellnum+1;


    end

    %Adds the pin surface numbers to the inside duct definition

    for kk=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(kk,mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for kk=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(kk,mm)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc, UNIC,...
            '$ Sec #',num2str(mm), 'Sodium coolant']);

    if mm==1

        if ~BorD

        %Appends the upper extensions cards

        cellctemp=[IDc; UEc; CRc;];

        else
%***************************** Benchmark *********************************

        %Appends the upper lower extensions and Core Region cards

        if LAT

            cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
                LATcname; LATNac; UEcname; UEc; CRcname; CRc; DPc];

        else

            cellctemp=[ODcname; ODc; ODCylc; IDcname; IDc; IDCylc;...
              LATcname; UEcname; UEc; CRcname; CRc; DPc];

        end
%***************************** Benchmark *********************************
        end
```

```
    end

    if mm~=1 && mm~=size(hdsorg,1)

       % Appends the mid section cards together

       cellctemp=[cellctemp; IDc; CRc];

    end


    if  mm~=1 && mm==size(hdsorg,1)

       %Appends the lower extension cards together

       cellctemp=[cellctemp; IDc; CRc; cellctemplo;...
           cellctemplo1; IDCylc; ];

    end


end


%% Appending the cards together

surfc=[surfcUP; DPs];
cellc=[SAcellname; cellctemp];

%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_XY16_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end
end
```

B.63. X402AMaker.m

```
function [ cellc,surfc ] = X402AMaker(hdorg,mcnpfix,bend, MatMap, DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugX402ANFE DebugX402ANDE
global surfnum cellnum
global hexsurfnum planesurfnum fbPl SAsurfnum createplane
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC fewwdC nfeC fepC ndeC fetphC fesphC
global feoxC

%% Mat Vars Breakout

global FmC FdC FNamC FNadC FPGmC FPGdC FCladmC FCladdC FPwwrmC FPwwdC
global DmC DdC DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC  LAmC LadC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt); bh=DimMap{bhC};
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC};
fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
feox=DimMap{feoxC}; fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Fuel slug section heights
fssh=fsh/3;
```

```
% Dummy Elements, they are the same as fuel elements but solid ss
deod=feod; deid=feid; deh=feh; dewt=fewt; deoz=feoz; dewwd=fewwd;...
    dewwh=fewwh; nde=DimMap{ndeC};

%Convert The diameters in radi

hdor=hdod/2;
hdir=hdid/2;
feor=feod/2;
lcr=lcd/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;
deor=deod/2;
deir=deid/2;
dewwr=dewwd/2;

% Set the number of lattice elements if applicable
if LATp; nle=nfe+nde; end

% Override pin count if debug micka is true
if debugMICKA;  nfe=DebugX402ANFE ; nde=DebugX402ANDE; end

%Wire Wrap Switch 1 = on 0 = off.
dewws=1;

% Solid dummy element 1=solid 0=hollow;
dess=1;

%*************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LUN=hdorg(1)*1000+1;

    LUNu=strjoin({'U=' Num2StrM(LUN)},'');
    LUNf=strjoin({'fill=' Num2StrM(LUN)},'');

    LUNfp=LUN+1;
    LUNdp=LUNfp+1;

    % Since this is a driver, the pins are all the same universe number.
    % Latnums=repmat(LUNfp,1,nfe);
    % The previous line is commented out because a HFW driver has its own pin
    % map that it needs to follow.

    % Debugging line to pad the universemap with zeros.
    if debugMICKA; LatnumsT=...
            [repmat(LUNdp,1,nde), repmat(LUNfp,1,nfe), zeros(1,nle-nfe-nde)]; end

    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatElezo=-feh/2;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice array size
```

```
    LATr=hdir-(hdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;

    LATzo=mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins
    nde=1;

    % The universe assigned to the pins also needs to be reset.

    UNICfp=strjoin({' U=' Num2StrM(LUNfp) ' IMP:N=1' },'');
    UNICdp=strjoin({' U=' Num2StrM(LUNdp) ' IMP:N=1' },'');

else

    UNICfp=UNIC;
    UNICdp=UNIC;

end

%**********************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=fslas+fsh+feoz;

if createplane

    [ fbPl, surfPlane ] = PinPlaneMaker( nfslas, fslas, fssh );

end

pinparm={hdorg(1),fbPl,nfslas,UNICfp,mcnpfix};
dpinparm={hdorg(1),fbPl,nfslas,UNICdp,mcnpfix};

%**********************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%**********************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap

FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas
```

```
FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

% Dummy Pins

% Dummy Fuel Slug
Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Dummy Fuel Slug
Deim=Dm;
Deid=Dd;

% Wire Wrap
Dewwrm=FWwrm;
Dewwrd=FWwrd;

%**********************************************************************
%Assigns dimensions and materials for the pins.
pind=[nfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
dpind=[nfslas;NaN;NaN;dewt;deir;deh;deor;dewwr;dewwh;dewws;dess];

%Passed to the pins
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];
dpinm=[Dm,Dd;Deim,Deid;Dewwrm,Dewwrd];

%**********************************************************************
%The following is a halfworth driver map to map where the dummy pins go.

pinmap=[1;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2];

hwdmap=[linspace(1,nfe+nde,nfe+nde)' pinmap(1:nfe+nde)];


% Changes the pin map and hwdmap to a lattice.

if LATp;

    hwdmap=[1,1;1,2];
    Latnums=pinmap';

    % Create the lattice map from the pinmap by replacing 1 and 2's with
    % universe numbers.

    Latnums(Latnums==1)=LUNdp;
    Latnums(Latnums==2)=LUNfp;

end

% Debug
if debugMICKA && LATp; Latnums=LatnumsT; end

%%
%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);


%%
%Debug only
%
% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['EXP Half Worth Driver MKIIA M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%%
% The following for loop creates the core region sections.

%This for loop creates the surface cards for the hex duct sections

% Outer Duct Assembly
 ODOWs=CharChecker({SAsurfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)-leh-lch-hdwt,0,0,crh+ueh+leh+lch+2*hdwt,...
 hdor, '$ Outer Wall of Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf1=SAsurfnum;
```

```
%Update the surnum number
 SAsurfnum=SAsurfnum+1;

if (LAT || LATp)

    LATNas=CharChecker({surfnum,'    SO',2*hdh,...
        '$ Surr Na For Lat'});

    surf15=surfnum;

    surfnum=surfnum+1;

end

%Creates the inner duct for the upper extension.

% Inner Duct Upper Extention

 UEs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4)+crh,0,0,ueh,hdir,...
 '$ Upper Extent Inner Hex'});

%Assign the surface number to be used later in the cell creation
UEInum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Inner Duct Core Region
 CRs=CharChecker({surfnum,'    RHP',hdsorg(2),...
 hdsorg(3),hdorg(4),0,0,crh,hdir,...
 '$ Inner Wall Hex Duct'});

 %Assign the surface number to be used later in the cell creation
 surf2=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

 if createplane

     CRs=[CRs; surfPlane];

 end

 if (LAT || LATp); surfctemp=[ODOWs; LATNas; UEs; CRs ];
 else    surfctemp=[ODOWs; UEs; CRs ]; end

%%
% Lower Extension Surface Cards

% Inner Duct
LEs=CharChecker({surfnum,'    RHP',hdorg(2),...
    hdorg(3),hdorg(4),0,0,-leh,hdir,' $ Lower Extent Inner Hex'});

%Assign the surface number to be used later in the cell creation
LEInum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

ODLPls=CharChecker({surfnum,'    PZ',-leh-hdwt,...
                '$ Plane Separation Duct to Cylinder'});

%I need this facet for later.
ODLPlnum=surfnum;

%Update the surnum number
 surfnum=surfnum+1;

% Lower Cylinder Extension

ODCyls=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-leh-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

%Assign the surface number to be used later in the cell creation
ODCylnum=surfnum;

%Update the surnum number
surfnum=surfnum+1;

% Lattice surfaces.

if LATp

    %Naming card
    LATname=NameCard('Pin Lattice Cards for Elements','Divider');

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LATzo ...
        ,0,0,LATh,LATr,'$ Element Lattice Bounding Surface'});

    LATpArnum=surfnum;
```

```
    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdorg(2),hdorg(3),hdorg(4)+LatElezo,0,0,LatEleh,0,LatEler,...
        0,'$ Lattice Window'});

    LatWinElenum=surfnum;

    surfnum=surfnum+1;

    %Append to the surface array

    LATsurfc=[LATname; LATArs; LatWinEle];

end

% Append the surfaces to the surface card.
surfctemp=[surfctemp; LEs;  ODLPls;  ODCyls];

%%
if isempty(hexsurfnum{1,1})

    hexsurfnum={surf1};

else

    hexsurfnum=[hexsurfnum, {surf1}];

end

%% Pin Creator

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe+nde,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe+nde,size(hdsorg,1));

%Call the Pinmaker function to create the surface and cell cards for the
%individual pins.
%The for loop repeatadly calls the pinmaker function to create the cell and
%surface cards for the individual pins.

%The new HD section origin must be changed for porg to work

%Pin maker has to be adjust just like hex maker to account for the
%slices. This part must be done in a series of if statements so that
%the slicing does not cut half sections.

%Porg calculates the origin of all the pins in a particular section.

porg=PinOrg((nfe+nde),fep,hdsorg(2),hdsorg(3),hdorg(4)+deoz);

% The pin origin needs to be moved slightly
if LATp; porg(:,2)=hdsorg(2)-LATpino; porg(:,3)=hdsorg(3); end

for ii=1:(nfe+nde)

    %Like the main program, the origins of the pins need to be determined so
    %that they can be fed into a program to write the cards. This is what the
    %pinorg program does.

    %ocws = Oyter Cladding Wall Surface
    %wws = wire wrap surface

    %Dummy Pins

    if hwdmap(ii,2)==1

        [ pincellctemp, pinsurfctemp, ocws , wws]=...
            DummyPinMaker(porg(ii,:),dpind,dpinm,dpinparm);

    end

    %Fuel Pins

    if hwdmap(ii,2)==2

        [ pincellctemp, pinsurfctemp, ocws , wws ]=...
            PinMaker(porg(ii,:),pind,pinm,pinparm);

    end

    pinsurfnum(ii)=ocws;
    wirewrapsurfnum(ii)=wws;

    if (ii==1)

        FDPs=pinsurfctemp;
        FDPc=pincellctemp;

    end

    if (ii>1)
```

```
        FDPs=[FDPs; pinsurfctemp];
        FDPc=[FDPc; pincellctemp];

    end
end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['EXP Half Worth Driver M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%% %Hex Duct Cell cards

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Half Worth Driver #' num2str(hdorg(1))],'Divider');

%***************************** Benchmark *********************************

% Outter Duct Wall

ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -surf1,...
 surf2,UEInum,LEInum,ODLPlnum,...
UNIC,'$ Hex Duct Upper Ext'});

cellnum=cellnum+1;


% Inside Duct Upper Extent Homog

UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -UEInum...
UNIC ,'$ Homog Upper Ext'});

cellnum=cellnum+1;

%Inside duct Core Region

CRc={cellnum,'     ',Dnam,Dnad, -surf2};

cellnum=cellnum+1;

%Inside duct Lower Extension Homog

LEc=CharChecker({cellnum,'     ',SLOm,SLOd, -LEInum...
UNIC ,'$ Homog Lower Ext'});

cellnum=cellnum+1;

%Sodium around the lower extension cylinder

CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -surf1,...
-ODLPlnum,ODCylnum,UNIC ,'$ Na Surr Lower Ext'});

cellnum=cellnum+1;

% Lower Cylinder

Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
UNIC ,'$ Lower Cyn Homog'});

cellnum=cellnum+1;

if LAT==1

    LATNac=CharChecker({cellnum,'     ',Dnam,...
        Dnad,surf1(1),-surf15,UNIC,'$ INF Na for Lat'});

    cellnum=cellnum+1;

end

if LATp

    %name the area

    LATpcellname=NameCard('Pin Lattice Cards','Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum,'Lat=2',LUNu,...
        'IMP:N=1 $ Element Lattice'});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall
```

```
    [LATparln,LATar]=MCNPPOSMaker( LUN, nle ,nar, Latnums );

    LATpcLns=CharChecker({'     ' num2str(LATparln)});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum,LUNf,...
        UNIC,' $ Pin Lattice'});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

end

%***************************** Benchmark *********************************

if LATp

    PNaname=NameCard('Surrounding sodium Cell for Pins','Divider');

    % This defines the surrounding sodium around the dummy pin.

    DPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(1),...
        wirewrapsurfnum(1),UNICdp, '$ Fuel Dummy Pin Cell'});

    cellnum=cellnum+1;

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-surf15,pinsurfnum(2),...
        wirewrapsurfnum(2),UNICfp, '$ Fuel Dummy Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnum,UNIC,...
            '$ Sodium coolant']);


else

    %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        CRc=[CRc pinsurfnum(mm)];

    end

    %Adds the wirewrap surface numbers to the inside duct definition.

    for mm=1:size(wirewrapsurfnum,1)

        CRc=[CRc wirewrapsurfnum(mm)];

    end

    %Taks on the importance and comment

        CRc=CharChecker([CRc,UNIC,...
            '$ Sodium coolant']);

end


%***************************** Benchmark *********************************

%Appends the upper lower extensions and Core Region cards

if LATp

    if LAT

        cellctemp=[ODc; LATNac; UEc; CRc; FDPc; PNaname; DPNac;...
            FPNac; LATpc; LEc; CylNac; Cylc];

    else

        cellctemp=[ODc; UEc; CRc; FDPc; PNaname; DPNac;...
            FPNac; LATpc; LEc; CylNac; Cylc];

    end

else

    if LAT

        cellctemp=[ODc; LATNac; UEc; CRc; FDPc; LEc;...
            CylNac; Cylc];

    else
```

```
        cellctemp=[ODc; UEc; CRc; FDPc; LEc;...
            CylNac; Cylc];

    end
end

%***************************** Benchmark *********************************


%% Appending the cards together

if LATp

    surfc=[surfname; surfctemp; FDPs; LATsurfc];
    cellc=[cellname; cellname2; cellctemp ];

else

    surfc=[surfname; surfctemp; FDPs];
    cellc=[cellname; cellname2; cellctemp ];

end
%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_X402A_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.64. DriverMakerMKIIBend.m

```
function [ cellc,surfc ] = DriverMakerMKIIBend(hdorg,mcnpfix,bend,MatMap,DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugDrNFE KatanaPls
global surfnum cellnum
global hexsurfnum planesurfnum fbPl SAsurfnum createplane
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC feoxC fewwdC nfeC fepC fetphC fesphC

%% Material Vars

global FmC FdC FNamC FNadC FPGmC FPGdC FCladmC FCladdC FPwwrmC FPwwdC
global DuctmC DuctdC DnamC DnadC SUPmC SUPdC SLOmC SLOdC LAmC LadC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

SATy=DimMap{MSATyC};

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt); bh=DimMap{bhC};
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC}+mcnpfix;
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};

% Fuel slug section heights
fssh=fsh/3;

% A factor needs to be instroduced to reduce the overall diameter of the
% duct such that it can handle contact. This factor will be half of the
% HDWT
```

```
%Convert The diameters in radi
hdor=(hdod/2)-0.8*hdwt;
hdir=hdid/2;
feor=feod/2;
lcr=lcd/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;

%% Set the assembly origin

%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);
BendSA=[hdsorg(:,2)-hdorg(2),hdsorg(:,3)-hdorg(3)];

% Set the number of lattice elements if applicable
if LATp; nle=nfe; end

% Override pin count if debug micka is true
if debugMICKA; nfe=DebugDrNFE;  end

%*************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

if LATp

    %Lat Universe number assignment

    LatOffInd=1;
    LUNb=hdorg(1)*1000;

    for ff=1:size(hdsorg,1)


        LUN(ff)=LUNb+LatOffInd;
        LatOffInd=LatOffInd+1;

        LUNu{ff}=strjoin({'U=' Num2StrM(LUN(ff))},'');
        LUNf{ff}=strjoin({'fill=' Num2StrM(LUN(ff))},'');

        LUNfp(ff)=LUNb+LatOffInd;
        LatOffInd=LatOffInd+1;

        LatOffInd=LatOffInd+1;

        % The universe assigned to the pins also needs to be reset.
        UNICfp{ff}=strjoin({' U=' Num2StrM(LUNfp(ff)) ' IMP:N=1' },'');

        % Since this is a driver, the pins are all the same universe number.
        Latnums(ff,:)=repmat(LUNfp(ff),1,nfe);

    end

%     % Debugging line to pad the universemap with zeros.
%     if debugMICKA; Latnums=[repmat(LUNfp,1,nfe), zeros(1,nle-nfe)]; end

    % Lattice Window
    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatElezo=-feh/2;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice boundings
```

```
    LATr=hdir-(hdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;
    LATzo=mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins


else

    UNICfp=UNIC;

end

%**********************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=fslas+fsh+feoz;

if createplane

    [ fbPl, surfPlane ] = PinPlaneMaker( nfslas, fslas, fssh );

end

for tt=1:size(hdsorg,1)

    pinparm(tt,:)={hdorg(1),fbPl,nfslas,UNICfp{tt},mcnpfix};

end

%**********************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%**********************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap
FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

%%
%**********************************************************************
```

```
%Assigns dimensions and materials for the pins.
pind=[nfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];

%**********************************************************************

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

% If the assembly is experimental, then the comment line will reflect that

if MatMap{MSATyC}==10

    surfname=NameCard(['EXP Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    surfname=NameCard(['Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end

%%
% The following for loop creates the core region sections.

% This for loop creates the surface cards for the hex duct sections
% The minimum number of sections is 3, any more means each subsection will
% be divided further. The code will be built around three main sections and
% then subsections be treated within the loop.

for ii=1:size(hdsorg,1)

    % Outer Duct Assembly
    ODOWs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
    hdsorg(ii,3),hdorg(4)-leh-lch-hdwt,0,0,crh+ueh+leh+lch+2*hdwt,...
    hdor, ['$ Outer Wall of Hex Duct Sec: ' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    ODOWnums(ii,1)=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

    if (LAT || LATp) && ii==1

        LATNas=CharChecker({surfnum,'    SO',2*hdh,...
            '$ Surr Na For Lat'});

        LATNanum=surfnum;

        surfnum=surfnum+1;

    end

    %Creates the inner duct for the upper extension.

    % Inner Duct Upper Extention

    UEs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
    hdsorg(ii,3),hdorg(4)+crh,0,0,ueh,hdir,...
    ['$ Upper Extension Inner Hex: ' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    UEInum(ii,1)=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

    % Inner Duct Core Region
    CRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
    hdsorg(ii,3),hdorg(4),0,0,crh,hdir,...
    ['$ Inner Wall Hex Duct Sec:' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    CRnums(ii,1)=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

     if createplane

         CRs=[CRs; surfPlane];

     end

     if ii==1

        if LAT || LATp; surfctemp=[ODOWs; LATNas; UEs; CRs ];
         else    surfctemp=[ODOWs; UEs; CRs ]; end

     else
```

```
        if LAT || LATp; surfctemp=[ODOWs; CRs ];
         else    surfctemp=[ODOWs; CRs ]; end

     end

    %%

    if ii==1

        % Lower Extension Surface Cards

        % Inner Duct
        LEs=CharChecker({surfnum,'    RHP',hdsorg(end,2),...
        hdsorg(end,3),hdorg(4),0,0,-leh,hdir,[' $ Lower Extent Inner Hex Sec:' Num2StrM(size(hdsorg,1))]});

        %Assign the surface number to be used later in the cell creation
        LEInum=surfnum;

        %Update the surnum number
        surfnum=surfnum+1;

        ODLPls=CharChecker({surfnum,'    PZ',-leh-hdwt,...
            ['$ Plane Separation Duct to Cylinder Sec:' Num2StrM(size(hdsorg,1))]});

        %I need this facet for later.
        ODLPlnum=surfnum;

        %Update the surnum number
        surfnum=surfnum+1;

        % Lower Cylinder Extension

        ODCyls=CharChecker({surfnum,'    RCC',hdsorg(end,2),hdsorg(end,3),...
        hdorg(4)-leh-hdwt,0,0,-lch,lcr,[' $ Lower Extension Cylinder Sec:' Num2StrM(size(hdsorg,1))]});

        %Assign the surface number to be used later in the cell creation
        ODCylnum=surfnum;

        %Update the surnum number
        surfnum=surfnum+1;

    end

    % Lattice surfaces.

    if LATp

        % Lattice array surface.
        LATArs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),hdsorg(ii,3),hdorg(4)+LATzo ...
            ,0,0,LATh,LATr,['$ Element Lattice Bounding Surface Sec: ' Num2StrM(ii)]});

        LATpArnum(ii,1)=surfnum;

        surfnum=surfnum+1;

        % Creates the window element for the lattice
        LatWinEle=CharChecker({surfnum,'    RHP',hdsorg(ii,2),hdsorg(ii,3),hdorg(4)+LatElezo...
            ,0,0,LatEleh,0,LatEler,0,['$ Lattice Window Sec: ' Num2StrM(ii)]});

        LatWinElenum(ii,1)=surfnum;

        surfnum=surfnum+1;

        if ii==1

            %Naming card
            LATname=NameCard('Lattice Cards for Elements','Divider');

            %Append to the surface array

            LATsurfc=[LATname; LatWinEle; LATArs ];

        else

            LATsurfc=[LATsurfc; LatWinEle; LATArs];

        end

    end

    if ii==1

        % Append the surfaces to the surface card.
        ductsurfc=[surfctemp; LEs;  ODLPls;  ODCyls];

    else

        ductsurfc=[ductsurfc; surfctemp];

    end

end
%%
% Appending the Outerduct surface numbers to the hexsurfnum variable
```

```
if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe,size(hdsorg,1));

for uu=1:size(hdsorg,1)

    %% Pin Creator

    %Call the Pinmaker function to create the surface and cell cards for the
    %individual pins.
    %The for loop repeatadly calls the pinmaker function to create the cell and
    %surface cards for the individual pins.

    %The new HD section origin must be changed for porg to work

    %Pin maker has to be adjust just like hex maker to account for the
    %slices. This part must be done in a series of if statements so that
    %the slicing does not cut half sections.

    %Porg calculates the origin of all the pins in a particular section.

    porg=PinOrg(nfe,fep,hdsorg(uu,2),hdsorg(uu,3),hdorg(4)+feoz);

    % The pin origin needs to be moved slightly and centerd on the element
    % window

    if LATp

        % Add the "bend

%         porg(:,2)=hdsorg(uu,2)+BendSA(uu,1)-LATpino;
%         porg(:,3)=hdsorg(uu,3)+BendSA(uu,2);

        porg(:,2)=hdsorg(uu,2)-LATpino;
        porg(:,3)=hdsorg(uu,3);

    else

        porg(:,2)=hdsorg(2)-LATpino;
        porg(:,3)=hdsorg(3);

    end

    for mm=1:nfe

        %Like the main program, the origins of the pins need to be determined so
        %that they can be fed into a program to write the cards. This is what the
        %pinorg program does.

        %ocws = Oyter Cladding Wall Surface
        %wws = wire wrap surface

        %Fuel Pins

        [ pincellctemp, pinsurfctemp, ocws , wws ]=...
            PinMakerBend(porg(mm,:),pind,pinm,pinparm(uu,:));

        pinsurfnum(mm,uu)=ocws;
        wirewrapsurfnum(mm,uu)=wws;

        if mm==1

            FDPs=pinsurfctemp;
            FDPcT=pincellctemp;

        else

            FDPs=[FDPs; pinsurfctemp];
            FDPcT=[FDPcT; pincellctemp];

        end

    end

    %% Append the surface cards together

    if uu==1

       surfc=[surfname; ductsurfc; LATsurfc; FDPs;];
       FDPc=FDPcT;

    else
```

```
        surfc=[surfc; FDPs;];
        FDPc=[FDPc; FDPcT];

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

if MatMap{MSATyC}==10

    cellname=NameCard(['EXP Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    cellname=NameCard(['Driver MKIIA M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Driver Half Worth #' Num2StrM(hdorg(1))],'Divider');

% The slicing code is broken up into three major sections. Top Middle
% bottom. Each section will then be further broken down into slicing
% each section.

% The main sections are determined by the sections of the fuel slug. These
% are the minimum sections required to work

for dd=1:size(hdsorg,1)

    % The following variable will need to be changed for code that chooses
    % the assembly out of the katana planes

    if dd==1

        curPlnum=KatanaPls{dd,2};

    end

    if dd>1 && dd<size(hdsorg,1)

        TopPl=KatanaPls{dd-1,2};
        BotPl=KatanaPls{dd,2};

        curPlnum=[-TopPl,BotPl];

    end

    if dd==size(hdsorg,1)

        curPlnum=-KatanaPls{dd-1,2};

    end

    %% Top Section

    if dd==1

        % Outter Duct Wall

        ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
            CRnums(dd,1),UEInum(dd,1),curPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

        cellnum=cellnum+1;

        % Inside Duct Upper Extent Homog

        UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -UEInum(dd,1)...
            UNIC ,'$ Homog Upper Ext'});

        cellnum=cellnum+1;

        %Inside duct Core Region

        CRc={cellnum,'     ',Dnam,Dnad, -CRnums(dd)};

        cellnum=cellnum+1;

        %name the area

        LATpcellname=NameCard(['Pin Lattice Cards Sec: ' Num2StrM(dd)],'Divider');

        %Create the lattice fill card

        LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum(dd,1),'Lat=2',LUNu{dd},...
```

```
        ['IMP:N=1 $ Element Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN(dd), nle ,nar, Latnums(dd,:) );

    LATpcLns=CharChecker({'     ' [ '' Num2StrM(LATparln)]});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum(dd,1),...
        curPlnum,LUNf{dd},UNIC,[' $ Pin Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

    PNaname=NameCard(['Surrounding sodium Cell for Pins Sec: ' Num2StrM(dd)],'Divider');

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(dd),...
        wirewrapsurfnum(dd),UNICfp{dd}, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnum(dd,1),curPlnum,UNIC,...
            ['$ Sodium coolant Sec: ' Num2StrM(dd)]]);

    %Appends the upper lower extension and Core Region cards

    cellctemp=[ODc; UEc; CRc; FDPc; PNaname; FPNac; LATpc;];

end

%% Middle Sections

if dd>1 && dd<size(hdsorg,1)

    % Outter Duct Wall

    ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
        CRnums(dd,1),curPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    %Inside duct Core Region

    CRc={cellnum,'     ',Dnam,Dnad, -CRnums(dd)};

    %Taks on the importance and comment

    CRc=CharChecker([CRc,LATpArnum(dd,1),curPlnum,UNIC,...
        ['$ Sodium coolant Sec: ' Num2StrM(dd)]]);

    cellnum=cellnum+1;

    %name the area

    LATpcellname=NameCard(['Pin Lattice Cards Sec: ' Num2StrM(dd)],'Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum(dd,1),'Lat=2',LUNu{dd},...
        ['IMP:N=1 $ Element Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN(dd), nle ,nar, Latnums(dd,:) );

    LATpcLns=CharChecker({'     ' [ '' Num2StrM(LATparln)]});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum(dd,1),...
        curPlnum,LUNf{dd},UNIC,[' $ Pin Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    % Append the cards together.
```

```
    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

    cellnum=cellnum+1;

    PNaname=NameCard(['Surrounding sodium Cell for Pins Sec: ' Num2StrM(dd)],'Divider');

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(dd),...
        wirewrapsurfnum(dd),UNICfp{dd}, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Appends the upper lower extension and Core Region cards

    cellctemp= [LATpc; ODc; CRc; PNaname; FPNac;];

end

%% Bottom Section

if dd==size(hdsorg,1)

    % Outter Duct Wall

    ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
        CRnums(dd,1),curPlnum,LEInum,ODLPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    %Inside duct Core Region

    CRc={cellnum,'     ',Dnam,Dnad, -CRnums(dd)};

    cellnum=cellnum+1;

    %Taks on the importance and comment

    CRc=CharChecker([CRc,LATpArnum(dd,1),curPlnum,UNIC,...
        ['$ Sodium coolant Sec: ' Num2StrM(dd)]]);

    cellnum=cellnum+1;

    %name the area

    LATpcellname=NameCard(['Pin Lattice Cards Sec: ' Num2StrM(dd)],'Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum(dd,1),'Lat=2',LUNu{dd},...
        ['IMP:N=1 $ Element Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN(dd), nle ,nar, Latnums(dd,:) );

    LATpcLns=CharChecker({'     ' [ '' Num2StrM(LATparln)]});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum(dd,1),...
        curPlnum,LUNf{dd},UNIC,[' $ Pin Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

    PNaname=NameCard(['Surrounding sodium Cell for Pins Sec: ' Num2StrM(dd)],'Divider');

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(dd),...
        wirewrapsurfnum(dd),UNICfp{dd}, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    LEname=NameCard('Lower Extension and Adapter Cards','Divider');

    %Inside duct Lower Extension Homog

    LEc=CharChecker({cellnum,'     ',SLOm,SLOd, -LEInum...
        UNIC ,'$ Smeared Lower Ext'});

    cellnum=cellnum+1;

    %Sodium around the lower extension cylinder

    CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -ODOWnums(dd,1),...
```

```
            -ODLPlnum,ODCylnum,UNIC ,'$ Na Surr Lower Ext'});

        cellnum=cellnum+1;

        % Lower Cylinder

        Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
            UNIC ,'$ Lower Cyn Homog'});

        cellnum=cellnum+1;

        %Appends the upper lower extension and Core Region cards

        cellctemp=[LATpc; ODc; CRc; PNaname; FPNac; LEname; LEc; CylNac; Cylc];

    end

    if dd==1

        cellc=[cellname; cellname2; cellctemp];

    else

        cellc=[cellc; cellctemp];

    end

end

%%

global SaveSAMakerVars

if SaveSAMakerVars

    if MatMap{MSATyC}==10

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_EXP_' MatMap{SANomen} '_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    else

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Dr_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    end

end

end
```

B.65. PinMakerBend.m

```
function [ cellc, surfc , ocws , wr] = PinMakerBend(porg,pind,pinm,pinparm,SATy)

%This function creates the pins in the same manner as the hexmaker.
% MKIIA pin

%This function Designs a pin as follows

% A slug immersed in fluid
% Fluid Plenum above the immersed slug
% Cladding around both fluids
% Wire Wrap around the cladding

global surfnum
global cellnum
global planesurfnum
global BorD
global KatanaPls

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

PinPlanes=pinparm{2};

gasbarsurfnum=PinPlanes{1,1};
sec12surfnum=PinPlanes{2,1};
sec23surfnum=PinPlanes{3,1};

%Sets the MICKA num
hdnum=pinparm{1};

%Fuel sodium level
fslas=pinparm{3};

%Sets the universe number
UNIC=pinparm{4};
```

```
%Mcnpfix
mcnpfix=100*pinparm{end};

%%
%***************Define the pin material parameters**********************

%slug material and density

pinmln=3;

slugs1m=pinm(pinmln-2,1);
slugs1d=pinm(pinmln-2,2);
slugs2m=pinm(pinmln-1,1);
slugs2d=pinm(pinmln-1,2);
slugs3m=pinm(pinmln,1);
slugs3d=pinm(pinmln,2);

%cladding material and density
cladm=pinm(pinmln+1,1);
cladd=pinm(pinmln+1,2);

%wire wrap material and density
wwrm=pinm(pinmln+2,1);
wwrd=pinm(pinmln+2,2);

%Coolant material and density
nam=pinm(pinmln+3,1);
nad=pinm(pinmln+3,2);

%plenum gas material and density

PGm=pinm(pinmln+4,1);
PGd=pinm(pinmln+4,2);

%%
%*********************************Pin Dimensions***************

% Spade height This will change the pin height to accomodate a spade

%Height of fuel from the bottom of the spade to bottom of fuel slug

%Fuel slug height
fsh=pind(2);

%Fuel Slug Radius
fsr=pind(3);

%cladding wall thickness
fewt=pind(4);

%Cladding inner radius
feir=pind(5);

%Pin height
feh=pind(6);

%Pin radius
feor=pind(7);

%Wire Wrap radius
fewwr=pind(8);

%Wire Wrap Length
fewwh=pind(9);

% Fuel Plug Top Length
fetph=pind(10);

% Spade Plug length
fesph=pind(11);

%%
%*******************************Surface Cards*****************************

%Since the cards for the pin and the subsequent pin sections will become
%huge, I will put a title in the cards for each pin now.

surfname=NameCard(['Pin: ' num2str(porg(1)) ...
                   ' of SA: ' num2str(hdnum)],'Divider');

%Outer Diameter Cladding

surfctemp=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)...
    ,0,0,feh,feor,'$ Pin:',porg(1),'Outer Cladding wall'});

%Assign the surface number to be used later in the cell creation
ocws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Inner Diameter Cladding

surfctemp2=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+fesph...
    ,0,0,feh-(fetph+fesph),feir,...
```

```
    '$ Pin:',porg(1),'Inner cladding wall'});

%Assign the surface number to be used later in the cell creation
icws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%Fuel slug cylinder

surfctemp3=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+fesph+mcnpfix...
    ,0,0,fsh,fsr,'$ Pin:',porg(1),'Fuel slug boundary'});

%Assign the surface number to be used later in the cell creation
fs=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%WireWrap
%This will have to call another function to calculate and generate the
%wirewrap positions. For right now we are just going to do one cylinder
%running up the side of the outter pin cladding. The cylinder will be
%parrallel to the pin with a 90 degree offset in the xy plane.

surfctemp4=CharChecker({surfnum,'    RCC',porg(2)+feor+fewwr+mcnpfix,...
    porg(3),porg(4),0,0,fewwh,fewwr,'$ Pin:',porg(1),'Wire Wrap'});

%Since the wrap is on the outside of the pin, it also needs to give its
%number to pinsurf so that hex maker can know the sodium is on the outside.

wr=surfnum;

%Update the surnum number
surfnum=surfnum+1;

%%
%*****************************Cell Cards*********************************
% Pin number title to seperate the pins

cellname=NameCard(['Pin: ' num2str(porg(1)) ...
    ' of SA: ' num2str(hdnum)],'Divider');

% These if statements are to check if this is the top of the hex duct section.
% the impact of this is that the top plane does not need to be used.

% Top part above the top of the element and above the sodium height.

%***************************** Benchmark *********************************
%Creates a fuel element

% Cladding

cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
    icws,UNIC,'$ Pin:',porg(1),' Cladding'});

cellnum=cellnum+1;

% Helium

cellctemp1=CharChecker({cellnum,'     ',PGm,PGd, gasbarsurfnum,...
    -icws,UNIC,'$ Pin:',porg(1),' Plenum Gas'});

cellnum=cellnum+1;

%Fuel sec1

cellctemp2=CharChecker({cellnum,'     ',slugs1m,slugs1d, -fs,sec12surfnum,...
      UNIC,'$ Pin:',porg(1),' Fuel Slug Sec 1'});

 cellnum=cellnum+1;

if pinmln==3

    %Fuel sec2

    cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',slugs2m,slugs2d,...
        -fs,-sec12surfnum, sec23surfnum,...
        UNIC,'$ Pin:',porg(1),' Fuel Slug Sec 2'})];

     cellnum=cellnum+1;

    %Fuel sec2

    cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',slugs3m,slugs3d, -fs,...
          -sec23surfnum, UNIC,'$ Pin:',porg(1),' Fuel Slug Sec 3'})];

     cellnum=cellnum+1;

end

% Sodium

cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',nam,nad,...
     -gasbarsurfnum, -icws,fs,UNIC,'$ Pin:',porg(1),' Sodium'})];
```

```
cellnum=cellnum+1;

%Wire Wrap

cellctemp2=[cellctemp2; CharChecker({cellnum,'     ',wwrm,wwrd,...
    -wr,UNIC,'$ Pin:',porg(1),' Wire Wrap'})];

cellnum=cellnum+1;

%***************************** Benchmark *********************************

%Append the cell cards

cellc=[cellname; cellctemp; cellctemp1; cellctemp2];

%%
%Assign the cards to be returned to hexmaker.
surfc=[surfname; surfctemp; surfctemp2; surfctemp3; surfctemp4];

%%

global SaveSAMakerVars
persistent FirstRun

FirstRun=true;

if SaveSAMakerVars && FirstRun

    % Save TotDimDat
    save('Debug\SAmakerVar\Tot_PoisonPin_Dat.mat')
    dispPrint('     TotDat_PoisonPin_saved....');

    FirstRun=false;

end

end
```

B.66. DriverMakerMKIIHFWBend.m

```
function [ cellc,surfc ] = DriverMakerMKIIHFWBend(hdorg,mcnpfix,bend, MatMap, DimMap )
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global debugMICKA DebugDrHFWNFE DebugDrHFWNDE
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum fbPl createplane KatanaPls
global LAT LATp
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC bhC crhC uehC lehC lcdC lchC fshC fsdC
global fslasC fehC feodC fewtC feozC fewwdC nfeC fepC ndeC fetphC fesphC
global feoxC

%% Material Vars

global FmC FdC FNamC FNadC FPGmC FPGdC  FCladmC FCladdC FPwwrmC FPwwdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC LAmC LadC DmC DdC

DebugStruct=false;

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sub assembly dimensions and types.

%SA type for use with the pin maker

SATy=MatMap{MSATyC};

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt); bh=DimMap{bhC};
crh=DimMap{crhC}; ueh=DimMap{uehC}; leh=DimMap{lehC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Fuel Elements
fsh=DimMap{fshC}; fsd=DimMap{fsdC}; fslas=DimMap{fslasC}; feh=DimMap{fehC};
feod=DimMap{feodC}; fewt=DimMap{fewtC}; feid=feod-(2*fewt); feoz=DimMap{feozC}+mcnpfix;
feox=DimMap{feoxC}; fewwh=feh; fewwd=DimMap{fewwdC}; nfe=DimMap{nfeC}; fep=DimMap{fepC};
fetph=DimMap{fetphC}; fesph=DimMap{fesphC};
```

```
% Dummy Elements, they are the same as fuel elements but solid ss
deod=feod; deid=feid; deh=feh; dewt=fewt; dewwd=fewwd;...
    dewwh=fewwh; nde=DimMap{ndeC};

% Fuel slug section heights
fssh=fsh/3;

% A factor needs to be instroduced to reduce the overall diameter of the
% duct such that it can handle contact. This factor will be half of the
% HDWT

%Convert The diameters in radi
hdor=(hdod/2)-0.8*hdwt;
hdir=hdid/2;
feor=feod/2;
lcr=lcd/2;
fsr=fsd/2;
feir=feid/2;
fewwr=fewwd/2;
deor=deod/2;
deir=deid/2;
dewwr=dewwd/2;

%% Set the assembly origin

%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);
BendSA=[hdsorg(:,2)-hdorg(2),hdsorg(:,3)-hdorg(3)];

% Set the number of lattice elements if applicable
if LATp; nle=nfe+nde; end

% Override pin count if debug micka is true
if debugMICKA;  nfe=DebugDrHFWNFE; nde=DebugDrHFWNDE; end

%Wire Wrap Switch 1 = on 0 = off.
dewws=1;

% Solid dummy element 1=solid 0=hollow;
dess=1;

%*************************************************************************
%Sets the universe num

if LAT

    UNIC=strjoin({' U=' Num2StrM(hdorg(1)) ' IMP:N=1' },'');

    % Reset the origin of the SA
    hdorg(2:3)=0;


else

    UNIC=' IMP:N=1';

end

if LATp

    % This section prepares the varaiables for the elements to be placed into a
    % lattice.

    %Lat Universe number assignment

    LatOffInd=1;
    LUNb=hdorg(1)*1000;

    for ff=1:size(hdsorg,1)


        LUN(ff)=LUNb+LatOffInd;
        LatOffInd=LatOffInd+1;

        LUNu{ff}=strjoin({'U=' Num2StrM(LUN(ff))},'');
        LUNf{ff}=strjoin({'fill=' Num2StrM(LUN(ff))},'');

        LUNfp(ff)=LUNb+LatOffInd;
        LatOffInd=LatOffInd+1;

        LUNdp(ff)=LUNfp(ff)+1;
        LatOffInd=LatOffInd+1;

        % The universe assigned to the pins also needs to be reset.

        UNICfp{ff}=strjoin({' U=' Num2StrM(LUNfp(ff)) ' IMP:N=1' },'');
        UNICdp{ff}=strjoin({' U=' Num2StrM(LUNdp(ff)) ' IMP:N=1' },'');

    end

    % Since this is a driver, the pins are all the same universe number.
    % Latnums=repmat(LUNfp,1,nfe);
    % The previous line is commented out because a HFW driver has its own pin
```

```
    % map that it needs to follow.

    % Debugging line to pad the universemap with zeros.
    %    if debugMICKA; LatnumsT=...
    %            [repmat(LUNdp,1,nde), repmat(LUNfp,1,nfe), zeros(1,nle-nfe-nde)]; end

    LatEler=fep/2;
    LatEleh=2*feh; % This height is increased to not give a perfect fit
    LatElezo=-feh/2;

    % This puts a slight offset in the pin origin so that the lattice does not
    % cut it off.

    LATpino=feox;

    %Geometry calculations

    %Number of hex duct rings.

    nr=ceil((3+sqrt(12*nle-3))/6)-1;

    %Number of rings needed to get circular motif

    %nar number of additional rings

    nar=2;

    fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

    % Lattice array size

    LATr=hdir-(hdir/100);

    if feh>=fewwh; FLatH=feh; else FLatH=fewwh; end

    LATh=FLatH+2*mcnpfix;
    LATzo=mcnpfix;

    % We now have to reset the number of fuel elements so that the pinmaker
    % function is only called once.

    nfe=1;   % New number of pins
    nde=1;

else

    UNICfp=UNIC;
    UNICdp=UNIC;

end


%**************************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

nfslas=fslas+fsh+feoz;

if createplane

    [ fbPl, surfPlane ] = PinPlaneMaker( nfslas, fslas, fssh );

end

for tt=1:size(hdsorg,1)

    pinparm(tt,:)={hdorg(1),fbPl,nfslas,UNICfp{tt},mcnpfix};
    dpinparm(tt,:)={hdorg(1),fbPl,nfslas,UNICdp{tt},mcnpfix};

end

%**************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Fuel Fluid Bond

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% Smeared Upper Extension Materials

SUPm=MatMap{SUPmC};
SUPd=MatMap{SUPdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};
```

```
% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%**********************************************************************
% Pin Materials

% Fuel Slug

Fm=MatMap{FmC}';
Fd=MatMap{FdC};

% Cladding

FCladm=MatMap{FCladmC};
FCladd=MatMap{FCladdC};

% Wire Wrap
FWwrm=MatMap{FPwwrmC};
FWwrd=MatMap{FPwwdC};

% Fuel Fluid Bond

Fnam=MatMap{FNamC};
Fnad=MatMap{FNadC};

% Fuel Plenum Gas

FPGm=MatMap{FPGmC};
FPGd=MatMap{FPGdC};

% Dummy Pins

% Dummy Fuel Slug
Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Dummy Fuel Slug
Deim=Dm;
Deid=Dd;

% Wire Wrap
Dewwrm=FWwrm;
Dewwrd=FWwrd;

%**********************************************************************
%Assigns dimensions and materials for the pins.
pind=[nfslas;fsh;fsr;fewt;feir;feh;feor;fewwr;fewwh;fetph;fesph];
dpind=[nfslas;NaN;NaN;dewt;deir;deh;deor;dewwr;dewwh;dewws;dess];

%Passed to the pins
pinm=[Fm,Fd;FCladm,FCladd;FWwrm,FWwrd;Fnam,Fnad;FPGm,FPGd];
dpinm=[Dm,Dd;Deim,Deid;Dewwrm,Dewwrd];

%**********************************************************************
%The following is a halfworth driver map to map where the dummy pins go.

if strcmp(MatMap{SANomen},'X402A')

pinmap=[1;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2;2;2;2;2;2;2;2;2;2;...
        2];

else

    pinmap=[1;1;2;2;1;2;2;2;2;1;1;2;1;2;2;1;1;2;1;2;1;1;2;2;1;1;2;1;2;1;1;2;2;1;...
        1;2;1;1;1;2;2;1;1;2;2;1;2;1;2;1;1;1;2;2;1;1;2;2;1;2;1;2;1;2;2;1;1;2;2;1;...
        1;2;2;1;2;1;2;1;2;1;2;2;1;1;2;2;1;1;2;2;1;2;1;2;1;2];

end

hwdmap=[linspace(1,nfe+nde,nfe+nde)' pinmap(1:nfe+nde)];

% Changes the pin map and hwdmap to a lattice.

if LATp

    hwdmap=[1,1;1,2];

    % Create the lattice map from the pinmap by replacing 1 and 2's with
    % universe numbers.

    for hh=1:size(hdsorg,1)

        LatnumsT=pinmap';

        LatnumsT(LatnumsT==1)=LUNdp(hh);
```

```
        LatnumsT(LatnumsT==2)=LUNfp(hh);

        if hh==1

            Latnums=LatnumsT;

        else

            Latnums=[Latnums;LatnumsT];

        end

    end

end

% Debug
% if debugMICKA && LATp; Latnums=LatnumsT; end

%%
%Debug only
%
% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Half Worth Driver MKIIA M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


%%
% The following for loop creates the core region sections.

% This for loop creates the surface cards for the hex duct sections
% The minimum number of sections is 3, any more means each subsection will
% be divided further. The code will be built around three main sections and
% then subsections be treated within the loop.

for ii=1:size(hdsorg,1)

    % Outer Duct Assembly
    ODOWs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
    hdsorg(ii,3),hdorg(4)-leh-lch-hdwt,0,0,crh+ueh+leh+lch+2*hdwt,...
    hdor, ['$ Outer Wall of Hex Duct Sec: ' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    ODOWnums(ii,1)=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

    if (LAT || LATp) && ii==1

        LATNas=CharChecker({surfnum,'    SO',2*hdh,...
            '$ Surr Na For Lat'});

        LATNanum=surfnum;

        surfnum=surfnum+1;

    end

    %Creates the inner duct for the upper extension.

    % Inner Duct Upper Extention

    UEs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
    hdsorg(ii,3),hdorg(4)+crh,0,0,ueh,hdir,...
    ['$ Upper Extension Inner Hex' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    UEInum(ii)=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

    % Inner Duct Core Region
    CRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
    hdsorg(ii,3),hdorg(4),0,0,crh,hdir,...
    ['$ Inner Wall Hex Duct Sec:' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    CRnums(ii,1)=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

     if createplane
```

```
     CRs=[CRs; surfPlane];

 end

 if ii==1

    if LAT || LATp; surfctemp=[ODOWs; LATNas; UEs; CRs ];
     else    surfctemp=[ODOWs; UEs; CRs ]; end

 else

    if LAT || LATp; surfctemp=[ODOWs; CRs ];
     else    surfctemp=[ODOWs; CRs ]; end

 end

%%

if ii==1

    % Lower Extension Surface Cards

    % Inner Duct
    LEs=CharChecker({surfnum,'    RHP',hdsorg(end,2),...
    hdsorg(end,3),hdorg(4),0,0,-leh,hdir,[' $ Lower Extent Inner Hex Sec:' Num2StrM(size(hdsorg,1))]});

    %Assign the surface number to be used later in the cell creation
    LEInum=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

    ODLPls=CharChecker({surfnum,'    PZ',-leh-hdwt,...
        ['$ Plane Separation Duct to Cylinder Sec:' Num2StrM(size(hdsorg,1))]});

    %I need this facet for later.
    ODLPlnum=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

    % Lower Cylinder Extension

    ODCyls=CharChecker({surfnum,'    RCC',hdorg(2),hdorg(3),...
    hdorg(4)-leh-hdwt,0,0,-lch,lcr,[' $ Lower Extension Cylinder Sec:' Num2StrM(ii)]});

    %Assign the surface number to be used later in the cell creation
    ODCylnum=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

end

% Lattice surfaces.

if LATp

    % Lattice array surface.
    LATArs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),hdsorg(ii,3),hdorg(4)+LATzo ...
        ,0,0,LATh,LATr,['$ Element Lattice Bounding Surface Sec: ' Num2StrM(ii)]});

    LATpArnum(ii,1)=surfnum;

    surfnum=surfnum+1;

    % Creates the window element for the lattice
    LatWinEle=CharChecker({surfnum,'    RHP',hdsorg(ii,2),hdsorg(ii,3),hdorg(4)+LatElezo...
        ,0,0,LatEleh,0,LatEler,0,['$ Lattice Window Sec: ' Num2StrM(ii)]});

    LatWinElenum(ii,1)=surfnum;

    surfnum=surfnum+1;

    if ii==1

        %Naming card
        LATname=NameCard('Lattice Cards for Elements','Divider');

        %Append to the surface array

        LATsurfc=[LATname; LatWinEle; LATArs ];

    else

        LATsurfc=[LATsurfc; LatWinEle; LATArs];

    end

end

if ii==1

    % Append the surfaces to the surface card.
    ductsurfc=[surfctemp; LEs;  ODLPls;  ODCyls];
```

```
    else

        ductsurfc=[ductsurfc; surfctemp];

    end

end

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nfe+nde,size(hdsorg,1));
wirewrapsurfnum=zeros(nfe+nde,size(hdsorg,1));

for uu=1:size(hdsorg,1)

    %% Pin Creator

    %Call the Pinmaker function to create the surface and cell cards for the
    %individual pins.
    %The for loop repeatadly calls the pinmaker function to create the cell and
    %surface cards for the individual pins.

    %The new HD section origin must be changed for porg to work

    %Pin maker has to be adjust just like hex maker to account for the
    %slices. This part must be done in a series of if statements so that
    %the slicing does not cut half sections.

    %Porg calculates the origin of all the pins in a particular section.

    porg=PinOrg((nfe+nde),fep,hdsorg(uu,2),hdsorg(uu,3),hdorg(4)+feoz);

    % The pin origin needs to be moved slightly and centerd on the element
    % window

    if LATp

        % Add the "bend

%         porg(:,2)=hdsorg(uu,2)+BendSA(uu,1)-LATpino;
%         porg(:,3)=hdsorg(uu,3)+BendSA(uu,2);

        porg(:,2)=hdsorg(uu,2)-LATpino;
        porg(:,3)=hdsorg(uu,3);

    else

        porg(:,2)=hdsorg(2)-LATpino;
        porg(:,3)=hdsorg(3);

    end

    for mm=1:(nfe+nde)

        %Like the main program, the origins of the pins need to be determined so
        %that they can be fed into a program to write the cards. This is what the
        %pinorg program does.

        %ocws = Oyter Cladding Wall Surface
        %wws = wire wrap surface

        %Dummy Pins

        if hwdmap(mm,2)==1

            pinparmT=dpinparm(uu,:);

            [ pincellctemp, pinsurfctemp, ocws , wws]=...
                DummyPinMakerBend(porg(mm,:),dpind,dpinm,pinparmT,hdsorg(uu,:),uu,SATy);

        end

        %Fuel Pins

        if hwdmap(mm,2)==2

            pinparmT=pinparm(uu,:);

            [ pincellctemp, pinsurfctemp, ocws , wws ]=...
                PinMakerBend(porg(mm,:),pind,pinm,pinparmT,SATy);
```

```
        end

        pinsurfnum(mm,uu)=ocws;
        wirewrapsurfnum(mm,uu)=wws;

        if mm==1

            FDPs=pinsurfctemp;
            FDPcT=pincellctemp;

        else

            FDPs=[FDPs; pinsurfctemp];
            FDPcT=[FDPcT; pincellctemp];

        end

    end

    %% Append the surface cards together

    if uu==1

       surfc=[surfname; ductsurfc; LATsurfc; FDPs;];
       FDPc=FDPcT;

    else

       surfc=[surfc; FDPs;];
       FDPc=[FDPc; FDPcT];

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['Half Worth Driver MKIIA M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

%% %Hex Duct Cell cards

%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Driver Half Worth #' Num2StrM(hdorg(1))],'Divider');

% The slicing code is broken up into three major sections. Top Middle
% bottom. Each section will then be further broken down into slicing
% each section.

% The main sections are determined by the sections of the fuel slug. These
% are the minimum sections required to work

KatPlSize=size(KatanaPls,2);

for dd=1:size(hdsorg,1)

    % The following variable will need to be changed for code that chooses
    % the assembly out of the katana planes

    if dd==1

        curPlnum=KatanaPls{dd,2};

    end

    if dd>1 && dd<size(hdsorg,1)

        TopPl=KatanaPls{dd-1,2};
        BotPl=KatanaPls{dd,2};

        curPlnum=[-TopPl,BotPl];

    end

    if dd==size(hdsorg,1)

        curPlnum=-KatanaPls{dd-1,2};

    end

    %% Top Section

    if dd==1

        % Outter Duct Wall

        ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
            CRnums(dd,1),UEInum(dd,1),curPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});
```

```
    cellnum=cellnum+1;

    % Inside Duct Upper Extent Homog

    UEc=CharChecker({cellnum,'     ',SUPm,SUPd, -UEInum(dd,1)...
        UNIC ,'$ Homog Upper Ext'});

    cellnum=cellnum+1;

    %Inside duct Core Region

    CRc={cellnum,'     ',Dnam,Dnad, -CRnums(dd)};

    cellnum=cellnum+1;

    %name the area

    LATpcellname=NameCard(['Pin Lattice Cards Sec: ' Num2StrM(dd)],'Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum(dd,1),'Lat=2',LUNu{dd},...
        ['IMP:N=1 $ Element Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN(dd), nle ,nar, Latnums(dd,:) );

    LATpcLns=CharChecker({'     ' [ '' Num2StrM(LATparln)]});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum(dd,1),...
        curPlnum,LUNf{dd},UNIC,[' $ Pin Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

    PNaname=NameCard(['Surrounding sodium Cell for Pins Sec: ' Num2StrM(dd)],'Divider');

    % This defines the surrounding sodium around the dummy pin.

    DPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(1,dd),...
        wirewrapsurfnum(1,dd),UNICdp{dd}, '$ Dummy Pin Cell'});

    cellnum=cellnum+1;

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(2,dd),...
        wirewrapsurfnum(2,dd),UNICfp{dd}, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Taks on the importance and comment

        CRc=CharChecker([CRc,LATpArnum(dd,1),curPlnum,UNIC,...
            ['$ Sodium coolant Sec: ' Num2StrM(dd)]]);

    %Appends the upper lower extension and Core Region cards

    cellctemp=[ODc; UEc; CRc; FDPc; PNaname; DPNac;...
        FPNac; LATpc;];

end

%% Middle Sections

if dd>1 && dd<size(hdsorg,1)

    % Outter Duct Wall

    ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
        CRnums(dd,1),curPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    %Inside duct Core Region

    CRc={cellnum,'     ',Dnam,Dnad, -CRnums(dd)};

    %Taks on the importance and comment

    CRc=CharChecker([CRc,LATpArnum(dd,1),curPlnum,UNIC,...
        ['$ Sodium coolant Sec: ' Num2StrM(dd)]]);
```

```
    cellnum=cellnum+1;

    %name the area

    LATpcellname=NameCard(['Pin Lattice Cards Sec: ' Num2StrM(dd)],'Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum(dd,1),'Lat=2',LUNu{dd},...
        ['IMP:N=1 $ Element Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
    %aka the wall

    [LATparln,LATar]=MCNPPOSMaker( LUN(dd), nle ,nar, Latnums(dd,:) );

    LATpcLns=CharChecker({'     ' [ '' Num2StrM(LATparln)]});

    % Fills the lattice into the core.

    LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum(dd,1),...
        curPlnum,LUNf{dd},UNIC,[' $ Pin Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    % Append the cards together.

    LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

    cellnum=cellnum+1;

    PNaname=NameCard(['Surrounding sodium Cell for Pins Sec: ' Num2StrM(dd)],'Divider');

    % This defines the surrounding sodium around the dummy pin.

    DPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(1,dd),...
        wirewrapsurfnum(1,dd),UNICdp{dd}, '$ Dummy Pin Cell'});

    cellnum=cellnum+1;

    % This defines the surrounding sodium around the fuel pin.

    FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(2,dd),...
        wirewrapsurfnum(2,dd),UNICfp{dd}, '$ Fuel Pin Cell'});

    cellnum=cellnum+1;

    %Appends the upper lower extension and Core Region cards

    cellctemp= [LATpc; ODc; CRc; PNaname; DPNac; FPNac;];

end

%% Bottom Section

if dd==size(hdsorg,1)

    % Outter Duct Wall

    ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
        CRnums(dd,1),curPlnum,LEInum,ODLPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    %Inside duct Core Region

    CRc={cellnum,'     ',Dnam,Dnad, -CRnums(dd)};

    cellnum=cellnum+1;

    %Taks on the importance and comment

    CRc=CharChecker([CRc,LATpArnum(dd,1),curPlnum,UNIC,...
        ['$ Sodium coolant Sec: ' Num2StrM(dd)]]);

    cellnum=cellnum+1;

    %name the area

    LATpcellname=NameCard(['Pin Lattice Cards Sec: ' Num2StrM(dd)],'Divider');

    %Create the lattice fill card

    LATpArc=CharChecker({cellnum,'     ',Dnam,Dnad,-LatWinElenum(dd,1),'Lat=2',LUNu{dd},...
        ['IMP:N=1 $ Element Lattice Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    LATpFlc=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

    %Creates the lattice line, fills in the outside zeros with the lat universe
```

```
        %aka the wall

        [LATparln,LATar]=MCNPPOSMaker( LUN(dd), nle ,nar, Latnums(dd,:) );

        LATpcLns=CharChecker({'     ' [ '' Num2StrM(LATparln)]});

        % Fills the lattice into the core.

        LATpcell=CharChecker({cellnum,'     ',Dnam,Dnad,-LATpArnum(dd,1),...
            curPlnum,LUNf{dd},UNIC,[' $ Pin Lattice Sec: ' Num2StrM(dd)]});

        cellnum=cellnum+1;

        % Append the cards together.

        LATpc=[LATpcellname; LATpArc; LATpFlc; LATpcLns; LATpcell];

        PNaname=NameCard(['Surrounding sodium Cell for Pins Sec: ' Num2StrM(dd)],'Divider');

        % This defines the surrounding sodium around the dummy pin.

        DPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(1,dd),...
            wirewrapsurfnum(1,dd),UNICdp{dd}, '$ Dummy Pin Cell'});

        cellnum=cellnum+1;

        % This defines the surrounding sodium around the fuel pin.

        FPNac=CharChecker({cellnum,'     ',Dnam,Dnad,-LATNanum,pinsurfnum(2,dd),...
            wirewrapsurfnum(2,dd),UNICfp{dd}, '$ Fuel Pin Cell'});

        cellnum=cellnum+1;

        LEname=NameCard('Lower Extension and Adapter Cards','Divider');

        %Inside duct Lower Extension Homog

        LEc=CharChecker({cellnum,'     ',SLOm,SLOd, -LEInum...
            UNIC ,'$ Smeared Lower Ext'});

        cellnum=cellnum+1;

        %Sodium around the lower extension cylinder

        CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -ODOWnums(dd,1),...
            -ODLPlnum,ODCylnum,UNIC ,'$ Na Surr Lower Ext'});

        cellnum=cellnum+1;

        % Lower Cylinder

        Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
            UNIC ,'$ Lower Cyn Homog'});

        cellnum=cellnum+1;

        %Appends the upper lower extension and Core Region cards

        cellctemp=[LATpc; ODc; CRc; PNaname; DPNac; FPNac; LEname; LEc; CylNac; Cylc];

    end

    if dd==1

        cellc=[cellname; cellname2; cellctemp];

    else

        cellc=[cellc; cellctemp];

    end

end


%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_DrHFW_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

end

end
```

B.67. DummyMakerBend.m

```
function [ cellc,surfc ] = DummyMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap)
%This function writes the cell and surface cards for the hex duct and the
%pins
```

```
%declare the global variables

global debugMICKA DebugDumNDE KatanaPls
global surfnum cellnum
global hexsurfnum planesurfnum SAsurfnum fbPl
global LAT
global BorD

 %% Reactor Map Vars Breakout

global MSATyC SAPosC SANomen

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC crhC lcdC lchC sazoC dehC deodC dewtC
global deoC ndeC depC

%% Material Vars

global PPGmC PPGdC
global DuctmC DuctdC DnamC DnadC  SUPmC SUPdC  SLOmC SLOdC LAmC LadC DmC DdC

%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).


%************************************************************************
%Sets the universe num

if LAT==1

    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

%************************************************************************
%Sub assembly dimensions and types.

SATy=DimMap{MSATyC};

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
crh=DimMap{crhC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

% Dummy Elements, they are the same as fuel elements but solid ss
deh=DimMap{dehC}; deod=DimMap{deodC}; dewt=DimMap{dewtC}; deid=deod-(2*dewt);  ...
    deo=DimMap{deoC}+mcnpfix; nde=DimMap{ndeC}; dep=DimMap{depC}; dslas=NaN; dsh=NaN;

%Convert The diameters in radi

hdor=(hdod/2)-0.8*hdwt;
hdir=hdid/2;
lcr=lcd/2;
deor=deod/2;
deir=deid/2;

%Wire Wrap Switch 1 = on 0 = off.
dewws=0;

% Solid dummy element 1=solid 0=hollow;
dess=1;

% Override pin count if debug micka is true
if debugMICKA; nde=DebugDumNDE; end

%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);
BendSA=[hdsorg(:,2)-hdorg(2),hdsorg(:,3)-hdorg(3)];

%************************************************************************
%This code sets the barrier between gas/shield and sodium

%Recalculate the pin origin

ndslas=dslas+dsh+deo;

dpinparm={hdorg(1),fbPl,ndslas,UNIC,mcnpfix};

%************************************************************************
%Subassembly movement

%Since the dummy is just one long core region, I decided to adjust the
%origin of the whole assembly like a control rod.
```

```
sazo=DimMap{sazoC};

%**************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Duct Fluid

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

%**************************************************************************
% Pin Materials

% Dummy Element

Dm=MatMap{DmC};
Dd=MatMap{DdC};


% Dummy Fuel Slug
Deim=MatMap{PPGmC};
Deid=MatMap{PPGdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%%
%**************************************************************************

%Assigns dimensions and materials for the pins.
dpinm=[Dm,Dd;Deim,Deid;NaN,NaN];
dpind=[ndslas;NaN;NaN;dewt;deir;deh;deor;NaN;NaN;dewws;dess];

%**************************************************************************
%%

%This section is the dummy origin adjustment.

%Offest plus movement
hdorg(4)=hdorg(4)-sazo;

%**************************************************************************
%%

%This next line performs the expansion. It moves the pins slightly outward
%from the center of the core. This will be replaced by pin section movement
%later on in PinSectionMaker but for not it will reside here as a proof of
%concept.

%porg=ThermalX(porgtemp);


%The surfaces must be written first for the duct and all the pins, then the
%cell cards.

%%
%Debug only

% save('hdorg.txt','hdorg','-ASCII')
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')


%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

% If the assembly is experimental, then the comment line will reflect that

if MatMap{MSATyC}==10

    surfname=NameCard(['EXP Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    surfname=NameCard(['Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end
```

```
for ii=1:size(hdsorg,1)

    %%
    % The following for loop creates the core region sections.

    %This for loop creates the surface cards for the hex duct sections

    % Outer Duct Assembly
     ODOWs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4)-lch-hdwt,0,0,crh+lch+2*hdwt,...
     hdor, ['$ Outer Wall of Hex Duct Sec: ' Num2StrM(ii)]});

     %Assign the surface number to be used later in the cell creation
     ODOWnums(ii,1)=surfnum;

    %Update the surnum number
     surfnum=surfnum+1;

    % Inner Duct Core Region
     CRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4),0,0,crh,hdir,...
     ['$ Inner Wall Hex Duct Sec: ' Num2StrM(ii)]});

     %Assign the surface number to be used later in the cell creation
     CRnums(ii,1)=surfnum;

    %Update the surnum number
     surfnum=surfnum+1;

    surfctemp=[ODOWs; CRs];

    if ii==1

        % Lower Extension Surface Cards

        ODLPls=CharChecker({surfnum,'    PZ', hdorg(4)-hdwt,...
                        '$ Plane Separation Duct to Cylinder'});

        %I need this facet for later.
        ODLPlnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        % Lower Cylinder Extension

        ODCyls=CharChecker({surfnum,'    RCC',hdsorg(end,2),hdsorg(end,3),...
            hdorg(4)-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

        %Assign the surface number to be used later in the cell creation
        ODCylnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

    end

    if ii==1

        % Append the surfaces to the surface card.
        surfc=[surfctemp;  ODLPls;  ODCyls];

    else

        surfc=[surfc; surfctemp];

    end

end

%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end

%Pin surfnum is used as a place to hold all of the outter cladding surface
%numbers to be used in the cell cards.

pinsurfnum=zeros(nde,size(hdsorg,1));

for uu=1:size(hdsorg,1)

    %% Pin Creator

    %Call the Pinmaker function to create the surface and cell cards for the
```

```
    %individual pins.
    %The for loop repeatadly calls the pinmaker function to create the cell and
    %surface cards for the individual pins.

    %The new HD section origin must be changed for porg to work

    %Pin maker has to be adjust just like hex maker to account for the
    %slices. This part must be done in a series of if statements so that
    %the slicing does not cut half sections.

    %Porg calculates the origin of all the pins in a particular section.

    porg=PinOrg(nde,dep,hdsorg(uu,2),hdsorg(uu,3),hdorg(4)+deo);

    % Assign the planes to be used for the pin cells

        % The following variable will need to be changed for code that chooses
    % the assembly out of the katana planes

    if uu==1

        curPlnum=KatanaPls{uu,2};

    end

    if uu>1 && uu<size(hdsorg,1)

        TopPl=KatanaPls{uu-1,2};
        BotPl=KatanaPls{uu,2};

        curPlnum=[-TopPl,BotPl];

    end

    if uu==size(hdsorg,1)

        curPlnum=-KatanaPls{uu-1,2};

    end

    dpinparm{2}=curPlnum;

    for ii=1:nde

        %Like the main program, the origins of the pins need to be determined so
        %that they can be fed into a program to write the cards. This is what the
        %pinorg program does.

        %ocws = Outter Cladding Wall Surface
        %wws = wire wrap surface

        [ pincellctemp, pinsurfctemp, ocws , ~ ]=...
            DummyPinMakerBend(porg(ii,:),dpind,dpinm,dpinparm,hdsorg(uu,:),uu,SATy);

        pinsurfnum(ii,uu)=ocws;

        if (ii==1) && (uu==1)

            pinsurfc=pinsurfctemp;
            pincellc=pincellctemp;

        end

        if (ii>1) || (uu>1)

            pinsurfc=[pinsurfc; pinsurfctemp];
            pincellc=[pincellc; pincellctemp];

        end

    end

end

%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['Dummy M#' num2str(hdorg(1)) ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

if MatMap{MSATyC}==10

    cellname=NameCard(['EXP Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

else

    cellname=NameCard(['Dummy M#' num2str(hdorg(1))...
        ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

end

%% %Hex Duct Cell cards
```

```
%Need to add cutplanes

%Put a comment card in for the hex coolant and du

cellname2=NameCard(['Hex Duct, Dummy: ' num2str(hdorg(1))],'Divider');

for dd=1:size(hdsorg,1)

    % The following variable will need to be changed for code that chooses
    % the assembly out of the katana planes

    if dd==1

        curPlnum=KatanaPls{dd,2};

    end

    if dd>1 && dd<size(hdsorg,1)

        TopPl=KatanaPls{dd-1,2};
        BotPl=KatanaPls{dd,2};

        curPlnum=[-TopPl,BotPl];

    end

    if dd==size(hdsorg,1)

        curPlnum=-KatanaPls{dd-1,2};

    end

    % Outter Duct Wall

    ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
         CRnums(dd,1),curPlnum,ODLPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    %Inside duct Core Region

    DPc={cellnum,'     ',Dnam,Dnad, -CRnums(dd,1), curPlnum};

    cellnum=cellnum+1;

        %Adds the pin surface numbers to the inside duct definition

    for mm=1:size(pinsurfnum,1)

        DPc=[DPc pinsurfnum(mm,dd)];

    end

    %Taks on the importance and comment

    DPc=CharChecker([DPc,UNIC,...
        '$ Sodium coolant']);

    if dd==1

        %Sodium around the lower extension cylinder

        CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -ODOWnums(end,1),...
            -ODLPlnum,ODCylnum,UNIC,'$ Na Surr Lower Ext'});

        cellnum=cellnum+1;

        % Lower Cylinder

        Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
            UNIC,'$ Lower Cyn Homog'});

        cellnum=cellnum+1;

        %Appends the upper lower extensions and Core Region cards

        cellctemp=[ODc; DPc; CylNac; Cylc];

        cellc=[cellname; cellname2; cellctemp];

    else

        cellctemp=[ODc; DPc];

        cellc=[cellc; cellctemp];

    end

end

surfc=[surfname; surfc; pinsurfc];
cellc=[cellc; pincellc];

%%
```

```
global SaveSAMakerVars

if SaveSAMakerVars

    if MatMap{MSATyC}==10

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_EXP_' MatMap{SANomen} '_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    else

        % Save TotDimDat
        save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Dummy_Dat.mat'])
        dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);

    end

end

end
```

B.68. DummyPinMakerBend.m

```
function [ cellc, surfc , ocws , wr ] = DummyPinMakerBend(porg,dpind,dpinm,dpinparm,hdsorg,secnum,SATy)
%This function creates the pins in the same manner as the hexmaker.

global surfnum
global cellnum
global planesurfnum
global BorD
global KatanaPls

%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

%Sets the MICKA num
hdnum=dpinparm{1};

% Sets the planes for katana
if SATy==6

    KatPl=dpinparm{2};

else

    KatPl='';

end

%Universe / IMP
UNIC=dpinparm{4};

%Mcnpfix
mcnpfix=100*dpinparm{end};

% Initialize wire wrap surface number
wr=0;

%%
%***************Define the pin material parameters**********************

%cladding material and density
cladm=dpinm(1,1);
cladd=dpinm(1,2);

%Inner material
deim=dpinm(2,1);
deid=dpinm(2,2);

%wirewrap material
dewwrm=dpinm(3,1);
dewwrd=dpinm(3,2);

%%
%*********************************Pin Dimensions**************

%Dummy element wall thickness.
dewt=dpind(4);

%Pin Inner radius
deir=dpind(5);

%Pin height
deh=dpind(6);

%Pin outter radius
deor=dpind(7);

%Wire Wrap radius
dewwr=dpind(8);
```

```
% Dummy Element Wire Wrap height
dewwh=dpind(9);

%Wire Wrap Switch
% This switch controls if a wire wrap is created or not.
dewws=dpind(10);

% Dummy Element Solid switch determines if a solid dummy pin is being made
% or a hollow one.
dess=dpind(11);

%%
%*****************************Surface Cards*****************************

%Since the cards for the pin and the subsequent pin sections will become
%huge, I will put a title in the cards for each pin now.

surfname=NameCard(['Pin: ' num2str(porg(1)) ' of SA: ' num2str(hdnum) ' Sec: ' num2str(secnum)],'Divider');

%Outer Diameter Cladding

surfctemp=CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)...
    ,0,0,deh,deor,'$ Pin:',porg(1),'Dummy Pin Outter Clad'});

%Assign the surface number to be used later in the cell creation
ocws=surfnum;

%Update the surnum number
surfnum=surfnum+1;

if dess==0

    %Inner Diameter Cladding

    surfctemp=[surfctemp; CharChecker({surfnum,'    RCC',porg(2),porg(3),porg(4)+dewt...
        ,0,0,deh-2*dewt,deir,'$ Pin:',porg(1),'Dummy Pin Inner Clad'})];

    icws=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

end

if dewws==1

    %WireWrap
    %This will have to call another function to calculate and generate the
    %wirewrap positions. For right now we are just going to do one cylinder
    %running up the side of the outter pin cladding. The cylinder will be
    %parrallel to the pin with a 90 degree offset in the xy plane.

    surfctemp2=CharChecker({surfnum,'    RCC',porg(2)+deor+dewwr+mcnpfix,...
        porg(3),porg(4),0,0,dewwh,dewwr,'$ Pin:',porg(1),'Wire Wrap'});

    %Since the wirewrap is on the outside of the pin, it also needs to give its
    %number to pinsurf so that hex maker can know the sodium is on the outside.

    wr=surfnum;

    %Update the surnum number
    surfnum=surfnum+1;

end

%%
%*****************************Cell Cards*******************************
% Pin number title to seperate the pins

cellname=NameCard(['Pin: ' num2str(porg(1)) ' Sec: ' num2str(secnum)...
                   ' of SA: ' num2str(hdnum)],'Divider');

% These if statements are to check if this is the top of the hex duct section.
% the impact of this is that the top plane does not need to be used.

% Top part above the top of the element and above the sodium height.


%***************************** Benchmark *********************************
%Creates a dummy pin element

        if dess==0

            % Outter Clad

            cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
                icws,KatPl,UNIC,'$ Pin:',porg(1),'Dummy Pin OClad'});

            cellnum=cellnum+1;

            % Inner Clad

            cellctemp=[cellctemp; CharChecker({cellnum,'     ',deim,deid, -icws,...
                KatPl,UNIC,'$ Pin:',porg(1),'Dummy Pin Inside Region'})];
```

```
            % Update the cell number
             cellnum=cellnum+1;

        end

        if dess==1

            % Outter Clad

            cellctemp=CharChecker({cellnum,'     ',cladm,cladd, -ocws,...
                KatPl,UNIC,'$ Pin:',porg(1),'Dummy Pin'});

            % Update the cell number
             cellnum=cellnum+1;

        end

        if dewws==1

            %Wire Wrap

            cellctemp2= CharChecker({cellnum,'     ',dewwrm,dewwrd,...
                    -wr,KatPl,UNIC,'$ Pin:',porg(1),' Wire Wrap'});

            % Update the cell number
             cellnum=cellnum+1;

        end

%***************************** Benchmark *********************************



%%
%Assign the cards to be returned to hexmaker.

if dewws==0

    % No Wire Wrap
    surfc=[surfname; surfctemp];
    cellc=[cellname; cellctemp];

end

if dewws==1

    % Wire Wrap
    surfc=[surfname; surfctemp; surfctemp2];
    cellc=[cellname; cellctemp; cellctemp2];


end

end
```

B.69. ReflectorMakerBend.m

```
function [ cellc,surfc ] = ReflectorMakerBend(hdorg,mcnpfix,bend,MatMap,DimMap)
%This function writes the cell and surface cards for the hex duct and the
%pins

%declare the global variables
global surfnum
global cellnum
global hexsurfnum
global planesurfnum
global SAsurfnum
global LAT
global BorD
global KatanaPls

 %% Reactor Map Vars Breakout

global SAPosC SANomen MSATyC

%% Dimensional Vars

global upphC uppdC hdhC hdodC hdwtC crhC lcdC lchC sazoC

%% Material Vars

global DuctmC DuctdC DnamC DnadC  SLOmC SLOdC LAmC LadC DmC DdC

%*************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD = true (Bench) or not (Dissert).

%*************************************************************************
%Sets the universe num

if LAT==1
```

```
    UNIC=strjoin({' U=' num2str(hdorg(1)) ' IMP:N=1' },'');
    hdorg(2:3)=0;

else

    UNIC=' IMP:N=1';

end

%************************************************************************
%Sub assembly dimensions and types.

SATy=DimMap{MSATyC};

%Upper Pole Piece
upph=DimMap{upphC}; uppd=DimMap{uppdC};

%Hex Duct
hdh=DimMap{hdhC}; hdod=DimMap{hdodC}; hdwt=DimMap{hdwtC}; hdid=hdod-(2*hdwt);
crh=DimMap{crhC}; lcd=DimMap{lcdC}; lch=DimMap{lchC};

%Convert The diameters in radi

hdor=(hdod/2)-0.8*hdwt;
hdir=hdid/2;
lcr=lcd/2;

%hdsorg calculates the dimensions of the hex duct sections to be created.

hdsorg=ThermalXHex(crh,hdorg,bend);
BendSA=[hdsorg(:,2)-hdorg(2),hdsorg(:,3)-hdorg(3)];

% if hdorg(1)==169; keyboard;end

%************************************************************************
%Subassembly movement

%Since the dummy is just one long core region, I decided to adjust the
%origin of the whole assembly like a control rod.

sazo=DimMap{sazoC};

%************************************************************************
%This section is the reflector origin adjustment.

%Offest plus movement
hdorg(4)=hdorg(4)-sazo;

%************************************************************************
% Duct Materials

% Duct and Cylinder

Ductm=MatMap{DuctmC};
Ductd=MatMap{DuctdC};

% Duct Fluid

Dnam=MatMap{DnamC};
Dnad=MatMap{DnadC};

% HexBlocks

Dm=MatMap{DmC};
Dd=MatMap{DdC};

% Smeared Lower Extension Materials

SLOm=MatMap{SLOmC};
SLOd=MatMap{SLOdC};

% Smeared Lower Adapter

SLCm=MatMap{LAmC};
SLCd=MatMap{LadC};

%************************************************************************

%%

%This next line performs the expansion. It moves the pins slightly outward
%from the center of the core. This will be replaced by pin section movement
%later on in PinSectionMaker but for not it will reside here as a proof of
%concept.

%porg=ThermalX(porgtemp);

%The surfaces must be written first for the duct and all the pins, then the
%cell cards.

%%
%Debug only

% save('hdorg.txt','hdorg','-ASCII')
```

```
% save('hdsorg.txt','hdsorg','-ASCII')
% save('hexsurfnum.txt','hexsurfnum','-ASCII')
% save('planesurfnum.txt','planesurfnum','-ASCII')

%% %Surface Card Creator

%Surfname puts comment lines in the input file which differentiate the sub
%assemblies

surfname=NameCard(['Reflector M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');


for ii=1:size(hdsorg,1)

     %%
    % The following for loop creates the core region sections.

    %This for loop creates the surface cards for the hex duct sections

    % Outer Duct Assembly
     ODOWs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4)-lch-hdwt,0,0,crh+lch+2*hdwt,...
     hdor, ['$ Outer Wall of Hex Duct Sec: ' Num2StrM(ii)]});

     %Assign the surface number to be used later in the cell creation
     ODOWnums(ii,1)=surfnum;

    %Update the surnum number
     surfnum=surfnum+1;

    % Inner Duct Core Region
     CRs=CharChecker({surfnum,'    RHP',hdsorg(ii,2),...
     hdsorg(ii,3),hdorg(4),0,0,crh,hdir,...
     ['$ Inner Wall Hex Duct Sec: ' Num2StrM(ii)]});

     %Assign the surface number to be used later in the cell creation
     CRnums(ii,1)=surfnum;

    %Update the surnum number
     surfnum=surfnum+1;

    surfctemp=[ODOWs; CRs];

    if ii==1

        %%
        % Lower Extension Surface Cards

        ODLPls=CharChecker({surfnum,'    PZ', hdorg(4)-hdwt,...
                        '$ Plane Separation Duct to Cylinder'});

        %I need this facet for later.
        ODLPlnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

        % Lower Cylinder Extension

        ODCyls=CharChecker({surfnum,'    RCC',hdsorg(end,2),hdsorg(end,3),...
            hdorg(4)-hdwt,0,0,-lch,lcr,' $ Lower Extension Cylinder'});

        %Assign the surface number to be used later in the cell creation
        ODCylnum=surfnum;

        %Update the surnum number
         surfnum=surfnum+1;

    end

    if ii==1

        % Append the surfaces to the surface card.
        surfc=[surfname;surfctemp;  ODLPls;  ODCyls];

    else

        surfc=[surfc; surfctemp];

    end

end
%%
% Appending the Outerduct surface numbers to the hexsurfnum variable

if isempty(hexsurfnum{1,1})

    hexsurfnum={ODOWnums};

else

    hexsurfnum=[hexsurfnum, {ODOWnums}];

end
```

```
%% Cell Card creator

%cellname puts comment lines in the input file which differentiate the sub
%assemblies

cellname=NameCard(['Reflector M#' num2str(hdorg(1))...
    ' POS:' MatMap{SAPosC} ' ID:' MatMap{SANomen}],'Title');

cellname2=NameCard(['Hex Duct, Reflector: ' num2str(hdorg(1))],'Divider');

for dd=1:size(hdsorg,1)

%% %Hex Duct Cell cards

    % The following variable will need to be changed for code that chooses
        % the assembly out of the katana planes

    if dd==1

        curPlnum=KatanaPls{dd,2};

    end

    if dd>1 && dd<size(hdsorg,1)

        TopPl=KatanaPls{dd-1,2};
        BotPl=KatanaPls{dd,2};

        curPlnum=[-TopPl,BotPl];

    end

    if dd==size(hdsorg,1)

        curPlnum=-KatanaPls{dd-1,2};

    end

    % Outter Duct Wall

    ODc=CharChecker({cellnum,'     ',Ductm,Ductd, -ODOWnums(dd,1),...
         CRnums(dd,1),curPlnum,ODLPlnum,UNIC,['$ Hex Duct Sec: ' Num2StrM(dd)]});

     cellnum=cellnum+1;

    % Hex Blocks

    HBc=CharChecker({cellnum,'     ',Dm,Dd, -CRnums(dd,1),curPlnum,...
      UNIC,['$ Hex Block Sec: ' Num2StrM(dd)]});

    cellnum=cellnum+1;

    if dd==1

        %Sodium around the lower extension cylinder

        CylNac=CharChecker({cellnum,'     ',Dnam,Dnad, -ODOWnums(end,1),...
            -ODLPlnum,ODCylnum,UNIC,'$ Na Surr Lower Ext'});

        cellnum=cellnum+1;

        % Lower Cylinder

        Cylc=CharChecker({cellnum,'     ',SLCm,SLCd, -ODCylnum,...
            UNIC,'$ Lower Cyn Homog'});

        cellnum=cellnum+1;

        %Appends the upper lower extensions and Core Region cards

        cellctemp=[ODc; HBc; CylNac; Cylc];

        cellc=[cellname; cellname2; cellctemp];

    else

        cellctemp=[ODc; HBc];

        cellc=[cellc; cellctemp];

    end

end

%%
global SaveSAMakerVars

if SaveSAMakerVars

    % Save TotDimDat
    save(['Debug\SAmakerVar\Tot_' MatMap{SAPosC} '_Reflector_Dat.mat'])
    dispPrint(['     TotDat_' MatMap{SAPosC} '_saved....']);
```

```
end

end
```

B.70. SAlphaBeta.m

```
function [ sabout ] = SAlphaBeta( matnum, matdb, SATy )
% This function identifies materials designated to have s(a,b) cross
% sections added.

global SAlphaBetas
persistent firstrun
persistent SATYsab
global xcrossset


if isempty(firstrun);

% s(a,b) additions

    SATYsab=cell2mat(SAlphaBetas(:,2));

end

sabout=0;
foundind=1;

if ~isempty(matdb)

    ZAIDSin=cell2mat(matdb(1,:));

    [sabrow,sabcol]=find(SATy==SATYsab);

    if ~isempty(sabrow)

        ZAIDS=SAlphaBetas{sabrow,3};

        for gg=1:size(ZAIDS,1)

            curzaidn=str2num(ZAIDS{gg,1});

            curzaids=ZAIDS{gg,1};

            found=sum(ZAIDSin==curzaidn);

            if found~=0

                sab=ZAIDtoSAB(curzaids);

                if foundind==1

                    sabout={['mt' Num2StrM(matnum)],sab};
                    foundind=foundind+1;

                else

                    PrevSAB=sabout{1,2};
                    sabout{1,2}=[PrevSAB ' ' sab];

                end

            end

        end


    end

end



end

function [mcnpsabform]=ZAIDtoSAB(zaidin)

global sabxcrossset
persistent Carbon
persistent Iron
global SABSet

if isempty(sabxcrossset);

    sabxcrossset=cell(1,1);
    Carbon=false;
    Iron=false;

end

switch zaidin

%     case '6000'
%
```

```
%         mcnpsabform='grph.10t';
%
%         if ~Carbon
%             sabxcrossset=[sabxcrossset;'.10t'];
%         end
%
%         Carbon=true;

    case '26056'

        if SABSet==1

            mcnpsabform='fe56.12t';

            if ~Iron
                sabxcrossset=[sabxcrossset;'.12t'];
            end

        end

        if SABSet==2

            mcnpsabform='fe56.22t';

            if ~Iron
                sabxcrossset=[sabxcrossset;'.22t'];
            end

        end


        if SABSet==3

            mcnpsabform='fe56.60t';

            if ~Iron
                sabxcrossset=[sabxcrossset;'.60t'];
            end

        end

        Iron=true;

end

end
```

B.71. DelNan.m

```
function [ cellin ] = delNaN( cellin, ~ , type )
% takes in an arbitrary cell array and removes the NaN's if they exist.

if isstruct(cellin)

    StrcTx=cellin.textdata;
    StrcDattemp=cellin.data;

    StrcDat=num2cell(StrcDattemp);

    colshift=abs(size(StrcTx,2)-size(StrcDat,2));
    rowshift=abs(size(StrcTx,1)-size(StrcDat,1));

    %Need to find any straggeling text and put it in the data part

    txlocar=cellfun(@isempty,(StrcTx(1+rowshift:end,1+colshift:end)));

    [txlocrow,txloccol]=find(txlocar==0);

    for kk=1:length(txlocrow)

        StrcDat(txlocrow(kk),txloccol(kk))=...
            StrcTx(rowshift+txlocrow(kk),colshift+txloccol(kk));

    end

    StrcTx(1+rowshift:end,1+colshift:end)=StrcDat;

    cellin=StrcTx;

end

stryes=false;
numyes=false;

if strcmp(type,'str')

    stryes=true;

end

if strcmp(type,'num')

    numyes=true;
```

```
end

[cellrow,cellcol]=size(cellin);


for ii=1:cellrow

    for ll=1:cellcol

        cellt=cellin{ii,ll};

        if isnan(cellt)

            cellin{ii,ll}=[];

        else

            if stryes

                cellin{ii,ll}=num2str(cellt);

            end

            if numyes

                numtemp=str2double(cellt);

                if ~isnan(numtemp)

                    cellin{ii,ll}=numtemp;

                end

            end

        end

    end


end


end
```

B.72. CoreMaker.m

```
function [ cellc, surfc ] = CoreMaker( hdp,nsa,DimMap,ReactMap,MatMap)
%This function writes the cards for the core

global debugMICKA
global hexsurfnum
global surfnum
global cellnum
global planesurfnum
global LAT
global pax
global pax2
global MatLabOld
global BorD KatanaPls

%% Dim Vars breakout
% This section breaks out all of the variables into seperate variables.

global hdodC bhC crhC uehC
global ihdodC sazoC saatC

%% Material Vars Breakout

global FmC FdC PmC PdC ShieldPmC ShieldPdC

%*************************************************************************

%Lat Universe number assignment

LUN=nsa+1;

LUNu=strjoin({'U=' num2str(LUN)},'');
LUNf=strjoin({'fill=' num2str(LUN)},'');

%*************************************************************************
%I need to ammend the parameters later

%Core Diameter and height
sd=DimMap{hdodC};
sh=DimMap{uehC};
szo=DimMap{saatC};

ch=DimMap{crhC};
czo=DimMap{sazoC};

LatEled=DimMap{ihdodC}+(2*DimMap{bhC});
```

```
LatEleh=ch+(10*2.54);
LatElezo=czo+(10*2.54);

if LAT

    hdp=LatEled;

    % This helps spread the assemblies out so that they are easier to see.

    if debugMICKA; hdp=10; LatEled=10; CorePerIncr=0; end

end

sr=sd/2;
LatEler=LatEled/2;

%Coolant material and density
%  the column assignments look strange, but this is how they are listed in
%  MatMapper. Refer to the generic materials section in matmapper right at
%  the end of the function to see which materials are being assigned here.
%  They are not what is listed here.
cwm=MatMap{FmC};
cwd=MatMap{FdC};

%%In sub assembly area

cmm=MatMap{PmC};
cmd=MatMap{PdC};

%Shield blocks

smm=MatMap{ShieldPmC};
smd=MatMap{ShieldPdC};

%*****************************************************************************
%Geometry calculations

%Number of hex duct rings.

nr=ceil((3+sqrt(12*nsa-3))/6)-1;

%Number of rings needed to get circular motif

%nar number of additional rings

nar=2;

fillc=strjoin({num2str(-(nr+nar)) ':' num2str(nr+nar)},'');

if nsa==721

   nr=nr-2;
   ExCr=hdp*0.2;

else

   ExCr=0;

end

% This is the radius of the vertex not the perdendicular
hdvr=hdp/(2*cosd(30));

%Core radius calculation

cr=nr*hdp+hdvr+ExCr;

if ~MatLabOld

    hold off
    annulus(cr,sr,0,0,0,0,pax);

    if ~debugMICKA; annulus(cr,sr,0,0,0,0,pax2); end
    drawnow

end

if ~BorD

    KatPlInd=0;
    KatPlOff=5;
    KatPlSize=size(KatanaPls,2)+1;
    KatPlIndar=linspace(KatPlOff,KatPlSize-1,2);

%     % Reassign the shield blocks to be sodium
%     smm=cmm;
%     smd=cmd;

end

if LAT==0 % Non lattice core

    %Naming card
    surfname=NameCard('Core Barrel','Title');
```

```
surfctemp={surfnum,'         RCC',0,0,-czo ...
    ,0,0,LatEleh,cr,'     $ Core Boundary'};

OutCore=surfnum;

%Write that surface number to the hexsurfarray

surfnum=surfnum+1;

surfctemp1={surfnum,'         RCC',0,0,-czo-(10*2.54) ...
    ,0,0,LatEleh+(10*2.54),sr,'     $ Shield Boundary'};

ShieldWall=surfnum;

%Write that surface number to the hexsurfarray

surfnum=surfnum+1;

surfc=[surfname; CharChecker(surfctemp); CharChecker(surfctemp1)];

%%

%Create the cell cards for the core barrel

cellname=NameCard('Core Cell Cards','Title');

for ii=1:KatPlSize

    if ~BorD

        % Katana Slicing code

        if ii==1

            curPlnum=KatanaPls{ii,2};

        end

        if ii>1 && ii<KatPlSize

            TopPl=KatanaPls{ii-1,2};
            BotPl=KatanaPls{ii,2};

            curPlnum=[-TopPl,BotPl];

        end

        if ii==KatPlSize

            curPlnum=-KatanaPls{ii-1,2};

        end

        % If this is not a sliced core, then it only runs once adding all
        % of the ODOWs

        cellname2=NameCard(['Katana Slice ' Num2StrM(ii)],'Title');

        ReactSlicec={cellnum,'       ',cwm,cwd,curPlnum};

        cellnum=cellnum+1;

    else

        cellname2=NameCard('Core Lattice Cards','Title');

        ReactSlicec={cellnum,'       ',cwm,cwd};

        cellnum=cellnum+1;

    end

    for kk=1:size(hexsurfnum,2)

        CurSA=hexsurfnum{kk}(ii);

        if length(CurSA)>1

            NotCell=['#', Num2StrM(CurSA)];

            NotCell=strrep(NotCell(1:end-1),' ',' #');

            curODOWs=[Num2StrM(cell2mat(CurSA(1:2))) NotCell];

        else

            curODOWs=Num2StrM(CurSA);

        end

        ReactSlicec=[ReactSlicec,{curODOWs}];

    end

    if ~BorD
```

```
            ReactSliceImpc={-OutCore,'IMP:N=1','$ Reactor Coolant','Sec #',Num2StrM(ii)};

        else
%***************************** Benchmark *********************************

            ReactSliceImpc={-OutCore,'IMP:N=1','$ Reactor Coolant'};

%***************************** Benchmark *********************************
        end

        if ii==1

            cellc=[cellname; cellname2; CharChecker([ReactSlicec, ReactSliceImpc])];

        else

            cellc=[cellc; cellname2; CharChecker([ReactSlicec, ReactSliceImpc])];

        end

    end

    cellc=[cellc; CharChecker({cellnum,'       ',smm,smd,...
        OutCore,-ShieldWall,'IMP:N=1 $ Shield Blocks'})];

    cellnum=cellnum+1;

    cellc=[cellc; CharChecker({cellnum,'       ',0,ShieldWall,...
        'IMP:N=0 $ Out of Core'})];

    cellnum=cellnum+1;

else

%% Surface Cards for Lattice core

%Naming card
surfname=NameCard('Core Barrel','Title');

surfctemp1=CharChecker({surfnum,'    RCC',0,0,-czo ...
    ,0,0,ch,cr,'     $ Core Boundary'});

OutCore=surfnum;

%Write that surface number to the hexsurfarray

surfnum=surfnum+1;

surfctemp=CharChecker({surfnum,'    RCC',0,0,-szo ...
    ,0,0,sh,sr,'     $ Shield Boundary'});

ShieldWall=surfnum;

%Write that surface number to the hexsurfarray

surfnum=surfnum+1;

% Creates the window element for the lattice

surfctemp2=CharChecker({surfnum,'    RHP',0,0,-LatElezo,0,0,LatEleh,LatEler,...
    '$ Lattice Window'});

LatWinElenum=surfnum;

surfnum=surfnum+1;

%Append to the surface array

surfc=[surfname; surfctemp; surfctemp1; surfctemp2];

%% Cell Cards

%name the area

cellname=NameCard('Core Cell Cards','Title');

%Create the lattice fill card

cellctemp=CharChecker({cellnum,'     ',cmm,cmd,-LatWinElenum,'Lat=2',LUNu,...
    'IMP:N=1 $ SA Lattice'});

cellnum=cellnum+1;

cellctemp1=CharChecker({'      ','fill ',fillc,fillc,'0:0'});

% Latnums is an array of the universe numbers that need to be placed inside of
% the lattice.

Latnums=linspace(1,nsa,nsa);

%Creates the lattice line, fills in the outside zeros with the lat universe
%aka the wall
[LATarln,LATar]=MCNPPOSMaker( LUN, nsa ,nar, Latnums );
```

```
%Find the blanks and replace them with LUN

for kk=1:nsa

    SATy=ReactMap{kk+1,2};
    Mnum=ReactMap{kk+1,1};

    if SATy==0

       LATarln(LATarln==Mnum)=LUN;
       LATar(LATar==Mnum)=LUN;

    end

end

cellctemp2=CharChecker({'     ' num2str(LATarln)});


% Fills the lattice into the core.

cellctemp3=CharChecker({cellnum,'     ',cmm,cmd,-OutCore,LUNf,...
    'IMP:N=1 $ SA Lattice'});

cellnum=cellnum+1;

% Shield blocks

cellctemp4=CharChecker({cellnum,'     ',smm,smd,OutCore,-ShieldWall,...
    'IMP:N=1 $ PoolShield Blocks'});

cellnum=cellnum+1;

% Outside of the problem

cellctemp5=CharChecker({cellnum,'     ','0',ShieldWall,...
    'IMP:N=0 $ Oblivion'});

cellnum=cellnum+1;

cellc=[cellname; cellctemp; cellctemp1; cellctemp2; cellctemp3; cellctemp4;...
     cellctemp5];

end

%%
global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotCoreMakerDat.mat')
    dispPrint('     TotReactDat saved....');

end


end
```

B.73. MCNPStart.m

```
function [  ] = MCNPStart( MCNPpara, FeaturesM, cellc, surfc, datac, ProbSet )
% This is a wrapper function to perform intial MCNP setup and execution

global Title debugMICKA

if strcmp(MCNPpara{5},'N') && strcmp(MCNPpara{6},'N') && strcmp('N',FeaturesM{1})

    if debugMICKA

        fileID = fopen('Debug\DebugMode.i','w+');
        Title='Debug';

    else

    %fileID is the Matlab variable that associates the opened file to be
    %written

    fileID = fopen(['RunFolders\InputFiles\' Title '.i'],'w+');

    end

    CardWriter(cellc,surfc,datac,fileID,Title);

    dispPrint(' ')

    fclose(fileID);

    dispPrint('Input File Created')

    fclose('all');

else
```

```
    InputFile=[{cellc};{surfc};{datac};{Title}];

    [keff, keffstd]=MCNPConfig(Title,MCNPpara,FeaturesM,InputFile,ProbSet);


    if isnumeric(keff)

            dispPrint(['Keff = ' num2str(keff)])
            dispPrint([' std = ' num2str(keffstd)])

    else

            dispPrint(keff);
            dispPrint(keffstd);

    end


    fclose('all');

end

end
```

B.74. CardWriter.m

```
function [ ] = CardWriter( cellc,surfc,datac,fileID,Title )
%This program writes the cell surface and data cards to a text file.

global comout
global planesurfnum

%Before this program does anything, it needs to know how many cards there
%are in surface cell and data cards.

ccs=size(cellc);
scs=size(surfc);
dcs=size(datac);

%The first card is the title card with the data attached

NewTitle=[Title ' '  datestr(datetime('now'))];

fscct='%s\r\n';
fprintf(fileID,fscct,NewTitle);

%%
%The Cell cards

%Give the user something to look at.

dispPrint(' ')

%The following for loop is for all of the rest of the cell cards

for ii=1:ccs(1,1)

   dispPrint(['Writing Cell Cards.....';{[ii,ccs(1,1)]}]);

   fscca='%i %s\r\n';

   if(iscellstr(cellc(ii,1)))

       fscca='%s %s\r\n';

   end

   try

       fprintf(fileID,fscca,cellc{ii,:});

   catch

       dispPrint('Random writing error claiming it cant write a cell.')
       dispPrint('This usually appears from something strange in the description block.')
       keyboard;

   end
end

%%
%Surface cards

dispPrint(' ')
%Title

fssct='\r\n%s\r\n';
fprintf(fileID,fssct,'c Surface Cards');

%Now for the cards
```

```
for kk=1:scs(1,1)

    dispPrint(['Writing Surface Cards.....';{[kk,scs(1,1)]}]);

    fscca='%i %s\r\n';

   if(iscellstr(surfc(kk,1)))

       fscca='%s %s\r\n';

   end

   fprintf(fileID,fscca,surfc{kk,:});

end

%%
%Data Cards

dispPrint(' ')

%Title

fsdct='\r\n%s\r\n';
fprintf(fileID,fsdct,'c Data Cards');

fsdca='%s %s\r\n';

for mm=1:dcs(1,1)

    dispPrint(['Writing Data Cards.....';{[mm,dcs(1,1)]}]);

   fprintf(fileID,fsdca,datac{mm,:});

end

end
```

B.75. MCNPConfig.m

```
function [keff,keffstd]=MCNPConfig(filename,MCNPpara,FeaturesM,InputFile,ProbSet)

global DelTemp
global FindOldLib
global CtoKconv
global RoomTemp

% Turn off text syntax warning

warning('off','MATLAB:handle_graphics:exceptions:SceneNode')

recycle('on');

keff='MCNP Was not Executed';
keffstd='MCNP Was not Executed';

disp(' ')

tasks=MCNPpara{1};
mcnpst=MCNPpara{6};
MCNPplot=MCNPpara{7};

cellc=InputFile{1};
surfc=InputFile{2};
datac=InputFile{3};
Title=InputFile{4};

TempCor=FeaturesM{1};
Temp=Num2StrM(DelTemp+CtoKconv+RoomTemp);

%% Create a run folder for the current problem.

% Create the folder

RunFoldMPath=['RunFolders\' filename ];

CurRunFold=dir('RunFolders');

ExistKind=0;

for ll=1:size(CurRunFold,1)

    CurFoldName=CurRunFold(ll,1).name;

    if strcmp(CurFoldName,filename)

        ExistKind=true;

    end

end

dispPrint(' ')
```

```
if ~ExistKind

    if strcmp(mcnpst,'Y');

        [s,message,messid]=mkdir(RunFoldMPath);
        addpath(RunFoldMPath);

    end

else

    OriginalFileName=filename;

    gg=1;

    while ExistKind

        NewFileName=[OriginalFileName  Num2StrM(gg) ];

        message='A folder with that filename already exists';
        dispPrint(message)
        dispPrint(' ')

        message=['Trying to create folder ' NewFileName];
        dispPrint(message)
        dispPrint(' ')

        for ll=1:size(CurRunFold,1)

            ExistKind=false;

            CurFoldName=CurRunFold(ll,1).name;

            if strcmp(CurFoldName,NewFileName)

                ExistKind=true;
                gg=gg+11;

            end

        end


        if ~ExistKind

            filename=NewFileName;

            RunFoldMPath=['RunFolders\' filename ];

            if strcmp(mcnpst,'Y');

            [~,message,~]=mkdir(RunFoldMPath);

            end

        end

    end

end

dispPrint(['Folder ' filename ' created.'])

%% Load the computer type and MCNP varaibles

ManSet=false;

[ thpath, libPath, PATHMc, DATAPATHMc, MCver, tasks, runmcfind, MICKAPath, hostname, ModelDev] =...
    MCNPCompSetup( tasks,ProbSet,ManSet );

if ModelDev

    mcnpst='N';

end

InCompthpath=[thpath '\InCompleteRuns'];

if runmcfind==1

    [ PATHMc, DATAPATHMc ] = FindMCNP(  )

end

%% Sets the Folder Paths

RunFoldPath=[InCompthpath '\' filename];

%% Temperature correction

MCverTemp=MCver;

% Save the datac variable to be used later for library creation
```

```
dispPrint('Exporting datac');
dispPrint(' ')
delete('Specs\MCNP\datac.mat');
save('Specs\MCNP\datac.mat','datac');

if TempCor

    if strcmp(MCver,'mcnp611')

        MCverTemp=MCver;

        MCver='mcnp6';

    end

    datacbeforetempadj=datac;

    [ datac, xsdir ] = XSecAdj( filename, datac, Temp,...
        RunFoldPath, RunFoldMPath, MCver, PATHMc, DATAPATHMc, libPath, FindOldLib );

    MCver=MCverTemp;

end

%% Creates the Batch file to run mcnp

% Set where a set of runs are being created.

RunSet=true;

line0={['cd /d ' RunFoldPath]};
line1={['@ set MCNPPATH=' PATHMc]};
line2={'@ PATH %MCNPPATH%;%PATH%'};

line3={'@ set DISPLAY=:0.'};
line4={['@ set DATAPATH=' DATAPATHMc]};

% Remove the .i from the inputfile

line5={['copy /Y ' filename '.i ' filename '.']};

%Now for the execution lines

if TempCor && strcmp(mcnpst,'Y')

    mcnp6lninit={[MCver ' ix n='  filename '. x=' xsdir  ' tasks 1']};
    mcnp6ln={[MCver  ' c'...
                ' o=' filename '.o'...
                ' r=' filename '.r'...
                ' s=' filename '.s'...
                ' mesh=' filename '.imsht'...
                ' mctal=' filename '.mctal'...
                ' x=' xsdir  ' tasks ' tasks]};

else

    mcnp6lninit={[MCver ' ix n='  filename '. tasks 1']};
    mcnp6ln={[MCver  ' c'...
                ' o=' filename '.o'...
                ' r=' filename '.r'...
                ' s=' filename '.s'...
                ' tasks ' tasks]};

end

copyflln0={['copy /Y '        filename '.o '...
                              filename '.Inito']};
copyfllnr={['copy /Y '        filename '.r '...
                              filename '.Initr']};
cleanflln={['del /S /Q '      filename '.o '...
                              filename '.s ']};

if TempCor && strcmp(mcnpst,'Y')

   movedir={['move /-Y ' InCompthpath '\' filename ' ' filename]};
   cddir={'cd .. & cd ..'};

else

   movedir={' '};
   cddir={' '};

end

mcnpenv=[line0; line1; line2; line3; line4; line5; mcnp6lninit;...
         copyflln0; copyfllnr; cleanflln; mcnp6ln; cddir; movedir];

kextr='N';

%%

if mcnpst=='Y'

%% Write the Input file & Batch file to the run folder.
```

```
fileID2=fopen([RunFoldMPath '\' filename '_MCrun.bat'],'w');

fprintf(fileID2,'%s\r\n',mcnpenv{:,1});

fclose(fileID2);

dispPrint(' ')

dispPrint('Batch Execution File Created')

fileID = fopen([ RunFoldMPath '\' filename '.i'],'w+');

CardWriter(cellc,surfc,datac,fileID,Title);

fclose(fileID);

dispPrint('Input File Created')

dispPrint(' ')

dispPrint('Executing Copy...')

fclose('all');

% Supress a warning

% Copy the run folder to the new place.

[status , cmdout]=dos(['xcopy /E /I /-Y ' RunFoldMPath ' ' RunFoldPath]);

dispPrint(' ')

dispPrint(cmdout)

dispPrint(' ')

% These lines execute the mcnp input file.

status1=0;
cmdout1=0;

if ~RunSet

    dispPrint('Executing MCNP...')

    dispPrint(' ')

    [status1 , cmdout1]=system([RunFoldPath '\' filename '_MCrun.bat &'],'-echo');

    dispPrint(cmdout1)

    dispPrint(' ')

else

    ExecLn={['call InCompleteRuns\' filename '\' filename '_MCrun.bat']};

    quitChk=false;

    BatExecName=[thpath '\ProbSetExec_' hostname '.bat'];

    BatExecNameOldWoExt=BatExecName(1:(strfind(BatExecName,'.'))-1);

    ff=1;

    if ~mislocked('MICKALock');

        [ isFirstRun ] = MICKALock(  );

    else

        isFirstRun=false;

    end

%     The following code is meant to check for a previous iteration of the
%     problem set execution batch file

    while ~quitChk

         if exist(BatExecName, 'file') == 2

            if isFirstRun;

               BatExecName=[BatExecNameOldWoExt '_' Num2StrM(ff) '.bat'];

               ff=ff+1;

            else

                quitChk=true;

            end

         else
```

```
                quitChk=true;

          end

    end

    fileID3=fopen(BatExecName,'a');

    fprintf(fileID3,'%s\r\n',ExecLn{:,1});

    fclose(fileID3);

end

dispPrint('Copying Maps to the Run Folder')

dispPrint(' ')

% Copy the Maps to the run folder.

[status , cmdout]=dos(['xcopy /E /I /-Y Maps' ' ' RunFoldPath '\Maps']);

dispPrint(cmdout)

dispPrint(' ')

% Move the material data to a new folder

dispPrint('Copying Material Vars to the Vars folder')

dispPrint(' ')

% Copy the Maps to the run folder.

[status , cmdout]=dos(['move /Y "' MICKAPath '\Maps\' filename 'TotMatDat.mat" "' MICKAPath '\Vars\' filename
'TotMatDat.mat"']);

dispPrint(' ')

dispPrint(cmdout)

dispPrint(' ')

%%

    if (status==0)||(status1==0)

        dispPrint(' ')
        dispPrint('File Execution Sucessful')
        dispPrint(' ')

        keff='MCNP Was Executed';
        keffstd='MCNP Was Executed';

    end

    if (status==1)||(status1==1)

        dispPrint('File Execution Unsucessful')

        save('ExeError.txt','cmdout','cmdout1','-ascii')

        keff='MCNP Was not Executed';
        keffstd='MCNP Was not Executed';

    end

    if strcmp(kextr,'Y')

        % this part finds the keff and STD value.

        dispPrint('Extracting Keff and Standard Deviation....')

        kind=strfind(output,'final k(col/abs/trk len)');

        keff=str2double(output((kind+27):(kind+27+6)));

        kstdind=strfind(output,'std dev = ');

        keffstd=str2double(output((kstdind+10):(kstdind+10+6)));

        dispPrint(' ')
        dispPrint(['Name: ' filename '    keff = ' num2str(keff)...
            '    std dev = ' num2str(keffstd)])
        dispPrint(' ')

    end


else

% if no execution is selected then just write the input file and batch
% file to the input folder.
```

```
% Redirect RunFoldPath

RunFoldPath=[MICKAPath '\RunFolders\' filename];

fileID2=fopen([RunFoldPath '\' filename '_MCrun.bat'],'w');

fprintf(fileID2,'%s\r\n',mcnpenv{:,1});

fclose(fileID2);

dispPrint(' ')

dispPrint('Batch Execution File Created')

fileID = fopen([ RunFoldPath '\' filename '.i'],'w+');

CardWriter(cellc,surfc,datac,fileID,Title);

fclose(fileID);

dispPrint(' ')

dispPrint('Input File Created')

dispPrint(' ')

dispPrint('Executing Move...')

dispPrint('Copying Maps to the Run Folder')

dispPrint(' ')

% Copy the Maps to the run folder.

[status , cmdout]=dos(['xcopy /E /I /-Y Maps' ' ' RunFoldPath '\Maps']);

dispPrint(cmdout)

dispPrint(' ')

% Move the material data to a new folder

dispPrint('Copying Material Vars to the Vars folder')

dispPrint(' ')

% Copy the Maps to the run folder.

[status , cmdout]=dos(['move /Y "' MICKAPath '\Maps\' filename 'TotMatDat.mat" "' MICKAPath '\Vars\' filename
'TotMatDat.mat"']);

dispPrint(' ')

dispPrint(cmdout)

dispPrint(' ')

fclose('all');

end

if MCNPplot=='Y'

    dispPrint(' ')

    dispPrint('Executing Plot...')

        mcnpenv=[line0; line1; line2; line3; line4; line5];

        [ status ] = MCNPplotStart( MCver, filename, mcnpenv, RunFoldPath );

        keff='Input File Was PLotted';
        keffstd='Input File Was PLotted';

end

%% Saving variable names

global SaveOtherFuncVars

if SaveOtherFuncVars

    % Save TotDimDat
    save('Debug\TotMCNPStartDat.mat')
    dispPrint('     TotReactDat saved....');

end
```

B.76. FindMCNP.m

```
function [ PATHMc, DATAPATHMc ] = FindMCNP(  )
```

```
% This function finds mcnp on the local computer

%This commands lists the users partition
[~, output]=system('wmic logicaldisk get name');

%Grabs the drive letters
k=strfind(output, ':');

dltemp=output(k-1);

%This loops tests the letters to see if they are active.

for i=1:length(dltemp)

    [status, output]=system(['cd /D ' dltemp(i) ':']);

    if status==0

       if i==1

            driveletters=dltemp(i);

       else

            driveletters=[driveletters dltemp(i)];

       end

    end

end


%%

dispPrint('Searching your PC for MCNP...')
dispPrint(' ')

%This for loop runs system commands to search for the MCNP directory on every
%partition.

for i=1:length(driveletters)

    dispPrint(['Searching Drive ' driveletters(i) ': for MCNP.'])
    dispPrint('...Please Standby...')


    [status, PATHMc]=system(['cd \&' ...
        'cd /D ' driveletters(i) ':&cd \&' 'dir MCNP_CODE /AD /B /s ']);


    %This line removes the extra space the cmd introduces.
    PATHMc=[PATHMc(1:end-1) '\bin'];


    [status1, DATAPATHMc]=system(['cd \&' ...
        'cd /D ' driveletters(i) ':&cd \&' 'dir MCNP_DATA /AD /B /s ']);

    DATAPATHMc=DATAPATHMc(1:end-1);

    if (status==0)||(status1==0)

        dispPrint(['MCNP Directory found on drive ' driveletters(i) ':'])
        dispPrint(' ')

        break

    else

        dispPrint(['MCNP Directory NOT found on drive ' driveletters(i) ':'])
        dispPrint(' ')

    end

end

if strcmp('File Not Found',PATHMc)

    dispPrint('Could not find MCNP directory!')
    dispPrint('Katana Terminated')
    return

end

end
```

B.77. XSecAdj.m

```
function [ datacout, newDirname ] = XSecAdj( filename, datac, Temp, Runpath,...
    RunMpath, MCver, PATHMc, DATApath, libPath, FindOldLib )
%This function will call makxsf and then perform the temperature correction
%on the cross sections and then output the new cross section set to be used
%by our problem.
```

```
% The 80 char limits needs to be changed temprorarily for the xsec
% adjustment


global CharLen
global RoomTemp
global DelTemp
global CtoKconv


% AbsTemp=DelTemp+CtoKconv+RoomTemp;

%%

if strcmp(Runpath,'NULL')

    LibPath=pwd;
    cd('E:\Inputs\MCNP\Benchmark')

end

% Removal is no longer needed because these isotopes are removed before
% they enter MICKA.

ProbISO=[46110, 54136, 54138, 56138, 58140, 0];

global xcrossset
global sabxcrossset
global SAlphaBetas
global SABcorr
global SABSet

% Set the ENDF version

if strcmp(xcrossset,'.80c'); NV='N71'; end

if strcmp(xcrossset,'.70c'); NV='N7'; end

if strcmp(xcrossset,'.66c'); NV='N7'; end

if SABSet==2; SABV='SAB71'; end

if SABSet==1; SABV='SAB7'; end

if SABSet==3; SABV='SAB6'; end

oldnewxcrossset=xcrossset;
oldnewsabxcrossset=sabxcrossset;

sabfound=false;
sabfirstInd=0;
sabZAID=[];

oldsabxsecl=size(oldnewsabxcrossset,1);

newxcrossset='.00c';
newxsabcrossset='.00t';

lcrossset=strrep(oldnewxcrossset,'0','1');
hcrossset=strrep(oldnewxcrossset,'0','2');

for dd=2:size(oldnewsabxcrossset,1)

    [hsabcrossset{dd,1} lsabcrossset{dd,1}]=...
    RTtoNewTempLibs(oldnewsabxcrossset{dd,1},str2double(Temp));

end

datacin=datac(:,2);

jj=1;

%Need to preserve problematic Isotopes

if FindOldLib; RunNum=1; else RunNum=2; end

for ff=1:RunNum

    if ~FindOldLib

        for ii=1:size(datacin,1)

            dispPrint(['Finding Unique Isotopes...';{[ii,size(datacin,1)]}]);

            curln=datacin{ii,1};

            newln=strrep(curln,oldnewxcrossset,newxcrossset);

            for ff=2:oldsabxsecl

                oldsabxsec=oldnewsabxcrossset(ff,1);

                sabInd=strfind(newln,oldsabxsec);
```

```
    if ~isempty(sabInd)

        if sabfirstInd==0; FirstSAB=true; end

        sabfirstInd=sabfirstInd+1;

        sabfound=true;

        newln=cell2mat(strrep(newln,oldsabxsec,newxsabcrossset));

    end

end

datacin{ii,1}=newln;

ZaidInd=strfind(newln,newxcrossset);

if ~isempty(ZaidInd) && ~sabfound


    if ZaidInd(1)==5

        ZLen1=4;

    else

        ZLen1=5;

    end

    if length(ZaidInd)==2

        if strcmp(newln(ZaidInd(2)-5),' ')

            ZLen2=4;

        else

            ZLen2=5;

        end

    end

    if length(ZaidInd)>1

        Zaid1=str2double(newln(ZaidInd(1)-ZLen1:ZaidInd(1)-1));
        Zaid2=str2double(newln(ZaidInd(2)-ZLen2:ZaidInd(2)-1));

        if jj==1

            AllZaids=[Zaid1;Zaid2];
            jj=jj+1;

        else

            AllZaidst=[AllZaids; Zaid1; Zaid2];
            AllZaids=AllZaidst;
            jj=jj+1;

        end

    else

        Zaid1=str2double(newln(ZaidInd(1)-ZLen1:ZaidInd(1)-1));

        if jj==1

            AllZaids=Zaid1;
            jj=jj+1;

        else

            AllZaidst=[AllZaids; Zaid1];
            AllZaids=AllZaidst;
            jj=jj+1;

        end

    end


end

if sabfound

    sabspc=strfind(curln,' ');

    if iscell(sabspc); sabspc=cell2mat(sabspc); keyboard; end

    if ~isempty(sabspc); sabs=strsplit(curln,' ')'; else sabs={curln}; end

    if FirstSAB;
```

```
            sabZAID=sabs;

            FirstSAB=false;

        else

            sabZAID=[sabZAID;sabs];

        end


    end

     sabfound=false;

end

uniqZaids=unique(AllZaids,'rows','sorted');

if SABcorr

    uniqSabs=unique(sabZAID,'sorted');

end

%% Future work

% Must include a lookup table for the ACE file or something to autofind
% which cross section sets can be used based upon temp.

%%

dispPrint(' ')
dispPrint(['Found ' num2str(length(uniqZaids)) ' Unique Isotopes'])

if SABcorr

    dispPrint(['Found ' num2str(length(uniqSabs)) ' Unique s(a,b)'])

end

dispPrint(' ')

%The following loop is needed to remove problematic isotopes that wont
%broaden. These ISO are listed above

OlduniqZaids=uniqZaids;

for ww=1:(length(ProbISO)-1)

    curZAID=num2str(ProbISO(1,ww));

    for bb=1:size(datacin,1)

        curln=datacin{bb,1};

        clnI=strfind(curln,curZAID);

        if ~iscell(clnI)

            if clnI==1

                PL=length(curZAID);

                newln=strrep(curln,[curln(clnI:clnI+PL-1) newxcrossset],...
                    [curln(clnI:clnI+PL-1) oldnewxcrossset]);

                datacin{bb,1}=newln;

            end

        end

    end

end

% Change back the problematic iso to the old cross section set.

for uu=1:length(ProbISO)

    uniqZaidstemp=uniqZaids;

    ProbISOInd=find(uniqZaidstemp==ProbISO(1,uu));

    if ~isempty(ProbISOInd)

        uniqZaids=[uniqZaidstemp(1:ProbISOInd-1); uniqZaidstemp(ProbISOInd+1:end)];

        dispPrint(['Removed ZAID from Broadening ' num2str(ProbISO(1,uu))])

    end

end
```

```
datacout=[datac(:,1),datacin];

%% The following arrays create the "specs" file used for makxsf

% Need to get the run folder without a SPACE

[~,ShortPathout]=dos('for %I in (.) do echo %~sI');
ColonInd=strfind(ShortPathout,':');

ShortPath=ShortPathout(ColonInd(end)-1:end-1);

newDirname=['EBRII' NV SABV Num2StrM(Temp) 'dir'];
newLib=['EBRII' NV SABV Num2StrM(Temp) 'lib'];
newLibPath=[PATHMc(1:end-3) MCver '\bin\' newLib];

line=cell(20,1);

line(1)={'# Location of current MCNP data folder'};
line(2)={['datapath=' DATApath]};
line(3)={'#'};
line(4)={'# old-xsdir  new-xsdir'};
line(5)={['xsdir_mcnp6.1_endfb-7.1' '     ' newDirname]};
line(6)={'#'};
line(7)={'# Create New Lib Folder'};
line(8)={[ newLib ' 2']};
line(9)={'# Location of the folder to place the new file'};
line(10)={newLibPath};
line(11)={'#'};
line(12)={'# List the Zaids to be placed into the new dataset'};

% Lists the ZAIDS into the file.

lineZ=cell(length(OlduniqZaids),1);

for kk=1:length(OlduniqZaids)

   curZaid=num2str(OlduniqZaids(kk));

   lineZ(kk,1)={[curZaid oldnewxcrossset]};

end

uniqSabsl=0;

if ~isempty(sabZAID)

    lineZ=[lineZ;uniqSabs];
    uniqSabsl=size(uniqSabs,1);

    olduniqSabs=uniqSabs;

    for bb=1:size(uniqSabs)

        oldcurln=olduniqSabs{bb,1};

        ExtIND=strfind(oldcurln,'.');

        if ~isempty(ExtIND); curln=oldcurln(1,1:ExtIND-1); end

        uniqSabsnoext{bb,1}=curln;

    end



end

line(14)={'#'};
line(15)={'# New Zaid   New T  Zaid Low T   Zaid High T'};

% List new ZAIDS with temperature bounds.

lineNZ=cell(length(uniqZaids)+uniqSabsl,1);

for mm=1:length(uniqZaids)

   curZaid=num2str(uniqZaids(mm));

   lineNZ(mm,1)={[curZaid newxcrossset '    '...
                 Temp ' ' curZaid lcrossset '    ' curZaid hcrossset]};

end

if ~isempty(sabZAID)

    for uu=1:length(uniqSabsnoext)

       curZaid=uniqSabsnoext{uu};

       lineNZ(uu+mm,1)={[curZaid newxsabcrossset '    '...
                     Temp ' ' curZaid lsabcrossset{uu+1,1} '    ' curZaid hsabcrossset{uu+1,1}]};

    end

end
```

```
line(17)={' '};

% Append all the lines together

SpecsFile=[line(1:12);lineZ;line(14);line(15);lineNZ;line(16)];

%% Write the specs file and copy to the MCNP code folder

dispPrint(' ')
dispPrint('Temperature Corrections shall now be performed...')
dispPrint(' ')

fileID=fopen([RunMpath '\specs'],'w');

fprintf(fileID,'%s\r\n',SpecsFile{:,1});

fclose(fileID);

% Copy specs file over to MCNP_Code
[status , cmdout]=dos(['copy /Y ' RunMpath '\specs '...
                    PATHMc(1:end-3) MCver '\bin\specs'],'-echo');

LongPath=pwd;

cd([PATHMc(1:end-3) MCver '\bin'])

% Execute the makxsf
[status1 , cmdout1]=system('makxsf','-echo');

% Copy new dir
[status2 , cmdout2]=dos(['copy /Y ' newDirname ' "' ShortPath '\' RunMpath '\' newDirname '" '],'-echo');

% Copy new lib
[status3 , cmdout3]=dos(['copy /Y ' newLib ' "' ShortPath '\' RunMpath '\' newLib '" '],'-echo');

cd(ShortPath)

dispPrint('Cleaning temporary Files')

% Delete Old dir file
[status4 , cmdout4]=dos(['del ' PATHMc(1:end-3) MCver '\bin\' newDirname],'-echo');

% Delete Old Lib file
[status5 , cmdout5]=dos(['del ' PATHMc(1:end-3) MCver '\bin\' newLib],'-echo');

% Delete Old specs file
[status6 , cmdout6]=dos(['del ' PATHMc(1:end-3) MCver '\bin\specs'],'-echo');

% Need to open the dir file and replace the old location with the
% newlocation.

%%

fileID=fopen([RunMpath '\' newDirname],'r');

dirint=textscan(fileID,'%s','delimiter','\n');

fclose(fileID);

dirin=dirint{:};

for nn=1:size(dirin,1)

    curln=dirin{nn,1};

    dirin{nn,1}=strrep(curln,newLibPath,[Runpath '\' newLib]);

end

% New File name

newDirname=['EBRII' NV SABV Temp  'dir'];
libName=['EBRII' NV SABV Temp  'lib'];

% Copy the dir file

[status , cmdout]=...
    dos(['xcopy /E /I /-Y "' RunMpath '\' newDirname '" "' libPath '"']);

cmdoutSlashcols=strfind(cmdout,'\');

dispPrint(cmdout(cmdoutSlashcols(end)+1:end))


% Copy the lib file

[status , cmdout]=...
    dos(['xcopy /E /I /-Y "' RunMpath '\' libName '" "' libPath '"']);

dispPrint(cmdout(cmdoutSlashcols(end)+1:end))

% Rerun The forloop only for finding the library that was just
% created
```

```
        FindOldLib=true;

    else
       % This part is used if an old lib file is present. The file will be
       % opened and the dir file will be changed for the current execution
       % path. The data file will then be opened and all the .70c extensions
       % will be replaced by .00c

        % Copy the dir file to the new location

        % Output the dir/lib name

        newDirname=['EBRII' NV SABV Temp  'dir'];
        libName=['EBRII' NV SABV Temp  'lib'];

        % Copy the dir file

        [status , cmdout]=...
            dos(['xcopy /E /I /-Y "' libPath '\' newDirname '" ' RunMpath]);

        cmdoutSlashcols=strfind(cmdout,'\');

        dispPrint(cmdout(cmdoutSlashcols(end)+1:end))

        % Copy the lib file

        [status , cmdout]=...
            dos(['xcopy /E /I /-Y "' libPath '\' libName '" ' RunMpath]);

        dispPrint(cmdout(cmdoutSlashcols(end)+1:end))

        for ii=1:size(datacin,1)

            dispPrint(['Changing Xsec set... ';{[ii,size(datacin,1)]}]);

            curln=datacin{ii,1};

            newln=strrep(curln,oldnewxcrossset,newxcrossset);

            datacin{ii,1}=newln;

        end

        dispPrint(' ')

        for jj=2:size(oldnewsabxcrossset,1)

            for kk=1:size(datacin,1)

                dispPrint([['Changing s(a,b) set...' Num2StrM(jj)];{[kk,size(datacin,1)]}]);

                curln=datacin{kk,1};

                newln=strrep(curln,oldnewsabxcrossset{jj},newxsabcrossset);

                datacin{kk,1}=newln;

            end

        end

        datacout=[datac(:,1),datacin];

        if strcmp(Runpath,'NULL')

            cd(LibPath);

        end

    end

end

end

function [hxcrossset, lxcrossset]=RTtoNewTempLibs(crossset, temp)

switch crossset
    case '.80c'
        breakpoints=[293.6;600;900;1200;2500];
        bpxset={'.80c';'.81c';'.82c';'.83c';'.84c'};
    case '.70c'
        breakpoints=[293.6;600;900;1200;2500];
        bpxset={'.70c';'.71c';'.72c';'.73c';'.74c'};
    case '.12t'
        breakpoints=[293.6;400;600;800];
        bpxset={'.12t';'.13t';'.14t';'.15t'};
    case '.22t'
        breakpoints=[20;80;293.6;400;600;800];
        bpxset={'.20t';'.21t';'22t';'.23t';'.24t';'.25t'};
    case '.20t'
        breakpoints=[293.6;400;500;600;700;800;1000;1200;1600];
        bpxset={'.20t';'.21t';'22t';'.23t';'.24t';'.25t';'.26t';'.27t';'.28t'};
    case '.60t'
        breakpoints=[294;400;600;800];
```

```
        bpxset={'.60t';'.61t';'.62t';'.63t';'.64t';'.65t'};
    case '.66c'
        breakpoints=[77;293.6;3000.1];
        bpxset={'.64c';'.66c';'.65c'};
end

[newxsetsl,~]=max(breakpoints(breakpoints<temp));
[newxsetsh, ~]=min(breakpoints(breakpoints>=temp));
lxcrossset=bpxset{breakpoints==newxsetsl};
hxcrossset=bpxset{breakpoints==newxsetsh};

end
```

B.78. ElementPlotter.m

```
function [  ] = ElementPlotter(  )
% This plot creates a 3D elements from FEA

clear all

NDist1=1;
NDist2=2*NDist1;

Vtcolor='blue';
Ndcolor='black';

FaceTp=.5;

DistortFactX=NDist1*0;
DistortFactY=NDist1*0;
DistortFactZ=NDist1*0;
%% 20  node Brick element

%% Face 1 nodes

VT1=[0 0 0];
VT2=[0 0 NDist2];
VT3=[NDist2 0 NDist2];
VT4=[NDist2 0 0];

MVT1=[0 0 NDist1];
MVT2=[NDist1 0 NDist2];
MVT3=[NDist2 0 NDist1];
MVT4=[NDist1 0 0];

%% Face 2 nodes

VT5=[0 NDist2 0];
VT6=[0 NDist2 NDist2];
VT7=[NDist2 NDist2 NDist2];
VT8=[NDist2 NDist2 0];

MVT5=[0 NDist2 NDist1];
MVT6=[NDist1 NDist2 NDist2];
MVT7=[NDist2 NDist2 NDist1];
MVT8=[NDist1 NDist2 0];

%% Mid Face nodes

MVT9=[0+DistortFactX NDist1+DistortFactY 0+DistortFactZ];
MVT10=[0+DistortFactX NDist1+DistortFactY NDist2+DistortFactZ];
MVT11=[NDist2+DistortFactX NDist1+DistortFactY NDist2+DistortFactZ];
MVT12=[NDist2+DistortFactX NDist1+DistortFactY 0+DistortFactZ];

%% Create Figure

f1=figure;

EleAx=axes('parent',f1);

axis(EleAx,'normal');

view(3)

grid on

hold(EleAx,'on')

axis on

% Axis lengths

xlim(EleAx,[0-NDist1,NDist2+NDist1]);
ylim(EleAx,[0-NDist1,NDist2+NDist1]);
zlim(EleAx,[0-NDist1,NDist2+NDist1]);

% %% Plot Brick Element

% Face 1

Face1=[VT1;VT2;VT3;VT4];

patch(EleAx,Face1(:,1),Face1(:,2),Face1(:,3),Vtcolor,'FaceAlpha',FaceTp)
```

```
% Face 2

Face2=[VT1;VT2;VT6;VT5];

patch(EleAx,Face2(:,1),Face2(:,2),Face2(:,3),Vtcolor,'FaceAlpha',FaceTp);

% Face 3

Face3=[VT3;VT4;VT8;VT7];

patch(EleAx,Face3(:,1),Face3(:,2),Face3(:,3),Vtcolor,'FaceAlpha',FaceTp);

% Face 4

Face4=[VT5;VT6;VT7;VT8];

patch(EleAx,Face4(:,1),Face4(:,2),Face4(:,3),Vtcolor,'FaceAlpha',FaceTp);

% Face 5

Face5=[VT2;VT3;VT7;VT6];

patch(EleAx,Face5(:,1),Face5(:,2),Face5(:,3),Vtcolor,'FaceAlpha',FaceTp);

% Face 6

Face6=[VT1;VT4;VT8;VT5];

patch(EleAx,Face6(:,1),Face6(:,2),Face6(:,3),Vtcolor,'FaceAlpha',FaceTp);

%% Node Plotter

Nodes=[VT1;VT2;VT3;VT4;VT5;VT6;VT7;VT8;...
    MVT1;MVT2;MVT3;MVT4;MVT5;MVT6;MVT7;MVT8;...
    MVT9;MVT10;MVT11;MVT12];

scatter3(EleAx,Nodes(:,1),Nodes(:,2),Nodes(:,3),Ndcolor,'filled');

%% Text annotation

text(EleAx,VT1(1),VT1(2),VT1(3),'J','VerticalAlignment','bottom','FontSize',16)
text(EleAx,VT2(1),VT2(2),VT2(3),'N','VerticalAlignment','top','FontSize',16)
text(EleAx,VT3(1),VT3(2),VT3(3),'O','VerticalAlignment','top','FontSize',16)
text(EleAx,VT4(1),VT4(2),VT4(3),'K','VerticalAlignment','bottom','FontSize',16)
text(EleAx,VT5(1),VT5(2),VT5(3),'I','VerticalAlignment','top','FontSize',16)
text(EleAx,VT6(1),VT6(2),VT6(3),'M','VerticalAlignment','bottom','FontSize',16)
text(EleAx,VT7(1),VT7(2),VT7(3),'P','VerticalAlignment','bottom','FontSize',16)
text(EleAx,VT8(1),VT8(2),VT8(3),'L','VerticalAlignment','top','FontSize',16)

text(EleAx,MVT1(1),MVT1(2),MVT1(3),'Z','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT2(1),MVT2(2),MVT2(3),'V','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT3(1),MVT3(2),MVT3(3),'A','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT4(1),MVT4(2),MVT4(3),'R','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT5(1),MVT5(2),MVT5(3),'Y','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT6(1),MVT6(2),MVT6(3),'X','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT7(1),MVT7(2),MVT7(3),'B','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT8(1),MVT8(2),MVT8(3),'T','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT9(1),MVT9(2),MVT9(3),'Q','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT10(1),MVT10(2),MVT10(3),'U','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT11(1),MVT11(2),MVT11(3),'Y','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT12(1),MVT12(2),MVT12(3),'S','VerticalAlignment','bottom','FontSize',16)

keyboard
%% 10 node Tetrahedron

% Plot Options
axis(EleAx,'equal');

axis off
% Axis lengths

xlim(EleAx,[-1.5,2]);
ylim(EleAx,[-2,2]);
zlim(EleAx,[-1,2]);

%% Nodes

VT1=[sqrt(8/9) 0 (-1/3)]*NDist2;
VT2=[-sqrt(2/9) sqrt(2/3) (-1/3)]*NDist2;
VT3=[-sqrt(2/9) -sqrt(2/3)  (-1/3)]*NDist2;

% Mid nodes

MVT1=[(sqrt(8/9)+sqrt(2/9))*0.5-sqrt(2/9) sqrt(2/3)*0.5 (-1/3)]*NDist2;
MVT2=[-sqrt(2/9) 0 (-1/3)]*NDist2;
MVT3=[(sqrt(8/9)+sqrt(2/9))*0.5-sqrt(2/9) -sqrt(2/3)*0.5  (-1/3)]*NDist2;

MVT4=[sqrt(8/9)*0.5 0 (-1/3)+(2/3)]*NDist2;
MVT5=[-sqrt(2/9)*0.5 sqrt(2/3)*0.5 (-1/3)+(2/3)]*NDist2;
MVT6=[-sqrt(2/9)*0.5 -sqrt(2/3)*0.5  (-1/3)+(2/3)]*NDist2;

% Top

VT4=[0 0 1]*NDist2;
```

```
%% Faces

% Base Face

Face1=[VT1;VT2;VT3];

patch(EleAx,Face1(:,1),Face1(:,2),Face1(:,3),Vtcolor,'FaceAlpha',FaceTp)

% Side 1

Face2=[VT1;VT2;VT4];

patch(EleAx,Face2(:,1),Face2(:,2),Face2(:,3),Vtcolor,'FaceAlpha',FaceTp)

% Side 2

Face3=[VT1;VT3;VT4];

patch(EleAx,Face3(:,1),Face3(:,2),Face3(:,3),Vtcolor,'FaceAlpha',FaceTp)

% Side 3

Face4=[VT2;VT3;VT4];

patch(EleAx,Face4(:,1),Face4(:,2),Face4(:,3),Vtcolor,'FaceAlpha',FaceTp)

%% Mid Nodes

Nodes=[VT1;VT2;VT3;VT4;...
    MVT1;MVT2;MVT3;MVT4;MVT5;MVT6];

scatter3(EleAx,Nodes(:,1),Nodes(:,2),Nodes(:,3),Ndcolor,'filled');

%% Annotations

text(EleAx,VT1(1),VT1(2),VT1(3),'K','VerticalAlignment','bottom','FontSize',16)
text(EleAx,VT2(1),VT2(2),VT2(3),'I','VerticalAlignment','top','FontSize',16)
text(EleAx,VT3(1),VT3(2),VT3(3),'J','VerticalAlignment','top','FontSize',16)
text(EleAx,VT4(1),VT4(2),VT4(3),'L','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT1(1),MVT1(2),MVT1(3),'O','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT2(1),MVT2(2),MVT2(3),'M','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT3(1),MVT3(2),MVT3(3),'N','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT4(1),MVT4(2),MVT4(3),'R','VerticalAlignment','bottom','FontSize',16)
text(EleAx,MVT5(1),MVT5(2),MVT5(3),'P','VerticalAlignment','top','FontSize',16)
text(EleAx,MVT6(1),MVT6(2),MVT6(3),'Q','VerticalAlignment','bottom','FontSize',16)

end
```

B.79. PowerPlotter.m

```
function [  ] = PowerPlotter(  )
% this function plots the japanese power data.

% read in the power data and the reactor map.

PowerMapPath='E:\Cloud Data\Dropbox\Thesis\Codes\matlab\EBRII\MICKA\Main\Katana\ANSYS\ANSYS Data';

[PowerMapN,PowerMapT]=xlsread([PowerMapPath '\PowerData.xlsx']);

PowerMapT(2:end,1:2)=num2cell(PowerMapN);

ReactMap=PowerMapT(:,1:end-1);

PowerData=PowerMapT(:,end);

% create the hdorg

nsa=127;
hdp=5.8;

[ hdorg,~ ] = HexOrg( nsa,hdp,ReactMap );

for ii=1:size(hdorg,1)

    InfoPlot( hdorg(ii,:),ReactMap(ii+1,:),hdp,PowerData{ii+1,:})

    drawnow limitrate

end

end
```

B.80. CoreLatMaker.m

```
function [ cellc, surfc ] = CoreLatMaker( nsa ,MatMap,DimMap)
%This function writes the cards for the core

global hexsurfnum
global surfnum
global cellnum
```

```
global planesurfnum

 %% Dim Vars breakout
% This section breaks out all of the variables into seperate variables.

global hdodC crhC sazoC

%************************************************************************
%Sets whether this is benchmark or Dissertation

%BorD=1 means benchmark and BorD=0 means dissertation

BorDCor=0;

if length(planesurfnum)==1

BorDCor=1;

end

%I need to ammend the parameters later

%Core Diameter and height
cd=DimMap{hdodC};
ch=DimMap{crhC};
czo=DimMap{sazoC};

cr=cd/2;

%Coolant material and density

cwm=2;
cwd=-0.968;

%Naming card
surfname=NameCard('Core Barrel','Title');

surfctemp={surfnum,'         RCC',0,0,czo ...
    ,0,0,ch,cr,'     $_Core_Barrel'};

surf1=surfnum;

%Write that surface number to the hexsurfarray

surfnum=surfnum+1;

surfc=[surfname; CharChecker(surfctemp)];

%%

%Create the cell cards for the core barrel

cellname=NameCard('Core Cell Cards','Title');


for ii=1:size(hexsurfnum,2)

    cellname2=NameCard(['Bulk Na for core Sec:' num2str(ii)],'Divider');

    %Top Section

    if ii==1

        cellctemp1={cellnum,'       ',cwm,cwd};

        cellnum=cellnum+1;

        for kk=2:size(hexsurfnum,1)

            if kk==2

               cellctemp2=[cellctemp1 hexsurfnum(kk,ii)];

            else

               cellctemp2=[cellctemp2 hexsurfnum(kk,ii)];

            end

        end

        if BorDCor==0

            cellctemp3={-surf1,planesurfnum(ii),...
                'IMP:N=1','$ Core Coolant','Sec #',num2str(ii)};

        else
%***************************** Benchmark *********************************

            cellctemp3={-surf1,'IMP:N=1','$ Core Coolant'};

%***************************** Benchmark *********************************
        end
```

```
        cellctemp=CharChecker([cellctemp2, cellctemp3]);

        cellc=[cellname; cellname2; cellctemp];

    end

    %Middle Sections

    if ii ~= 1 && ii ~= size(hexsurfnum,2)

            cellname2=NameCard(['Bulk Na for core Sec:' num2str(ii)],'Divider');

            cellctemp1={cellnum,'       ',cwm,cwd};

            cellnum=cellnum+1;

            for kk=2:size(hexsurfnum,1)

                if kk==2

                   cellctemp2=[cellctemp1 hexsurfnum(kk,ii)];

                else

                   cellctemp2=[cellctemp2 hexsurfnum(kk,ii)];

                end

            end

            cellctemp3={-surf1,-planesurfnum(ii-1),planesurfnum(ii)...
                'IMP:N=1','$ Core Coolant','Sec #',num2str(ii)};

            cellctemp=CharChecker([cellctemp2, cellctemp3]);

            cellc=[cellc; cellname2; cellctemp];

    end

    %Bottom Section.

    if ii ~= 1 && ii==size(hexsurfnum,2)

        cellctemp1={cellnum,'       ',cwm,cwd};

        cellnum=cellnum+1;

        for kk=2:size(hexsurfnum,1)

            if kk==2

               cellctemp2=[cellctemp1 hexsurfnum(kk,ii)];

            else

               cellctemp2=[cellctemp2 hexsurfnum(kk,ii)];

            end

        end

        cellctemp3={-surf1,-planesurfnum(ii-1),...
            'IMP:N=1','$ Core Coolant','Sec #',num2str(ii)};

        cellctemp=CharChecker([cellctemp2, cellctemp3]);

        cellc=[cellc; cellname2; cellctemp];

    end


end

cellc=[cellc; CharChecker({cellnum,'       ',0,surf1,'IMP:N=0 $ Outside of EBR-II'})];

cellnum=cellnum+1;

end
```

B.81. FindInFile.m

```
function [ LineNum, CharNum ] = FindInFile( fileID, VarIn )
% This function will find a specific string inside of a file and then
% output the line number and colume that is the beingging of that string.

ii=1;

FirstRun=true;

while ~feof(fileID)

    CurLn=fgetl(fileID);
```

```
    StrCol=strfind(CurLn,VarIn);

    if ~isempty(StrCol)

        CurLn

        if FirstRun

            LineNum=ii; CharNum=StrCol;

            FirstRun=false;

        else

            LineNum(end+1)=ii; CharNum(end+1)=StrCol;

        end

    end

    ii=ii+1;

end

frewind(fileID)
```

B.82. VectorRotation.m

```
function [ VecIn, POSIn, VecOut, POSOut ] = VectorRotation( AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, Vu, Vv, Vw,
AngD,Sigs )
% This program performs a vector rotation about an axis.
% Ax is the axis of rotation
% V is the point being rotated

% Copy all of the incoming data.

global MagAx

VecScale=MagAx*0.2;

% Readjust the vector to scale with size

Vu=Vu*VecScale;
Vv=Vv*VecScale;
Vw=Vw*VecScale;

AXuOld=AXu;
AXvOld=AXv;
AXzOld=AXw;
AngDOld=AngD;

VxOld=Vx;
VyOld=Vy;
VzOld=Vz;

VecIn=round([Vu, Vv, Vw]/VecScale,Sigs,'significant');
POSIn=round([Vx, Vy, Vz],Sigs,'significant');

[Ln1, Ln2, Ln3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD);

% Need more points to make a nice curve

[QP1, QP2, QP3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD/4);
[HP1, HP2, HP3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD/2);
[TQP1, TQP2, TQP3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, (AngD/2)+(AngD/4));

MaxPt=max([abs(Vu);abs(Vv);abs(Vw)]);

scale=2;
LnWidth=2;

% Old Point
quiver3(Vx , Vy , Vz, Vu , Vv , Vw,scale,'k','filled','LineWidth',LnWidth)

% % Half Point
% quiver3(HP1 , HP2 , HP3, Vu , Vv , Vw,scale,'k','filled','LineWidth',LnWidth)

% New Point
hold on
quiver3(Ln1 , Ln2 , Ln3, Vu , Vv, Vw,scale, 'b','filled','LineWidth',LnWidth)

% Create connecting curve

CurvePts=[[Vx ; Vy ; Vz,],...
          [QP1 ; QP2 ; QP3],...
          [HP1 ; HP2 ; HP3],...
          [TQP1 ; TQP2 ; TQP3],...
          [Ln1 ; Ln2 ; Ln3]];

TravLn=fnplt(cscvn(CurvePts(:,1:end)),'g',1);
plot3(TravLn(1,:),TravLn(2,:),TravLn(3,:),'LineStyle','--','Color',[0.16425 0.6714986 0.16425]);
```

```
% Calculate the New Vector

AXx=Ln1;
AXy=Ln2;
AXz=Ln3;

Vx=Vu+Ln1;
Vy=Vv+Ln2;
Vz=Vw+Ln3;

[Ln12, Ln22, Ln32]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD);

% Rotate the vector using the new points starting location.

quiver3(Ln1 , Ln2 , Ln3, Ln12-Ln1 , Ln22-Ln2 , Ln32-Ln3,scale,'r','filled','LineWidth',LnWidth)

% Plot Path

% Need more points to make a nice curve

[QP1, QP2, QP3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD/4);
[HP1, HP2, HP3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD/2);
[TQP1, TQP2, TQP3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, (AngD/2)+(AngD/4));

% Create connecting curve

CurvePts=[[Vx ; Vy ; Vz,],...
          [QP1 ; QP2 ; QP3],...
          [HP1 ; HP2 ; HP3],...
          [TQP1 ; TQP2 ; TQP3],...
          [Ln12 ; Ln22 ; Ln32]];

TravLn=fnplt(cscvn(CurvePts(:,1:end)),'g',1);
plot3(TravLn(1,:),TravLn(2,:),TravLn(3,:),'LineStyle','--','LineWidth',1,'Color',[0.3966 0.3017 0.3017]);

legend('Original', 'New Pt','Path Point','Path Vector', 'New Vector')

VecOut=round([Ln12-Ln1, Ln22-Ln2, Ln32-Ln3]/VecScale,Sigs,'significant');
POSOut=round([Ln1, Ln2, Ln3],Sigs,'significant');

axis('equal')

% Create 3D xis

% xlim([-MagAx MagAx]);
% ylim([-MagAx MagAx]);
% zlim([-MagAx MagAx]);

lims=axis*1.3;

xAXIS=plot3(lims(1:2),[0 0],[0 0]); % for x-axis
yAXIS=plot3([0 0],lims(3:4),[0 0]); % for y-axis
zAXIS=plot3([0 0],[0 0],lims(5:6)); % for z-axis

xAXIS.Color='black';
yAXIS.Color='black';
zAXIS.Color='black';

xlabel('X');
ylabel('Y');
zlabel('Z');

end

function [R1, R2, R3]=RotMat(AXx, AXy, AXz, AXu , AXv , AXw, Vx , Vy , Vz, AngD)

% First rotate the point within the axis
R1=(AXx*(AXv^2 + AXw^2) - AXu*(AXy*AXv + AXz*AXw - AXu*Vx - AXv*Vy - AXw*Vz))...
    *(1 - cosd(AngD)) + Vx*cosd(AngD) + (-AXz*AXv + AXy*AXw - AXw*Vy + AXv*Vz)*sind(AngD);

R2=(AXy*(AXu^2 + AXw^2) - AXv*(AXx*AXu + AXz*AXw - AXu*Vx - AXv*Vy - AXw*Vz))...
    *(1 - cosd(AngD)) + Vy*cosd(AngD) + (AXz*AXu - AXx*AXw + AXw*Vx - AXu*Vz)*sind(AngD);

R3=(AXz*(AXu^2 + AXv^2) - AXw*(AXx*AXu + AXy*AXv - AXu*Vx - AXv*Vy - AXw*Vz))...
    *(1 - cosd(AngD)) + Vz*cosd(AngD) + (-AXy*AXu + AXx*AXv - AXv*Vx - AXu*Vy)*sind(AngD);
```

B.83. FluxPlotter.m

```
function [ ] = FluxPlotter(  )

global MatLabOld
global debugMICKA
global SAtocol

 SAtocol=[1,[1 0 0];...  % Driver     (Red)
     2,[0.75 0 0];...    % DriverHFW  (Light Red)
     3,[1 0 1];...       % Safety     (Magenta)
     4,[0 1 0];...       % HWCR       (Green)
     5,[1 1 0];...       % Control    (Yellow)
     6,[0 0 0.75];...    % Dummy      (Light Blue)
```

```
     7,[0 0 1];...       % Reflector  (Blue)
     8,[0 1 1];...       % Blanket    (Cyan)
     9,[0 0 1];...       % Spare      (Blue)
     10,[0.5 0.5 0];...  % Experiment (Light Yellow)
     11,[0 0 0];];       % Wall       (Black)

MatLabOld=false;

debugMICKA=false;

createnewmats=false;
SAplot=true;

% This file reads in meshtal filkes, usually with the extension .imsht


if createnewmats
%%

    d = dir;
    str = {d.name};
    str=str(3:end);
    [s,~] = listdlg('PromptString','Select a file:',...
                    'SelectionMode','single',...
                    'ListString',str);

    %% Data Import

    filename=str{s};

    disp('Reading In Data');

    DatIn=importdata(filename);

    DatInText=DatIn.textdata;

    DatInTextLen=size(DatInText,1);

    FindStr='        X         Y         Z     Result     Rel Error';
    FindStr2='Mesh';
    FindStr3='';

    ii=1;
    gg=1;
    mm=1;

    quit=false;

    ProgBar=waitbar(0,'Reading In Flux Data....');
    % TalBar=waitbar(0,'Reading In Tally 1');

    iints=200;
    jints=200;
    kints=1;

    NumTalPts=iints*jints*kints-1;

    Tal=zeros(NumTalPts,5);

    for ii=1:DatInTextLen

        CurLn=DatInText(ii,1:3);

        if ~isempty(strfind(CurLn{1},FindStr2))

            CurTalNum=strjoin(DatInText(ii,1:4));

            disp([CurTalNum ' is being read...']);

        end

        if strfind(strjoin(CurLn),'X Y Z')

            iiInd=ii+1;

            ii=ii+1+NumTalPts;

            FluxData=DatInText(iiInd:(iiInd+NumTalPts),:);

            parfor gg=1:NumTalPts

                TalX=str2double(FluxData{gg,1});
                TalY=str2double(FluxData{gg,2});
                TalZ=str2double(FluxData{gg,3});
                TalDat=str2double(FluxData{gg,4});
                TalDatUnc=str2double(FluxData{gg,5});

                Tal(gg,:)=[TalX,TalY,TalZ,TalDat,TalDatUnc];

            end

            save(CurTalNum,'Tal');

            Tal=zeros(NumTalPts,5);
```

```
            mm=mm+1;

            disp([CurTalNum ' written.']);
            disp(' ');

        end

        waitbar(ii/DatInTextLen,ProgBar);

    end

end
%% Plotter

if SAplot

    % nsa=331;
    nsa=721;
    hdp=5.88772;

    hdorg=HexOrg(nsa,hdp);
    load('MatMap.mat');

    %**********************Hex Duct card maker*******************************

    for ii=1:nsa

        if ii==1;

            nr=ceil((3+sqrt(12*nsa-3))/6);
            nnsa=3*((nr^2)+(nr+1))-2;

        end

        MovePlot(hdorg(ii,:),MatMap(ii+1,:),0,[]);

    end

end

d = dir;
str = {d.name};
str=str(3:end);
[s,~] = listdlg('PromptString','Select a file:',...
                'SelectionMode','single',...
                'ListString',str);

filename=str{s};

load(filename);

ff=1;

TalLen=size(Tal,1);

TalF=Tal(:,4);

UnqTalF=unique(TalF);

UniqueTalLen=size(UnqTalF,1)-1;

% NTal=zeros(UniqueTalLen,5);
% NTalColor=zeros(UniqueTalLen,3);
% NTalSize=zeros(UniqueTalLen,1);

FluxMax=max(TalF);

ZeroTal=0;

TalF=(TalF/FluxMax)*50+ZeroTal;
Tal(:,4)=(Tal(:,4)/FluxMax)*50+ZeroTal;

TalSizeMax=30;

FluxMax=max(TalF);
FluxMin=min(TalF);
FluxDiff=FluxMax-FluxMin;

FTalLen=round(0.05*TalLen);

TalColorT=colormap(hot(TalLen+FTalLen));

TalColor=TalColorT(1:TalLen,:);

TalSize=linspace(1,TalSizeMax,TalLen)';


TalFrac=(Tal(:,4)-FluxMin)./FluxDiff;

TalColorInd=round(TalFrac.*TalLen);

TalColorInd(TalColorInd==0)=1;

%NTalColor=TalColor(TalColorInd,:);
```

```
disp('Removing Zeros')

for jj=1:TalLen

    if Tal(jj,4)<=(ZeroTal+2);

    else

        NTal(ff,:)=Tal(jj,:);
        NTalColor(ff,:)=TalColor(TalColorInd(jj),:);
        NTalSize(ff,1)=TalSize(TalColorInd(jj));

        ff=ff+1;

    end

end

NTalLen=size(NTal,1);

TalSize=10;
%TalSize=ones(TalLen,1);

CurAx=gca;

disp('Drawing Graph')

% tri = delaunay(NTal(:,1),NTal(:,2));
%plot(x,y,'.')

% Plot it with TRISURF

%h = trisurf(tri,NTal(:,1),NTal(:,2),NTal(:,4));
%axis off
%axis equal

%set(gca,'CameraPosition',[208 -50 7687])

%scatter3(CurAx,Tal(:,1),Tal(:,2),Tal(:,4),TalSize,NTalColor,'.')

%scatter3(CurAx,NTal(:,1),NTal(:,2),NTal(:,4),NTalSize,NTalColor,'.')

keyboard

scatter3(CurAx,NTal(:,1),NTal(:,2),NTal(:,4),NTalSize,NTalColor,'o','MarkerEdgeColor','k','MarkerFaceColor','flat')

% pause
%
% l = light('Position',[-50 -15 29]);
% %set(gca,'CameraPosition',[208 -50 7687])
% lighting phong
% shading interp

%drawnow

% disp(' ')
% disp('Saving Figure')
%
% %savefig(gcf,'Flux','compact')
%
% disp(' ')
% disp('Saving .pdf figure')
%
% print('-opengl','FluxMapFig.pdf','-dpdf','-r600');
%
% disp(' ')
% disp('Saving .eps figure')
%
% print('FluxMapFig.eps','-depsc','-loose');

clear all

end
```

B.84. MatZaidConv.m

```
function [ S1Zaids, S2Zaids, S3Zaids ] = MatZaidConv( OutDatType, DataFormat )
%% The purpose of this program is to read in depletion data from SCALE and
% triton, and convert it to ZAID material composition information.

% The program will then output both the raw data and the normalized
% composition.

% OutDatType=1 Excel Doc Out
% OutDatType=2 Data Array Out
% OutDatType=3 Both Doc Out

%% User input for while material file to use

d = dir;
str = {d.name};
str=str(3:end);
[s,~] = listdlg('PromptString','Select a file:',...
```

```
                'SelectionMode','single',...
                'ListString',str);

%% Initialize variable
ExcelOut=false;
DataOut=false;

if OutDatType==1; ExcelOut=true; end
if OutDatType==2; DataOut=true; end
if OutDatType==3; DataOut=true; ExcelOut=true; end

S2Zaids=0;
S3Zaids=0;

FoundPreserve=false;
PreserveZAID=[60148;57139;25010];
RemoveZAIDs=[58140;56138;54136;14028];

%% Data Import

filename=str{s};

fileID=fopen(filename,'r');

FindStr='                          depletion material no. ';
FindStr2='1';

ii=1;
kk=1;

quit=false;

ProgBar=waitbar(0,'Reading In SCALE Data....');

while ~feof(fileID)

    DatIn = fgetl(fileID);

    if ~isempty(strfind(DatIn,FindStr))

        zonenumtemp=strrep(DatIn,FindStr,'');

        zoneNum=str2double(zonenumtemp(1:2));

        fgetl(fileID); fgetl(fileID); fgetl(fileID); DatIn = fgetl(fileID);

        if isempty(strfind(DatIn(6:10),' '))

            while ~quit

                NucIn=fgetl(fileID);

                if ~strcmp(NucIn(1),FindStr2)

                    DatOutTemp{ii,1}=zoneNum;
                    DatOutTemp{ii,2}=NucIn;

                    ii=ii+1;

                else

                    quit=true;

                end

            end

            DatOut{kk,1}=DatOutTemp;

            waitbar(kk/6,ProgBar);

            ii=1;
            quit=false;
            kk=kk+1;

            clear DatOutTemp

        end

    end

end

DatOutLength=size(DatOut,1);

DataAtom=DatOut(1:DatOutLength/2);
DataGram=DatOut(DatOutLength/2+1:end);

if strcmp(DataFormat,'Fuel'); S2=true; S3=true; DatCols=7; end
if strcmp(DataFormat,'Blanket'); S2=false; S3=false; DatCols=14;end

S1Dat=DataAtom{1,1};
S1Dsize=size(S1Dat,1)-2;
S1DatSplit=cell(S1Dsize,DatCols);
S1Zaids=cell(S1Dsize,3);
```

```
S1ZaidOut=cell(S1Dsize,2);
S1DatAden=cell(S1Dsize,1);

if S2

    S2Dat=DataAtom{2,1};
    S2Dsize=size(S2Dat,1)-2;
    S2DatSplit=cell(S2Dsize,DatCols);
    S2Zaids=cell(S2Dsize,3);
    S2ZaidOut=cell(S2Dsize,2);
    S2DatAden=cell(S2Dsize,1);

end

if S3

    S3Dat=DataAtom{3,1};
    S3Dsize=size(S3Dat,1)-2;
    S3DatSplit=cell(S3Dsize,DatCols);
    S3Zaids=cell(S3Dsize,3);
    S3ZaidOut=cell(S3Dsize,2);
    S3DatAden=cell(S3Dsize,1);

end

waitbar(0,ProgBar,'Converting Aden Data....');

for ii=1:S1Dsize;


%% Section 1

   waitbar(ii/S1Dsize,ProgBar);

   S1DatSplit(ii,:)=strsplit(S1Dat{ii,2},' ');
   S1Zaids(ii,2)=S1DatSplit(ii,2);
   S1ZaidsTemp=S1Zaids{ii,2};

   LenChk=str2double(S1ZaidsTemp(2));

    if isnan(LenChk) || LenChk==1i

        ZnumLen=3;
        S1Zaids(ii,1)={S1ZaidsTemp(1:2)};

    else

        ZnumLen=2;
        S1Zaids(ii,1)={S1ZaidsTemp(1)};

    end

    S1Zaids(ii,3)={S1ZaidsTemp(ZnumLen:end)};

    S1DatAden(ii)=S1DatSplit(ii,end);
    [S1Zaids{ii,2}, m1]=ZnumConv(S1Zaids{ii,1});
    S1iso=S1Zaids{ii,3};



     if strcmp(S1iso(end),'m');

         S1isoT=S1iso(1:end-1);

         if length(S1isoT)<3

            S1iso=[num2str(m1) S1isoT];

         else

            S1iso=[num2str(str2double(S1isoT(1))+m1) S1isoT(2:end)];

         end

     end

    if length(S1iso)>2;

        S1ZaidOut(ii,1)={[S1Zaids{ii,2}, S1iso]};

    else

        if length(S1iso)==1

            S1ZaidOut(ii,1)={[S1Zaids{ii,2}, '00', S1iso ]};

        else

            S1ZaidOut(ii,1)={[S1Zaids{ii,2}, '0', S1iso ]};

        end

    end

   S1ZaidOut(ii,2)=num2cell(str2double(S1DatAden{ii}));
```

```
%% Section 2
   if S2

       S2DatSplit(ii,:)=strsplit(S2Dat{ii,2},' ');
       S2Zaids(ii,2)=S2DatSplit(ii,2);
       S2ZaidsTemp=S2Zaids{ii,2};

        LenChk=str2double(S2ZaidsTemp(2));

        if isnan(LenChk) || LenChk==1i

            ZnumLen=3;
            S2Zaids(ii,1)={S2ZaidsTemp(1:2)};

        else

            ZnumLen=2;
            S2Zaids(ii,1)={S2ZaidsTemp(1)};

        end

        S2Zaids(ii,3)={S2ZaidsTemp(ZnumLen:end)};

        S2DatAden(ii)=S2DatSplit(ii,end);
        [S2Zaids{ii,2}, m2]=ZnumConv(S2Zaids{ii,1});
        S2iso=S2Zaids{ii,3};


         if strcmp(S2iso(end),'m') ;

             S2isoT=S2iso(1:end-1);

             if length(S2isoT)<3

                S2iso=[num2str(m2) S2isoT];

             else

                S2iso=[num2str(str2double(S2isoT(1))+m2) S2isoT(2:end)];

             end

        end

        if length(S2iso)>2;

            S2ZaidOut(ii,1)={[S2Zaids{ii,2}, S2iso]};

        else

            if length(S2iso)==1

                S2ZaidOut(ii,1)={[S2Zaids{ii,2}, '00', S2iso ]};

            else

                S2ZaidOut(ii,1)={[S2Zaids{ii,2}, '0', S2iso ]};

            end

        end

        S2ZaidOut(ii,2)=num2cell(str2double(S2DatAden{ii}));

   end

%% Section 3

   if S3

       S3DatSplit(ii,:)=strsplit(S3Dat{ii,2},' ');
       S3Zaids(ii,2)=S3DatSplit(ii,2);
       S3ZaidsTemp=S3Zaids{ii,2};

        LenChk=str2double(S3ZaidsTemp(2));

         if isnan(LenChk) || LenChk==1i

            ZnumLen=3;
            S3Zaids(ii,1)={S3ZaidsTemp(1:2)};

        else

            ZnumLen=2;
            S3Zaids(ii,1)={S3ZaidsTemp(1)};

        end

        S3Zaids(ii,3)={S3ZaidsTemp(ZnumLen:end)};

        S3DatAden(ii)=S3DatSplit(ii,end);
        [S3Zaids{ii,2}, m3]=ZnumConv(S3Zaids{ii,1});
```

```
        S3iso=S3Zaids{ii,3};

         if strcmp(S3iso(end),'m') && S3;

             S3isoT=S3iso(1:end-1);

             if length(S3isoT)<3

                S3iso=[num2str(m3) S3isoT];

             else

                S3iso=[num2str(str2double(S3isoT(1))+m1) S3isoT(2:end)];

             end

         end

        if length(S3iso)>2 && S3;

            S3ZaidOut(ii,1)={[S3Zaids{ii,2}, S3iso]};

        else

            if length(S3iso)==1

                S3ZaidOut(ii,1)={[S3Zaids{ii,2}, '00', S3iso ]};

            else

                S3ZaidOut(ii,1)={[S3Zaids{ii,2}, '0', S3iso ]};

            end

        end

        S3ZaidOut(ii,2)=num2cell(str2double(S3DatAden{ii}));

    end

end

%% Normalized Composition

% Run the NormProgram

S1ZaidOutNorm=Normalize(S1ZaidOut,1);

% Prepare the data for MICKA
% The actinides will be removed and a new Normalization will be performed
% and then that new nrom will be placed into an excel doc

% Convert the data to all numbers

LFPS1=cell2mat(delNaN(S1ZaidOut,'Conv','num'));

% Remove the actinides

LFPS1new=[LFPS1(LFPS1(:,1)<90000,1), LFPS1(LFPS1(:,1)<90000,2)];

% Find any preserved ZAIDS

for jj=1:length(PreserveZAID)

    PZAIDsT=[LFPS1(LFPS1(:,1)==PreserveZAID(jj,1),1), LFPS1(LFPS1(:,1)==PreserveZAID(jj,1),2)];

    if jj==1; PZAIDs=PZAIDsT; else PZAIDs=[PZAIDs;PZAIDsT]; end

end

% Normalize

LFPS1newNorm=Normalize(LFPS1new,1);

% Remove low yield LFPS the threshold will be 1E-3

LFPS1newNormMSTemp=[LFPS1newNorm(LFPS1newNorm(:,2)>1E-2,1),...
                LFPS1newNorm(LFPS1newNorm(:,2)>1E-2,2)];

for ff=1:length(PZAIDs)

    curPZAID=[LFPS1newNormMSTemp(LFPS1newNormMSTemp(:,1)==PZAIDs(ff,1),1), ...
        LFPS1newNormMSTemp(LFPS1newNormMSTemp(:,1)==PZAIDs(ff,1),2)];

    % Add on the preserved ZAIDS if they were cut

    if isempty(curPZAID); LFPS1newNormMSTemp=[LFPS1newNormMSTemp;PZAIDs(ff,:)]; end

end

% rind and remove the problematic ZAIDS

for dd=1:length(RemoveZAIDs)

        RZmatrix=LFPS1newNormMSTemp==RemoveZAIDs(dd);
```

```
        RZmatrix=[RZmatrix(:,1),RZmatrix(:,1)];

        if ~(sum(RZmatrix)==0)

            LFPS1newNormMSTemp=[LFPS1newNormMSTemp(~RZmatrix(:,1),1),LFPS1newNormMSTemp(~RZmatrix(:,2),2)];

        end
end

clear PZAIDs

% ReNormalize

LFPS1newNormMS=Normalize(LFPS1newNormMSTemp,1);

if S2

    S2ZaidOutNorm=Normalize(S2ZaidOut,1);

    LFPS2=cell2mat(delNaN(S2ZaidOut,'Conv','num'));

    LFPS2new=[LFPS2(LFPS2(:,1)<90000,1), LFPS2(LFPS2(:,1)<90000,2)];

    % Find any preserved ZAIDS

    for vv=1:length(PreserveZAID)

        PZAIDsT=[LFPS2(LFPS2(:,1)==PreserveZAID(vv,1),1), LFPS2(LFPS2(:,1)==PreserveZAID(vv,1),2)];

        if vv==1; PZAIDs=PZAIDsT; else PZAIDs=[PZAIDs;PZAIDsT]; end

    end

    LFPS2newNorm=Normalize(LFPS2new,1);

    LFPS2newNormMSTemp=[LFPS2newNorm(LFPS2newNorm(:,2)>1E-2,1),...
                LFPS2newNorm(LFPS2newNorm(:,2)>1E-2,2)];

    for ff=1:length(PZAIDs)

        curPZAID=[LFPS2newNormMSTemp(LFPS2newNormMSTemp(:,1)==PZAIDs(ff,1),1), ...
            LFPS2newNormMSTemp(LFPS2newNormMSTemp(:,1)==PZAIDs(ff,1),2)];

        % Add on the preserved ZAIDS if they were cut

        if isempty(curPZAID); LFPS2newNormMSTemp=[LFPS2newNormMSTemp;PZAIDs(ff,:)]; end

    end

    % rind and remove the problematic ZAIDS

    for dd=1:length(RemoveZAIDs)

            RZmatrix=LFPS2newNormMSTemp==RemoveZAIDs(dd);
            RZmatrix=[RZmatrix(:,1),RZmatrix(:,1)];

            if ~(sum(RZmatrix)==0)

                LFPS2newNormMSTemp=[LFPS2newNormMSTemp(~RZmatrix(:,1),1),LFPS2newNormMSTemp(~RZmatrix(:,2),2)];

            end
    end

    LFPS2newNormMS=Normalize(LFPS2newNormMSTemp,1);

    clear PZAIDs

end % Section 2

if S3

    S3ZaidOutNorm=Normalize(S3ZaidOut,1);

    LFPS3=cell2mat(delNaN(S3ZaidOut,'Conv','num'));

    LFPS3new=[LFPS3(LFPS3(:,1)<90000,1), LFPS3(LFPS3(:,1)<90000,2)];

    % Find any preserved ZAIDS

    for vv=1:length(PreserveZAID)

        PZAIDsT=[LFPS3(LFPS3(:,1)==PreserveZAID(vv,1),1), LFPS3(LFPS3(:,1)==PreserveZAID(vv,1),2)];

        if vv==1; PZAIDs=PZAIDsT; else PZAIDs=[PZAIDs;PZAIDsT]; end

    end

    LFPS3newNorm=Normalize(LFPS3new,1);

    LFPS3newNormMSTemp=[LFPS3newNorm(LFPS3newNorm(:,2)>1E-2,1),...
                LFPS3newNorm(LFPS3newNorm(:,2)>1E-2,2)];

    for ff=1:length(PZAIDs)

        curPZAID=[LFPS3newNormMSTemp(LFPS3newNormMSTemp(:,1)==PZAIDs(ff,1),1), ...
```

```
            LFPS3newNormMSTemp(LFPS3newNormMSTemp(:,1)==PZAIDs(ff,1),2)];

        % Add on the preserved ZAIDS if they were cut

        if isempty(curPZAID); LFPS3newNormMSTemp=[LFPS3newNormMSTemp;PZAIDs(ff,:)]; end

    end

    % rind and remove the problematic ZAIDS

    for dd=1:length(RemoveZAIDs)

            RZmatrix=LFPS3newNormMSTemp==RemoveZAIDs(dd);
            RZmatrix=[RZmatrix(:,1),RZmatrix(:,1)];

            if ~(sum(RZmatrix)==0)

                LFPS3newNormMSTemp=[LFPS3newNormMSTemp(~RZmatrix(:,1),1),LFPS3newNormMSTemp(~RZmatrix(:,2),2)];

            end
    end

    LFPS3newNormMS=Normalize(LFPS3newNormMSTemp,1);

    clear PZAIDs

end % Section 3


%% Convert to MICKA standard

% GraB all the ZAID numbers

AllLFPZaidsTemp=...
unique(LFPS1newNormMS(:,1)','sorted');

AllLFPZaids=cell(2,length(AllLFPZaidsTemp));

if S2

    AllLFPZaidsTemp=...
    unique([LFPS1newNormMS(:,1)',LFPS2newNormMS(:,1)'],'sorted');

    AllLFPZaids=cell(3,length(AllLFPZaidsTemp));

end

if S3

    AllLFPZaidsTemp=...
    unique([LFPS1newNormMS(:,1)',LFPS2newNormMS(:,1)',LFPS3newNormMS(:,1)'],'sorted');

    AllLFPZaids=cell(4,length(AllLFPZaidsTemp));

end

% The following loop, goes through all of the unique zaid numbers and finds
% them inside of the LFP

for kk=1:size(AllLFPZaids,2)

    waitbar(kk/size(AllLFPZaids,2),ProgBar,'Converting to MICKA standard...')

    % Current ZAID number

    curZAID=AllLFPZaidsTemp(kk);

    % Find that ZAID in the data

    [S1row, ~]=find(LFPS1newNormMS(:,1)==curZAID);

    if isempty(S1row); S1rowComp=num2cell(0);...
        else S1rowComp=num2cell(LFPS1newNormMS(S1row,2)); end

    AllLFPZaids(1:2, kk)=[{num2str(curZAID)}; S1rowComp];

    if S2

        [S2row, ~]=find(LFPS2newNormMS(:,1)==curZAID);

        if isempty(S2row); S2rowComp=num2cell(0);...
            else S2rowComp=num2cell(LFPS2newNormMS(S2row,2)); end

        AllLFPZaids(1:3, kk)=[{num2str(curZAID)}; S1rowComp; S2rowComp];

    end % Section 2

    if S3

        [S3row, ~]=find(LFPS3newNormMS(:,1)==curZAID);

        if isempty(S3row); S3rowComp=num2cell(0);...
            else S3rowComp=num2cell(LFPS3newNormMS(S3row,2)); end

       AllLFPZaids(1:4, kk)=[{num2str(curZAID)}; S1rowComp; S2rowComp; S3rowComp];
```

```
    end % Section 3

end

% Summation Checks

SumS1=sum(cell2mat(AllLFPZaids(2, :)));

if S2; SumS2=sum(cell2mat(AllLFPZaids(3, :))); end

if S3; SumS3=sum(cell2mat(AllLFPZaids(4, :))); end

switch size(AllLFPZaids,1)

    case 2

        if RoundM(SumS1,5)~=1;
            disp('Section 1'); disp('Sum of LFP != 1'); keyboard; end

        AllLFPZaids=[[{'Section'}; {'Sec 1'}], AllLFPZaids,...
            [{'Total Should be 1'}; SumS1]];

    case 3

        if RoundM(SumS2,5)~=1;
           disp('Section 2'); disp('Sum of LFP != 1'); keyboard; end

        AllLFPZaids=[[{'Section'}; {'Sec 1'}; {'Sec 2'}], AllLFPZaids,...
            [{'Total Should be 1'}; SumS1; SumS2]];


    case 4

        SumS3=sum(cell2mat(AllLFPZaids(4, :)));

        if RoundM(SumS3,5)~=1;
            disp('Section 3'); disp('Sum of LFP != 1'); keyboard; end

        AllLFPZaids=[[{'Section'}; {'Sec 1'}; {'Sec 2'}; {'Sec 3'}], AllLFPZaids,...
         [{'Total Should be 1'}; SumS1; SumS2; SumS3]];

end


%% Output data

% Format the out Data

% Raw Data

S1ZaidOuttemp=[{'Section 1'},{''}; {'ZAID'}, {'Atom Density'}; S1ZaidOut];
ZAIDout=S1ZaidOuttemp;
spacer=cell(size(S1ZaidOuttemp,1),1);

% Normalized Data

S1ZaidOutNormtemp=[{'Section 1'},{''}; {'ZAID'}, {'Atom Density'}; S1ZaidOutNorm];
ZAIDoutNorm=S1ZaidOutNormtemp;

if S2;
    S2ZaidOuttemp=[{'Section 2'},{''}; {'ZAID'}, {'Atom Density'}; S2ZaidOut];
    ZAIDout=[S1ZaidOuttemp,spacer,S2ZaidOuttemp];
    S2ZaidOutNormtemp=[{'Section 2'},{''}; {'ZAID'}, {'Atom Density'}; S2ZaidOutNorm];
    ZAIDoutNorm=[S1ZaidOutNormtemp,spacer,S2ZaidOutNormtemp];
end

if S3;
    S3ZaidOuttemp=[{'Section 3'},{''}; {'ZAID'}, {'Atom Density'}; S3ZaidOut];
    ZAIDout=[S1ZaidOuttemp,spacer,S2ZaidOuttemp,spacer,S3ZaidOuttemp];
    S3ZaidOutNormtemp=[{'Section 3'},{''}; {'ZAID'}, {'Atom Density'}; S3ZaidOutNorm];
    ZAIDoutNorm=[S1ZaidOutNormtemp,spacer,S2ZaidOutNormtemp,spacer,S3ZaidOutNormtemp];
end

% Write Data to excel or arrays

if ExcelOut

    waitbar(1,ProgBar,'Writing Raw DataExcel File');

    xlswrite('Section Data Raw',ZAIDout);

    waitbar(1,ProgBar,'Writing Normalized DataExcel File');

    xlswrite('Section Data Normalized',ZAIDoutNorm);

end

if DataOut

    waitbar(1,ProgBar,'Writing MICKA standard LFP');

    ExtInds=strfind(filename, '.');

    if strcmp(DataFormat,'Fuel'); xlswrite([filename(1:ExtInds-1) '.xlsx'],AllLFPZaids); end
```

```
    if strcmp(DataFormat,'Blanket'); xlswrite([filename(1:ExtInds-1) '.xlsx'],AllLFPZaids); end

end

close(ProgBar)
fclose('all');

end

function [znumout, mlev ]= ZnumConv(Znamein)

% This function is just a lookup table to change element names into their
% respective znumber
% The ml stands for metalevel

switch Znamein

    case 'h' ; znumout='1'; mlev=0;
    case 'he'; znumout='2'; mlev=0;
    case 'li'; znumout='3'; mlev=0;
    case 'be'; znumout='4'; mlev=0;
    case 'b';  znumout='5'; mlev=0;
    case 'c';  znumout='6'; mlev=0;
    case 'n';  znumout='7'; mlev=0;
    case 'o';  znumout='8'; mlev=0;
    case 'f';  znumout='9'; mlev=0;
    case 'ne'; znumout='10'; mlev=0;
    case 'na'; znumout='11'; mlev=0;
    case 'mg'; znumout='12'; mlev=0;
    case 'al'; znumout='13'; mlev=0;
    case 'si'; znumout='14'; mlev=0;
    case 'p';  znumout='15'; mlev=0;
    case 's';  znumout='16'; mlev=0;
    case 'cl'; znumout='17'; mlev=0;
    case 'ar'; znumout='18'; mlev=0;
    case 'k';  znumout='19'; mlev=0;
    case 'ca'; znumout='20'; mlev=0;
    case 'sc'; znumout='21'; mlev=0;
    case 'ti'; znumout='22'; mlev=0;
    case 'v';  znumout='23'; mlev=0;
    case 'cr'; znumout='24'; mlev=0;
    case 'mn'; znumout='25'; mlev=0;
    case 'fe'; znumout='26'; mlev=0;
    case 'co'; znumout='27'; mlev=0;
    case 'ni'; znumout='28'; mlev=0;
    case 'cu'; znumout='29'; mlev=0;
    case 'zn'; znumout='30'; mlev=0;
    case 'ga'; znumout='31'; mlev=0;
    case 'ge'; znumout='32'; mlev=0;
    case 'as'; znumout='33'; mlev=0;
    case 'se'; znumout='34'; mlev=3;
    case 'br'; znumout='35'; mlev=3;
    case 'kr'; znumout='36'; mlev=3;
    case 'rb'; znumout='37'; mlev=3;
    case 'sr'; znumout='38'; mlev=0;
    case 'y';  znumout='39'; mlev=3;
    case 'zr'; znumout='40'; mlev=0;
    case 'nb'; znumout='41'; mlev=3;
    case 'mo'; znumout='42'; mlev=0;
    case 'tc'; znumout='43'; mlev=3;
    case 'ru'; znumout='44'; mlev=0;
    case 'rh'; znumout='45'; mlev=3;
    case 'pd'; znumout='46'; mlev=3;
    case 'ag'; znumout='47'; mlev=3;
    case 'cd'; znumout='48'; mlev=3;
    case 'in'; znumout='49'; mlev=3;
    case 'sn'; znumout='50'; mlev=3;
    case 'sb'; znumout='51'; mlev=3;
    case 'te'; znumout='52'; mlev=3;
    case 'i';  znumout='53'; mlev=3;
    case 'xe'; znumout='54'; mlev=3;
    case 'cs'; znumout='55'; mlev=3;
    case 'ba'; znumout='56'; mlev=0;
    case 'la'; znumout='57'; mlev=3;
    case 'ce'; znumout='58'; mlev=0;
    case 'pr'; znumout='59'; mlev=0;
    case 'nd'; znumout='60'; mlev=0;
    case 'pm'; znumout='61'; mlev=3;
    case 'sm'; znumout='62'; mlev=0;
    case 'eu'; znumout='63'; mlev=0;
    case 'gd'; znumout='64'; mlev=0;
    case 'tb'; znumout='65'; mlev=0;
    case 'dy'; znumout='66'; mlev=0;
    case 'ho'; znumout='67'; mlev=0;
    case 'er'; znumout='68'; mlev=0;
    case 'tm'; znumout='69'; mlev=0;
    case 'yb'; znumout='70'; mlev=0;
    case 'lu'; znumout='71'; mlev=0;
    case 'hf'; znumout='72'; mlev=0;
    case 'ta'; znumout='73'; mlev=0;
    case 'w';  znumout='74'; mlev=0;
    case 're'; znumout='75'; mlev=0;
    case 'os'; znumout='76'; mlev=0;
    case 'ir'; znumout='77'; mlev=0;
    case 'pt'; znumout='78'; mlev=0;
```

```
    case 'au'; znumout='79'; mlev=0;
    case 'hg'; znumout='80'; mlev=0;
    case 'tl'; znumout='81'; mlev=0;
    case 'pb'; znumout='82'; mlev=0;
    case 'bi'; znumout='83'; mlev=0;
    case 'po'; znumout='84'; mlev=0;
    case 'at'; znumout='85'; mlev=0;
    case 'rn'; znumout='86'; mlev=0;
    case 'fr'; znumout='87'; mlev=0;
    case 'ra'; znumout='88'; mlev=0;
    case 'ac'; znumout='89'; mlev=0;
    case 'th'; znumout='90'; mlev=0;
    case 'pa'; znumout='91'; mlev=0;
    case 'u';  znumout='92'; mlev=0;
    case 'np'; znumout='93'; mlev=3;
    case 'pu'; znumout='94'; mlev=0;
    case 'am'; znumout='95'; mlev=3;
    case 'cm'; znumout='96'; mlev=0;
    case 'bk'; znumout='97'; mlev=0;
    case 'cf'; znumout='98'; mlev=0;
    case 'es'; znumout='99'; mlev=3;
    case 'fm'; znumout='100'; mlev=0;
    case 'md'; znumout='101'; mlev=0;
    case 'no'; znumout='102'; mlev=0;
    case 'lr'; znumout='103'; mlev=0;
    case 'rf'; znumout='104'; mlev=0;
    case 'db'; znumout='105'; mlev=0;
    case 'sg'; znumout='106'; mlev=0;
    case 'bh'; znumout='107'; mlev=0;
    case 'hs'; znumout='108'; mlev=0;
    case 'mt'; znumout='109'; mlev=0;
    case 'ds'; znumout='110'; mlev=0;

end



end

function [ DataOut ] = Normalize( DataIn, NormFact )

% This function brings in a data set and normalizes it. The input should
% be a 2xn or a nx2. The first row or column can be titles or whatever,
% they are not change and are retained in the ouput.

% Norm fact is used if a different normalization is needed. If NormFact is
% set to 1 then the data is noirmalize to itself, if other then the data is
% normalized to that constant.

% Initialize switches

CustNorm=true;
NumIn=false;
CellIn=false;
RowAr=false;
ColAr=false;
ConvDat=false;

if NormFact==1; CustNorm=false; end

 DataOut=DataIn;

 %% Analyze the incoming data

 % Get the incoming data size
 [DatRow, DatCol]=size(DataOut);

 if DatRow>DatCol; RowAr=true; RowLen=DatRow; ColLen=1; end
 if DatRow<DatCol; ColAr=true; RowLen=1; ColLen=DatCol; end
 if DatRow==DatCol; disp('Square Array!'); keyboard; end

 % Check to see what kind of array is DataIn

 if isnumeric(DataOut); NumIn=true; end
 if iscell(DataOut); CellIn=true; end

 % Check to see if the data is strings or nums

 if ~NumIn; if ischar(DataOut{2,2}); ConvDat=true; end; end

 %% Process Cell array

 if CellIn

     % Remove the titles

     Titles=DataOut(1:RowLen,1:ColLen);

     if RowAr; Data=DataOut(:,2); end
     if ColAr; Data=DataOut(2,:); end

     % Convert the Data if needed

     if ConvDat
```

```
         DataTemp=Data;

         Data=cellfun(@str2double, DataTemp, 'un', 0);

        if RowAr; DataOut(:,2)=Data; end
        if ColAr; DataOut(2,:)=Data; end

         DataNums=cell2mat(Data);

     else

         DataNums=cell2mat(Data);

     end

     % Determine the summation

     if CustNorm; DataSum=NormFact; else DataSum=sum(DataNums); end

     % Normalize the Cell Data

     if RowAr; DataOut(:,2)=cellfun(@(x) x./DataSum, DataOut(:,2), 'un', 0); end
     if ColAr; DataOut(2,:)=cellfun(@(x) x./DataSum, DataOut(2,:), 'un', 0); end

     % Change the data back to strings if needed

     if ConvDat

        if RowAr; DataOut(:,2)=...
                cellfun(@(x) num2str(x, '%10.4e\n'), DataOut(:,2), 'un', 0); end
        if ColAr; DataOut(2,:)=...
                cellfun(@(x) num2str(x, '%10.4e\n'), DataOut(2,:), 'un', 0); end

     end

 end

 %% Process Num array

 if NumIn

    % Determine the summation

    if CustNorm;

        DataSum=NormFact;

    else

            if RowAr; DataSum=sum(DataOut(:,2)); end
            if ColAr; DataSum=sum(DataOut(2,:)); end

    end

    if RowAr; DataOut(:,2)=DataOut(:,2)./DataSum; end
    if ColAr; DataOut(2,:)=DataOut(2,:)./DataSum; end

 end

end

function [ cellin ] = delNaN( cellin, ~ , type )
% takes in an arbitrary cell array and removes the NaN's if they exist.

if isstruct(cellin)

    StrcTx=cellin.textdata;
    StrcDattemp=cellin.data;

    StrcDat=num2cell(StrcDattemp);

    colshift=abs(size(StrcTx,2)-size(StrcDat,2));
    rowshift=abs(size(StrcTx,1)-size(StrcDat,1));

    %Need to find any straggeling text and put it in the data part

    txlocar=cellfun(@isempty,(StrcTx(1+rowshift:end,1+colshift:end)));

    [txlocrow,txloccol]=find(txlocar==0);

    for kk=1:length(txlocrow)

        StrcDat(txlocrow(kk),txloccol(kk))=...
            StrcTx(rowshift+txlocrow(kk),colshift+txloccol(kk));

    end

    StrcTx(1+rowshift:end,1+colshift:end)=StrcDat;

    cellin=StrcTx;

end

stryes=false;
numyes=false;
```

```
if strcmp(type,'str')

    stryes=true;

end

if strcmp(type,'num')

    numyes=true;

end

[cellrow,cellcol]=size(cellin);


for ii=1:cellrow

    for ll=1:cellcol

        cellt=cellin{ii,ll};

        if isnan(cellt)

            cellin{ii,ll}=[];

        else

            if stryes

                cellin{ii,ll}=num2str(cellt);

            end

            if numyes

                numtemp=str2double(cellt);

                if ~isnan(numtemp)

                    cellin{ii,ll}=numtemp;

                end

            end

        end

    end


end


end

function [ NumnOut ] = RoundM( NumIn, Dig, Type)

global MatLabOld


% Simple program to round numbers not builtin to matlab 2014
% Type only works for new rounding,

if MatLabOld

IntNum=floor(NumIn);

DecNum=NumIn-IntNum;

numtxt=num2str(DecNum,'%1.12f');

DigTempfr=numtxt(1:Dig+2);

DigTempbk=numtxt(Dig+3:end);

DigTempbkNum=str2double(['0.' DigTempbk]);

RdBk=round(DigTempbkNum)*(10^-(Dig));

NumnOut=(IntNum+str2double(DigTempfr)+RdBk);

else

    if exist('Type','var');

        if strcmp(Type,'S');
            Type='significant';
        end

    else

        Type='decimals';
    end
```

```
    if exist('Dig','var')

        NumnOut=round(NumIn, Dig, Type);

    else

        NumnOut=round(NumIn);

    end

end


end
```

B.85. MCNPKeffReader.m

```
function [  ] = MCNPKeffReader(  )
% this function reads MCNP output files and grabds the keff

OutputDir='R:\MCNP\Benchmark\Results';

debugOut=false;
ControlF=false;
orgpath=pwd;
FullFileP=false;
cd(OutputDir)

% Read all of the folder names
DIRS = dir;

folds=cell2mat({DIRS.isdir}');
foldsnames={DIRS.name}';

%% Find sub folders

DirDats=cell(1,11);
DirDats{1,1}='Name';
DirDats{1,2}='Data Input Files';
DirDats{1,3}='Data Output Files';
DirDats{1,4}='Change Files';
DirDats{1,5}='keff';
DirDats{1,6}='Uncert';
DirDats{1,7}='Perturbation Type';
DirDats{1,8}='Zaid or Dimension';
DirDats{1,9}='Fraction or Dimension Change';
DirDats{1,10}='Scaling Factor';
DirDats{1,11}='N';

DirDats2=DirDats;

ProgBar1=waitbar(0,'Reading Output Directories....');

for ii=3:length(DIRS)

    if folds(ii)==1

        curfold=foldsnames(ii);

        DataDirs(ii-2,1)=curfold;

        % Grab the filenames

        subfoldnames=dir(cell2mat(curfold));

        curfoldnames={subfoldnames.name}';

        % Add the folder name to the filenames

        for kk=3:length(curfoldnames)

            cursubfold=curfoldnames{kk,1};

            filenames=dir([cell2mat(curfold) '\' cursubfold]);

            curfilenames={filenames.name}';

            filenamePert=0;

            for ff=3:length(curfilenames)

                curfile=curfilenames{ff,1};

                extindO=strfind(curfile,'.o');
                extindI=strfind(curfile,'.i');

                if strfind(curfile,'.im') | strfind(curfile,'.in')

                    extindI=[];

                end
```

```
                if ~isempty(extindO)

                    filenameOut=strjoin([curfold '\' cursubfold '\' curfile],'');
                    NoPfilenameOut=curfile;

                    if strcmp(curfold,'Dimensions')

                        filenamePert=strjoin([curfold '\' cursubfold '\Maps\DimMap_Perturbed'],'');

                    end

                end

                if ~isempty(extindI)

                    filenameIn=strjoin([curfold '\' cursubfold '\'  curfile],'');
                    NoPfilenameIn=curfile;

                end

            end



            DirDats{end+1,1}=cursubfold;
            DirDats{end,2}=filenameIn;
            DirDats{end,3}=filenameOut;
            DirDats{end,4}=filenamePert;

            DirDats2{end+1,1}=cursubfold;
            DirDats2{end,2}=NoPfilenameIn;
            DirDats2{end,3}=NoPfilenameOut;
            DirDats2{end,4}=filenamePert;

        end


    end

    waitbar(ii/length(DIRS),ProgBar1);

end


waitbar(0,ProgBar1 ,'Reading In MCNP Data....');
PB2I=1;
PB2ITotO=50000;
PB2ITotI=10000;
WBIncr=500;
ProgBar2=waitbar(0,'Reading Output Data....');

for nn=2:size(DirDats,1)

    InFileOpen=true;
    OutFileOpen=true;

    disp(['Reading file: ' DirDats{nn,2}])

    fileID=fopen(DirDats{nn,3},'r');

    waitbar(0,ProgBar2,['Reading Output Data....' DirDats{nn,1}]);

    while ~feof(fileID) && OutFileOpen

        curln=fgetl(fileID);

        strindS=strfind(curln,'final result     ');

        if ~isempty(strindS)

            % Remove multiple spaces

            ncurln = regexprep(curln,' +',' ');

            ncurlnspcs=strfind(ncurln,' ');

            keff=ncurln(ncurlnspcs(3)+1:ncurlnspcs(4)-1);
            Uncert=ncurln(ncurlnspcs(4)+1:ncurlnspcs(5)-1);

            DirDats{nn,5}=keff;
            DirDats{nn,6}=Uncert;

            OutFileOpen=false;

        end

        PB2I=PB2I+1;

        if mod(PB2I,WBIncr)==0;

            waitbar(PB2I/PB2ITotI,ProgBar2);

            if debugOut

                curln
```

```
        end
    end

    if PB2I==PB2ITotO; PB2I=1; end

end

disp(['Reading file: ' DirDats{nn,3}])

fileID2=fopen(DirDats{nn,2},'r');

FoundPert=false;

PB2I=1;

waitbar(0,ProgBar2,['Reading Input Data....' DirDats{nn,1}]);

while ~feof(fileID2) && InFileOpen

    curln=fgetl(fileID2);

    strindsM=strfind(curln,'Perturbed ZAID:');
    strindsD=strfind(curln,'Perturbed Dim:');
    strindsDen=strfind(curln,'Perturbed Density Material:');
    strindsSmear=strfind(curln,'Change in Sodium WT%:');
    strindsOM=strfind(curln,'Perturbed Other Material:');

    curln2=curln;

    if ~isempty(strindsM)

        % Remove multiple spaces

        ncurlnColn=strfind(curln,':');

        ZAID=curln(ncurlnColn+1:end);

        fgetl(fileID2); fgetl(fileID2);
        curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

        FractChng=curln(ncurlnColn+1:end);

        curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

        ScaleFact=curln(ncurlnColn+1:end);

        PertType='Material';

        DirDats{nn,7}=PertType;
        DirDats{nn,8}=ZAID;
        DirDats{nn,9}=FractChng;
        DirDats{nn,10}=ScaleFact;

        FoundPert=true;

    end

    if ~isempty(strindsD)

        % Remove multiple spaces

        ncurlnColn=strfind(curln,':');

        DIM=curln(ncurlnColn+1:end);

        curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

        Sign=curln(ncurlnColn+1:end);

        curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

        EXPsPert=curln(ncurlnColn+1:end);

        curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

        ScaleFact=curln(ncurlnColn+1:end);

        if strcmp('Ex',DirDats{nn,1}(end-1:end))

            PertType='Measured Dimension';

        else

            PertType='Dimensional';

        end

        DirDats{nn,7}=PertType;
        DirDats{nn,8}=DIM;
        DirDats{nn,9}=Sign;
        DirDats{nn,10}=ScaleFact;

        FoundPert=true;
```

```
end

if ~isempty(strindsDen)

    ncurlnColn=strfind(curln,':');

    DIM=curln(ncurlnColn+1:end);

    curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

    NewDen=str2double(curln(ncurlnColn+1:end));

    curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

    OldDen=str2double(curln(ncurlnColn+1:end));

    curln=fgetl(fileID2); curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

    ScaleFact=curln(ncurlnColn+1:end);

    PertType='Density';

    DirDats{nn,7}=PertType;
    DirDats{nn,8}=DIM;
    DirDats{nn,9}=num2str(OldDen-NewDen);
    DirDats{nn,10}=ScaleFact;

    FoundPert=true;

end

if ~isempty(strindsOM)

    ncurlnColn=strfind(curln,':');

    curmat=curln(ncurlnColn+1:end);

    curln2=fgetl(fileID2);
    ncurlnColn2=strfind(curln2,':');

    Chng=curln(ncurlnColn2+1:end);

    curiso=curln2(ncurlnColn2-3:ncurlnColn2-1);

    PertType='Impurity';

    DirDats{nn,7}=PertType;
    DirDats{nn,8}=curiso;
    DirDats{nn,9}=Chng;
    DirDats{nn,10}='Na';

    FoundPert=true;

end

if ~isempty(strindsSmear)

    ncurlnColn=strfind(curln,':');
    noldlnColn=strfind(OldLn,':');

    DIM=OldLn(noldlnColn+1:end);
    NaSmear=curln(ncurlnColn+1:end);

    curln=fgetl(fileID2); ncurlnColn=strfind(curln,':');

    ScaleFact=curln(ncurlnColn+1:end);

    PertType='Smear';

    DirDats{nn,7}=PertType;
    DirDats{nn,8}=DIM;
    DirDats{nn,9}=NaSmear;
    DirDats{nn,10}=ScaleFact;

    FoundPert=true;

end

PB2I=PB2I+1;

if mod(PB2I,WBIncr)==0;

    waitbar(PB2I/PB2ITotI,ProgBar2);

    if debugOut

        curln

    end

end

if PB2I==PB2ITotI; PB2I=1; end

if FoundPert; InFileOpen=false; end
```

```
    OldLn=curln2;

end

if FoundPert

    if ~isempty(strfind(PertType,'Dimension'))

        DimChngPath=DirDats{nn,4};

        DimChngtrue=false;

        try

            [~,~,DimChng]=xlsread(DimChngPath);

            DimChngtrue=true;

        catch

            DimChngtrue=false;

        end

        if DimChngtrue

            for gg=1:size(DimChng,2)

                curDim=DimChng{1,gg};

                curDimInd=strfind(curDim,'times');

                if ~isempty(curDimInd)

                   spcInd=strfind(curDim,' ');

                   Npert=str2double(curDim(spcInd(end-1):spcInd(end)));

                   Npert=PinMod(DimChng,Npert,DIM,gg);

                   DirDats{nn,11}=Npert;

                end


            end

        end

  end

end

if ~FoundPert

    %% Bank Rod Extension

    FoundPert2=false;

    strindsCU=strfind(DirDats{nn,1},'ControlUncert');



    if ~isempty(strindsCU)

        if strcmp(DirDats{nn,1}(end),'P')

            Sign='Plus';

        else

            Sign='Minus';

        end

        DirDats{nn,7}='Bank Rod Extension';
        DirDats{nn,8}='Control Rod Position Uncertainty';
        DirDats{nn,9}=Sign;
        DirDats{nn,10}='1';

        ControlF=true;

        FoundPert2=true;

    end

   %% Control Rods

   strindsR=strfind(DirDats{nn,1},'Control');
   strindsR1=strfind(DirDats{nn,1},'HWCR');
   strindsR2=strfind(DirDats{nn,1},'control');
   strindsR3=strfind(DirDats{nn,1},'Safety');

    if (~isempty(strindsR1) || ~isempty(strindsR2) ||...
```

```
                ~isempty(strindsR) || ~isempty(strindsR3)) && ~ControlF

            if strfind(DirDats{nn,1},'FullIn')

                DIM='Rod at full insertion';
                Sign='+14"';
                ScaleFact='1';

            else

                DIM='Rod at SCRAM insertion';
                Sign='0"';
                ScaleFact='1';
            end

            DirDats{nn,7}='Rod Worth';
            DirDats{nn,8}=DIM;
            DirDats{nn,9}=Sign;
            DirDats{nn,10}=ScaleFact;

            FoundPert2=true;

        end

        %% Features

        strindsF=strfind(DirDats{nn,1},'No');

        if ~isempty(strindsF)

            DirDats{nn,7}='Model Feature';
            DirDats{nn,8}=DirDats{nn,1}(strindsF:end);
            DirDats{nn,9}='-';
            DirDats{nn,10}='NA';

            FoundPert2=true;

        end

        %% LFPS

        strindsL=strfind(DirDats{nn,1},'LFP');

        if ~isempty(strindsL)


            if strcmp(DirDats{nn,1}(end),'P')

                Sign='Plus';

            else

                Sign='Minus';

            end

            DirDats{nn,7}='Lumped Fission Product';
            DirDats{nn,8}=DirDats{nn,1}(strindsL+3:end-1);
            DirDats{nn,9}=Sign;
            DirDats{nn,10}='NA';

            FoundPert=true;
            FoundPert2=true;

        end

    end



    clear ScaleFact Sign DIM PertType

    ControlF=false;

    fclose(fileID);
    fclose(fileID2);

    waitbar(nn/size(DirDats,1) ,ProgBar1);

end

fclose('all');

cd(orgpath)

DirDats2=[DirDats(:,1),DirDats2(:,2:4),DirDats(:,5:end)];

if FullFileP

    xlswrite([OutputDir '\Results.xlsx'],DirDats)

else

    xlswrite([OutputDir '\Results.xlsx'],DirDats2)
```

```
end

[status,cmdout] = system([OutputDir '\Results.xlsx']);

end

function Npert=PinMod(SpreadSheet,OldNPert,DIM,DIMCol)

Npert=OldNPert;

if strcmp(DIM(1),'f')

    PertDim=SpreadSheet(2:end,DIMCol+1);

    Rows=cellfind(PertDim,'This dim',1,'No Match')+1;

    Npert=sum(cell2mat(SpreadSheet(Rows,159)));

end

if strcmp(DIM(1),'d')

    PertDim=SpreadSheet(2:end,DIMCol+1);

    Rows=cellfind(PertDim,'This dim',1,'No Match')+1;

    nde=cell2mat(SpreadSheet(Rows,192));

    Npert=sum(nde);

end

if strcmp(DIM(1),'e') || strcmp(DIM(1),'b')

    PertDim=SpreadSheet(2:end,DIMCol+1);

    Rows=cellfind(PertDim,'This dim',1,'No Match')+1;

    Npert=sum(cell2mat(SpreadSheet(Rows,192)));

end

if strcmp(DIM(1),'p')

    PertDim=SpreadSheet(2:end,DIMCol+1);

    Rows=cellfind(PertDim,'This dim',1,'No Match')+1;

    Npert=sum(cell2mat(SpreadSheet(Rows,111)));

end

end

function [ inds ] = cellfind( cellin,varin,rowcol,Match )
%This function takes in a cell array and a variable to be found and outputs
%its index number inside of the cell.
%This only works for cell arrays that do not contain subcell arrays.
%multiple finds will yeild a numerical array with rows in column 1 and
%columns in column 2.

%rowcol=1 means output row number
%rowcol=2 means output col number
%rowcol=3 means output both numbers

%An Output of 0 means it did not find it.


% Match means match exact case and length

if strcmp(Match,'Match')

    Matchcase=true;

else

    Matchcase=false;

end

%Subindex
kk=1;

if isnumeric(varin)


    for ii=1:size(cellin,1)

        %Extracts lines from the cell array
        cellintemp=cellin(ii,:);

        %converts the finds in 0 and the emptycells into 1
        cellinindarnum=cellfun(@isnumeric,cellintemp);

        %Gets the column number
```

```
        colnum=find(cellinindarnum==1);

        if ~(colnum==0)

            if cellintemp{colnum}==varin

                   if kk>1

                       indouttemp=[ii,colnum];

                       indout=[indout; indouttemp];

                   end

                   if kk==1

                      indout=[ii,colnum];

                      kk=kk+1;

                   end

            end

        end


    end

end

if ischar(varin)


    for ii=1:size(cellin,1)

        %Extracts lines from the cell array
        cellintemp=cellin(ii,:);
        try
        %converts everything to a string
        cellintemp2=cellfun(@(x) Num2StrM( x,'%.8g'),cellintemp,'un',0);
        catch;keyboard;end
        %Finds if varin is in the line
        cellinindar=strfind(cellintemp2,varin);

        %converts the finds in 0 and the emptycells into 1
        cellinindarnum=cellfun(@isempty,cellinindar);

        %Gets the column number
        colnum=find(cellinindarnum<1);

        if isempty(colnum)

            colnum=0;

        end

        if ~(colnum(1,1)==0)

            for ll=1:length(colnum)

                if Matchcase

                    if length(cellintemp{colnum(ll)})==length(varin)

                           if kk>1

                               indouttemp=[ii,colnum(ll)];

                               indout=[indout; indouttemp];

                           end

                           if kk==1

                              indout=[ii,colnum(ll)];

                              kk=kk+1;

                           end

                    end

                else

                           if kk>1

                               indouttemp=[ii,colnum(ll)];

                               indout=[indout; indouttemp];

                           end

                           if kk==1
```

```
                            indout=[ii,colnum(ll)];

                            kk=kk+1;

                        end

                end

            end

        end


    end

end

if ~exist('indout','var'); indout=[0,0]; end

if rowcol==1; inds=indout(:,1); end

if rowcol==2; inds=indout(:,2); end

if rowcol==3; inds=indout; end

end

function [ OutVar ] = Num2StrM( VarIn, FS )
% This is a custom function to rewrite the num2str function to be more
% efficient.

% fs should probably be '%.8g\n', this just directly converts up to 8 digits
% past the decimal point without any leading or trailing zeros.

% if ~exist('FS','var'); FS='%.8g'; end

if ~ischar(VarIn)

    if nargin==1; FS='%.8g'; end

    SpcR=0;

    if size(VarIn,2)>1; FS='%.8g '; SpcR=0; end

    OutVar1=sprintf(FS,VarIn);

    OutVar=OutVar1(1:end-SpcR);

%    if size(VarIn,2)>1; keyboard; end

else

    OutVar=VarIn;

end

end
```

B.86. TallyReader.m

```
function [  ] = TallyReader(  )
% this function reads an MCNP output file and finds the f4 and f5 talkly
% results

%% Code to pick a file

d = dir;
str = {d.name};
str=str(3:end);
[s,~] = listdlg('PromptString','Select a file:',...
                'SelectionMode','single',...
                'ListString',str);

FileTitle=str{s};

%% Open up the selected file ID

fileID=fopen(FileTitle);

%% Find the line numbers and the column numbers

LineIDstr='                           tally      ';
CharIDstr='tally';
CharTalNum=11;


[LineNum,CharNum]=FindInFile( fileID, LineIDstr, CharIDstr);

%% Find the tally numbers

TalNums=cell(size(LineNum,1),2);
```

```
OldLnNum=0;

for hh=1:size(LineNum)

   CurCharNum=CharNum{hh,1};

   CurLnNum=LineNum(hh,1);

   for gg=1:CurLnNum-OldLnNum

       curLn=fgetl(fileID);

       if gg==CurLnNum-OldLnNum;

           TalNums{hh,1}=CurLnNum;

           for nn=1:length(CurCharNum)

               CurTalChar=CurCharNum(nn)+CharTalNum;

               if nn==1

                   TalNumsTemp={curLn(CurTalChar:CurTalChar+2)};

               else

                   TalNumsTemp=[TalNumsTemp,{curLn(CurTalChar:CurTalChar+2)}];

               end

           end

           TalNums{hh,2}=TalNumsTemp;

           clear TalNumsTemp

       end

   end

   OldLnNum=CurLnNum;

end

frewind(fileID);

%% Find the end of the flux values

% Only one tally is needed to determine where the flux value is located.

FluxLineNum=LineNum(1);
FluxNumLn=0;

QuitLoop=false;

kk=0;

for rr=1:FluxLineNum

    fgetl(fileID);

    if rr==FluxLineNum

        while ~QuitLoop

            curLn=fgetl(fileID);

            if length(curLn)<=1

                FluxNumLn=kk;

                QuitLoop=true;

            end

            kk=kk+1;

        end

    end

end

frewind(fileID);

%% Grab the flux and uncertainty

OldFluxNum=0;
ColFlux1=17;
ColFlux2=57;
ColFlux3=97;

for bb=1:size(TalNums,1)

    FluxNum=TalNums{bb,1}+FluxNumLn;
```

```
    TallyNums=TalNums{bb,2};

    Tallies=size(TallyNums,2);

    for dd=1:FluxNum-OldFluxNum

        curLn=fgetl(fileID);

        if dd==FluxNum-OldFluxNum

            SpcInds=strfind(curLn,'.');

            switch Tallies

                case 2

                    TallyNums{2,1}=curLn(ColFlux1:ColFlux1+9);
                    TallyNums{3,1}=curLn(ColFlux1+11:ColFlux1+16);
                    TallyNums{2,2}=curLn(ColFlux2:ColFlux2+9);
                    TallyNums{3,2}=curLn(ColFlux2+11:ColFlux2+16);

                case 3

                    TallyNums{2,1}=curLn(ColFlux1:ColFlux1+9);
                    TallyNums{3,1}=curLn(ColFlux1+11:ColFlux1+16);
                    TallyNums{2,2}=curLn(ColFlux2:ColFlux2+9);
                    TallyNums{3,2}=curLn(ColFlux2+11:ColFlux2+16);
                    TallyNums{2,3}=curLn(ColFlux3:ColFlux3+9);
                    TallyNums{3,3}=curLn(ColFlux3+11:ColFlux3+16);

            end


        end

    end

    TalNums{bb,2}=TallyNums;

    OldFluxNum=FluxNum;

end

for oo=1:size(TalNums,1)

   CurFluxLn=TalNums{oo,2}';

   if oo==1

       Fluxes=[{FileTitle,'',''};CurFluxLn];

   else

       Fluxes=[Fluxes; CurFluxLn];

   end

end

xlswrite('FluxOut',Fluxes)

end


function [ LineNum, CharNum ] = FindInFile( fileID, VarInLn, VarInCn )
% This function will find a specific string inside of a file and then
% output the line number and colume that is the beingging of that string.

ii=1;

FirstRun=true;

while ~feof(fileID)

    CurLn=fgetl(fileID);

    StrCol=strfind(CurLn,VarInLn);

    if ~isempty(StrCol)

        % Find the CharNumbers

        StrChr=strfind(CurLn,VarInCn);

        if FirstRun

            LineNum=ii; CharNum={StrChr};

            FirstRun=false;

        else

            LineNum(end+1,1)=ii;
            CharNum{end+1,1}=StrChr;
```

```
        end
    end
    ii=ii+1;
end
frewind(fileID)
end
```

APPENDIX C EBR-II DESCRIPTION

Appendix C.1 is section 1.0 from the International Reactor Physics Benchmark handbook evaluation EBR2-LMFR-RESR-001 CRIT. [14]. At the time of writing this benchmark had been submitted by the author and was pending comment resolution. The benchmark contains detailed uncertainty quantification of run 138B.

C.1. EBR-II Detailed Description

The dimensions obtained from engineering drawings were reported in inches and feet. Where referenced in this report, the original dimensions are then followed by their converted values in units of meters or centimeters, in parentheses.

C.1.1. In Vessel

EBR-II was a sodium cooled fast reactor, which was operational from 1964 through 1994. It had a maximum heat output of 62.5 MW (about 20 MW electric). Although initially designed to breed more fuel than it consumed, it was later reconfigured to operate as an irradiation facility where a variety of fuels and structural materials were tested.

The EBR-II consisted of 637 vertical, hexagonally shaped, removable assemblies. As shown in Figure 49, these assemblies were divided into three regions (moving outward from the middle): the core, an inner blanket, and an outer blanket. The number of assemblies within each region varied over the years with changing configurations, due to the experimental nature of the reactor. In the core region were the driver assemblies containing 91 fuel elements each. The fuel was made of enriched uranium metal, alloyed with a small percentage of other elements to improve fuel properties, and clad within stainless steel. Also in the core region were 2 safety and 8 control type assemblies. Both types of control assemblies contained 61 fuel elements, and could be inserted to increase reactor power, or lowered from the core to reduce the reactor power. The control rods were later upgraded to a high worth version, which included a B_4C poison region above the fuel to maximize the reactivity swing. Other assemblies in this region included stainless-steel dummies, half worth drivers, and experimental/instrumentation assemblies. The inner blanket region initially consisted of assemblies which contained depleted uranium, for

breeding new fissile fuel, as well as reflecting neutrons back toward the center of the core. After proving the breeding concept, these were replaced with stainless-steel reflectors more compatible with the goal of an irradiation facility. The outer blanket region consisted almost entirely of depleted uranium assemblies, again for breeding and reflection.

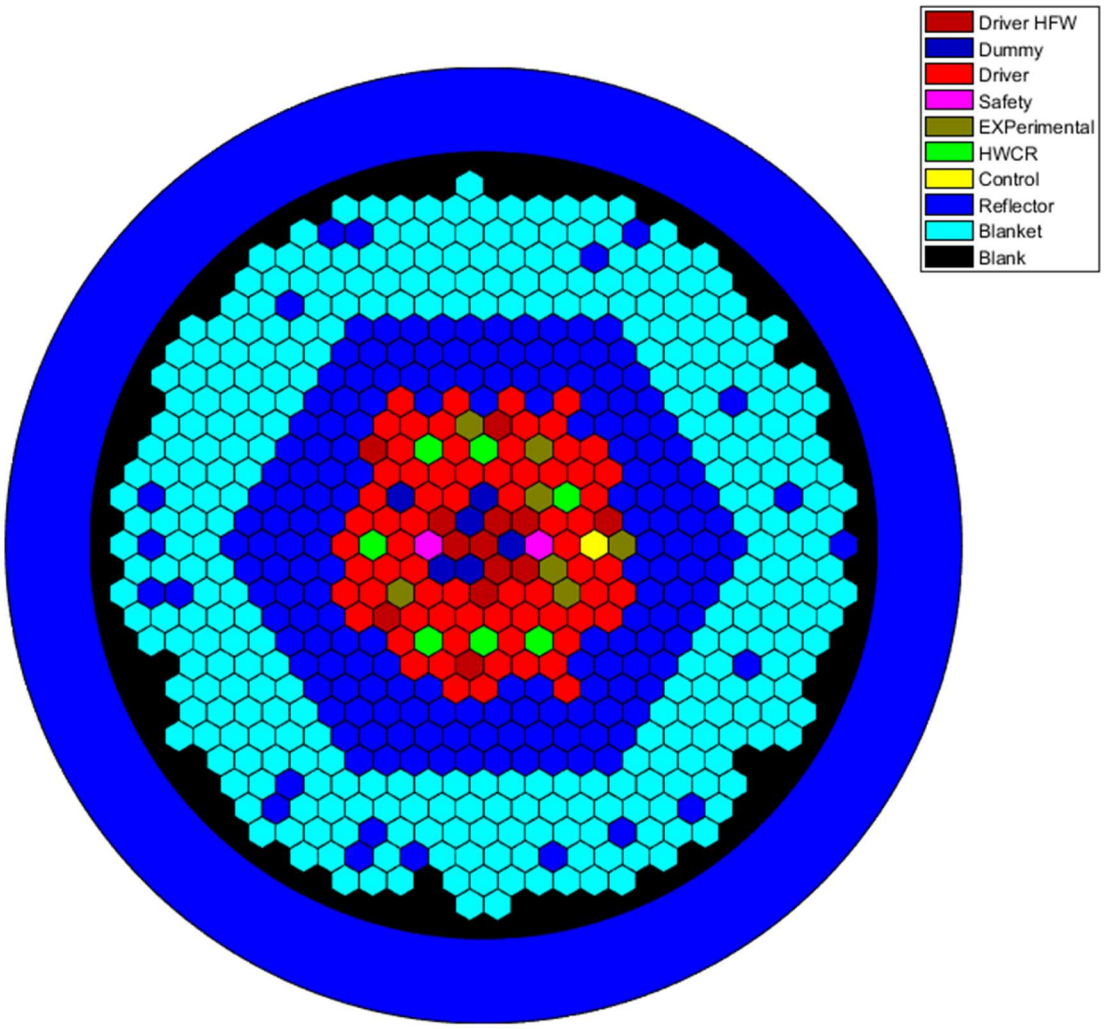

Figure 49. EBR-II Run 138B core configuration.

Assembly locations were denoted by three parameters: row, section, and position within the core. If a horizontal slice of the core was taken, the central assembly would be row 1. Row 2 follows with six assemblies immediately surrounding row 1. Row 3 and on follow the same pattern, with each row growing by six until the row 14 of the core is

reached. The last two rows, 15 and 16, have 66 and 24 assemblies, respectively. The core is then split into six sections labeled A through F, as can be seen in Figure 50. A line is drawn from the central assembly and through each assembly towards the outside edge, in approximately 60-degree angles, which split the core evenly. An assembly position is determined by the number of assemblies from each line of the six sectors. For example, in Figure 51, position 12A05 contains a blanket assembly. To find it on the map, the first section (which corresponds to A) is selected. This is followed by moving up five assemblies on the 180-degree line, starting with the central assembly. This will be assembly 12A01; then move to the right four assemblies starting with 12A05. Figure 51 also shows additional assemblies and their corresponding position references.

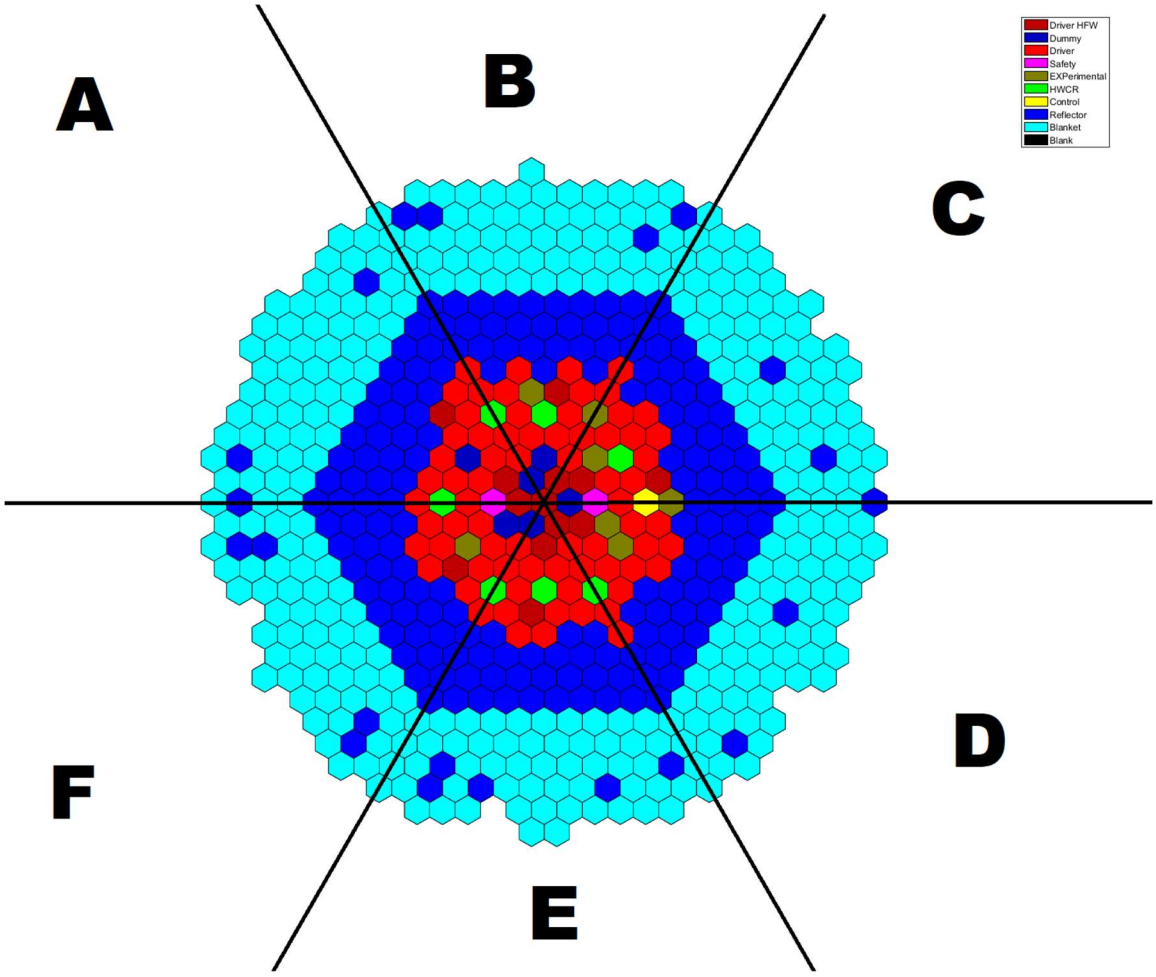

Figure 50. EBR-II core segments.

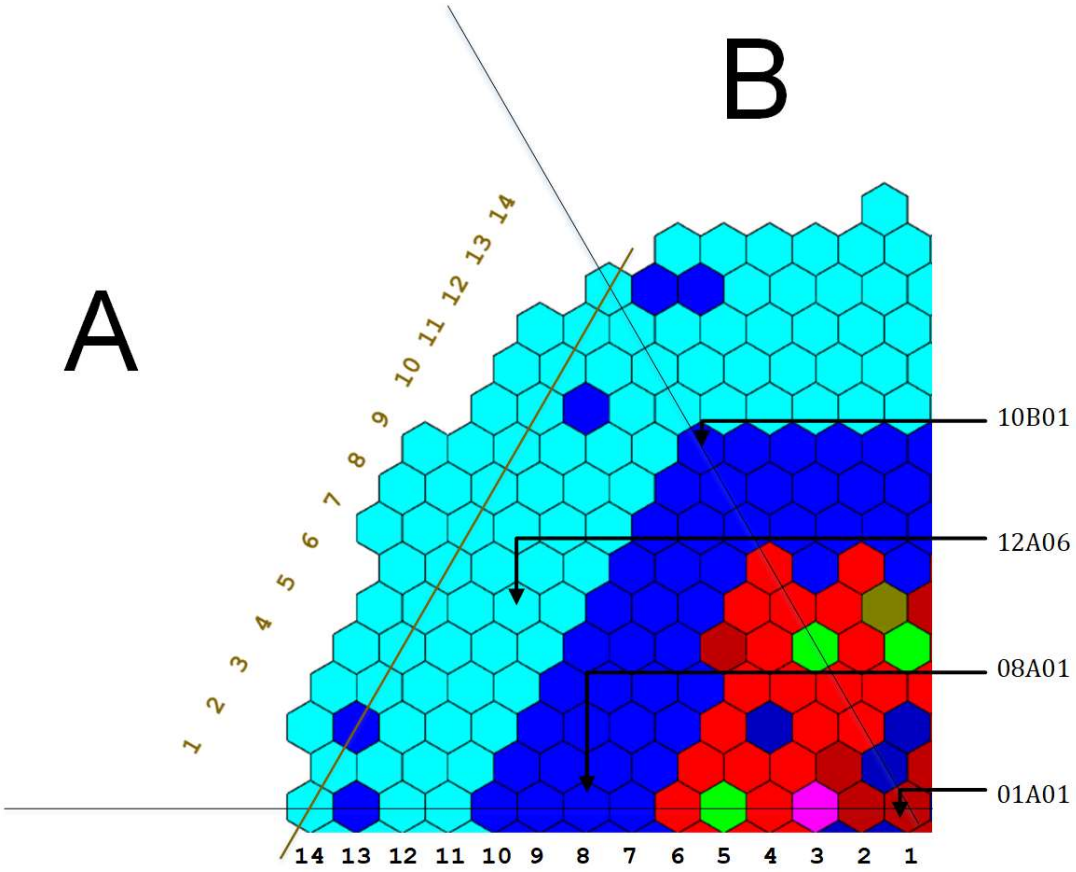

Figure 51. EBR-II position reference.

The assemblies were approximately 92 inches long, although only about 14 inches were uranium fuel. Above and beneath the fuel were neutron reflectors, which began as depleted uranium but were replaced with stainless steel. At the top of each assembly was fixture for assembly removal, and at the bottom was an adapter which fit into a grid plenum support structure. Orifices at the bottom of each assembly allowed sodium coolant to flow upward, with larger holes (and therefore greater flow) for those assemblies in the center region which produced the greatest heat.

Each assembly was fixed in the reactor via a lower adapter which connected into the grid plates, which was supported by a stainless-steel reactor vessel. Surrounding the core region was the stainless-steel reactor vessel which was comprised of a grid plenum

assembly, reactor vessel shell, and the reactor vessel cover. Surrounding the reactor vessel was a radial neutron shield, which was comprised of graphite and borated graphite blocks shown in Figure 53. [18] The entire reactor vessel is in the lower central area of the primary tank and is submerged below ten feet of sodium. Above the reactor vessel was a top cover which also contained a neutron shield. The top cover could be removed when required to enable replacement and handling of assemblies, and contained penetrations to allow for the control rod drive mechanism. Surrounding this reactor vessel was the remainder of the primary sodium tank. The reactor core, two primary coolant centrifugal pumps, and intermediate heat exchanger were all contained within the primary sodium tank which was filled with 337,000 liters of primary sodium coolant. With the pool type design, any leaks within the primary coolant piping would simply drain into the primary coolant pool. While such a leak would impact plant efficiency, no leakage of primary sodium coolant outside the vessel would occur. Heat from the primary coolant was transferred to a secondary sodium loop through a heat exchanger submerged within the primary pool. Thus, heat from the reactor was removed to the secondary sodium loop while minimizing neutron activation of the secondary sodium. Finally, the secondary sodium was used to generate superheated steam for electricity generation.

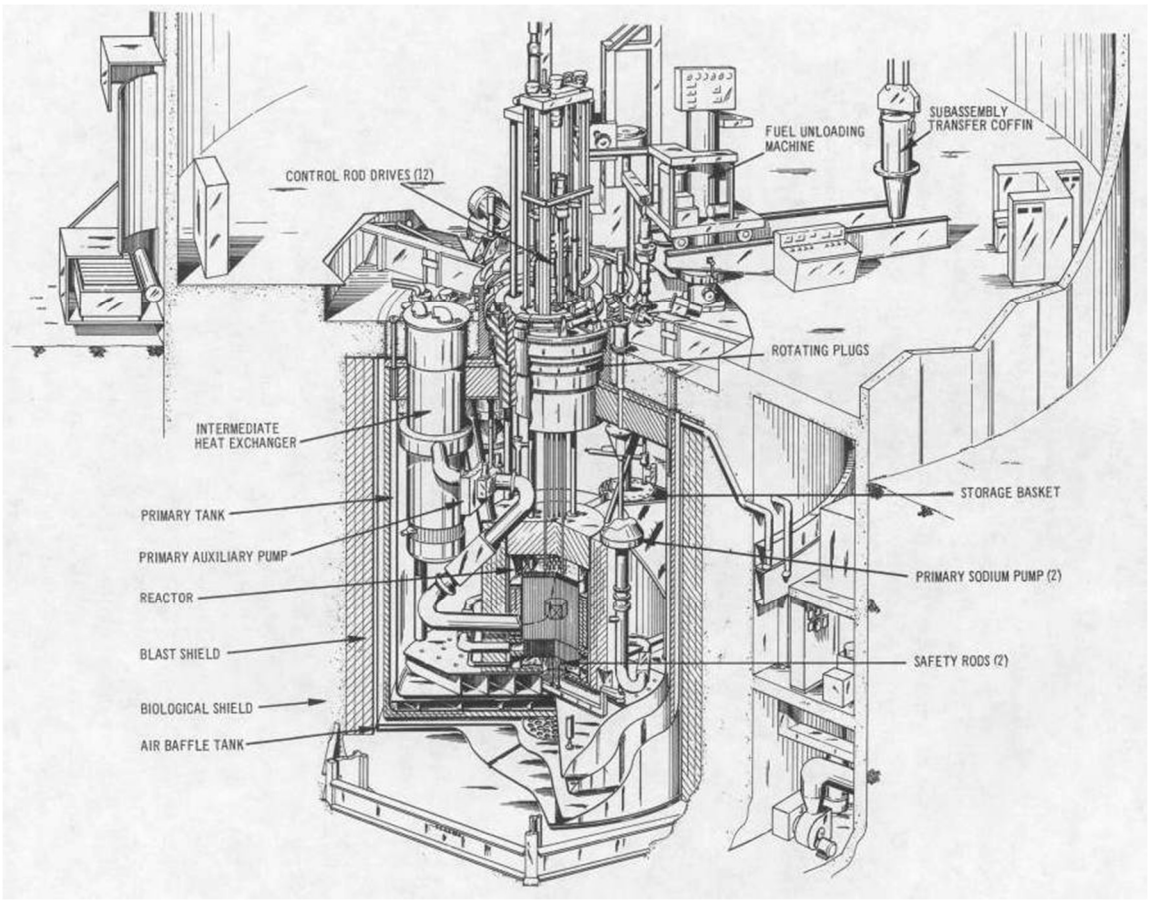

Figure 52. EBR-II cross-sectional view.

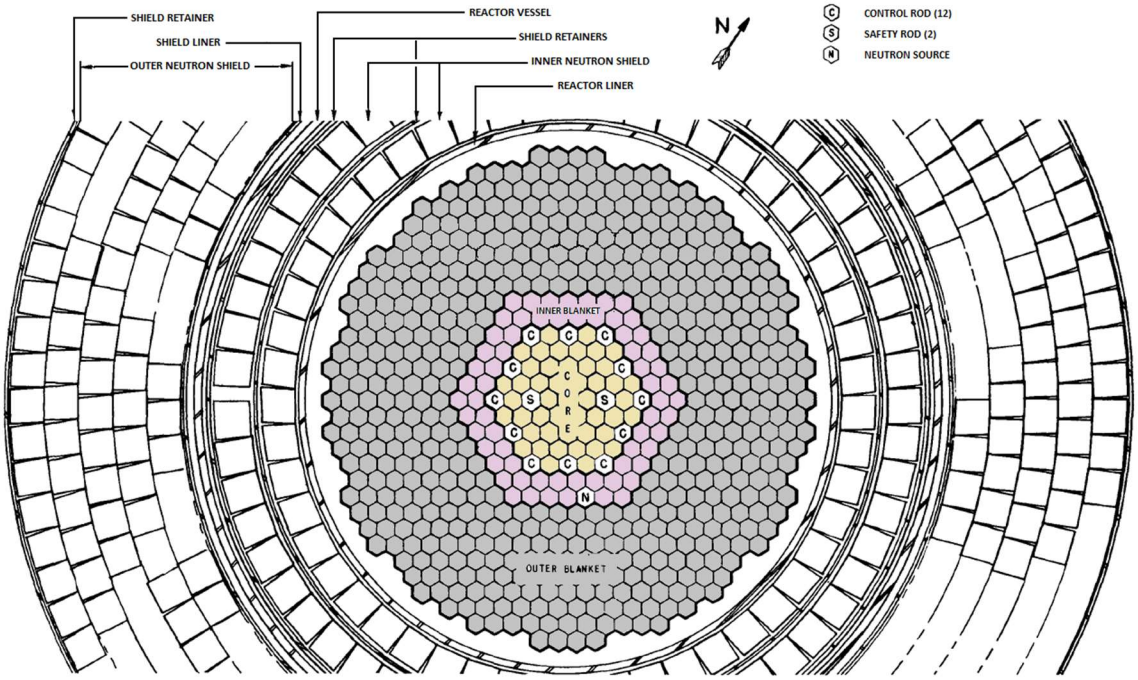

Figure 53. EBR-II reactor core.

C.1.2. Assemblies

The EBR-II reactor assemblies are hexagonally shaped and consists of upper extension, lower extension and core region. There were driver, control, safety, reflector, outer blanket, dummy, instrument, and experiment assemblies.

C.1.3. Core Driver Assemblies

There were two types of driver assemblies used in the EBR-II core, MK-II and MK-IIA. Their overall dimensions did not differ and they have the same composition. The difference between the two are the manufacturer and the fuel element length. MK-II AI was a MK-II assembly manufactured by Atomics International and MK-IIA was manufactured by Argonne National Laboratory with longer fuel elements. The following description applies to both assembly types with the different fuel element length noted. The assembly was comprised of three sections, the upper preassembly, core region and the lower extension. The three sections shared the same stainless steel 304L hexagonal duct with an outside flat-to-flat distance of 2.29 in (5.8166 cm) and a wall thickness of 0.04 ±

0.001 in (0.1016 ± 0.00254 cm). The overall length of the duct was 65.796 ± 0.062 in (167.1218 ± 0.15748 cm). Figure 54 shows a segmented photograph of a driver assembly.

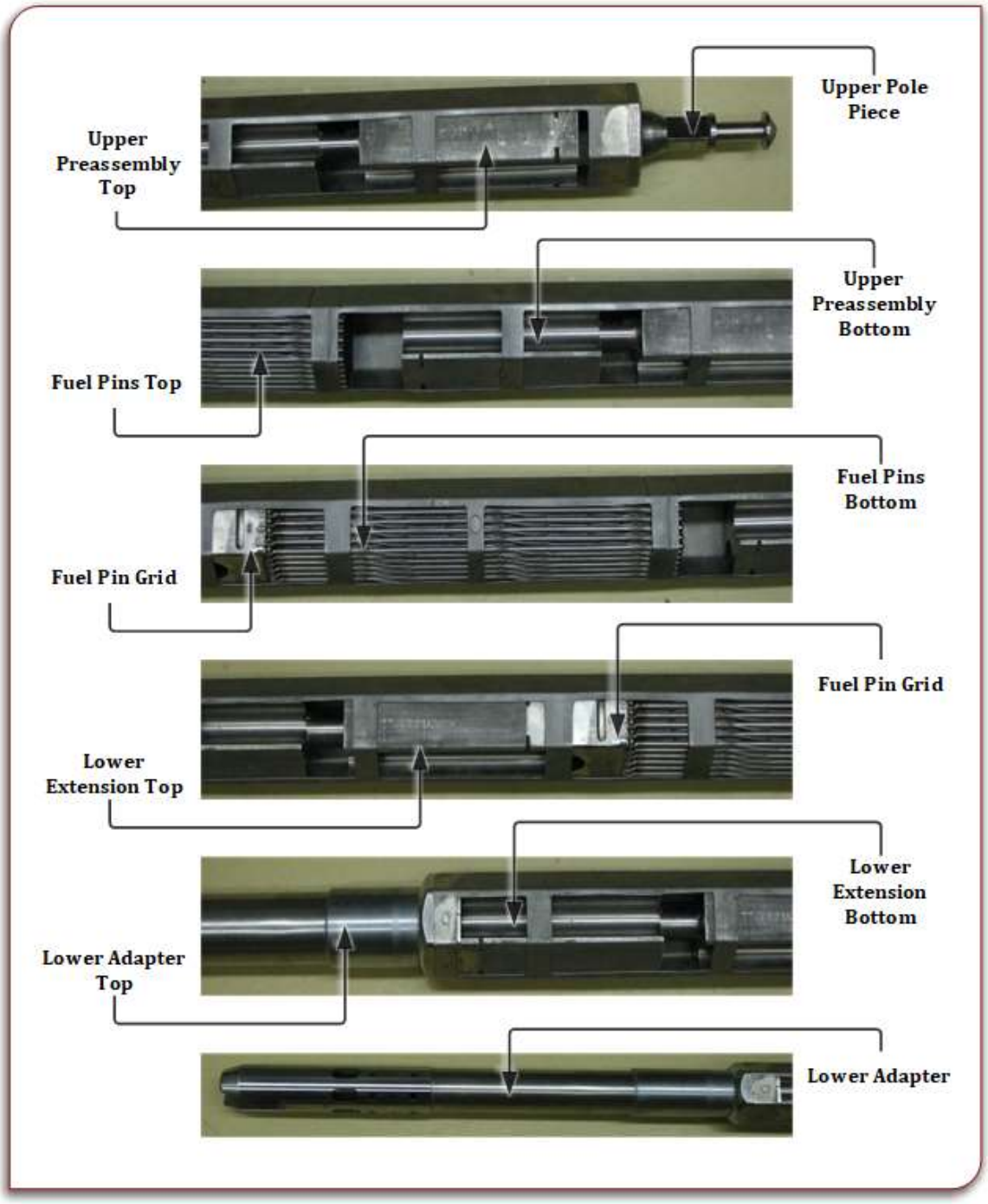

Figure 54. Assembly photos.

The upper preassembly hexagonal duct had a length above the top of the fuel elements to the upper pole piece of 17.312 in (43.9725 cm). The upper pole piece was a

cylindrical piece of stainless steel 304L with a total length of 5.609 in (14.2469 cm) and a diameter of 0.1248 in (0.3169 cm).

The core region hexagonal duct length that contained the fuel elements was 25.94 in (65.8876 cm) long and made of stainless steel 304L. Inside of the core region were 91 fuel elements in a hexagonal layout. The pitch of the elements was 0.2229 in (0.5661 cm).

The MK-II fuel element is shown in Figure 55. Each fuel element consisted of a fuel slug containing 66.72% enriched uranium surrounded by sodium with a gas plenum of helium above the sodium. The diameter of the fuel slug was 0.130 ± 0.003 in (0.3302 ± 0.0076 cm) and had a length of 13.5 ± 0.1 in (34.29 ± 0.254 cm). The fuel slug was surrounded by sodium which filled the void between the outer diameter of the fuel slug and the inner diameter of the cladding. The length of the sodium was 13.75 ± 0.35 in (34.925 ± 0.889 cm). Above the sodium was a helium gas plenum which filled the rest of the internal volume of the cladding. The cladding was made of stainless steel 316 which had an outer diameter of 0.1740 ± 0.0005 in (0.4420 ± 0.00127 cm) and an inner diameter of 0.1500 ± 0.0005 in (0.3810 ± 0.00127 cm). The length of the cladding from the top of the spade to the top of the element for a MK-IIA was 24.3 in (61.72 cm) and for a MK-II/MKIIAI was 24.425 in (62.0395 cm). The cladding had an outer wire wrap with a diameter of 0.0490 ± 0.0005 in (0.1245 ± 0.00127 cm) and a length of 23.820 ± 0.031 in (60.5028 ± 0.07874 cm). The pitch of the wire was 6.000 ± 0.250 in (15.24 ± 0.635 cm).

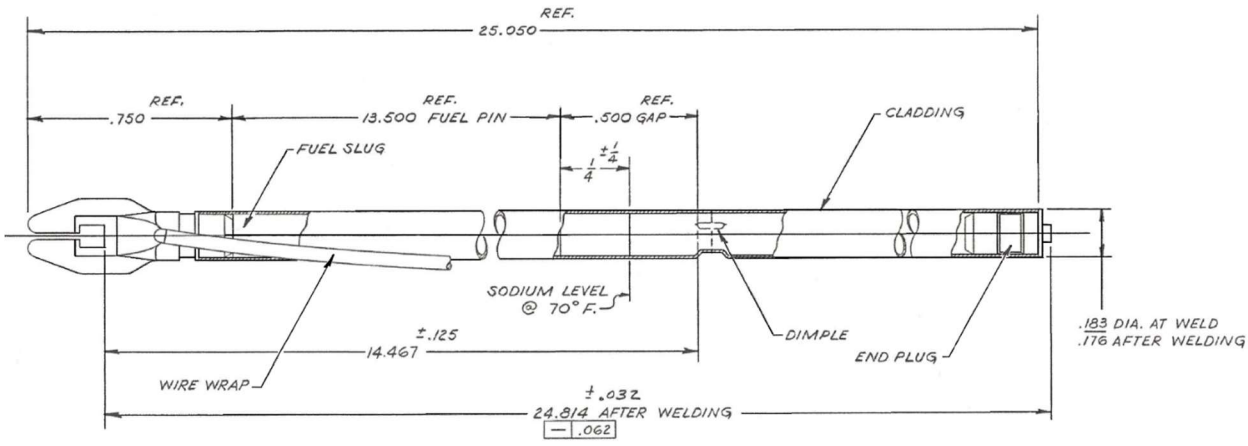

Figure 55. MK-II fuel element.

The hexagonal duct of the lower extension had a length of 24.625 ± 0.031 in (62.5475 ± 0.07874 cm) from the bottom of the fuel slug to the top of reactor grid plate. Inside of the lower extension were stainless steel 304L channels for the sodium to flow up to the fuel elements.

There were half worth core drivers that were geometrically identical to the drivers but had 46 fuel elements and 45 stainless-steel dummy elements. The pattern of loading was starting from the top vertex of the assembly and moving in line 30 degrees down to the right. Every other element was a dummy element starting with a normal element. This pattern continued to the end of the row and then starting again with the next row and subsequent fuel element.

High flow drivers were also used which were geometrically identical to full worth drivers but had more holes drilled into the lower extension and upper preassembly to facilitate higher sodium flow through the assembly.

C.1.4. Control Assemblies

EBR-II had three types of control assemblies, 7 high-worth control rods, 1 control rod, and 2 safety rods. In general, the difference between the two types of control rods was the addition of B_4C followers to the high-worth control rods. The high worth rods were comprised of two sections; a fueled region and a poison region. The combination of the

fueled section and the poison region was 57.725 in (146.6215 cm) long. The poison region, as shown in Figure 57, was 36.093 ± 0.078 in (91.6762 ± 0.1981 cm) long and was comprised of 7 poison elements in a hexagonal array. Movement of the control rods was performed using independent control rod drive mechanisms located above the core, whereas the safety rod drive mechanism was located below the core and moved the rods in tandem. Figure 56 shows the control assemblies movement in reference to the core plane. For the safety rods and control rod, when the rod is inserted the fueled region of the assembly is placed within the core region of the reactor. This is counter typical control rods where insertion means the poison region is placed in the core. The high worth control rods follow a similar nature. When the rod is inserted, the fueled region is placed within the core region of the reactor and in when the high worth control was retracted the poison region of the assembly was within the core region of the reactor.

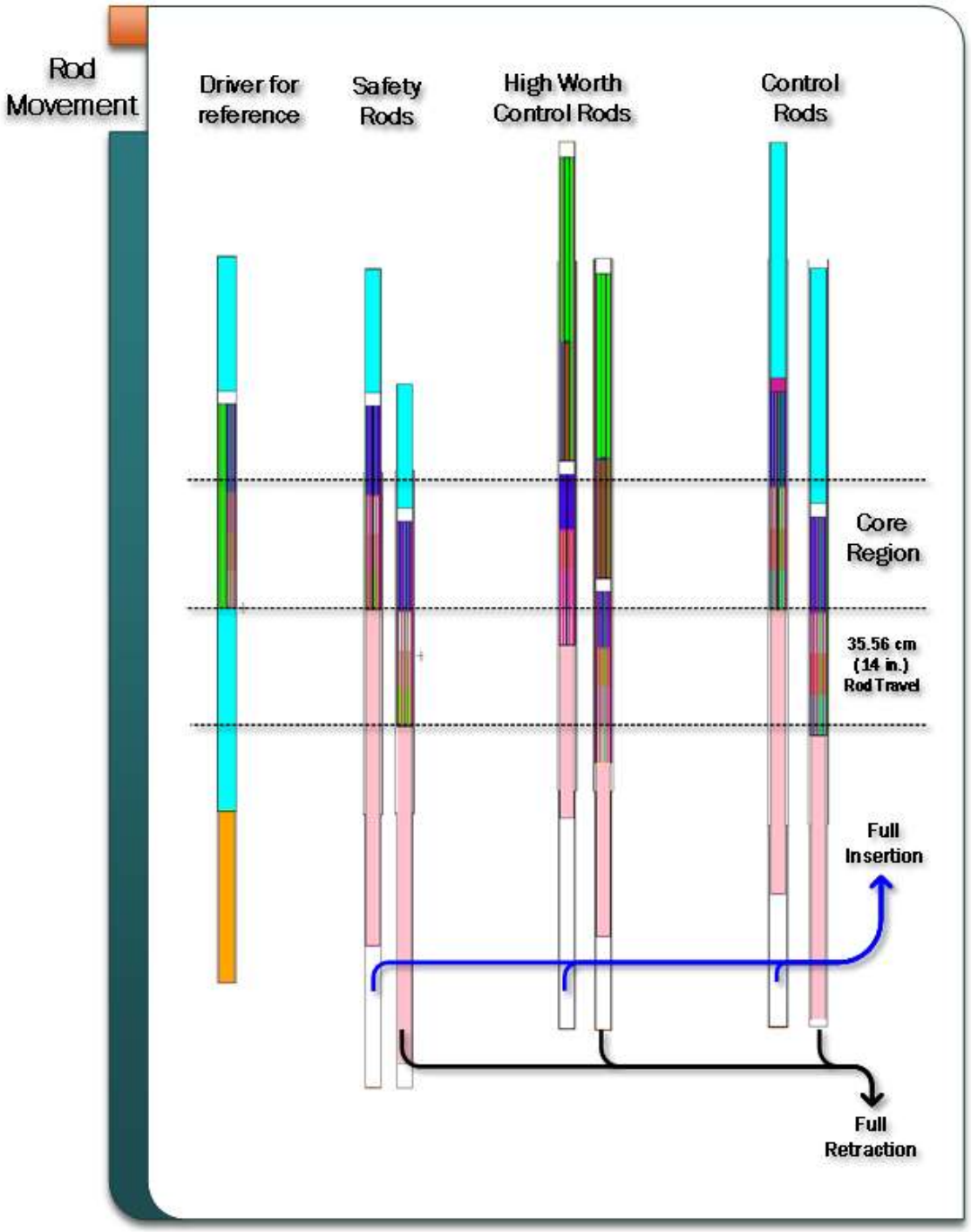

Figure 56. Control type assembly movement.

Each poison element was made up of a steel cap on top, gas, stainless-steel shielding, spring, boron carbide poison, and a lower steel cap all surrounded by steel cladding. The top steel cap was 2.406 in (6.1112 cm) long. The gas took up 9.750 in (24.765

cm) between the upper steel cap and the stainless-steel shield. The stainless-steel shield was 8 in (20.32 cm) with a diameter of 0.5490 in (1.3945 cm). The spring below the stainless-steel shield was 1.375 in (3.4925 cm) long with an inner diameter of 0.465 in (1.1811 cm) and an outer diameter of 0.54 in (1.3716 cm). The B_4C poison was 14 in (35.56 cm) long with a diameter of 1.4728 in (3.7410 cm). The steel cap below it was 0.5 in (1.27 cm). The cladding that surrounded each poison element had an inner diameter of 0.555 (0.554/0.556) in (1.4097 cm) and an outer diameter of 0.625 (0.627/0.623) in (1.5875 cm).

The fueled region was 21.05 in (53.467 cm) long and comprised of 61 MK-IIS fuel elements in a hexagonal array. Each fuel element was made up of a fuel slug of 0.130 ± 0.003 in (0.33020 ± 0.00762 cm) diameter and 13.500 ± 0.100 in (34.290 ± 0.254 cm) long. Each fuel element was filled with sodium to 13.75 ± 0.25 in (34.925 ± 0.635 cm). There was a sodium bond between the fuel slug and the inner diameter of the stainless-steel cladding. The inner diameter of the cladding is 0.1500 ± 0.0005 in (0.381 ± 0.00127 cm) and the outer diameter of the cladding is 0.1740 ± 0.0005 in (0.4420 ± 0.00127 cm).

The assembly lower extension was 28.340 in (71.9836 cm) long in total from the bottom of the fueled section to the top of the reactor grid plate. Below the lower extension was the lower adapter which slid between the upper and lower grid plate and had a length of 1.4081 ft (0.4292 m).

The single EBR-II control assembly was hexagonally-shaped. It was comprised of three sections; an upper extension, a fueled region, and a lower extension; with a combined length of 7.2136 ft (2.1987 m). The upper extension was 1.6862 ft (0.5140 m) long. It was comprised of stainless steel with 6 holes to allow for sodium coolant flow.

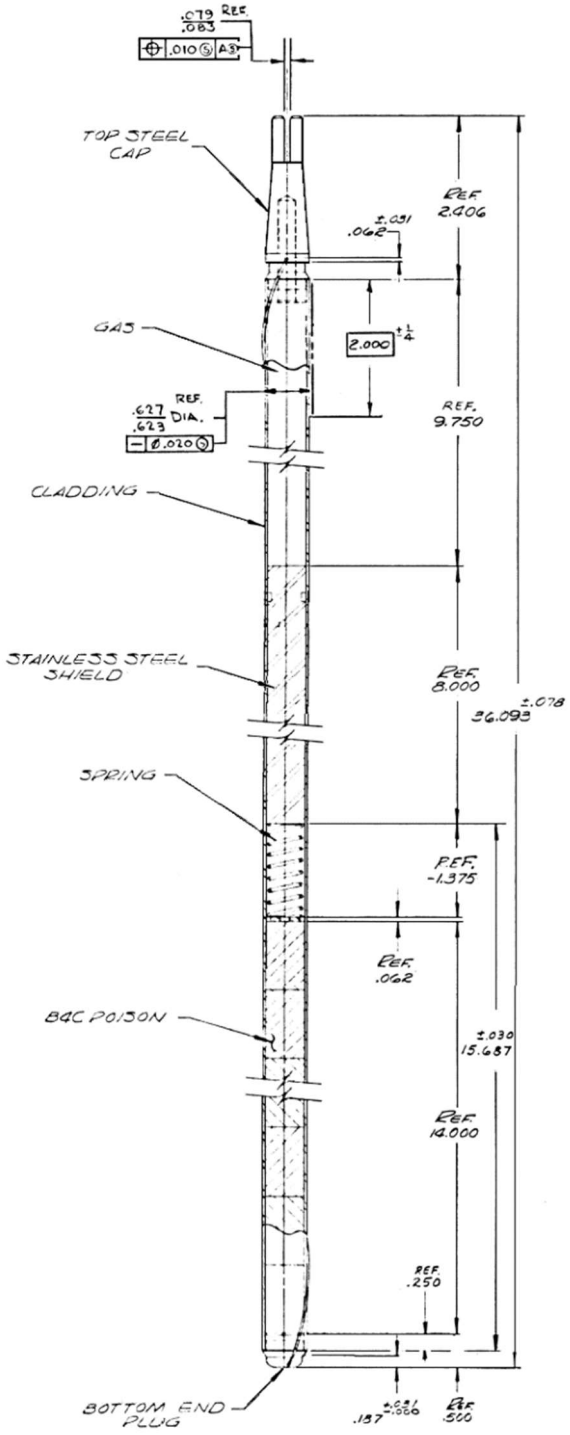

Figure 57. B4C capsule in high worth control rod.

The fueled region was 2.0873 ft (0.6362 m) tall and was comprised of 61 MK-IIA fuel elements in a hexagonal array. The lower extension was 3.2019 ft (0.9759 m) long and

was comprised of a hexagonal portion and a cylindrical portion. The hexagonal duct was 1.3413 ft (0.4088 m) long and the cylindrical portion is 1.4375 ft (0.4382 m) long. The outer diameter of the cylindrical portion is 1.36 in (3.4544 cm).

Safety assemblies were similar to the control assembly with the exception of using MK-II fuel elements rather than MK-IIA fuel elements and the drive mechanism was located at the bottom of the core.

C.1.5. Reflector Assemblies

The EBR-II stainless-steel reflector assemblies were hexagonally-shaped components. They were comprised of two sections; a core region, and a lower extension, with a combined length of 7.6537 ft (2.3328 m).

The core region was 65.796 ± 0.062 in (167.1218 ± 0.1575 cm) and comprised of a stainless-steel core. The stainless-steel core was separated into three separate pieces as shown in Figure 58. The lower piece was 20.5 in (52.07 cm) while the top was 5.5478 in (14.0913 cm).

The lower extension was comprised of a hexagonal portion and a cylindrical portion. The hexagonal portion was 1.8125 in (4.6038 cm) long, with a flat to flat width of 2.29 in (5.817 cm), and the cylindrical portion was 1.7043 ft (51.95 cm) long. The outer diameter of the cylindrical portion changes at points along the length. The cylindrical portion of the top 3.952 in (10.04 cm) had an outer diameter of 1.508 in (3.83 cm). The second portion was 10.5 in (26.67 cm) in length and had an outer diameter of 1.469 in (3.73 cm). The third portion was 2.000 in (5.08 cm) in length and had an outer diameter of 1.469 in (3.73 cm). The final cylindrical portion was 4.000 in (10.16 cm) in length and had an outer diameter of 1.437 in (3.65 cm).

The second stainless-steel reflector type had minor differences only in the lower extension. The second type of reflector had a total lower extension length of 22.315 in (56.68 cm) and was comprised of a hexagonal portion and cylindrical portion. The hexagonal portion was 1.861 in (4.727 cm) long. The cylindrical portion changes along the length. The top cylindrical portion of the lower extension was 3.953 in (10.04 m) long and

had a diameter of 1.726 in (4.384 m). The second portion was 12.5 in (31.75 cm) long and had a diameter of 1.585 in (4.206 cm).

The sectional view of the reflector assembly is shown in Figure 59.

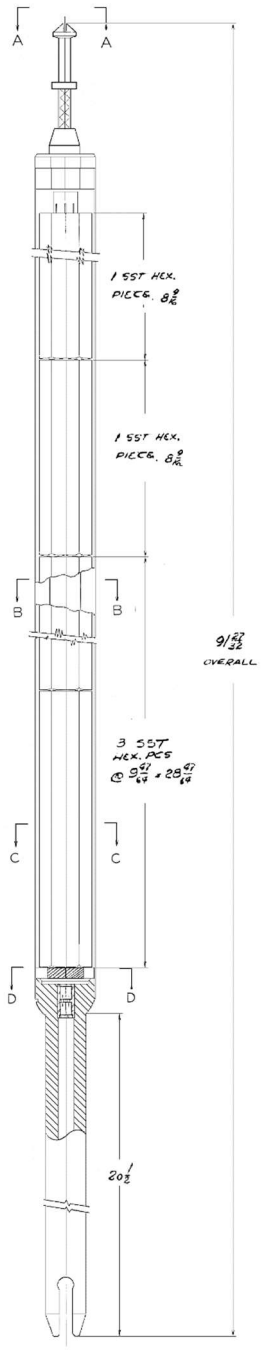

Figure 58. Reflector assembly.

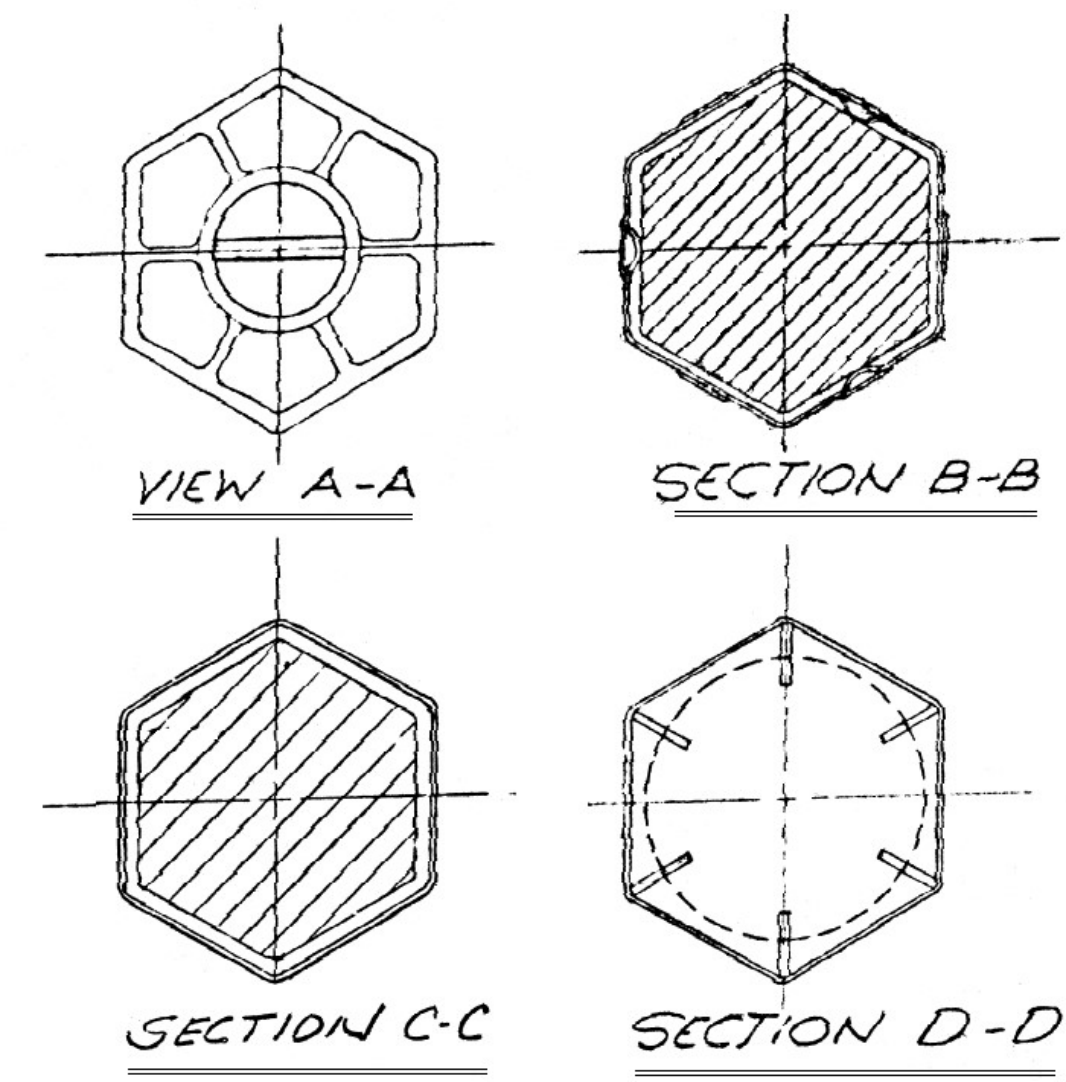

Figure 59. Sectional view of reflector assembly.

C.1.6. Outer Blanket Assemblies

The EBR-II depleted uranium outer blanket assemblies were hexagonally shaped components. The blanket assemblies were comprised of two sections; a core region and lower extension, with a combined length of 7.654 ft (2.3328 m). The lower extension, core region and upper extension had a flat-to-flat distance of of 2.29 in (5.817 cm).

The core region was 5.168 ft (1.5753 m) tall and was comprised of 19 blanket fuel elements. The blanket element slug height was 4.583 ft (1.397 m) with a diameter of 0.433

in (1.100 cm). The fuel elements were filled with sodium up to 4.698 ft (1.4318 m). There was a sodium bond between uranium slug and the inner diameter of the stainless-steel cladding. The inner diameter of the cladding was 0.457 in (1.162 cm). The outer diameter of the cladding was 0.493 in (1.252 cm).

The lower adapter had a height of 24.4 in (61.98 cm) and was comprised of a hexagonal portion and a cylindrical portion. The hexagonal portion was 13.61 in (34.57 cm) height from the top. The first cylindrical portion from the top was 3.953 in (10.04 cm) tall and had a diameter of 1.508 in (3.83 cm). The second portion was 3.949 in (10.03 cm) height and had a diameter of 14.68 in (37.29 cm). The final portion was 6.00 in (15.24 cm) height and had a diameter of 1.437 in (3.65 cm).

C.1.7. Core Dummy Assemblies

The EBR-II core dummy assemblies were hexagonally shaped components. They were comprised of two sections; a core region and a lower extension with a combined length of 7.638 ft (2.3281 m) as shown in Figure 60. The lower extension, core region and upper extension had a flat-to-flat distance of 2.29 in (5.817 cm).

The core region was 5.340 ft (1.6276 m) tall and was comprised of 7 solid stainless-steel elements. The elements were 4.983 ft (1.5189 m) long and had a diameter of 0.785 in (1.994 cm). There was a gap between the bottom of the rods and the upper part of the reactor grid plate 1.653 in (4.199 cm) long. The upper part of the reactor grid plate was 17.181 in (43.64 cm) long.

The lower extension was a total length of 20.437 in (51.91 cm) tall and was cylindrical. The diameter of the lower extension changed in size at certain lengths. The upper portion of the lower extension was 1.936 in (4.917 cm) in diameter and was 4.126 in (10.48 cm) in height. The second portion was 1.750 in (4.445 cm) in diameter and was 8.126 in (20.64 cm) tall. The final portion was 1.875 in (4.763 cm) in diameter and was 8.189 in (20.80 cm) tall. Figure 61 shows the sectional view of the dummy assembly.

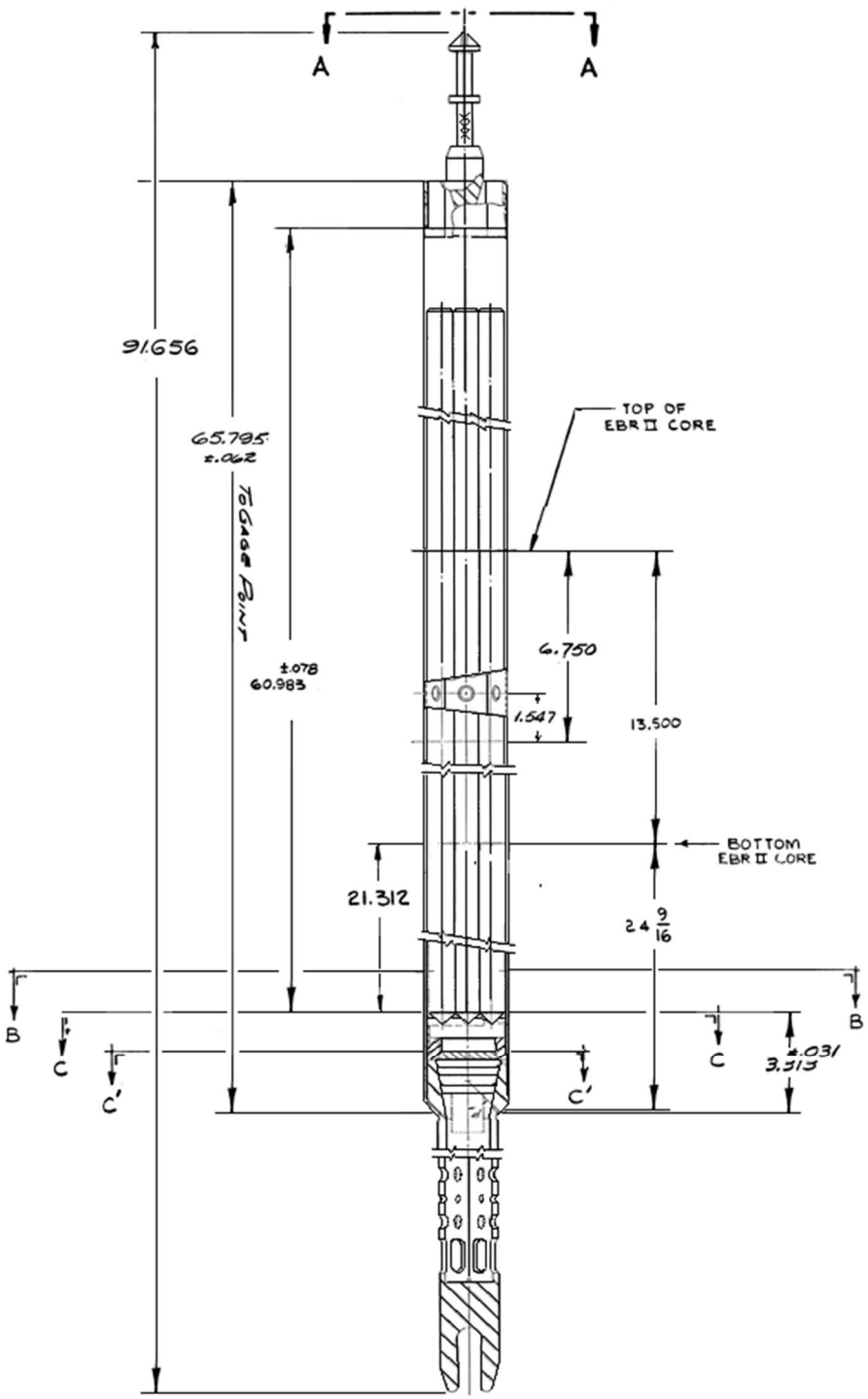

Figure 60. Dummy assembly.

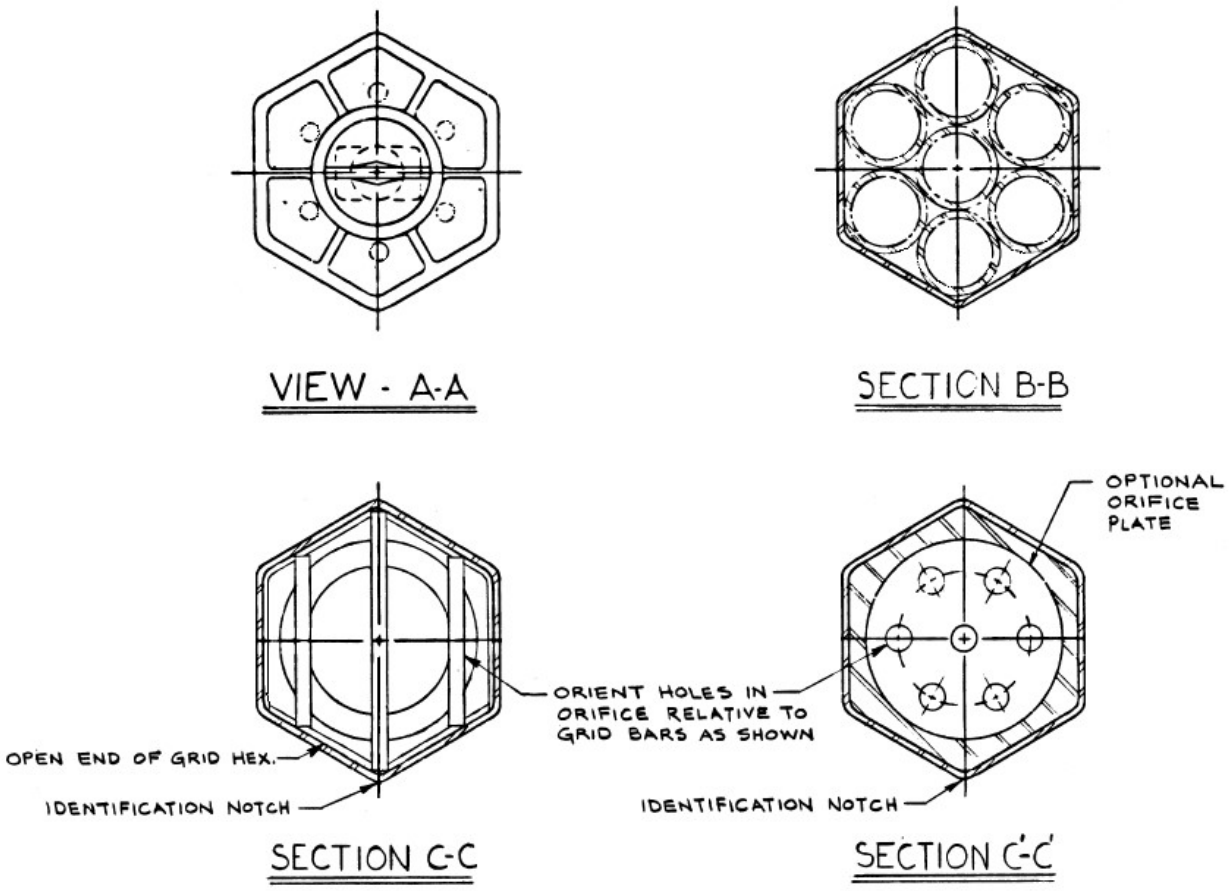

Figure 61. Sectional view dummy assembly.

C.1.8. Instrumented Assemblies

There were two types of instrumented assemblies in EBR-II reactor core during Shutdown Heat Removal Tests. One was a driver type (XX09), and the other was a blanket type (XX10). Both assemblies were designed to fit in the control rod positions. The instrumented assembly consisted of the assembly, an extension tube, and a terminal box. The XX09 instrumented rod was made the same as a standard EBR-II driver. The extension tube connects the assembly to outside the primary tank. [18]

The XX10 is a 19-element unfueled instrumented assembly. [19] Eighteen elements were solid SS316 rods with the remaining position being used as a conduit tube for the passage of instrument leads. Each rod has an outer diameter of 0.881 cm and is 61.2 cm

long. The assembly was in a control rod position. Figure 62 shows the details of the in-core thermocouple positioning.

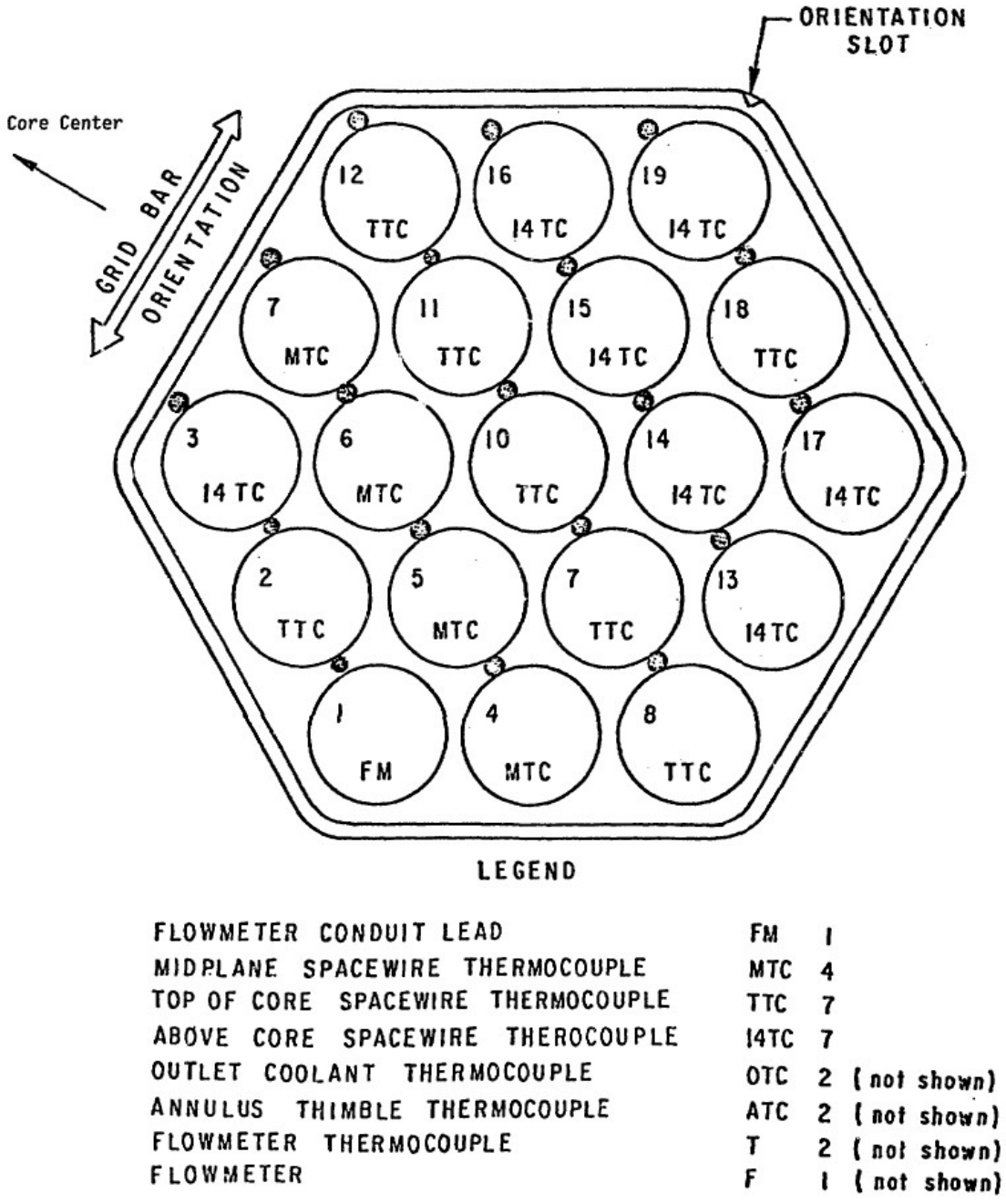

Figure 62. XX10 in-core thermocouple loading.

C.1.9. Experimental Assemblies

There were several experiments loaded into the core during Run 138B. While the experiments constituted special use assemblies, their main difference was material loading. C2776A was a driver that had xenon tags loaded into the fuel elements. X412 and X402A were driver assemblies. XY-16 was a driver that had 61 fuel elements and was placed into a control rod position. X320C was a dummy assembly.

C.1.10. Fuel Element Swelling

Radiation induced swelling affects the metal fuel slug, cladding, and hexagonal flow duct. At the time of manufacture, the 0.25-mm radial gap between the metal fuel slug and the cladding is filled with sodium metal to enhance heat transfer. After ~3.2 at. % burnup, the metal fuel slug swells to the point where it encounters the cladding. [20] Most of the sodium bonding between the metal fuel slug and cladding is pushed up above the top of the metal fuel slug. The initial gap between the metal fuel slug and cladding, along with the swelling property, is part of the overall fuel design. Once the metal fuel slug swells ~30%, interconnection of pores within the slug occurs, which provides a release pathway for fission product gases to the volume above the slug. This gap between the metal fuel slug and the cladding led to fuel lifetimes in the 80,000 to 150,000 MWd per ton HM range. The fuel element cladding also swells as a function of burnup. The cladding diameter swells ~3% at 9.4 at. % burnup. Finally, the hexagonal flow duct also swells because of irradiation.

C.1.11. Material Data

Sodium Coolant

The main reactor vessel and primary cooling systems of EBR-II were all submerged in liquid sodium. This provided many advantages both from a thermodynamics perspective and a neutronic perspective. In 1967 chemical analysis was done on the pool sodium to assess what chemical contaminates and impurities residing in both the primary and secondary loop. The contaminates were grouped into three categories, non-metals, trace metals and radionuclides. Table 16, table 17, and table 18 show the results of the chemical analysis of the sodium.

Table 16. Non-metal Trace Elements in Pool Sodium

Element	Primary Loop Concentration (ppm)	Secondary Loop Concentration (ppm)
C	<2.0	<2.0
Cl	<5.0	<5.0
CN	<0.2	<0.2
H	<0.1	<0.1
N	~0.2	<0.1
O	<2.0	<2.0
S	<2.0	<2.0

Table 17. Metal Trace Elements in Pool Sodium

Element	Primary Loop Concentration (ppm)	Secondary Loop Concentration (ppm)
Ag	0.04	0.01
Al	<0.06	<0.06
Bi	2.0	<0.1
Ca	<0.02	<0.02
Cd	0.08	<0.02
Co	<0.02	<0.02
Cr	<0.02	0.01-0.5
Cu	<0.02	<0.02
Fe	0.1-1.0	0.1-1.0
In	<0.06	<0.06
Mg	<0.005	0.005-0.05
Mn	<0.005	0.005-0.05
Mo	<0.07	<0.07
Ni	<0.04	0.01-0.2
Pb	10.0	0.6
Sn	20.0	<0.3
Zn	<0.06	<0.06

Table 18. Radionuclides in Pool Sodium

Element	Primary Loop Activity, μCi/g	Secondary Loop Activity, μCi/g
^{24}Na	1400	4.0E-2
^{22}Na	5E-2	
^{137}Cs	1.2E-2	
^{131}I	7.0E-4	

Apart from samples pulled from the pool, the sodium composition was not changed significantly from the original filling.

Stainless Steel

EBR-II used two different types of stainless steel, SS316, and SS304L. Most of the internal structure of the reactor was SS304L. After irradiation experiments it was determined that SS316 was more appropriate to use in the cladding of the fuel elements due to fuel cladding interactions. During run 138B, only the fuel cladding was made of SS316, while the remainder of the core used SS304L.

Table 19 lists the minima and maxima composition for the two types of stainless steels.

Table 19. SS304L and SS316 Compositions.

Element	AISI SA SS304L wt.% balance Fe	AISI SA SS316 wt.% balance Fe
C	0.07-0.07	0.07-0.07
Si	0.5-0.5	0.5-0.5
P	0.011-0.011	0.011-0.011
S	0.006-0.006	0.006-0.006
Mn	0.05-0.2	0.05-0.2
Ni	8.0-10.5	10.0-14.0
Cr	18.0-20.0	16.0-18.0
Mo	0	2.0-3.0

Plenum Gas

The inert gas that was primarily used for EBR-II operation was argon; however, the engineering drawings for fuel elements indicate that a mix of 75 wgt. % helium, 25 wgt. % argon was also acceptable. There are no records which indicate what the mix was on a per fuel element basis. The plenum gas was tagged in some fuel elements to identify what fuel element breached during operation without shutting down the reactor. Table 20 is a list of maximum impurities in the plenum gas.

Table 20. Acceptance Criteria for Inert Atmosphere in Plenum Gas.

Element	Maximum Impurity (ppm)
N	5000
O	50
H_2O	50

Beginning of Life Uranium

The fueled assemblies regardless of the type were fueled with similar beginning of life uranium. The uranium itself was 66.72% enriched ^{235}U with total fuel composition consisting of 95.00 ± 1.0 wgt. % uranium and 5.00 ± 0.32 wgt. % fissium. Where fissium

was comprised of elements that were meant to simulate dominant mid-life fission products. Table 22 is a list of the composition of the uranium isotopes. Table 21 is a list of the beginning of life fuel composition with the respective uncertainties of the components. Niobium and tantalum are difficult to analyze separately and therefore were combined in the analysis. The uncertainties were measured analytically using an unknown method. The acceptance criteria allowed for the use of average chemical sample values meaning that elements that breach acceptance can still be allow such that the average of the batch fell within criteria.

The impurities acceptance criteria were based upon exceeding any specific element in table 23 but also state that a total impurity amount exceeding 1600 ppm would be unacceptable.

Table 21. Beginning of Life Fuel Composition

Element	Weight Percent	Weight Percent Uncertainty
Mo	2.44	0.17
Ru	1.94	0.25
Rh	0.28	0.05
Pd	0.19	0.04
Zr	0.085	0.06
Nb + Ta	0.02	0.012
Si	0.054	0.031
U	95.00	1.00

Table 22. Uranium Composition

Isotope	Weight Percent	Weight Percent Uncertainty
^{235}U	66.72	0.50
$^{234}U + ^{236}U + ^{238}U$	33.28	0.50
$^{234}U + ^{236}U$	< 1.0	NA

Table 23. Uranium Impurities List

Element	Flag Level, PPM
Al	400
Be	100
B	100
C	600
Cr	300
Fe	300
Ni	250
N	200
O	200
Th	100
Y	100

Fuel Composition Derivation

The EBR-II Run 138B core configuration used depleted fuel, thus the initial mass uncertainty must be combined with the depletion analysis uncertainty. The initial mass uncertainty is based on discussions with D. L. Porter, an expert associated with EBR-II fuel manufacturing activities, and reported manufacturing data. Nearly 30,000 Mk-II fuel elements were manufactured at ANL-W. According to D. L. Porter, the Mk-II driver slug mass specification was quite broad at 50.7 g to 54.7 g per slug. The slug mass inspection limits were set at 50.7 g to 52.7 g indicating a reduction of 2 g per slug below the upper fuel slug mass specification value noted by D. L. Porter.

An excessively conservative initial mass assumption would be to assume an initial mass uncertainty of ± 4 % based on the 52.7 g ± 2 g fuel slug mass specification stated by D. L. Porter. A somewhat more realistic initial mass uncertainty of ± 2 % could be used based on the 51.7 g ± 1 g inspection limit. However, consideration should be given to the reduction of the fuel slug specification mass range (50.7 g to 54.7 g) to the fuel slug acceptance mass limit (50.7 g to 52.7 g), indicating a manufacturing bias toward lower fuel slug mass values. Additionally, it must be recognized that the manufacturing process involved injection casting of metal slugs which is prone to slug internal and external defects

resulting in reduced slug mass. Manufacturing rejection rates of slugs following shearing slugs to length were approximately 15%. Rejections following shearing were due almost exclusively to internal and external defects which resulted in unacceptably low slug mass values.

Considering the reduced slug mass acceptance limit range, a manufacturing technique which produces slug defects resulting in reduced slug mass, and slug rejections dominated by unacceptably low mass values; a reduced and skewed initial mass uncertainty of -1.0% and +0.5% was selected.

Detailed core-follow depletion analysis for EBR-II was performed in the early 1990s using the Argonne National Laboratory developed REBUS-3 computer code. The REBUS-3 EBR-II model explicitly modeled each assembly as a homogenized unit with three axial depletion zones. The neutronics solution was performed using DIF3D and ENDF/B version V.2 cross-sections. Region dependent cross sections with 9 energy groups were generated for each reactor run. The full core model contained 33 x 33 x 35 mesh intervals and 1360 depletion zones.

Detailed core-follow analysis using REBUS-3 was performed starting with EBR-II Run 130A which started in August 1984. The core-follow analysis continued to EBR-II Run 154F ending in September 1990 for a total of 67 reactor runs including sub runs like Run 138B which occurred in March and April 1986. For each reactor run or sub-run, a single constant power burn step was used. All core configurations changes including control rod positions were included with each depletion calculation. Material compositions at the end of the previous run were carried forward into the next depletion run.

Burnup measurements were determined using chemistry measurements of ^{139}La and ^{148}Nd from destructive analysis of three assemblies. The chemistry measurement uncertainty was estimated to be ± 4%. Additionally, the reactor power measurement uncertainty was estimated to be ± 4%. The agreement between the calculated burnup value and the measured burnup value was therefore limited to ± 6%. That is, there is a 6% uncertainty in the burnup prediction when using the REBUS-3 for EBR-II core-follow depletion analysis. However, the mass uncertainty component due to the burnup calculation

is significantly mitigated by the fact most of the initial heavy metal atoms do not undergo fission. For example, if the burnup was 10%, and the initial mass uncertainty was 1%, the post burnup mass uncertainty would be 1.06%. For EBR-II Run 138B, the maximum driver assembly burnup was 6.08% and the average burnup was just 1.64%. Since the burnup contribution to the overall mass uncertainty is small compared to the initial mass uncertainty, a conservative 10% burnup assumption was made. Thus, when the mass uncertainty associated with the burnup is combined with the skewed initial mass uncertainty estimate, identified above; the EBR-II Run 138B mass uncertainty is -1.06% and +0.56%. The blanket assembly initial mass uncertainty and burnup mass uncertainty is not known. However, the blanket assembly burnup values are all trivial compared to driver assemblies. The driver assembly related mass uncertainty was applied to the blanket assembly mass uncertainty based on engineering judgement.

Element Bond

The element bond was the sodium that surrounded the fuel slug. The oxygen and water content were of importance. The manufacturer was given the instruction that "If there is any question at any time that the sodium in the equipment has been contaminated with oxygen or H_2O or impurities beyond allowable limits, that sodium shall be disposed of or repurified." [21]

Table 24 were the limits to the impurities acceptable in the element bond sodium.

Table 24. Sodium Impurities List

Element	Flag Level, PPM
Al	400
Be	100
B	100
C	600
Cr	300
Fe	300
Ni	250
N	200
O	200
Th	100
Y	100

Boron Carbide

B_4C was used as a poison in the high worth control rod. The only information reported as to its composition is from the engineering drawing. It states that the ^{10}B content per 14 in stack must be 18.04 ± 0.500 g. It was also reported that the boron was naturally enriched.

No impurities or acceptance criteria were reported.

Temperature Data

Bulk Sodium Temperature

The critical stamp located in the reactor operator's logbook state the bulk experiment temperature to be 650.6°F (616.82 K). Figure 63 is a scan of the microfilm logbook entry taken during critical of Run 138B.

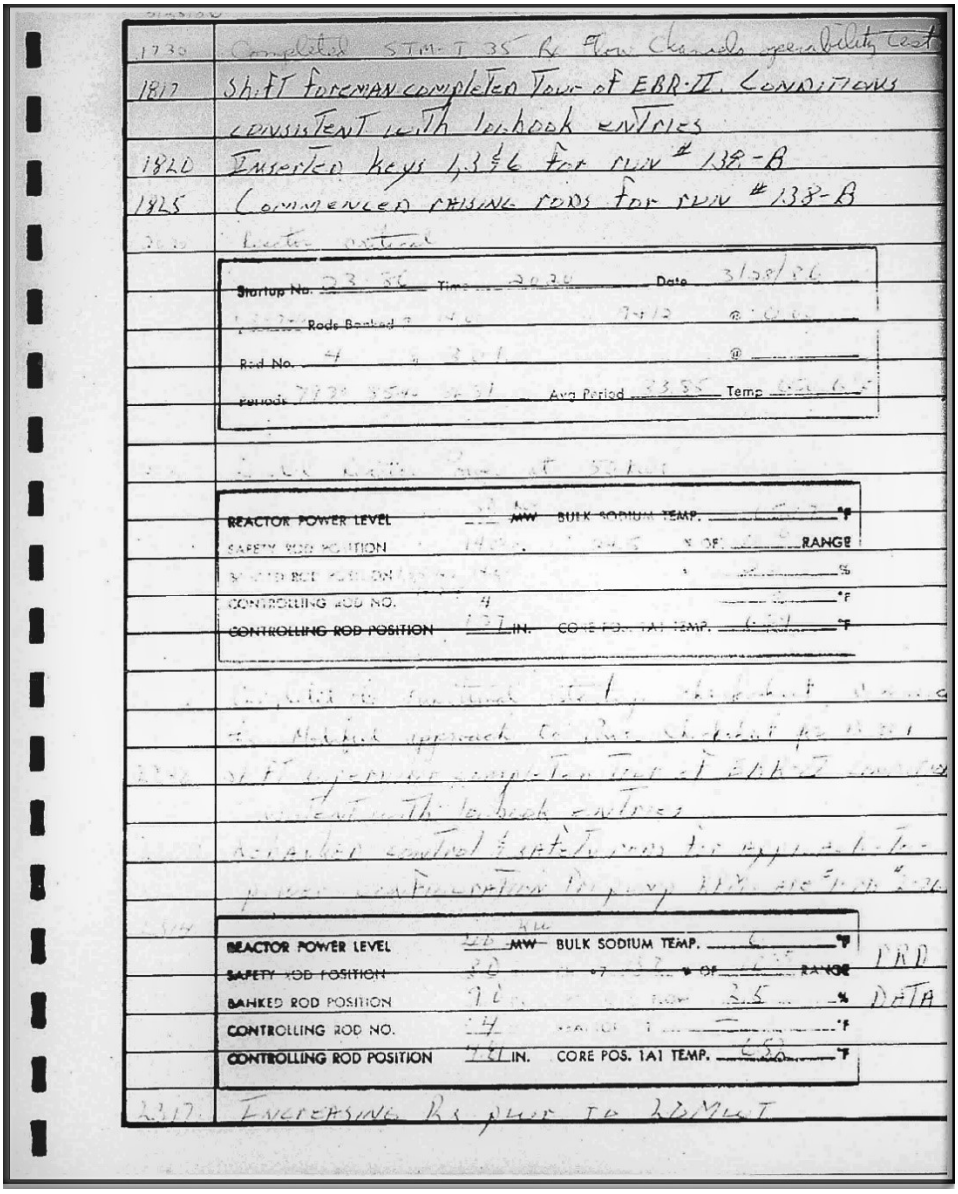

Figure 63. Run 138B critical stamp logbook entry.

Table 25 shows a list of the negative reactivity coefficients and their respective values. [2]

Table 25. Thermal Expansion Coefficients Used for the Safety Analysis of EBR-II Run 138B

Calculated Negative Reactivity Component	Values Used in Analysis (10^{-4} \$/°C)	Estimated Uncertainty (%)
Upper grid plate expansion	-14.5	20
Driver sodium and steel expansion	-14.1	7
Thermal bowing	+9.8	25
Control-rod bank extension	-7.16	5
Axial upper reflector sodium expansion	-6.32	7
Axial lower reflector sodium expansion	-6.32	7
Driver fuel expansion	-4.91	11
Radial reflector sodium expansion	-2.2	-
Doppler effect	-0.671	10

C.1.12. Critical Rod Heights

For run 138B, the control rods positions can be found in Table 26.

Table 26. Critical Rod Heights of Run 138B

Control Rod	Height (inch/cm)
03D01 Safety	14.0 / 35.56
03A01 Safety	14.0 / 35.56
05C03 HWCR	0.0 0.0
05D01 Control	14.0 / 35.56
05E01 HWCR	14.0 / 35.56
05E03 HWCR	3.01 / 7.65
05F01 HWCR	14.0 / 35.56
05A01 HWCR	14.0 / 35.56
05B01 HWCR	0.0 / 0.0
05B03 HWCR	14.0 / 35.56

C.2. EBR-II Part Photographs

C.2.1. Subassembly Cut-away

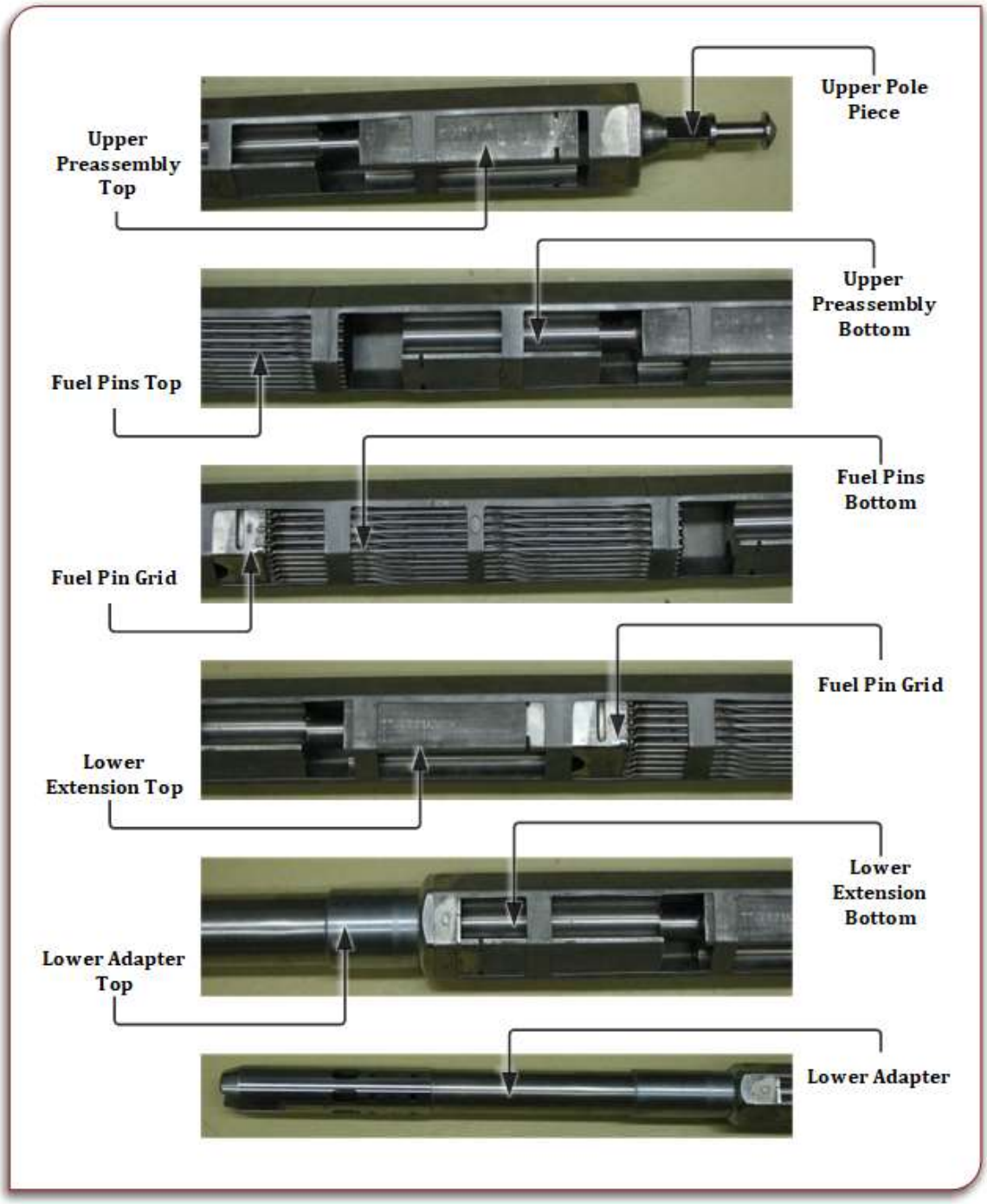

Figure 64. EBR-II subassembly annotated cut-away photos.

C.2.2. Lower Extension

Figure 65. EBR-II subassembly lower extension.

C.2.3. Lower Adapter

Figure 66. EBR-II lower adapter.

C.3. EBR-II Benchmark MCNP Model Figures

Figure 67 through figure 71 are MCNP XY plots of EBR-II run 138B benchmark model. The different colors represent 1400 materials.

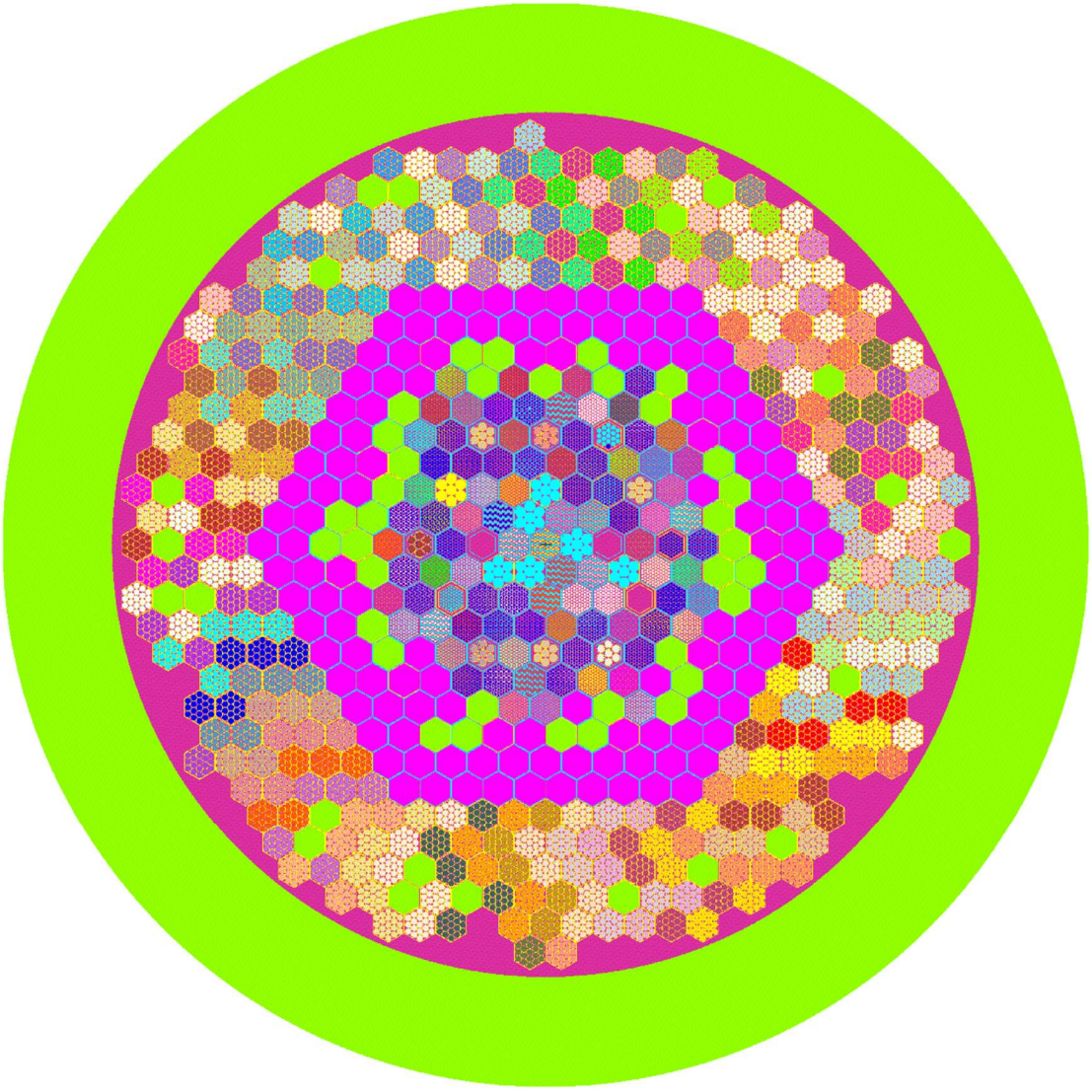

Figure 67. EBR-II run 138B MCNP figure at Z = 60 cm.

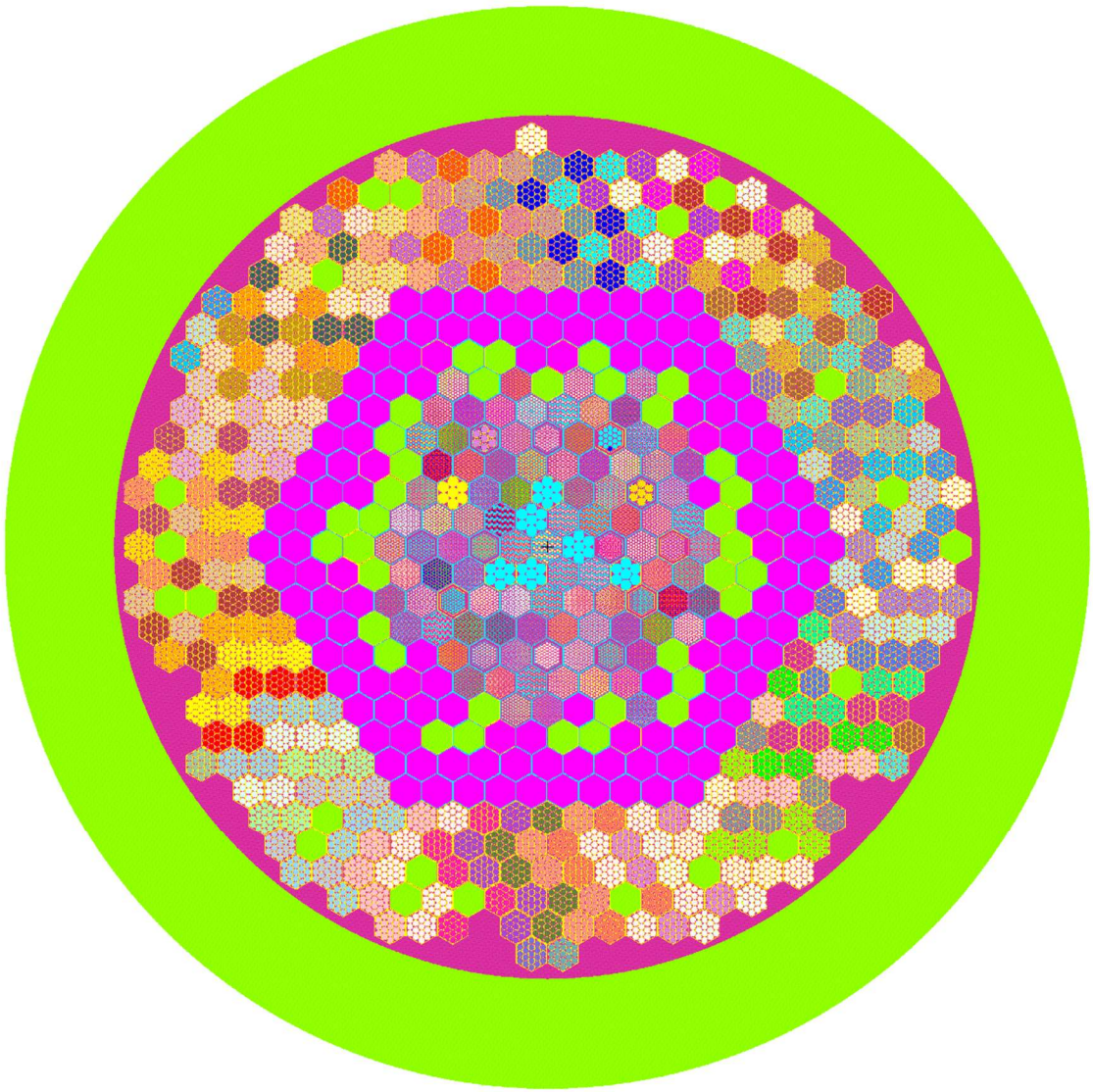

Figure 68. EBR-II run 138B MCNP figure at Z = 15 cm.

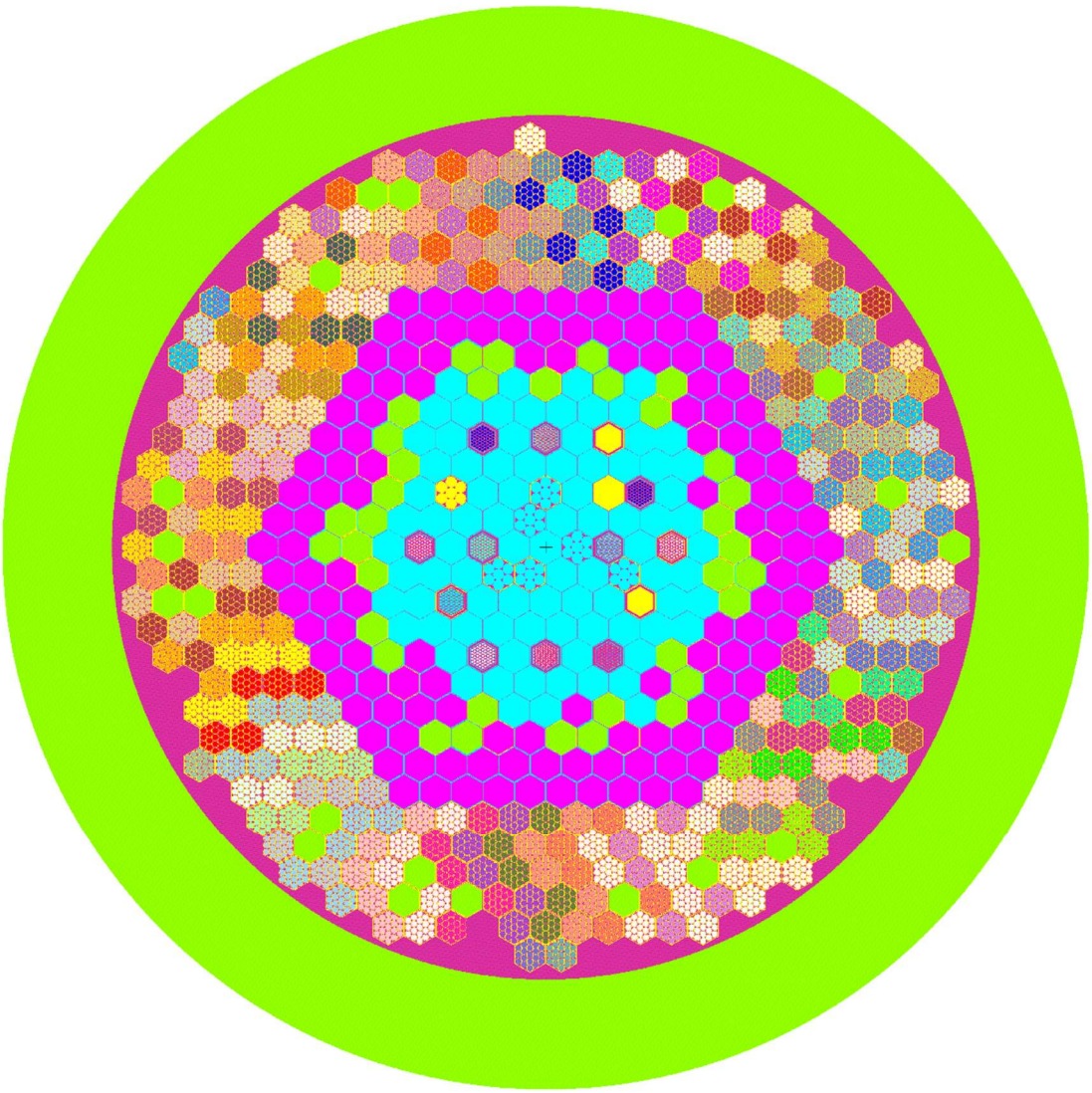

Figure 69. EBR-II run 138B MCNP figure at Z = 0 cm.

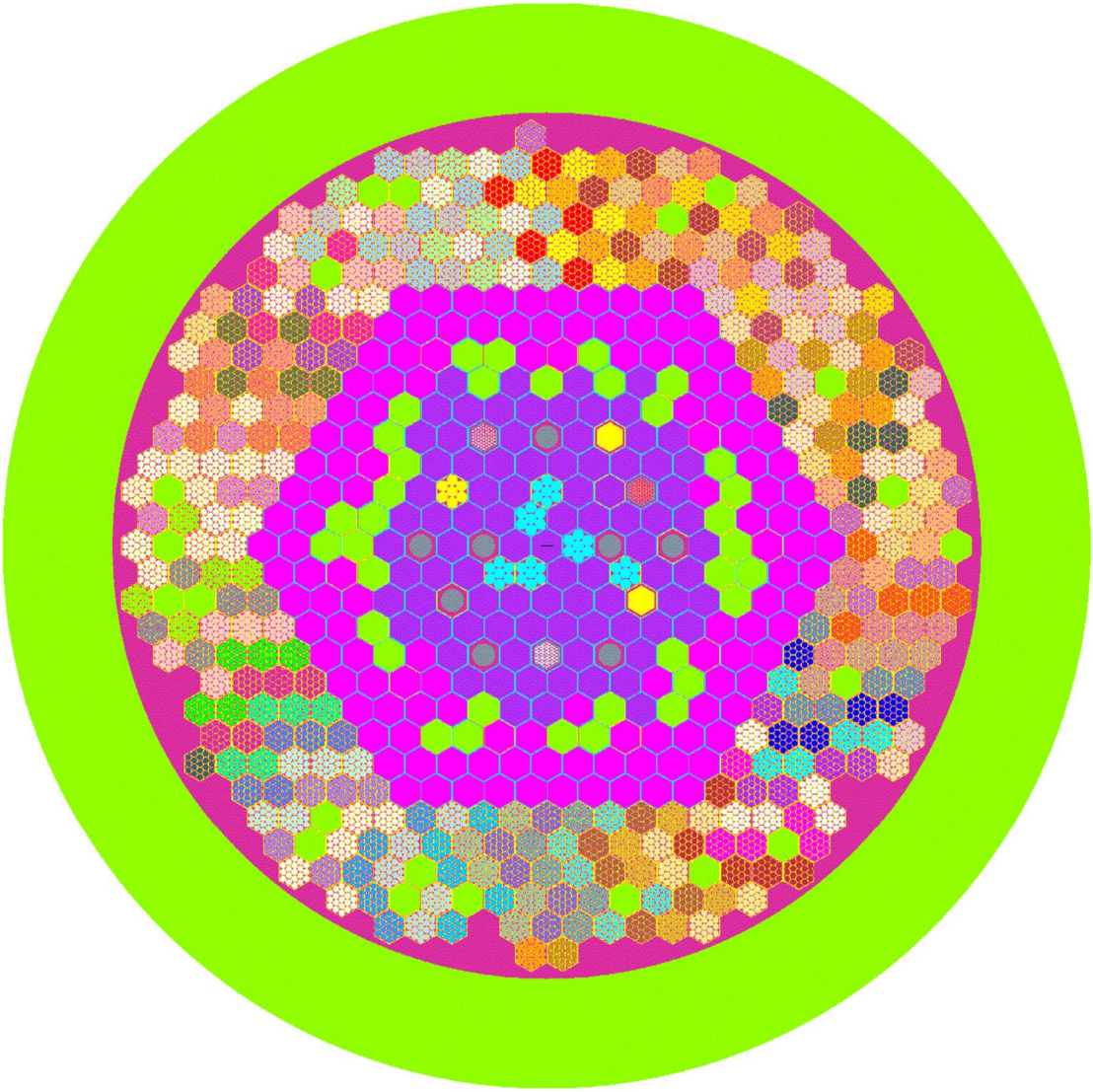

Figure 70. EBR-II run 138B MCNP figure at Z = -30 cm.

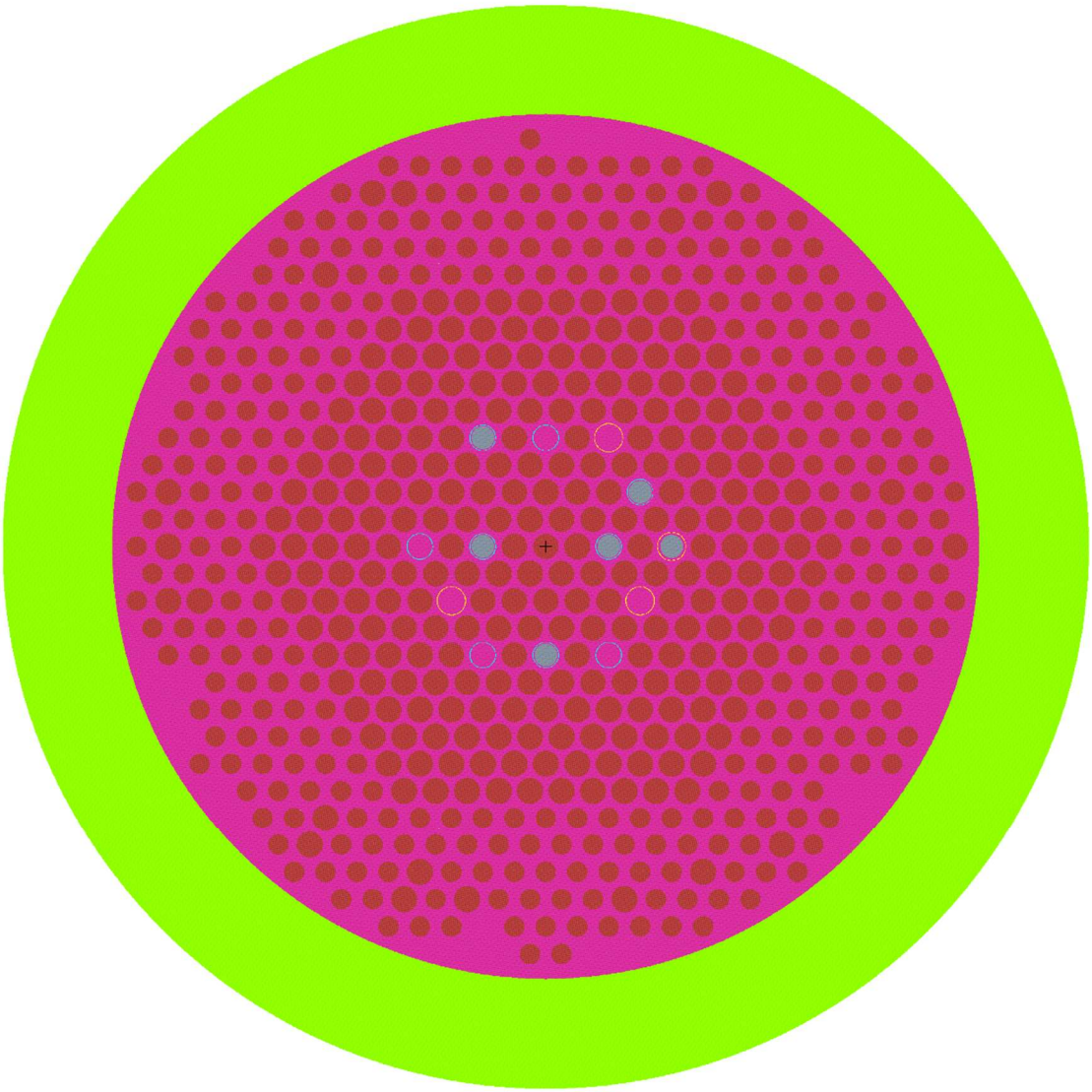

Figure 71. EBR-II run 138B MCNP figure at Z = -80 cm.

APPENDIX D KATANA EFFECT MODEL INPUT DATA

D.1. EBR-II Katana Effect Material Properties

Temperature dependent material properties were required to be used for the EBR-II model. Table 27 through Table 42 show the material properties for all material used in the model.

D.1.1. Sodium Material Properties

Table 27. Sodium Density Properties

Temperature (°K)	Density (kg/m^3) [22]
250	953.91
300	944.32
371	930.88
400	919
500	897
600	874
650	863.05
700	852
800	828
900	805
1000	781
1100	756
1200	732
1300[a]	708
1400[e]	684

[a] These data were extrapolated using the other temperatures and densities and are not a part of the original reference.

Table 28. Sodium Instantaneous Coefficient of Thermal Expansion

Temperature (°C)	Thermal Expansion Coefficient ($^{o}C^{-1}$) [22]
-23.15	7.42e-5
26.85	7.56e-5
97.85	7.76e-5
126.85	8.03e-5
226.85	8.33e-5
326.85	8.67e-5
376.85	8.83e-5
426.85	9.03e-5
526.85	9.4e-5
626.85	9.83e-5
726.85	1.03e-4
826.85	1.09e-4
926.85	1.15e-4

Table 29. Sodium Thermal Conductivity

Temperature (°C)	Thermal Conductivity $\left({}^{W}/_{m \cdot C}\right)$ [22]
-23.15	99.81
26.85	95.66
97.85	89.44
126.85	87.22
226.85	80.09
326.85	73.7
376.85	70.77
426.85	68
526.85	62.9
626.85	58.34
726.85	54.24
826.85	50.54
926.85	47.16

Table 30. Sodium Specific Heat

Temperature (°C)	Specific Heat $\left(J/_{kg\ \cdot C}\right)$ [22]
-23.15	1445.5
26.85	1421.2
97.85	1383
126.85	1372
226.85	1334
326.85	1301
376.85	1288
426.85	1277
526.85	1260
626.85	1252
726.85	1252
826.85	1261
926.85	1279

D.1.2. Stainless Steel 304

A temperature dependent density is not required because in FEA, it is only used to determine the mass. Ductility, strength, displacement are determined by other parameters.

Table 31. SS304 Density

Temperature (°C)	Density $\left(kg/_{m^3}\right)$ [23]
23	8000

Table 32. SS304 Instantaneous Coefficient of Thermal Expansion

Temperature (°K)	Thermal Expansion Coefficient (K^{-1}) [23]
250	1.70e-5
373.15	1.73e-5
588.15	1.78e-5
923.15	1.87e-5
1050[a]	1.89e-5
1150[f]	1.91e-5
1250[f]	1.93e-5
1350[f]	1.95e-5

Table 33. SS304 Elasticity Properties

Property	Value [23]
Young's Modulus	1.93e11
Poisson's Ratio	0.29
Bulk Modulus	1.5317e11
Shear Modulus	7.4806e10

Table 34. SS304 Tensile Properties

Property	Pressure (Pa) [23]
Tensile Yield Strength	2.15e8
Tensile Ultimate Strength	5.05e8

[a] These data were extrapolated using the other temperatures and densities and are not a part of the original reference.

Table 35. SS304 Thermal Conductivity

Temperature (°C)	Thermal Conductivity $\left({}^{W}/_{m \cdot C}\right)$ [23]
-23.15	14.6
100	16.2
500	21.5
900	26.774
1300	32.054

Table 36. SS304 Specific Heat

Temperature (°C)	Specific Heat $\left({}^{J}/_{kg \cdot C}\right)$ [23]
100	500

D.1.3. Stainless Steel 316

Table 37. SS316 Density

Temperature (°C)	Density $\left({}^{kg}/_{m^3}\right)$ [23]
23	8000

Table 38. SS316 Instantaneous Coefficient of Thermal Expansion

Temperature (°C)	Thermal Expansion Coefficient (K^{-1}) [23]
-23.15	1.59e-5
100	1.6e-5
315	1.62e-5
540	1.75e-5
776.85	1.9e-5
876.85	1.99e-5
976.85	2.1e-5
1076.85	2.2e-5

Table 39. SS316 Elasticity Properties

Property	Value [23]
Young's Modulus	1.9734e11
Poisson's Ratio	0.27
Bulk Modulus	1.43e11
Shear Modulus	7.7693e10

Table 40. SS316 Tensile Properties

Property	Pressure (Pa) [23]
Tensile Yield Strength	2.4e11
Tensile Ultimate Strength	5.5e8

Table 41. SS316 Thermal Conductivity

Temperature (°C)	Thermal Conductivity $\left({}^{W}/_{m \cdot C}\right)$ [23]
100	16.3

Table 42. SS316 Specific Heat

Temperature (°C)	Specific Heat $\left(J/_{kg\ \cdot C}\right)$ [23]
100	500

D.2. EBR-II Thermal Analysis Power Data

Table 43 was the power data used in the ANSYS model that came from an IAEA thermal hydraulics benchmark of EBR-II run 138B. [13]

Table 43. Heat Flux per Subassembly

EBR-II Position	Power (KW)
01A01	389.97
02C01	387.43
02E01	397.44
02A01	358.87
03C01	753.28
03C02	389.59
03D01	465.64
03D02	354.87
03E01	696.97
03F01	705.20
03A01	462.87
03A02	368.94
03B01	687.49
04C01	679.73
04C02	716.83
04C03	704.97
04D01	654.79
04D03	710.72
04E01	668.48
04E02	653.11
04E03	690.66
04F01	698.69
04F02	715.99

Table 43. Heat Flux per Subassembly Cont'd

EBR-II Position	Power (KW)
04F03	700.68
04A01	651.74
04A02	691.03
04A03	650.36
04B01	714.02
04B02	722.83
04B03	694.755
05C02	633.87
05C03	437.16
05C04	659.47
05D01	393.62
05D02	607.93
05D03	406.76
05D04	618.81
05E01	408.86
05E02	625.21
05E03	427.55
05E04	598.89
05F01	412.75
05F02	614.57
05F04	573.5
05A01	411.63
05A02	630.39
05A04	630.56
05B01	427.92
05B02	619.51
05B03	449.53

Table 43. Heat Flux per Subassembly Cont'd

EBR-II Position	Power (KW)
05B04	642.79
06C01	560.25
06C02	585.62
06C03	551.06
06C04	585.85
06C05	303.08
06D01	535.12
06D02	564.45
06D03	588.95
06D04	599.83
06D05	600.88
06E01	552.08
06E02	589.6
06E03	591.48
06E04	311.24
06E05	570.39
06F01	567.42
06F02	532.26
06F03	307.57
06F04	571.36
06F05	555.39
06A01	523.18
06A02	576.45
06A03	572.95
06A04	548.24
06A05	567.54
06B01	553.05

Table 43. Heat Flux per Subassembly Cont'd

EBR-II Position	Power (KW)
06B02	586.69
06B03	564.29
06B04	316.21
06B05	580.45
07C01	512.46
07C03	538.12
07C04	533.55
07D03	550.76
07D04	531.92
07E01	474.92
07E04	524.38
07E05	500.74
07F04	513.86
07F05	522.29
07A05	270.47
07A06	480.37
07B01	484.86
07B03	515.16
07B05	532.54

D.3. Other EBR-II Thermal Analysis Input Data

Table 44. Outer Duct Convective Heat Transfer Coefficient

Temperature (°C)	Heat Transfer Coefficient $\left(W/_{m^2 \cdot C}\right)$
-23.15	5.9e+013
26.85	5.66e+013
97.85	5.29e+013
126.85	5.16e+013
226.85	4.74e+013
326.85	4.36e+013
376.85	4.19e+013
426.85	4.02e+013
526.85	3.72e+013
626.85	3.45e+013
726.85	3.21e+013
826.85	2.99e+013
926.85	2.99e+013
1026.8	2.99e+012

Table 45. Temperature Boundary Conditions

Time	Temperature (°C)
0	22.
1	25.55
10	57.5
100	377
200	377
300	377
310	377

APPENDIX E ANSYS FINITE ELEMENT ANALYSIS THEORY

E.1. General Theory

The following sections are derivations provided with the ANSYS software. [24] They are provided as a legacy reference and are themselves compiled from reference literature.

E.1.1. Shape functions

Figure 72 shows a representation of a 10-node tetrahedron element. This element is primarily used to build circular and spherical geometries. The mid-nodes between the vertices allow the element to be skewed to achieve the desired shape.

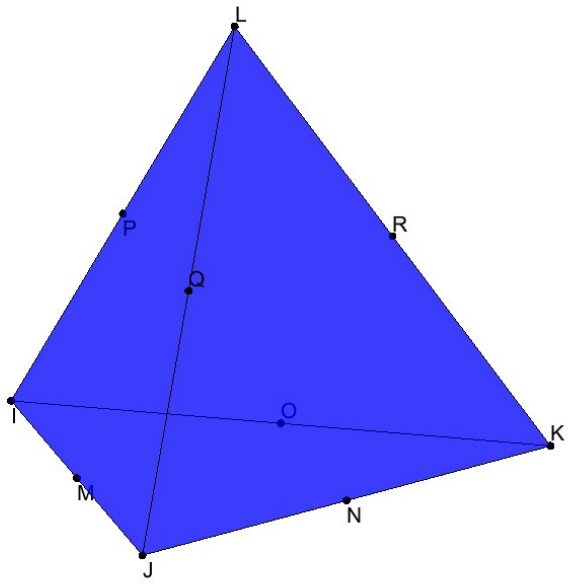

Figure 72. 10-node tetrahedron element

Sections E.1.2 and E.1.3 use variables defined in Table 46.

Table 46. Shape Function Variable Definition

Variable	Description
u	Translation in the x or s direction
v	Translation in the y or t direction
w	Translation in the z or r direction
T	Temperature
L_n	Normalized length (0.0 at vertex to 1.0 at the opposite side or face)
s	Normalized coordinates (-1.0 on one side of the element to +1.0 on the other side)
r	Normalized coordinates (-1.0 on one side of the element to +1.0 on the other side)
t	Normalized coordinates (-1.0 on one side of the element to +1.0 on the other side)

E.1.2. 10-node tetrahedron:

$$u = u_I(2L_1 - 1)L_1 + u_J(2L_2 - 1)L_2 + u_k(2L_3 - 1)L_3 + u_L(2L_4 - 1)L_4 + 4u_M L_1 L_2 + u_N L_2 L_3 + u_O L_1 L_3 + u_P L_1 L_4 + u_Q L_2 L_4 + u_R L_3 L_4$$

$$v = v_I(2L_1 - 1)L_1 + v_J(2L_2 - 1)L_2 + v_k(2L_3 - 1)L_3 + v_L(2L_4 - 1)L_4 + 4v_M L_1 L_2 + v_N L_2 L_3 + v_O L_1 L_3 + v_P L_1 L_4 + v_Q L_2 L_4 + v_R L_3 L_4$$

$$w = w_I(2L_1 - 1)L_1 + w_J(2L_2 - 1)L_2 + w_k(2L_3 - 1)L_3 + w_L(2L_4 - 1)L_4 + 4w_M L_1 L_2 + w_N L_2 L_3 + w_O L_1 L_3 + w_P L_1 L_4 + w_Q L_2 L_4 + w_R L_3 L_4$$

$$T = T_I(2L_1 - 1)L_1 + u_J(2L_2 - 1)L_2 + u_k(2L_3 - 1)L_3 + u_L(2L_4 - 1)L_4 + 4u_M L_1 L_2 + u_N L_2 L_3 + u_O L_1 L_3 + u_P L_1 L_4 + u_Q L_2 L_4 + u_R L_3 L_4$$

Figure 73 represents a 20-node brick element, it is primarily used for constructing cuboid type geometries.

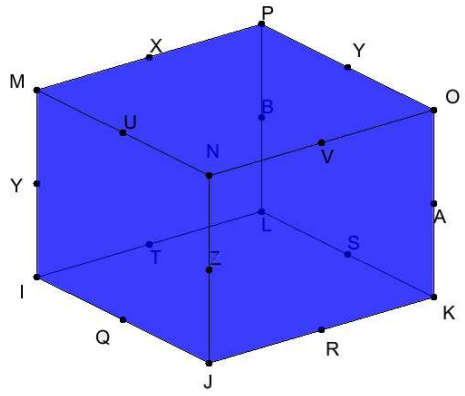

Figure 73. 20-node brick element

E.1.3. 20-node brick

$$
\begin{aligned}
u = \frac{1}{8}\Big(& u_I(1-s)(1-t)(1-r)(-s-t-r-2) \\
&+ u_J(1+s)(1-t)(1-r)(s-t-r-2) \\
&+ u_K(1+s)(1+t)(1-r)(s+t-r-2) \\
&+ u_L(1-s)(1+t)(1-r)(-s+t-r-2) \\
&+ u_M(1-s)(1-t)(1+r)(-s-t+r-2) \\
&+ u_N(1+s)(1-t)(1+r)(s-t+r-2) \\
&+ u_0(1+s)(1+t)(1+r)(s+t+r-2) \\
&+ u_P(1-s)(1+t)(1+r)(-s+t+r-2)\Big) \\
&+ \frac{1}{4}(u_Q(1-s^2)(1-t)(1-r) + u_R(1+s)(1-t^2)(1-r) \\
&+ u_S(1-s^2)(1+t)(1-r) + u_T(1-s)(1-t^2)(1-r) \\
&+ u_u(1-s^2)(1-t)(1+r) + u_v(1+s)(1-t^2)(1-r) \\
&+ u_w(1-s^2)(1+t)(1+r) + u_X(1-s)(1-t^2)(1+r) \\
&+ u_y(1-s)(1-t)(1-r^2) + u_z(1+s)(1-t)(1-r^2) \\
&+ u_A(1+s)(2+t)(1-r^2) + u_B(1-s)(1+t)(1-r^2))
\end{aligned}
$$

$$
\begin{aligned}
v = \frac{1}{8}\Big(& v_I(1-s)(1-t)(1-r)(-s-t-r-2) \\
& + v_J(1+s)(1-t)(1-r)(s-t-r-2) \\
& + v_K(1+s)(1+t)(1-r)(s+t-r-2) \\
& + v_L(1-s)(1+t)(1-r)(-s+t-r-2) \\
& + v_M(1-s)(1-t)(1+r)(-s-t+r-2) \\
& + v_N(1+s)(1-t)(1+r)(s-t+r-2) \\
& + v_0(1+s)(1+t)(1+r)(s+t+r-2) \\
& + v_P(1-s)(1+t)(1+r)(-s+t+r-2)\Big) \\
& + \frac{1}{4}(v_Q(1-s^2)(1-t)(1-r) + v_R(1+s)(1-t^2)(1-r) \\
& + v_S(1-s^2)(1+t)(1-r) + v_T(1-s)(1-t^2)(1-r) \\
& + v_u(1-s^2)(1-t)(1+r) + v_v(1+s)(1-t^2)(1-r) \\
& + v_w(1-s^2)(1+t)(1+r) + v_X(1-s)(1-t^2)(1+r) \\
& + v_y(1-s)(1-t)(1-r^2) + v_z(1+s)(1-t)(1-r^2) \\
& + v_A(1+s)(2+t)(1-r^2) + v_B(1-s)(1+t)(1-r^2))
\end{aligned}
$$

$$
\begin{aligned}
w = \frac{1}{8}\Big(& w_I(1-s)(1-t)(1-r)(-s-t-r-2) \\
& + w_J(1+s)(1-t)(1-r)(s-t-r-2) \\
& + w_K(1+s)(1+t)(1-r)(s+t-r-2) \\
& + w_L(1-s)(1+t)(1-r)(-s+t-r-2) \\
& + w_M(1-s)(1-t)(1+r)(-s-t+r-2) \\
& + w_N(1+s)(1-t)(1+r)(s-t+r-2) \\
& + w_0(1+s)(1+t)(1+r)(s+t+r-2) \\
& + w_P(1-s)(1+t)(1+r)(-s+t+r-2)\Big) \\
& + \frac{1}{4}(w_Q(1-s^2)(1-t)(1-r) + w_R(1+s)(1-t^2)(1-r) \\
& + w_S(1-s^2)(1+t)(1-r) + w_T(1-s)(1-t^2)(1-r) \\
& + w_u(1-s^2)(1-t)(1+r) + w_v(1+s)(1-t^2)(1-r) \\
& + w_w(1-s^2)(1+t)(1+r) + w_X(1-s)(1-t^2)(1+r) \\
& + w_y(1-s)(1-t)(1-r^2) + w_z(1+s)(1-t)(1-r^2) \\
& + w_A(1+s)(2+t)(1-r^2) + w_B(1-s)(1+t)(1-r^2))
\end{aligned}
$$

$$
\begin{aligned}
T = \frac{1}{8}\Big(& T_I(1-s)(1-t)(1-r)(-s-t-r-2) \\
&+ T_J(1+s)(1-t)(1-r)(s-t-r-2) \\
&+ T_K(1+s)(1+t)(1-r)(s+t-r-2) \\
&+ T_L(1-s)(1+t)(1-r)(-s+t-r-2) \\
&+ T_M(1-s)(1-t)(1+r)(-s-t+r-2) \\
&+ T_N(1+s)(1-t)(1+r)(s-t+r-2) \\
&+ T_0(1+s)(1+t)(1+r)(s+t+r-2) \\
&+ T_P(1-s)(1+t)(1+r)(-s+t+r-2)\Big) \\
&+ \frac{1}{4}(T_Q(1-s^2)(1-t)(1-r) + T_R(1+s)(1-t^2)(1-r) \\
&+ T_S(1-s^2)(1+t)(1-r) + T_T(1-s)(1-t^2)(1-r) \\
&+ T_u(1-s^2)(1-t)(1+r) + T_v(1+s)(1-t^2)(1-r) \\
&+ T_w(1-s^2)(1+t)(1+r) + T_X(1-s)(1-t^2)(1+r) \\
&+ T_y(1-s)(1-t)(1-r^2) + T_z(1+s)(1-t)(1-r^2) \\
&+ T_A(1+s)(2+t)(1-r^2) + T_B(1-s)(1+t)(1-r^2))
\end{aligned}
$$

E.2. Heat Fundamentals

E.2.1. Conduction and Convection

The first law of thermodynamics states that thermal energy is conserved.

Specializing this to a differential control volume:

$$\rho c\left(\frac{\partial T}{\partial t} + \{v\}^T\{L\}T\right) + \{L\}^T\{q\} = \ddot{q} \tag{9}$$

Where:

$\rho = density$

$c = specific\ heat$

$T = temperature = T(x, y, z, t))$

t =*time*

$$\{L\} = \begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{Bmatrix} = \textit{vector operator}$$

$$\{v\} = \begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix} = \textit{velocity vector for mass transport}$$

$\{q\} = \textit{heat flux vector}$

$\ddot{q}$ =*heat generation*

Next Fouriers law is used to relate the heat flux vector to the thermal gradients:

$$\{q\} = -[D]\{L\}T \tag{10}$$

Where:

$$[D] = \begin{bmatrix} K_{xx} & 0 & 0 \\ 0 & K_{yy} & 0 \\ 0 & 0 & K_{zz} \end{bmatrix} = \textit{conductivity matrix}$$

Where:

K = thermal conductivity

Combining equation 9 and equation 10

$$\rho c \left(\frac{\partial T}{\partial t} + \{v\}^T \{L\}T \right) = \{L\}^T ([D]\{L\}T) + \ddot{q} \tag{11}$$

It should be noted that $\{L\}T$ and $\{L\}^T\{q\}$ may also be interpreted as ∇T and $\nabla \cdot \{q\}$, respectively, where ∇ represents the grad operator and $\nabla \cdot$represents the divergence operator.

Expanding equation 11

$$\rho c\left(\frac{\partial T}{\partial t}+v_x\frac{\partial T}{\partial x}+v_y\frac{\partial T}{\partial y}+v_z\frac{\partial T}{\partial z}\right) = \ddot{q}+\frac{\partial}{\partial x}\left(K_x\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(K_y\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(K_z\frac{\partial T}{\partial z}\right) \quad (12)$$

Boundary Conditions

Element boundary conditions are considered to cover the entire surface of the element:

Specified temperatures acting over surface $\boldsymbol{S_1}$:

$$T = T^* \quad (13)$$

Where:

$T^*$is a specified temperature.

Specified heat flows acting over surface $\boldsymbol{S_2}$:

$$\{q\}^T\{n\} = -q^* \quad (14)$$

Where:

$\{n\}$ =unit outward normal vector

q^* =specified heat flow

Specified convection surfaces acting over surface $\boldsymbol{S_3}$(Newton's Law of Cooling):

$$\{q\}^T\{n\} = h_f(T_s - T_B) \quad (15)$$

Where:

h_f = film coefficient evaluated at $(T_B + T_S)/2$ unless otherwise specified for the element

T_B =bulk temperature of the adjacent fluid

T_S =temperature at the surface of the model

Combining equation 10 with equations 14 and 15.

$$\{n\}^T[D]\{L\}T = q^* \tag{16}$$

$$\{n\}^T[D]\{L\}T = h_f(T_s - T_B) \tag{17}$$

Premultiplying equation 11 by a virtual change in temperature, integrating over the volume of the element, and combining with equations 16 and 17 yields equation 8.

$$\int_{vol}\left(\rho c\delta T\left(\frac{\partial T}{\partial t} + \{v\}^T\{L\}T\right) + \{L\}^T(\delta T)([D]\{L\})\right)d(vol) = \int_{S_2}\delta T q^* d(S_2) + \int_{S_3}\delta T h_f(T_B - T)d(S_3) + \int_{vol}\delta T\ddot{q}d(vol) \tag{18}$$

Where:

vol =volume of an element.

δT =an allowable virtual temperature ($= \delta T(x, y, z, t)$)

E.2.2. Heat Flow Matrices

The variable T can vary in both space and time.

$$T = \{N\}^T\{T_e\} \tag{19}$$

Where:

$T = T(x, y, z, t)$ = temperature

$\{N\} = \{N(x, y, z)\}$ = element shape functions

$\{T_e\} = \{T_e(t)\}$ = nodal temperature vector of element

Time derivative of T

$$\dot{T} = \frac{\partial T}{\partial t} = \{N\}^T\{T_e\} \tag{20}$$

δT has the same form as T:

$$\delta T = \{\delta T_e\}^T\{N\} \tag{21}$$

The combination $\{L\}T$ is written as:

$$\{L\}T = [B]\{T_e\} \tag{22}$$

Where:

$[B] = \{L\}\{N\}T$

Now, the variational statement of equation 8 can be combined with equations 19, 20, 21, and 22 to yield.

$$\begin{aligned}
&\int_{vol} \rho c\{\delta T_e\}^T\{N\}\{N\}^T\{\dot{T}_e\}d(vol) \\
&\qquad + \int_{vol} \rho c\{\delta T_e\}^T\{N\}\{v\}^T[B]\{T_e\}d(vol) \\
&\qquad + \int_{vol} \{\delta T_e\}^T[B]^T[D][B]\{T_e\}d(vol) \\
&\qquad = \int_{S_2} \{\delta T_e\}^T\{N\}q^* d(S_2) \\
&\qquad + \int_{S^3} \{\delta T_e\}^T\{N\}h_f(T_B - \{N\}^T\{T_e\})d(S_3) \\
&\qquad + \int_{vol} \{\delta T_e\}^T\{N\}\ddot{q}d(vol)
\end{aligned} \tag{23}$$

Several simplifications can be made from equation 23.

Assumptions:

ρ is assumed to remain constant over the volume of the element.

$c\ and\ \ddot{q}$ may vary over the element.

Notes:

$\{\delta T_e\}^T$, $\{\dot{T}_e\}$, and $\{T_e\}$ are nodal quantities and do not vary over the element and can be removed out of their respective intergrals. All quantities are multiplied by the arbitrary vector $\{\delta T_e\}$ therefore it can be removed from the equation.

Thus, equation 23 can be reduced to:

$$\begin{aligned}\rho \int_{vol} c\{N\}\{N\}^T d(vol)\{\dot{T}_e\} + \rho \int_{vol} c\{N\}\{v\}^T [B] d(vol)\{T_e\} \\ + \int_{vol} [B]^T [D][B] d(vol)\{T_e\} \\ = \int_{S_2} \{N\} q^* d(S_2) \\ + \int_{S^3} T_B h_f \{N\} d(S_3) - \int_{S^3} h_f \{N\}\{N\}^T \{T_e\} d(S_3) \\ + \int_{vol} \ddot{q}\{N\} d(vol)\end{aligned} \tag{24}$$

Equation 24 can be rewritten as:

$$[C_e^t]\{\dot{T}_e\} + ([K_e^{tm}] + [K_e^{tb}] + [K_e^{tc}])\{T_e\} = \{Q_e^f\} + \{Q_e^c\} + \{Q_e^g\} \tag{25}$$

Where:

$[C_e^t] = \int_{vol} c\{N\}\{N\}^T d(vol) =$ element specific heat (thermal dampening) matrix.

$[K_e^{tm}] = \int_{vol} c\{N\}\{v\}^T [B] d(vol) =$ element mass transport conductivity matrix.

$[K_e^{tb}] = \int_{vol} [B]^T [D][B] d(vol) =$ element diffusion conductivity matrix.

$[K_e^{tc}] = \int_{S^3} h_f \{N\}\{N\}^T \{T_e\} d(S_3)\ =$ element convection surface conductivity matrix.

$\{Q_e^f\} = \int_{S_2} \{N\} q^* d(S_2) =$ element mass flux vector.

$\{Q_e^c\} = \int_{S^3} T_B h_f \{N\} d(S_3) \ =$ element convection surface heat flow vector.

$\{Q_e^g\} = \int_{vol} \ddot{q}\{N\} d(vol) =$ element heat generation load.

E.3. Structural Fundamentals

E.3.1. Stress-strain relationship

$$\{\sigma\} = [D]\{\varepsilon^{el}\} \tag{26}$$

Where:

$\{\sigma\} =$ stress vector $= \left[\sigma_x \ \sigma_y \ \sigma_z \ \sigma_{xy} \ \sigma_{yz} \ \sigma_{xz}\right]^T$

$[D] =$ elasticity or elastic stiffness matrix as defined in equations 39 through 44

$\{\varepsilon^{el}\} = \{\varepsilon\} - \{\varepsilon^{th}\} =$ elastic strain vector

$\{\varepsilon\} =$ total strain vector $= \left[\varepsilon_x \ \varepsilon_y \ \varepsilon_z \ \varepsilon_{xy} \ \varepsilon_{yz} \ \varepsilon_{xz}\right]^T$

$\{\varepsilon^{th}\} =$ thermal strain vector as defined in equation 28

Equation 26 may also be inverted to:

$$\{\varepsilon\} = \{\varepsilon^{th}\} + [D]^{-1}\{\sigma\} \tag{27}$$

Thermal strain vector

$$\{\varepsilon^{th}\} = \Delta T\left[\alpha_x^{se} \ \alpha_y^{se} \ \alpha_z^{se} \ 0 \ 0 \ 0\right]^T \tag{28}$$

Where:

$\alpha_x^{se} =$ secant coefficient in the x direction

$\Delta T = T - T_{ref}$

$T =$ current temperature at the point in question

$T_{ref} =$ reference (strain free temperature)

The Flexibility or Compliance matrix

$$[D]^{-1} = \begin{bmatrix} 1/E_x & -v_{xy}/E_x & -v_{xz}/E_x & 0 & 0 & 0 \\ -v_{yx}/E_y & 1/E_y & -v_{yz}/E_y & 0 & 0 & 0 \\ -v_{zx}/E_z & -v_{zy}/E_z & 1/E_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/G_{xy} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/G_{yz} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/G_{xz} \end{bmatrix} \tag{29}$$

Where:

$E_x =$ Young's modulus in the x direction.

$v_{xy} =$ major Poisson's ratio.

$v_{yx} =$ minor Poisson's ratio.

$G_{xy} =$ shear modulus in the xy plane.

$[D]^{-1}$ is assumed to be symmetric such that:

$$\frac{v_{yx}}{E_y} = \frac{v_{xy}}{E_x} \tag{30}$$

$$\frac{v_{zx}}{E_z} = \frac{v_{xz}}{E_x} \tag{31}$$

$$\frac{v_{zy}}{E_z} = \frac{v_{yz}}{E_y} \tag{32}$$

Expanding equation 27 with equations 28 through 32

$$\varepsilon_x = \alpha_x \Delta T + \frac{\sigma_x}{E_x} - \frac{v_{xy}\sigma_y}{E_x} - \frac{v_{xz}\sigma_z}{E_x} \tag{33}$$

$$\varepsilon_y = \alpha_y \Delta T - \frac{v_{xy}\sigma_x}{E_x} + \frac{\sigma_y}{E_y} - \frac{v_{yz}\sigma_z}{E_y} \tag{34}$$

$$\varepsilon_z = \alpha_z \Delta T - \frac{v_{xz}\sigma_x}{E_x} - \frac{v_{yz}\sigma_y}{E_v} + \frac{\sigma_z}{E_z} \tag{35}$$

$$\varepsilon_{xy} = \frac{\sigma_{xy}}{G_{xy}} \tag{36}$$

$$\varepsilon_{yz} = \frac{\sigma_{yz}}{G_{yz}} \tag{37}$$

$$\varepsilon_{xz} = \frac{\sigma_{xz}}{G_{xz}} \tag{38}$$

Where typical terms are:

$\varepsilon_x =$ direct strain in the x direction

$\sigma_x =$ direct stress in the x direction

$\varepsilon_{xy} =$ shear strain in the x-y plane

$\sigma_{xy} =$ shear stress in the x-y plane

Alternatively, equation 26 may be expanded by first inverting equation 29 and then combining that result with equation 28 and equations 30 through 32 yields:

$$\begin{aligned}\sigma_x = \frac{E_x}{h}\left(1 - \left(v_{yz}\right)^2 \frac{E_z}{E_y}\right)(\varepsilon_x - \alpha_x \Delta T) \\ + \frac{E_y}{h}\left(v_{xy} + v_{xz}v_{yz}\frac{E_z}{E_y}\right)\left(\varepsilon_y - \alpha_y \Delta T\right) \\ + \frac{E_z}{h}\left(v_{xz} + v_{yz}v_{xy}\right)(\varepsilon_z - \alpha_z \Delta T)\end{aligned} \tag{39}$$

$$\sigma_y = \frac{E_y}{h}\left(v_{xy} + v_{xz}v_{yz}\frac{E_z}{E_y}\right)(\varepsilon_x - \alpha_x \Delta T) + \frac{E_y}{h}\left(1 - (v_{xz})^2\frac{E_z}{E_x}\right)(\varepsilon_y - \alpha_y \Delta T) + \frac{E_z}{h}\left(v_{yz} + v_{xz}v_{xy}\frac{E_y}{E_x}\right)(\varepsilon_z - \alpha_z \Delta T) \quad (40)$$

$$\sigma_z = \frac{E_z}{h}\left(v_{xz} + v_{yz}v_{xy}\right)(\varepsilon_x - \alpha_x \Delta T) + \frac{E_z}{h}\left(v_{yz} + v_{xz}v_{xy}\frac{E_y}{E_x}\right)(\varepsilon_y - \alpha_y \Delta T) + \frac{E_z}{h}\left(1 - \left(v_{xy}\right)^2\frac{E_y}{E_x}\right)(\varepsilon_z - \alpha_z \Delta T) \quad (41)$$

$$\sigma_{xy} = G_{xy}\varepsilon_{xy} \quad (42)$$

$$\sigma_{yz} = G_{yz}\varepsilon_{yz} \quad (43)$$

$$\sigma_{xz} = G_{xz}\varepsilon_{xz} \quad (44)$$

Where:

$$h = 1 - \left(v_{xy}\right)^2\frac{E_y}{E_x} - \left(v_{yz}\right)^2\frac{E_z}{E_y} - (v_{xz})^2\frac{E_z}{E_x} - 2v_{xy}v_{yz}v_{xz}\frac{E_z}{E_x} \quad (45)$$

If the shear moduli G_{xy}, G_{yz}, and G_{xz} are not input for isotopic materials, they are computed as:

$$G_{xy} = G_{yz} = G_{xz} = \frac{E_x}{2(1 + v_{xy})} \quad (46)$$

E.3.2. Temperature-Dependent Coefficient of Thermal Expansion

Starting with the thermal strain from equation 28

$$\varepsilon^{th} = \alpha^{se}(T)\left(T - T_{ref}\right) \quad (47)$$

Where:

$\alpha^{se}(T)$= temperature-dependent scant coefficient of thermal expansion

$\alpha^{se}(T)$ is computed from $\varepsilon^{th}(T)$ by rearranging equation 47

$$\alpha^{se}(T) = \frac{\varepsilon^{ith}(T)}{T - T_{ref}} \tag{48}$$

$\varepsilon^{th}(T)$ (thermal strain) is related to $\alpha^{in}(T)$ by:

$$\varepsilon^{th}(T) = \int_{T_{ref}}^{T} \alpha^{in}(T)dT$$

Combining with equation 48 yields:

$$\alpha^{se}(T) = \frac{\int_{T_{ref}}^{T} \alpha^{in}(T)T}{T - T_{ref}} \tag{49}$$

An adjustment is needed if T_{ref} is defined at one value and the thermal strain is zero at a different temperature. Consider:

$$\varepsilon_0^{th} = \alpha_0^{se}(T)(T - T_0) = \int_{T_0}^{T} \alpha^{in}dT \tag{50}$$

$$\varepsilon_r^{th} = \alpha_r^{se}(T)\big(T - T_{ref}\big) = \int_{T_{ref}}^{T} \alpha^{in}dT \tag{51}$$

Equations 50 and 51 represent the thermal strain at a temperature T for two different starting points, T_0 and T_{ref}. Now let T_0 be the temperature about which the data has been generated (definition temperature). Thus, α_0^{se} is the supplied data, and α_r^{se} is what is needed as program input.

Equation 50 can be expanded as:

$$\int_{T_0}^{T} \alpha^{in} dT = \int_{T_0}^{T_{ref}} \alpha^{in} dT + \int_{T_{ref}}^{T} \alpha^{in} dT \tag{52}$$

Also,

$$\int_{T_0}^{T_{ref}} \alpha^{in} dT = \alpha_0^{se}(T_{ref})(T_{ref} - T_0) \tag{53}$$

Or

$$\int_{T_0}^{T_{ref}} \alpha^{in} dT = \alpha_r^{se}(T_0)(T_{ref} - T_0) \tag{54}$$

Combining equations 50 through 53

$$\alpha_r^{se}(T) = \alpha_0^{se}(T) + \frac{T_{ref} - T_0}{T_0 - T_{ref}}(\alpha_0^{se}(T) - \alpha_0^{se}(T_{ref})) \tag{55}$$

Equation 55 is used when an adjustment is needed for the definition temperature is different than the strain free temperature.

E.4. Structural Matrices

The principle of virtual work states that a virtual (very small) change of the internal stain energy must be offset by an identical change in external work due to the applied loads, or:

$$\delta U = \delta V \tag{56}$$

Where:

U = strain energy (internal work) = $U_1 + U_2$

V = external work = $V_1 + V_2 + V_3$

δ = virtual operator

The virtual strain energy is:

$$\delta U_1 = \int_{vol} \{\delta\varepsilon\}\{\sigma\} d\{vol\}^T \tag{57}$$

Where:

$\{e\}$ = strain vector

$\{\sigma\}$ = stress vector

vol = volume of element

Continuing the derivation assuming linear materials and geometry, combining equations 56 and 57.

$$\delta U_1 = \int_{vol} (\{\delta\varepsilon\}^T [D]\{\varepsilon\} - \{\delta\varepsilon\}^T [D]\{\varepsilon^{th}\}) d(vol) \tag{58}$$

The strains may be related to the nodal displacements by:

$$\{\varepsilon\} = [B]\{u\} \tag{59}$$

Where:

$[B]$ = strain-displacement matrix, based on the element shape functions

$\{u\}$ = nodal displacement vector

Assuming all effects are in the global Cartesian system. Combining equations 59and 58, and noting that $\{u\}$ does not vary over the volume:

$$\begin{aligned} \delta U_1 = \{\delta u\}^T \int_{vol} [B]^T [D][B] d(vol)\{u\} \\ - \{\delta u\}^T \int_{vol} [B]^T [D]\{\varepsilon^{th}\} d(vol) \end{aligned} \tag{60}$$

Another form of virtual strain energy is when a surface moves against a distributed resistance, as in a foundation stiffness. This may be written as:

$$\delta U_2 = \int_{area_f} \{\delta w_n\}^T \{\sigma\} d(area_f) \tag{61}$$

Where:

$\{w_n\}$ = motion normal to the surface

$\{\sigma\}$ = stress (or pressure) carried by the surface

$area_f$ = area of the distributed resistance

Both $\{w_n\}$ and $\{\sigma\}$ will usually have only one nonzero component. The point-wise normal displacement is related to the nodal displacements by:

$$\{w_n\} = [N_n]\{u\} \tag{62}$$

Where:

$[N_n]$ = matrix of shape functions for normal motions at the surface

The stress, $\{\sigma\}$, is

$$\{\sigma\} = k\{w_n\} \tag{63}$$

Where:

k = the foundation stiffness in units of force per length per unit area.

Combining equations 61 through 63, and assuming that k is constant over the area:

$$\delta U_2 = \{\delta u\}^{Tk} \int_{area_f} [N_n]^T [N_n] d\left(area_f\right)\{u\} \tag{64}$$

Next, the external virtual work will be considered. The inertial effects will be studied first:

$$\delta V_1 = -\int_{vol} \{\delta w\}^T \frac{\{F^a\}}{vol} d(vol) \tag{65}$$

Where:

$\{w\}$ = vector of displacements of a general point

$\{F^a\}$ = acceleration (D'Alember) force vector

According to Newton's second law

$$\frac{\{F^a\}}{vol} = \rho \frac{\partial^2}{\partial t^2} \{w\} \tag{66}$$

Where:

ρ = density

t = time

The displacements within the element are related to the nodal displacements by:

$$\{w\} = [N]\{u\} \tag{67}$$

Where:

$[N]$ = matrix of shape functions.

Combining equations 65, 66, and 67, and assuming that ρ is constant over the volume.

$$\delta V_1 = -\{\delta u\}^T \rho \int_{vol} [N]^T [N] d(vol) \frac{\partial^2}{\partial t^2} \{u\} \tag{68}$$

The pressure force vector formulation starts with:

$$\delta V_2 = \int_{area_p} \{\delta w_n\}^T \{P\} d(area_p) \tag{69}$$

Where:

$\{P\}$ = the applied pressure vector (normally contains only one nonzero component)

$area_p$ = area over which pressure acts.

Combining equations 67 and 69

$$\delta V_2 = \{\delta u\}^T \int_{area_p} [N_n]\{P\} d(area_p) \tag{70}$$

Unless otherwise noted, pressures are applied to the outside surface of each element and are normal to curved surfaces, if applicable. Nodal forces applied to the element can be accounted for by:

$$\delta V_3 = \{\delta u\}^T \{F_e^{nd}\} \tag{71}$$

Where:

$\{F_e^{nd}\}$ = nodal forces applied to the element.

Recall equation 56

$$\delta U_1 + \delta U_2 = \delta V_1 + \delta V_2 + \delta V_3$$

Combining equations 56, 60, 64, 68, 70, and 71 yields:

$$\{\delta u\}^T \int_{vol} [B]^T[D][B]d(vol)\{u\} - \{\delta u\}^T \int_{vol} [B]^T[D]\{\varepsilon^{th}\}d(vol) + \delta U_2 = \{\delta u\}^{Tk} \int_{area_f} [N_n]^T [N_n] d(area_f)\{u\} = -\{\delta u\}^T \rho \int_{vol} [N]^T[N]d(vol)\frac{\partial^2}{\partial t^2}\{u\} + \{\delta u\}^T \int_{area_p} [N_n]\{P\}d(area_p) + \{\delta u\}^T\{F_e^{nd}\} \quad (72)$$

Noting that the $\{\delta u\}^T$ vector is a set of arbitrary virtual displacements common in all the above terms, the condition required to satisfy equation 72:

$$\left([K_e] + \left[K_e^f\right]\right)\{u\} - \{F_e^{th}\} = [M_e]\{\ddot{u}\} + \left\{F_e^{pr}\right\} + \{F_e^{nd}\} \quad (73)$$

Where:

$[K_e] = \int_{vol}[B]^T[D][B]d(vol)$ = element stiffness matrix

$\left[K_e^f\right] = k\int_{are\ _f}[N_n]^T[N_n]d(area_f)$ = element foundation stiffness matrix

$\{F_e^{th}\} = \int_{vol}[B]^T[D]\{\varepsilon^{th}\}d(vol)$ = element thermal load

$[M_e] = \rho\int_{vol}[N]^T[N]d(vol)$ = element mass matrix

$\{\ddot{u}\} = \frac{\partial^2}{\partial t^2}\{u\}$ = acceleration vector (such as gravity effects)

$\left\{F_e^{pr}\right\} = \int_{area_p}[N_n]^T\{P\}d(area_p)$ = element pressure vector

$\{F_e^{nd}\}$ = nodal forces applied to the element.

E.5. Solver Equations

E.5.1. Sparse Direct Solver

The system of equations generated by either the thermal or structural analysis is solved by using a direct elimination method.

$$[K]\{u\} = \{F\} \tag{74}$$

Where:

$[K]$ = global stiffness/conductivity matrix

$\{u\}$ = global vector of nodal unknown

$\{F\}$ = global applied load vector

The direct elimination method decomposes the $[K]$ matrix found in equation 74 into upper and lower matrices.

$$[K] = [L][U] \tag{75}$$

These matrices are then used forward and back substitutions to compute the solution vector $\{u\}$. The main goal is to remove the zero entries and reformulate the $[K]$ matrix such that $[L]$ matrix is only filled with non-zero elements. This greatly increases the solver performance by reducing the matrix size to a bare minimum.

E.5.2. Newton-Raphson Procedure

The finite element discretization process yields a set of simultaneous equations:

$$[K]\{u\} = \{F^a\} \tag{76}$$

Where:

$[K]$ = coefficient matrix

$\{u\}$ = vector of unknown DOF values

$\{F^a\}$ = vector of applied loads

If the coefficient matrix $[K]$ is itself a function of the unknown DOF values (or their derivatives) then equation 76 is a nonlinear equation. The Newton-Raphson method is an iterative process of solving the nonlinear equations and can be written:

$$[K_i^T]\{\Delta u_i\} = \{F^a\} - \{F_i^{nr}\} \tag{77}$$

$$\{u_{i+1}\} = \{u_i\} + \{\Delta u_i\} \tag{78}$$

Where:

$[K_i^T]$ = Jacobian matrix (tangent matrix)

i = subscript representing the current equilibrium iteration

$\{F_i^{nr}\}$ = vector of restoring loads corresponding to the element internal loads

Both $[K_i^T]$ and $\{F_i^{nr}\}$ are evaluated based on the values given $\{u_i\}$. The right-hand side of equation 77 is the residual or out-of-balance load vector. This is the amount the system is out of equilibrium. For thermal analysis $[K_i^T]$ is the conductivity matrix, $\{u_i\}$ is the temperature vector, and $\{F_i^{nr}\}$ is the resisting load vector calculated from the element heat flows. For structural analysis $[K_i^T]$ is the tangent stiffness matrix, $\{u_i\}$ is the displacement vector, and $\{F_i^{nr}\}$ is the restoring force calculated from the element stresses.

Multiple iterations are usually required for convergence. The following algorithm is used:

Step 1. Assume $\{u_0\}$. $\{u_0\}$ is the converged solution from the previous time-step. For the first time-step $\{u_0\} = \{0\}$

Step 2. Compute the updated $[K_i^T]$ and the restoring load $\{F_i^{nr}\}$ from configuration $\{u_i\}$.

Step 3. Calculate $\{\Delta u_i\}$ from equation 77

Step 4. Add $\{\Delta u_i\}$ to $\{u_i\}$ to obtain the next approximation $\{u_{i+1}\}$ equation 78

Step 5. Repeat steps 2 through 4.

E.5.3. Convergence

The iteration process described in the previous section continues until convergence is achieved.

Convergence is assumed when

$$\|\{R\}\| < \varepsilon_R R_{ref} \text{ (out-of-balance convergence)} \tag{79}$$

And/or

$$\|\{\Delta u_i\}\| < \varepsilon_u u_{ref} \text{ (DOF increment convergence)} \tag{80}$$

Where:

$\{R\}$ is the residual vector.

$$\{R\} = \{F^a\} - \{F^{nr}\} \tag{81}$$

Equation 81 is the right-hand side of the equation 77. $\{\Delta u_i\}$, is the DOF increment vector, ε_u and ε_R are the tolerances. R_{ref} and u_{ref} are the reference values. The operator $\|\quad\|$ is the vector magnitude. Convergence is achieved when the size of the residual

(disequilibrium) is less than a tolerance times a reference value. For structural analysis both equations 79and 80 are utilized to determine convergence.

The vector normalization used by default is a Euclidean normalization.

$$\|\{R\}\| = \sqrt{\sum R_i^2} \tag{82}$$

The following are the criteria for use of default R_{ref}.

For structural DOFs minimum $R_{ref} = 1.0x10^{-2}$.

For thermal DOFs minimum $R_{ref} = 1.0x10^{-6}$.

Default $u_{ref} = \|\{u\}\|$

E.5.4. Predictor

Usually the converged solution from the previous step is used as the estimate for the solution to the next step. A predictor is usually implemented to make a better estimate of the next solution.

For thermal analysis, the prediction is based upon the trapezoidal formula:

$$\{u_{n,0}\} = \{u_{n-1}\} + \{1-\theta\}\{\dot{u}_{n-1}\}\Delta t_n \tag{83}$$

Where:

$\{u_{n-1}\}$ = current temperatures

$\{\dot{u}_{n-1}\}$ = current rates of temperature changes

θ = trapezoidal time integration parameter

For structural analysis:

$$\{u_{n,0}\} = \{u_{n-1}\} + \beta\{\Delta u_n\} \tag{84}$$

Where:

$\{\Delta u_n\}$ = displacement increment accumulated over the previous time-step

n = current time-step

$$\{\Delta u_n\} = \sum_{i=1}\{\Delta u_i\} \tag{85}$$

$$\beta = \frac{\Delta t_n}{\Delta t_{n-1}} \tag{86}$$

Where:

Δt_n = current time-step size

Δt_{n-1} = previous time-step size

E.5.5. Line search parameter

A further improvement on the Newton-Raphson solution is the line search parameter. The line search parameter is a scalar value which is applied to the solution vector.

Consider equation 78

$$\{u_{i+1}\} = \{u_i\} + \{\Delta u_i\} \tag{87}$$

The full Δu_i can lead to solution instabilities which necessitates the use of the line search parameter.

$$\{u_{i+1}\} = \{u_i\} + s\{\Delta u_i\} \tag{88}$$

Where:

s = line search parameter, $0.05 < s < 1.0$

s is determined by minimizing the energy of the system, which reduces to finding the zero of the nonlinear equation:

$$g_s = \{\Delta u_i\}^T(\{F^a\} - \{F^{nr}(s\{\Delta u_i\})\}) \tag{89}$$

Where:

g = gradient of the potential energy with respect to s

Equation 89 is iterated until one of the following conditions exists:

g_s is less than $0.5\ g_0$, where g_0 is the value of equation 89 at $s = 0.0$

g_s is not changing significantly between iterations

Six iterations have been performed.