NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPLEMENTATION OF THE CONFIGURABLE FAULT TOLERANT SYSTEM EXPERIMENT ON NPSAT - 1

by

Andrew S. Jackson

March 2016

Thesis Advisor: Herschel H. Loomis, Jr.
Co-Advisor: James H. Newman

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704–0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE IMPLEMENTATION OF THE CONFIGURABLE FAULT TOLERANT SYSTEM EXPERIMENT ON NPSAT - 1		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Andrew S. Jackson			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

Space is a harsh environment, full of high-energy radiation that can cause single-event effects (SEE) in orbiting satellites. Some of these effects, such as single-event upsets and single-event functional interrupts, occur when ionizing radiation causes the logical value in a memory element to change and are detectable and correctable through various error mitigation techniques.

In this thesis, we develop and implement a hardware solution to combat these SEE for deployment as an experimental module on Naval Postgraduate School Satellite 1 (NPSAT-1). Based on the relatively benign orbit of NPSAT-1, industrial-grade, commercial-off-the-shelf components that have shown tolerance to radiation were selected to keep costs low. The primary source of mitigation relies on a globally triple-modular redundant microprocessor system instantiated inside of a XILINX Kintex-7 field-programmable gate array. The system consists of an open-source microprocessor without interlocked pipeline stages (MIPS) based processor softcore, a cached memory structure capable of accessing double-data rate type three and secure digital card memories, an interface to the main satellite bus, and XILINX's soft error mitigation softcore. The hardware was tested both in and out of the system and verified to work on the ground with faults injected. Other techniques to mitigate errors due to SEE, such as memory scrubbing, are intended to be added before launch.

14. SUBJECT TERMS Configurable Fault Tolerant Processor (CFTP), Global Triple-Modular Redundancy (GTMR), NPSAT-1, Field-Programmable Gate Array (FPGA), Single Event Effect (SEE), cache, memory controller, error mitigation, hardware design, softcore design			**15. NUMBER OF PAGES** 265
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IMPLEMENTATION OF THE CONFIGURABLE FAULT TOLERANT SYSTEM EXPERIMENT ON NPSAT - 1

Andrew S. Jackson
Lieutenant, United States Navy
B.S., Missouri University of Science and Technology, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Approved by: Herschel H. Loomis, Jr.
Thesis Advisor

James H. Newman
Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Space is a harsh environment, full of high-energy radiation that can cause single-event effects (SEE) in orbiting satellites. Some of these effects, such as single-event upsets and single-event functional interrupts, occur when ionizing radiation causes the logical value in a memory element to change and are detectable and correctable through various error mitigation techniques.

In this thesis, we develop and implement a hardware solution to combat these SEE for deployment as an experimental module on Naval Postgraduate School Satellite 1 (NPSAT-1). Based on the relatively benign orbit of NPSAT-1, industrial-grade, commercial-off-the-shelf components that have shown tolerance to radiation were selected to keep costs low. The primary source of mitigation relies on a globally triple-modular redundant microprocessor system instantiated inside of a XILINX Kintex-7 field-programmable gate array. The system consists of an open-source microprocessor without interlocked pipeline stages (MIPS) based processor softcore, a cached memory structure capable of accessing double-data rate type three and secure digital card memories, an interface to the main satellite bus, and XILINX's soft error mitigation softcore. The hardware was tested both in and out of the system and verified to work on the ground with faults injected. Other techniques to mitigate errors due to SEE, such as memory scrubbing, are intended to be added before launch.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ARM	Advanced RISC Machine
BCLK	bus clock
BTMR	block triple modular redundancy
C&DH	command and data handling module
CFTP	Configurable Fault Tolerant Processor
CMT	clock management tile
COTS	commercial-of-the-shelf
CPI	cycles per instruction
CPU	central processing unit
CS	chip select
DDR3	double data rate type three
DTMR	distributed triple-modular redundancy
FIFO	first in, first out
FPGA	field-programmable gate array
FSM	finite-state machine
GTMR	global triple-modular redundancy
IC	integrated circuit
IP	intellectual property
ISE	Integrated Synthesis Environment
IRQ	interrupt request
JTAG	Joint Test Action Group
LED	light emitting diode
LTMR	local triple-modular redundancy
MidSTAR-1	Midshipmen Space Technology Applications Research-1
MIPS	Microprocessor without Interlocked Pipeline Stages
NPSAT-1	Naval Postgraduate School Satellite-1
OE	output enable
PC	personal computer
PC	program counter
RAM	random access memory

SAA	South Atlantic Anomaly
SD	secure digital
SEE	single-event effect
SEFI	single-event functional interrupt
SEL	single-event latch-up
SEM	soft error mitigation
SET	single-event transient
SEU	single-event upset
SOC	system on a chip
SSAG	Space Systems Academic Group
TAP	test access port
TID	total ionizing dose
TMR	triple-modular redundancy
UART	universal asynchronous receiver transmitter
WE	write enable
XUM	eXtensible Utah Multicore

ACKNOWLEDGMENTS

I would like thank several individuals for the tremendous support they lent to me throughout the development of this second iteration of the configurable fault tolerant processor. First, to my thesis advisors Herschel Loomis and Jim Newman, thank you for providing me the greatest of opportunities, for being a sounding board and allowing me the freedom to explore new ideas, and for believing in my abilities to accomplish such a grand project. Second, to the civilian staff of the Space Systems Academic Group, particularly David Rigmaiden, Dan Sakoda, and Jim Horning, thank you for helping me to understand exactly how the configurable fault tolerant processor would integrate into NPSAT-1, for helping me to understand the environment we would be operating in, and for helping with hardware and software design. Third, to Ron Aikins, thank you for providing me access to and understanding all of the previous configurable fault tolerant processor software programs. Your assistance drastically reduced the amount of time it would have taken me to achieve the most basic forms of communication between the configurable fault tolerant processor and NPSAT-1. To Rita Painter and Anne Pickens, thank you for your diligence and assistance in acquiring the necessary hardware components to make this project happen. Finally, I would like to thank my friends and family, who helped keep me sane and pulled me away from long nights of coding when it was absolutely needed.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Space is a harsh environment full of high-energy particles that can induce errors—sometimes with catastrophic effects—in satellite electronics. For example, TDRS-1, which launched in 1983, was plagued with single event upsets (SEU) in its attitude control system memory, forcing constant ground monitoring. Some of those errors, if they had not been corrected within 90 seconds, would have caused the satellite to tumble out of control [1]. While advancements in materials have progressed over the years, and radiation-hardened electronics are available to protect against the space environment, these parts tend to have a low degree of integration with technology several generations behind the current state-of-the-art, no on-hand supply with long lead times to manufacture, and a high cost per component [2]. To take advantage of cutting-edge technology, reduce the cost of the satellite with commercial-off-the-shelf (COTS) components, and still maintain a high degree of reliability, a designer of flight hardware must carefully consider the environment the spacecraft will operate in to determine specific COTS components to use and to build in the appropriate amount of redundancy and error correction capability.

A. OBJECTIVES

In this thesis, we detail the design, test, and implementation of an updated version of the Configurable Fault Tolerant Processor (CFTP) flight hardware. The CFTP is an experimental module to be flown on NPSAT-1 that will test the feasibility of using low-cost COTS hardware, vice radiation-hardened hardware, to create a reliable, reconfigurable, fault-tolerant system. This thesis is motivated by a subset of the design objectives of the CFTP program as defined by the creator of the original CFTP board, Dean Ebert, and centers its design on the use of a field-programmable gate array (FPGA).

The first objective is that the CFTP must be reconfigurable [3]. This is important because the softcore of the system on a chip (SOC) implemented on the FPGA is stored in its configuration memory and is susceptible to the effects of radiation. In the event the configuration is corrupted, the FPGA must be able to be reconfigured to restore the

design. Furthermore, the ability to reconfigure while on orbit allows the device to be upgraded should a more efficient softcore be developed after launch. It also allows a new softcore architecture to be uploaded and tested.

The second objective is that the CFTP must utilize COTS technology [3]. We want to prove that, through careful design, significant savings can be realized without incurring a significant amount of increased risk. The use of COTS technology also allows the use of more advanced, current generation technologies.

The third objective is that the CFTP must be reliable. That is, the design must be able to mitigate or recover from any non-destructive fault caused by the effects of radiation [3]. In this thesis, we consider both the reliability of individual hardware components as well as reliability of the softcore. When considering hardware, due to the complexity of interconnecting current generation integrated circuits (IC), we use a pre-configured daughter board. This ensures that for the most complex components, we begin our design from a state known to be in a reliable configuration in the absence of radiation effects. For similar reasons, we make extensive use of open-source designs and intellectual property (IP) in the development of our FPGA softcore. Starting from these known reliable states, we then modify our design to compensate for the effects of radiation.

The final objective is that the CFTP must satisfy budget considerations, specifically in regard to size, weight, power consumption, and heat [3]. The CFTP hardware design must fit within the space allocated within the command and data handling (C&DH) module. The C&DH utilizes a standard size circuit card of 7.3 inches by 5.3 inches with a spacing of 0.435 inches between printed circuit boards. The weight requirements for the CFTP are negligible. Power is regulated internally to the C&DH and generates 20 W at both 15 V and 5 V; however, the CFTP only utilizes the 5-V power rails. The 5-V rails are also shared with other boards in the C&DH, so power usage must be balanced. Thermally, due to the positioning of the CFTP at the top of the C&DH, we can sink waste heat directly into the aluminum casing of the entire module, greatly increasing the radiative surface for heat dissipation.

B. DEFINING THE PROBLEM

Before beginning any design, it is necessary to have a firm understanding of the problem. In the case of the CFTP, a precise understanding of the space environment in which it will operate and the effects of that particular environment on our hardware define the problem. In this section, we examine the radiation environment and then discuss the various effects we are likely to see manifested in our system.

1. The Space Environment

We used the European Space Agency's online tool, the Space Environment Information System, to characterize our environment. The primary inputs into this system are the orbital parameters of the satellite, the launch date, and mission duration. NPSAT-1, anticipated to be launched September 2016 as part of the Space Test Program-2, will have a circular orbit at 720 kilometers with an inclination of 24 degrees and a mission duration of 18 months [4]. Due to the low altitude and latitude of the mission, radiation originating as solar proton events are effectively shielded by Earth's geomagnetic field, so only trapped ions are considered [5]. We used the AE-8 and AP-8 models—the standard for modelling trapped radiation since the 1970s [5]—for electrons with energies greater than 1 MeV and protons greater than 10 MeV, respectively. In Figure 1, we see that the CFTP will primarily be affected by radiation from the South Atlantic Anomaly (SAA). For comparison, the same models were run at 3500 km, roughly the center of the Inner Van Allen Belt. We see that the flux we will experience in the SAA is more than two orders of magnitude less than would be experienced in the center of the belt, and a majority of the time the satellite will be located outside of the SAA, where energy levels of the ions are not as likely to have an impact.

The next model we ran was SHIELDOSE-2 to determine the total radiation dose that the CFTP will receive for the duration of its mission. The model takes into consideration the orbit and any previously defined sources for radiation. Again, our primary source will be from trapped ions. In addition, it will also determine the amount of radiation received for various thicknesses of aluminum shielding. Our results are displayed in Table 1.

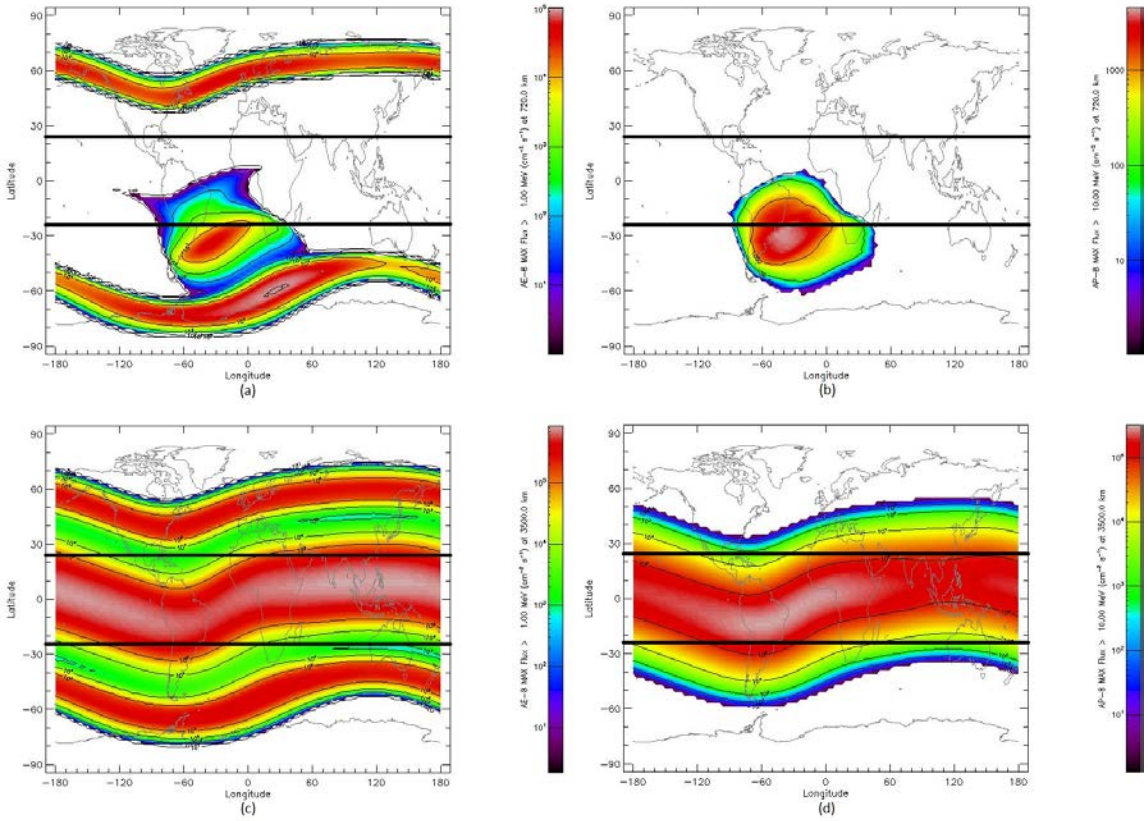


Figure 1. Flux of Trapped (a) Electrons at 720 km, (b) Protons at 720 km, (c) Electrons at 3500 km, and (d) Protons at 3500 km.

Table 1. Total Mission Dose for NPSAT-1 (rad)

Al absorber			Total	Trapped electrons	Brems-strahlung	Trapped protons	Tr. electrons+ Bremsstrahlung	Tr. el.+Bremss. +Tr. protons
(mm)	(mils)	(g cm^{-2})						
0.05	1.97	0.01	254000	251900	491.1	1635	252400	254000
0.1	3.94	0.03	155900	154000	316.1	1523	154400	155900
0.2	7.87	0.05	75440	73840	168.1	1433	74010	75440
0.5	19.69	0.14	10690	9422	42.61	1224	9465	10690
1	39.37	0.27	2298	1237	14.09	1046	1251	2298
1.5	59.06	0.41	1432	458.6	8.62	964.7	467.2	1432
2	78.74	0.54	1141	231.5	6.4	903.2	238	1141
2.5	98.43	0.68	997	136.4	5.16	855.5	141.5	997
3	118.11	0.81	906.7	86.02	4.37	816.3	90.39	906.7
4	157.48	1.08	794.6	36.1	3.4	755.1	39.49	794.6

The CFTP resides within the command and data-handling module of NPSAT-1. This module is housed within an aluminum rectangular parallelepiped, which has a thickness of 7 mm on four of its sides but only 1 mm on two of its sides. The command and data-handling module resides within the satellite's bus, which has an aluminum shell 1 mm thick at its thinnest. In total, at its thinnest, we can assume there is 2 mm of aluminum shielding. This results in an estimated mission dose of 1.14 krad.

2. Total Ionizing Dose and Single Event Effects

We determined in the previous subsection that, while NPSAT-1's orbit will be benign, it will still encounter radiation levels with high enough energies to cause concern while it passes through the SAA. These effects can be categorized into the long-term effects, or total dose, and the instantaneous effects, or single-event effects (SEE). Single-event effects can be further broken down into destructive and non-destructive effects with various subcategories defined for each. The primary source for SEE in NPSAT-1 will be due to high energy protons encountered in the SAA, while total dose is a cumulative effect of all radiation encountered while on orbit.

Total Ionizing Dose (TID) occurs due to ionization of a component over time, which leads to the component's failure. This effect can manifest itself by changing the electrical potential required for gate operation, increasing leakage currents, or changing characteristics of how a component functionally operates [6]. For the CFTP, with an estimated total mission dose of just over 1 krad, TID is not a major concern. Most commercial components have a TID rating of at least 5 krad [2].

The most common SEE and a description of their effects are shown in Table 2. Component selection for the CFTP's printed circuit board will primarily be concerned with mitigation of the destructive SEE, particularly with single-event latch-up and burnout. The design of the softcore, located inside the FPGA, will be the primary mitigating source for non-destructive SEEs: single-event transient, single-event upset, and single-event functional interrupt.

Table 2. Listing of Single-Event Effects

Acronym	Effect	Description
SEU	Upset	Digital circuit changes logic state
SEL	Latchup	Device switches to a destructive, high current state
SEGR	Gate Rupture	Destructive failure of a power transistor
SEB	Burnout	Another mode of destructive failure for a power transistor
SEFI	Functional Interrupt	Device enters mode where it is no longer performing the designed function
SEMBE	Multiple Bit Error	More than one logic state change from one ion
SET	Transient	Transient current in circuit
SEIDC	Induce Dark Current	Increased dark current in CCD arrays

Source: [6] J. W. Howard and D. M. Hardage, "Spacecraft environments interactions: space radiation and its effects on electronic systems," NASA, Hampton, VA, Tech. Pap. TP-1999-209373, Oct. 1999.

C. ORGANIZATION

Our motivation and an overview of the problem was discussed in this chapter. The remainder of the thesis is organized as follows. Background on various triple-modular redundancy (TMR) implementations and error correction techniques utilized throughout the history of the CFTP program and the softcore processor selected for this design are discussed in Chapter II. The design and implementation of both the printed circuit board and the softcore system instantiated in the FPGA is detailed in Chapter III. The results from simulation and in-system testing are presented in Chapter IV. Finally, conclusions and recommendation for future studies are provided in Chapter V.

D. ADDITIONAL DOCUMENTATION

Several appendices are included in this thesis to provide insight into the current iteration of CFTP. The modified Verilog source code of the eXtensible Utah Multicore (XUM), as well as the remainder of the source code for the CFTP softcore, are contained in Appendix A. Furthermore, sources for the software utilized from the XUM project to conduct testing and verification are included. The source code for programs executed on the satellite's Advanced RISC Machine (ARM) processors to interact with the CFTP is given in Appendix B. Finally, the schematics and Gerber drawing for the CFTP printed circuit board are contained in Appendix C.

II. BACKGROUND AND PRIOR WORK

In the first chapter, we looked at the radiation environment in which the second iteration of the CFTP will be operating and the possible effects that environment will have on our system. In this chapter, we discuss the history of the CFTP, which encompasses many variations of the triple-modular redundancy (TMR) scheme. We also provide an overview of the baseline softcore SOC, the XUM, used in our design. Finally, based on previous design work, we define the scope and direction of this thesis.

A. CFTP HISTORY

Before continuing with research on the CFTP project, it was necessary to assess what had previously been accomplished. The CFTP project has been active for nearly two decades at NPS and has already launched one iteration of its flight board into space. In the next few subsection, we examine the initial research into a COTS based TMR flight board, the resulting design work that led to the CFTP's first space flight, and the follow-on research conducted since.

1. Early Work

The CFTP program at NPS traces its origins to a thesis drafted in 1998 by John C. Payne. Payne's thesis [7] was motivated by a 1994 defense directive that mandated the use of COTS wherever possible in military systems. In his research, Payne provided the framework for a fault-tolerant system utilizing COTS components intended to overcome the radiation effects of a space environment. His solution implemented a block triple-modular redundant (BTMR) scheme, in which he used three separate hardware processors with external voters to majority vote the outputs, as shown in Figure 2. His design was later implemented and tested by Dave Summers [8] and Damen Hofheinz [9], respectively, in their theses research. The experiment ultimately ended in electrical failure of the board, and the experiment was plagued by major synchronization issues between the three processors throughout testing [10].

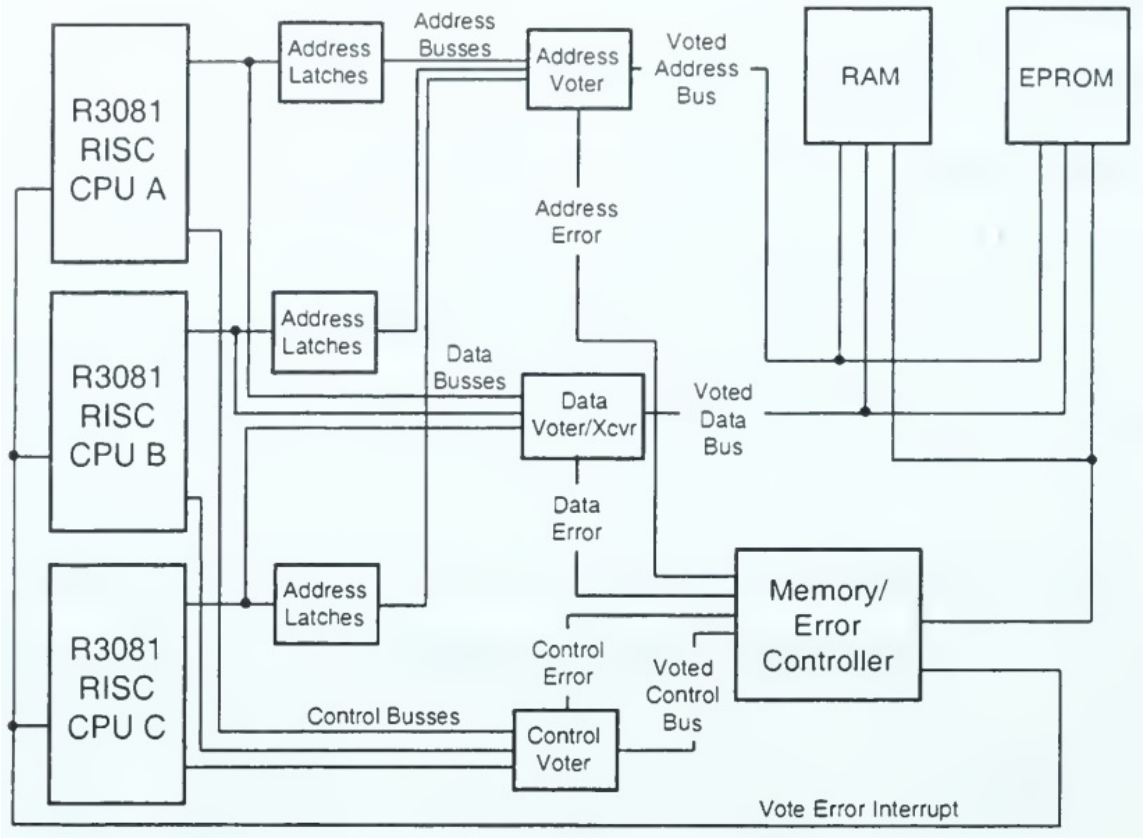


Figure 2. Triple-Modular Redundant R3081 Board Design

Source: [7] J. C. Payne, "Fault tolerant computing testbed: a tool for the analysis of hardware and software fault handling techniques," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 1998.

In 2002, Peter Lashomb [10] conducted an in-depth review of various alternative hardware technologies for implementing a BTMR structure of processors since the previous board had to be redesigned. He determined that FPGAs would make a suitable alternative and solve the synchronization issues previously encountered. Utilizing XILINX's 8-bit PicoBlaze softcore processor, he was able to re-create and successfully simulate a portion of Payne's hardware design as a functional softcore within XILINX's Integrated Synthesis Environment (ISE) design environment. The design consisted of a BTMR structure of three PicoBlaze processors with output voters. All memory, including the memory controller, was assumed to be implemented external to the FPGA. The successful simulation of the softcore set the stage for what became the first successful implementation of the CFTP.

2. MidSTAR-1 Implementation

The majority of research and development for the first successful implementation of the CFTP occurred in 2003 by Dean A. Ebert, Steven A. Johnson, and Rong Yuan. As mentioned in Chapter I, Ebert [3] developed the printed circuit board. His hardware solution, shown in Figure 3, consisted of two FPGAs. FPGA 1, X1, communicated with the satellite's main processing unit and acted as a control for the experimental BTMR softcore processor implemented in FPGA 2, X2. Johnson [11] developed the softcore for the BTMR processor, upgraded from the 8-bit PicoBlaze processor used in Lashomb's design to the 16-bit KDLX processor. In a traditional BTMR scheme, you do not have access to the internal registers of the individual modules being triplicated. This allows errors to collect over time and eventually causes the system to fail [12]. Johnson was able to overcome this through the implementation of his *Interrup* module. When an error was detected, an interrupt was generated to execute a service routine that caused the internal registers of the processor to be written to memory and then read back. Since register values pass through voters on the way to memory, they are corrected and incorrect values are replaced upon readback. Yuan [13] conducted system level integration and testing of the softcore. The launch of CFTP aboard Midshipmen Space Technology Applications Research-1 (MidSTAR-1) in March 2007 was the culmination of their combined efforts.

While the CFTP was operable in space, it still had several vulnerabilities. It was built on static random access memory (RAM) based FPGAs, whose configuration memories are susceptible to SEUs [12], with no method of configuration memory scrubbing. The TMR scheme was applied only to the processors, allowing SEEs that manifested in other parts of the system to cause the CFTP to become unresponsive and require a hard restart. Finally, the CFTP was unable to correct SEUs in real-time and required many clock cycles to return the processor to a normal operating state. These additional research areas still required exploration.

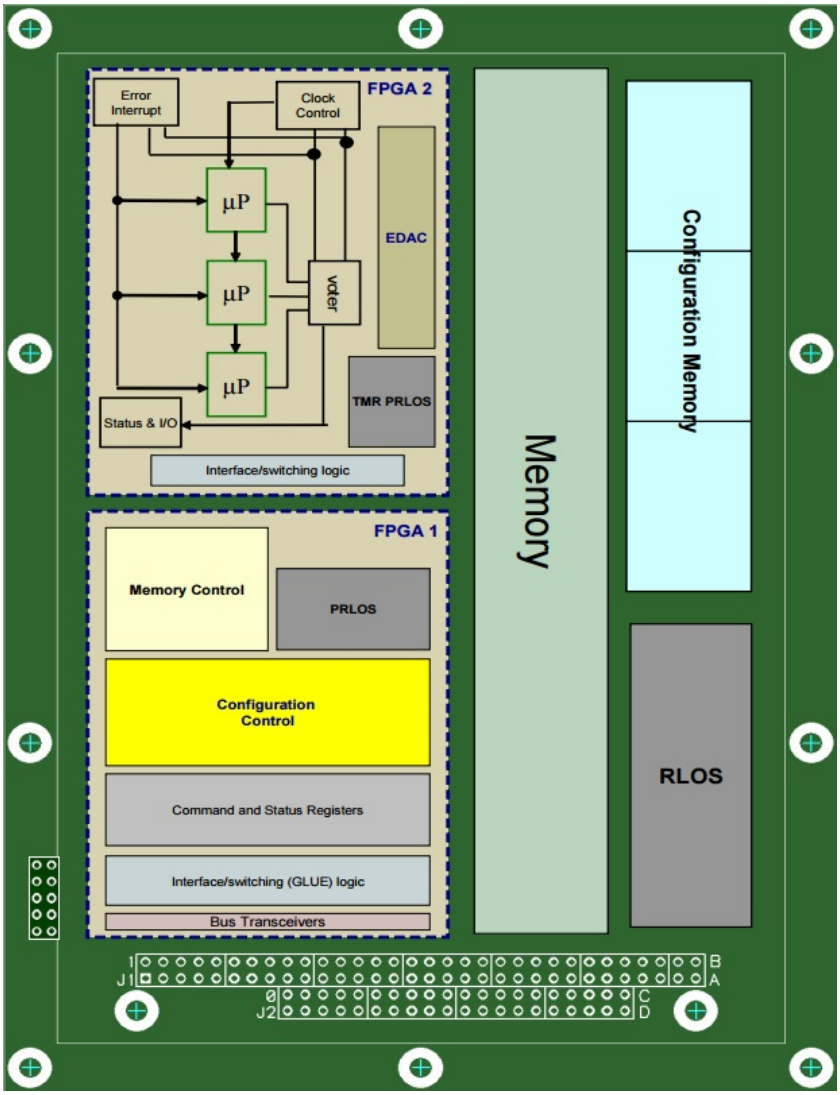


Figure 3. Final CFTP Concept Drawing

Source: [3] D. A. Ebert, "Design and development of a configurable fault-tolerant processor (CFTP) for space applications," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2003.

3. Post-MidSTAR-1 Recent Work

Though not included in the softcore for CFTP aboard MidSTAR-1, James C. Coudeyras and Peter J. Majewicz made significant contributions to the project in 2005 prior to the launch of MidSTAR-1. Coudeyras [14] was able to successfully implement a method for accessing the configuration memory of the FPGA while it was running to perform partial reconfiguration and error injection. Though scrubbing of the

configuration memory was overlooked in his thesis, this was an important first step toward that end. Meanwhile, Majewicz [15] began to explore what he termed Internal Triple-Modular Redundancy, which focused on implementing the TMR at the register level vice the module or block level and again upgraded the processor, this time to the 32-bit Pix processor. The specific form of his TMR was a cross between what is now known as Local Triple-Modular Redundancy (LTMR), shown in Figure 4, and Distributed Triple-Modular Redundancy (DTMR), shown in Figure 5. In his design, he triplicated the combinational logic and registers but still retained single voters out the register outputs. Like Coudeyras' work, this implementation was also a step in the right direction but was still susceptible to errors should a single-event transient (SET) occur on the voter output and be captured in following registers or a SEU occur in the configuration memory that affected the structure of the voting circuitry itself.

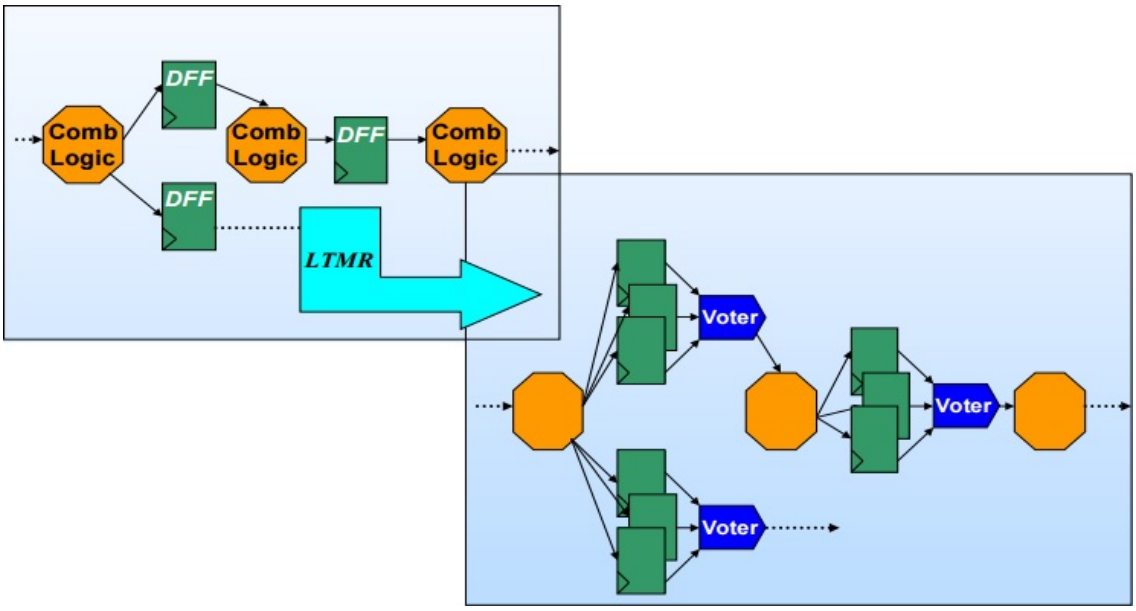


Figure 4. Local Triple-Modular Redundancy Implementation

Source: [12] M. D. Berg, "Single event effects in FPGA devices 2014–1015," presented at NASA Electronic Parts and Packaging Program Electronics Technology Workshop, Greenbelt, MD, 2015.

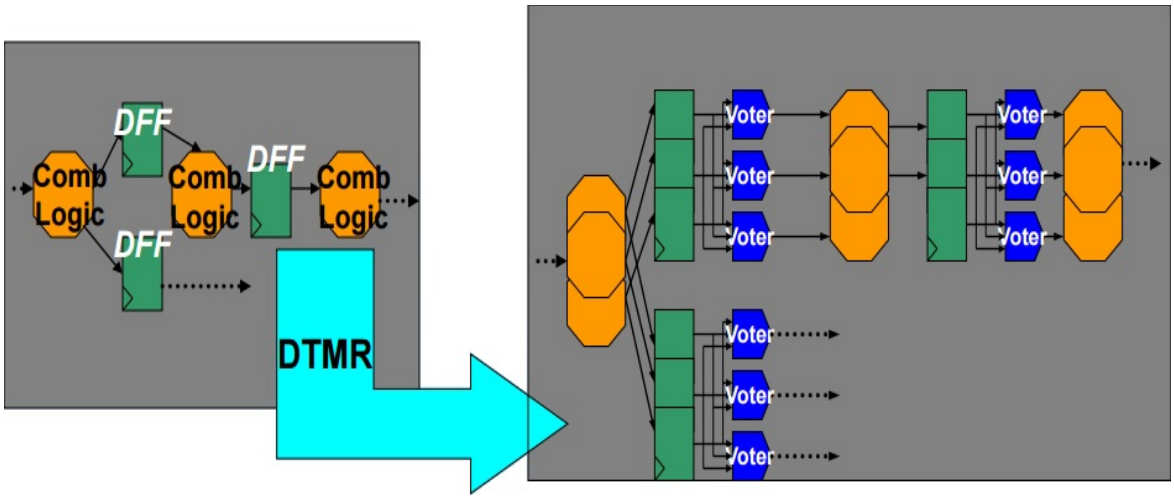


Figure 5. Distributed Triple-Modular Redundancy Implementation

Source : [12] M. D. Berg, "Single event effects in FPGA devices 2014–1015," presented at NASA Electronic Parts and Packaging Program Electronics Technology Workshop, Greenbelt, MD, 2015.

After the research conducted by Coudeyras and Majewicz, the CFTP project slipped into a lull period. In 2008, David E. Dwiggins [16] performed a massive overhaul of X1 and added in two key functionalities. He was able to implement a method for X1 to provide a continuous scrub of X2's configuration memory, and he also applied DTMR to the softcore design located within X1. While these modifications increased the reliability of CFTP, they consumed nearly all of its resources and still left major vulnerabilities within the design, notably that X1's configuration memory was not protected and the design for X2 was still not fully protected by DTMR. Finally, in 2014 Jordan K. Goff [17] began implementation of a processor based on the microarchitecture of the Microprocessor without Interlocked Pipeline Stages (MIPS), developed by MIPS Computer Systems, Inc, and a universal asynchronous receiver transmitter (UART), which were both fully protected by DTMR.

B. EXTENSIBLE UTAH MULTICORE

It was decided at the outset of this thesis research that this iteration of the CFTP would be developed from a pre-existing softcore for the processor. Any softcore selected needed to meet the following characteristics. First, it needs to be open source. While there are numerous softcore processors available, to implement the more advanced TMR

schemes we need to be able to modify the internal structures of the softcore; this is not possible for proprietary softcores. Second, it needs to utilize as few resources as possible. Any softcore instantiated would immediately take three times the amount of resources to triplicate, and an additional overhead of voting logic would be applied on top of that. After the processor softcore was triplicated and voted, additional resources are needed to access input and output ports to communicate with the satellite and off-chip memory. Third, a softcore processor that had high-level language support, such as C, was desired. With the exception of Dwiggins' design, all previous CFTP implementations relied on their authors to hand-generate assembly and machine code to execute on their processors. This is not practical when writing any meaningful piece of software to execute while on orbit. Finally, the softcore code needed to be well commented and easily understood. A MIPS-based architecture was preferred, as the CFTP has utilized that architecture in many of its iterations.

After researching several softcore processors, the XUM softcore [18], designed by Grant Ayers, was found to match all criteria. It is licensed under the GNU Lesser General Public License, which allows for modification of the original source code. It is small for a 32-bit processor, with a single core utilizing only five percent of the resources on the FPGA for which it was designed, a XILINX Virtex 5 110t FPGA. The XUM architecture, shown in Figure 6, implements release one of the 32-bit MIPS instruction set architecture, MIPS32R1, with interrupt handling via a full co-processor zero and utilizes a five-stage pipeline with hazard detection and forwarding. Finally, the source code is extremely well commented, and the supporting documentation contained instructions for building a MIPS cross-compiler utilizing CYGWIN, a Linux development environment run on windows, as well as a simple bootloader and various demonstration programs coded in C.

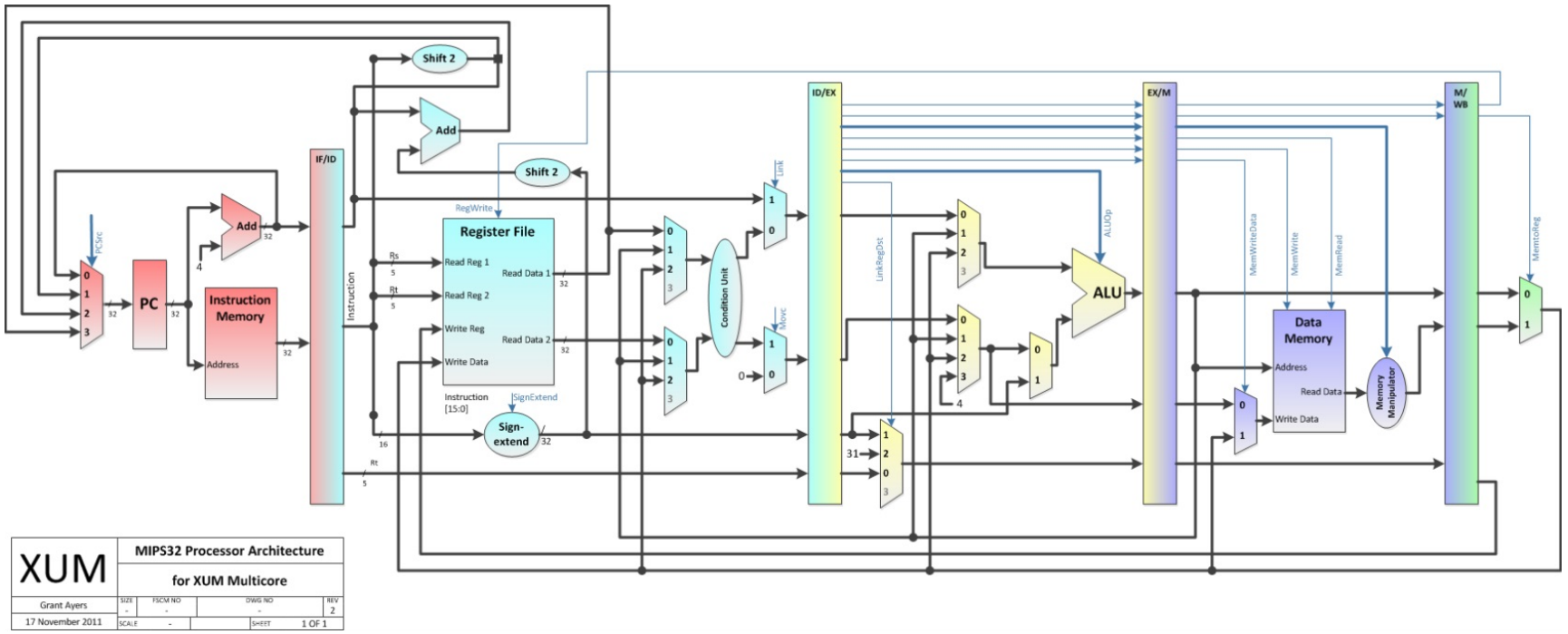


Figure 6. MIPS32R1 Processor Architecture for XUM

Source : [18] G. Ayers, Salt Lake City, UT. (2014). [Online]. *The eXtensible Utah Multicore*. Available: https://github.com/grantae/mips32r1_xum. Accessed May. 5, 2015.

C. DESIGN SCOPE AND DIRECTION

In preparation for the launch of the CFTP aboard NPSAT-1, we performed a complete overhaul of CFTP's 2003 design to include the latest research. As the control and experimental FPGA devices were nearly at maximum resource usage, a new FPGA is selected, and the printed circuit board is re-designed. This allows us to simplify the hardware layout by collapsing both the control and experimental designs into a single FPGA and to capitalize on technologies and IP cores integrated into FPGAs over the past decade. The primary TMR technique realized in our softcore is global triple-modular redundancy (GTMR), shown in Figure 7. GTMR provides further protection over DTMR in that the global clock lines for individual systems are isolated from each other; therefore, the TMR design is capable of surviving a SET on a global clock [12]. Furthermore, this GTMR extends to every module of the softcore capable of being triplicated. Finally, configuration memory scrubbing is implemented for the entire softcore design.

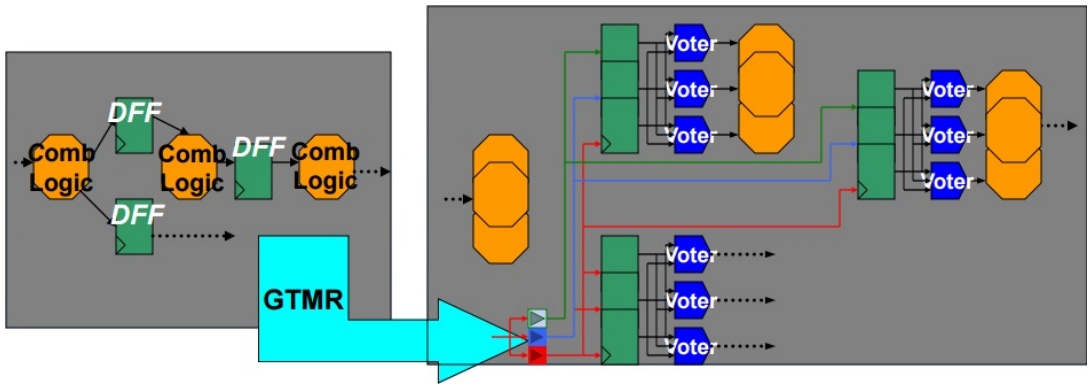


Figure 7. Global Triple-Modular Redundancy Implementation

Source : [12] M. D. Berg, "Single event effects in FPGA devices 2014–1015," presented at NASA Electronic Parts and Packaging Program Electronics Technology Workshop, Greenbelt, MD, 2015.

D. CHAPTER SUMMARY

In this chapter, we bounded the scope of our design for the second iteration of the CFTP based on previous design work. We discussed the implementation of both the hardware and softcore developed for the first iteration of the CFTP launched aboard MidSTAR-1 and continued research into more advanced triple-modular redundant

techniques and configuration memory scrubbing. We noted that building an entire MIPS-based processor from scratch would take a lengthy amount of time, and instead, we examined the open source XUM softcore MIPS processor that serves as the base of our design. In the next chapter, we discuss the actual implementation of both the hardware and softcore used for this iteration of the CFTP to be launched aboard NPSAT-1.

III. IMPLEMENTATION

The process and reasoning used in the implementation of this iteration of the CFTP are highlighted in this chapter. The method used to transform the XUM processor into one that is globally triple-modular redundant is discussed. We also detail how we expanded out from the processor to build a reliable memory hierarchy, how we interface with the satellite main processor, and how we protect our loaded configuration memory. We explore the hardware designed to run the new configurable fault tolerant processor. Finally, we examine storage of the FPGA configuration files and the method used to configure the FPGA.

A. SOFTCORE DEVELOPMENT

We used both XILINX ISE and XILINX Vivado at various stages during development of the softcore. The softcore was constructed piecewise by building a component, testing the component, and finally, integrating it into the larger design. This methodical approach became crucial as the design grew due to ever-increasing synthesis and implementation times. It also aided in identifying bugs earlier, as they were not obfuscated within the larger design. The complete source code for the CFTP softcore is given in Appendix A.

1. XUM Softcore Processor Modifications

We first verified the functionality of the XUM SOC softcore before performing any modifications. We conducted this on a XUPV5-LX110T development board, the same used to build the original core [18], to eliminate any design variables. Later, development was conducted on a Genesys 2 development board featuring a Kintex-7 325T FPGA. We found one minor issue involving the UART instantiated within the system and communicated this back to the author of the core for verification. We also noticed that memory accesses were not keeping up with the demand of the processor. Modifications were made to the core to resolve these two issues and then techniques were applied to the core to make it internally triple-modular redundant.

a. UART

The UART, which has a dual functionality of being able to either boot-load a new program into memory or allow a user program to communicate [18], was found to stop receiving data for user programs after sending its first transmission. We discovered that during the write back of any character, BootSwEnabled transitioned from low to high, and following that transition, the receive first in, first out (FIFO) module quickly emptied. The issue is caused by a race condition. When BootSwEnabled goes high, it switches to boot-load mode and empties the receive FIFO searching for a specific string of character and causing the user program miss all of its input data. This is shown in Figure 8. While we were able to correct the issue with the UART, and used it extensively in testing, we removed the entire module from the final design since NPSAT-1 communicates over its PC-104 bus; however, the concept of using the communication interface to control, temporarily, the instruction data bus for reprogramming the operating system was kept and adapted into our PC104 interface.

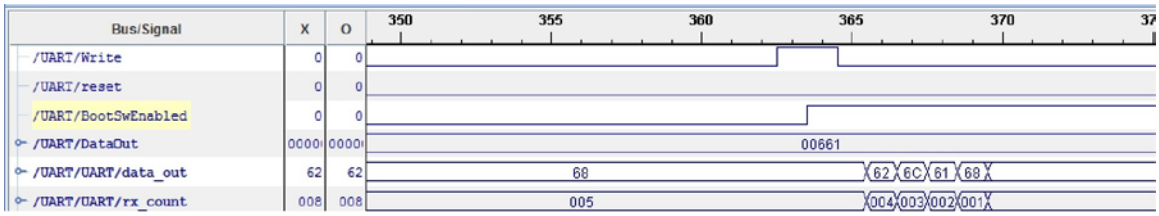


Figure 8. Waveform Demonstrating XUM UART Improperly Transitioning BootSwEnabled on Write

b. Adjusting the Memory Access Protocol

The XUM core utilizes a four-way handshake to retrieve instructions and data from memory. In the original core, memory is instantiated using block RAM with registered outputs. Examining both the code and waveforms produced when executing a simple program on the processor, we noticed that even with the memory clocked at twice the speed of the processor, 66 MHz and 33 MHz, respectively, the memory was unable to keep pace with the processor and took three central processing unit (CPU) clocks to retrieve one instruction as shown in Figure 9. The author himself notes that “instruction

memory fetches only once per handshake, the minimum theoretical CPI is greatly increased from 1 to between 3 and 4" [18].

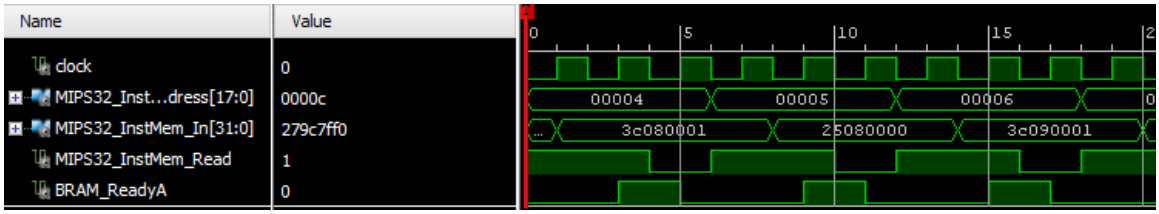


Figure 9. Waveform Capture of XUM Instruction Fetch Utilizing Four-way Handshake Protocol

To resolve this issue, we implemented a L1 cache onboard the FPGA and replaced the handshake protocol with a simple flagging protocol. In the new protocol, we assume that the processor makes requests to the L1 cache every CPU cycle, either a read or write. Based on this assumption, we designed our cache to be able to resolve a hit within the same CPU clock cycle it is received. Because the cache is able to respond every CPU clock cycle, only a single valid line needs to come from memory to the processor. If the valid line is invalid, the processor stalls; otherwise, it accepts the data on the data lines.

c. Application of the GTMR Architecture

The process of applying the GTMR architecture to XUM consisted of a two-step approach. We first BTMR the system as a whole. Then, we GTMR the system module by module. We also structure our code to resemble the BTMR hierarchy, with all voting, both BTMR and GTMR, taking place at the level where the systems are triplicated as shown in Figure 10. The benefit of this structuring is that there is little disturbance to the internal interconnect of the systems beneath the TMR hierarchical level, so any system viewed below the TMR hierarchical level can be viewed as a single operable system. The inputs and outputs of this single system can be directly mapped to the top-level ports, and the system functions. This allows us to test a single system for functionality and reduce compilation times.

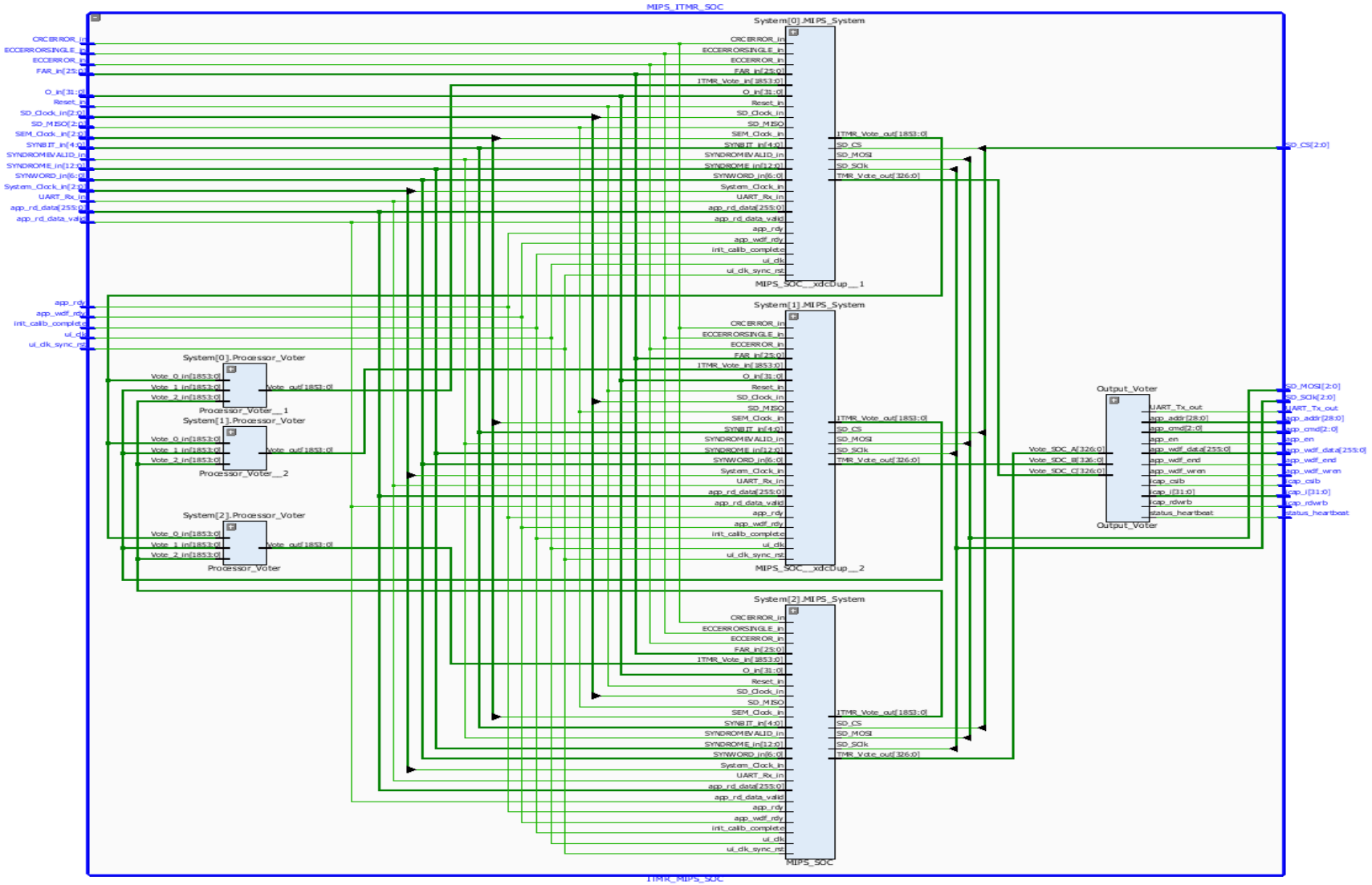


Figure 10. Schematic Drawing of the CFTP TMR Level

In applying BTMR, we must first identify all parts of the system that cannot be triplicated. These are instantiated on the top level. In the case of the original core, this was only the top-level pins; however, we also include the clock management tile (CMT) on the top and generate triplicated clock outputs from it. The reason for this is to minimize clock skew / jitter, which can be induced by chaining together multiple CMTs. The top module then routes the appropriate signals to the TMR level.

At the TMR level, the entire system was triplicated. Inputs from the top level are distributed to each of the three systems, and outputs from each system are voted on before being sent to the top level. This completes the BTMR implementation, and we verified that the system was operable with three processors executing synchronously.

To progress from our BTMR XUM system to a GTMR system, we had to first locate every register inside of the XUM softcore. These registers were then modified so that their outputs were directly routed to the TMR level vice the connecting logic. After progressing through the GTMR voter, the signal was then routed back to the connecting logic for which it was originally destined. Furthermore, all registers were modified to update every clock cycle. In the event the processor stalled or a register was not going to be actively written to, such as those in the register file, a voted on output is still updated in each. Since the process of modifying any single processor did not compromise the system functionally, we were able to apply the GTMR module by module and verify the system in between updating each module so that any errors produced could more readily be identified.

This standardized process of applying the BTMR and GTMR architectures was applied to every submodule later instantiated within CFTP with the exception of the following: the CMT, the internal configuration access port and frame error correction code primitives, and the double-data rate type three (DDR3) IP softcore.

2. Clock Distribution

Clock distribution is extremely important in FPGA designs that incorporate any sort of sequential logic. Improper implementation can lead to setup and hold time

violations and dramatically increase skew. Distribution becomes further complicated when multiple clock frequencies are used, which is the case for the CFTP.

The schematic for the CFTP's clock generation is shown in Figure 11. In it, we see the primary clock used to generate all derived clocks is based off a 200-MHz differential source. A differential source was selected as our primary source to mitigate the effects of SETs between the oscillator and input pins on the FPGA. Upon entering the system, both clock signals are globally buffered. This allows the CMT to be placed optimally within the design in a location to minimize clock skew across the entire design vice only being able to be placed within the clock region where the clock signals enter the FPGA. For similar reasons, to minimize clock skew and jitter, all derived clocks are generated within the same CMT, with the exception of the soft-error mitigation (SEM) clocks that do not interact with the SOC user design.

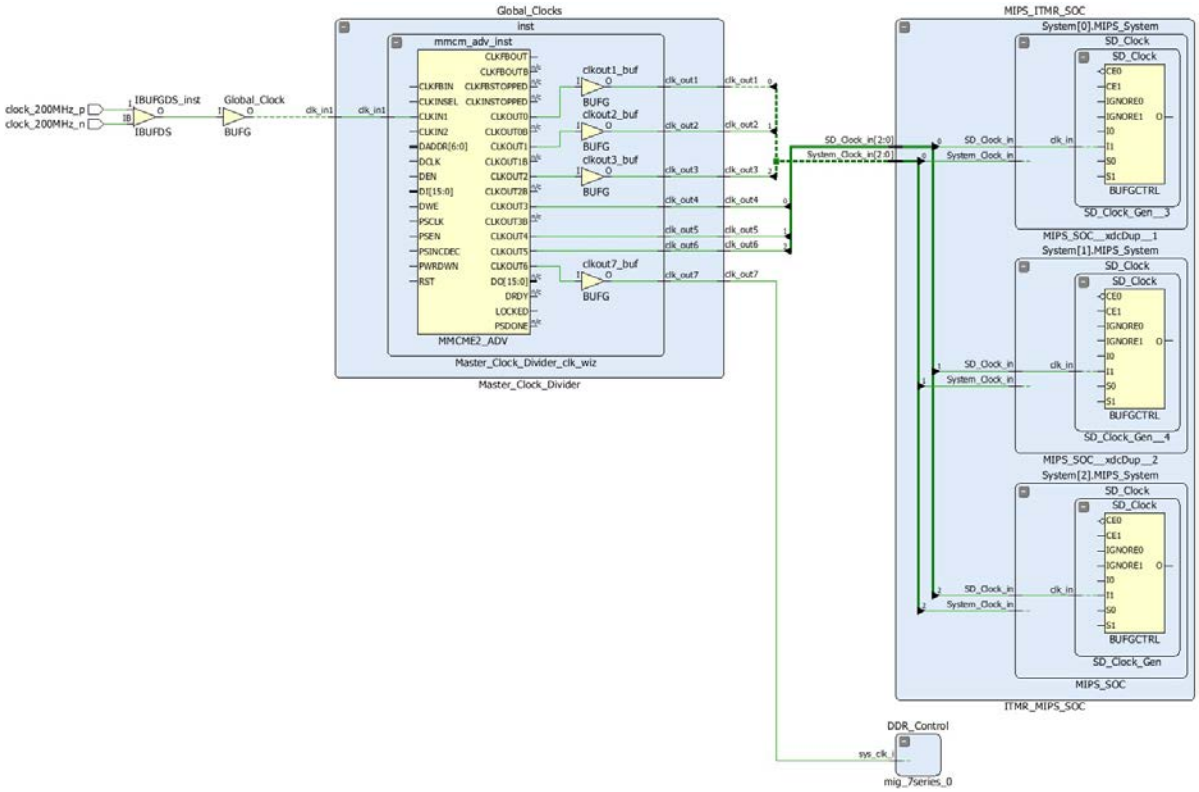


Figure 11. Schematic Drawing of the CFTP Clock Distribution

The clock management tile generates seven output clocks. Three buffered 50-MHz System_Clocks, three unbuffered 50-MHz SD_Clocks, and one buffered 200-MHz DDR_Clock. One System_Clock and one SD_Clock are then routed to each of the three systems. This distribution of the clock allows for individual systems to be affected by a SET on their clock lines, and as long as it does not happen to two of the systems within the same clock cycle, the SET is masked.

The reason the SD_Clocks are unbuffered is because the SD_Module requires two different clock speeds, a 400-kHz clock for initialization and 50 MHz for normal operations. Again, to minimize skew, it is desirable that global clocks pass through the same number of global buffers, as the buffers themselves can add a significant amount of delay. To accomplish this the unbuffered 50-MHz signal is further divided within the SD_Clock_Gen module to generate the 400-kHz signal, and a two-input, one-output global buffer is used to select the appropriate clock for distribution during operations based on the SD controller's current state.

The DDR3 Controller takes only a single clock for its input. Unfortunately, this means that the DDR3 module is susceptible to SETs. It is also susceptible to SEU and single-event functional interrupts (SEFI). The reason for this is that we were unable to apply any sort of TMR to this IP core. The primary issue preventing the TMR of this core is that the tristate buffers, which drive the bi-directional bus, are buried within the IP core and we do not have access to the signal that turns the tristate buffers on and off. This is a problem because if we run the bi-directional lines through a voter we have to specifically declare them as inputs or outputs and, thus, we are either always reading from or driving the lines at the top level, depending on how the voting was implemented, which is incorrect.

The DDR3 module also creates another clocking issue in that it runs at a faster clock speed than the rest of the system and access times vary from transaction to transaction. We were able to account for this by using the same four-way handshake protocol introduced in the original XUM SOC memory model. The four-way handshake accounts for any delays in the DDR3 transaction. Furthermore, the control lines guarantee that data is valid and held on the data lines long enough to be clocked into registers when

passing between the 50-MHz and 200-MHz clock domains. To ensure that these data paths are not marked as critical we must constrain the clocks to account for multicycle paths. An excerpt of these constraints is shown in Figure 12.

```
#Fast to Slow Clock
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] 3
#Slow to Fast Clock
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 3
```

Figure 12. Constraining Multicycle Paths Between System Clock 1 and DDR Clock in the CFTP

3. Memory Hierarchy

The memory hierarchy implemented in the CFTP is realized through multiple layers of caching. Instruction and data requests from the processor, as depicted in Figure 13, are serviced by separate L1 instruction and data caches. Should a miss occur in the L1 cache, the requests are then forwarded on through the bus arbiter to the L2 cache controller. Since the L2 cache is a shared cache, the bus arbiter ensures that only one access occurs at a time. The L2 cache utilizes the DDR3 memories as its data ram, and when a hit occurs, the requested data is retrieved from the DDR3 memories and backfilled into the requesting L1 cache. If both the L1 and L2 caches miss, the data is retrieved from the secure digital (SD) cards and backfilled into the DDR3 memory and requesting L1 cache.

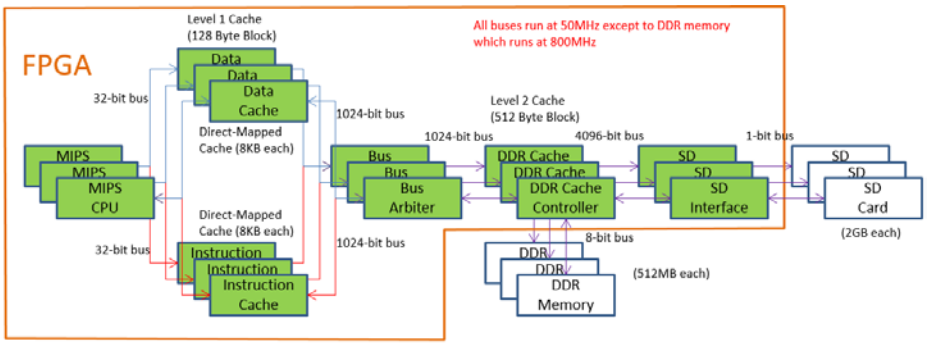


Figure 13. Concept Drawing of the CFTP Memory Architecture

The CFTP presents a 4-GB flat addressed space of which the bottom 2 GB are used for physical memory. The mapping of CFTP's address space at the various levels in the memory hierarchy is shown in Table 3, and the ranges for tags, indices, and offsets in the L1 and L2 caches are defined. It can be seen that the L2 cache utilizes address bits A-1 and A-2; this is because all of the DDR3 memory devices are seen as a single device and, when combined with a fourth DDR3 IC not used, to give a total of 2 GB of available DDR3 memory. To allow for triplication of the data, we shift its addressable space to the right by two and use the A-1 and A-2 address bits to produce duplicate copies of data in each of the individual DDR3 chips. This reduces the DDR3's addressable space to 512 MB. Similarly, all data is triplicated and stored on each of the 2-GB SD cards, but because each SD card is 2 GB, no shifting of the address space is necessary.

Table 3. Mapped Address Space within the CFTP Memory Hierarchy

Addressable	4 GB	2 GB	1 GB	512 MB	256 MB	128 MB	64 MB	32 MB	16 MB	8 MB	4 MB	2 MB	1 MB	512 KB	256 KB	128 KB	64 KB
CPU	A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16	A15
Level 1 Cache	I/O	T17	T16	T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2
Level 2 Cache	I/O	T1	T0	I19	I18	I17	I16	I15	I14	I13	I12	I11	I10	I9	I8	I7	I6
SD Card	I/O	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16	A15
Addressable	32 KB	16 KB	8 KB	4 KB	2 KB	1 KB	512 B	256 B	128 B	64 B	32 B	16 B	8 B	4 B	2 B	1 B	
CPU	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	A-1	A-2
Level 1 Cache	T1	T0	I5	I4	I3	I2	I1	I0	O4	O3	O2	O1	O0	X	X	X	X
Level 2 Cache	I5	I4	I3	I2	I1	I0	O1	O0	C1	C0	X	X	X	X	X	R1	R0
SD Card	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	A-1	A-2

At the time of writing, GTMR has been applied to all registers in the memory hierarchy except for those directly on the data path and inside of the DDR3 IP core, providing near full protection against SEFIs inside the FPGA. Due to the size of the buses, voting each way between each module on the data path requires excessive resources. For ease of initial implementation, large buses were used; however, if the memory access protocol is reworked, a minimum bus size of 256 bits can be used in place of all buses greater than that length as the DDR3 interface process 256 bits per each 50-MHz clock cycle. Regardless, the data is protected through BTMR of the entire memory architecture and additional error correcting code automatically applied while stored on the SD cards.

a. Level 1 Cache

Implementation of the L1 cache was initially based off of the simple 4-state direct-mapped cache finite-state machine (FSM) shown in Figure 14 and composed of the following modules: Valid RAM, Tag Ram, Dirty Ram, Data Ram, Hit Detector, and L1 Cache Controller. This initial design, however, requires two clock cycles to return valid data on a cache hit due to transitioning back and forth between the idle and compare tag states. We found that by utilizing distributed pipeline RAM, implemented using lookup tables, the compare was accomplished and data returned in the same cycle due to reads of pipeline distributed RAM having a zero clock latency. This allowed us to combine the Idle and Compare states into a single Request state and allowed the L1 cache to run at the same clock frequency as the processor while providing valid data to the processor every clock cycle on consecutive cache hits using only a single line to indicate whether or not the requested data was ready.

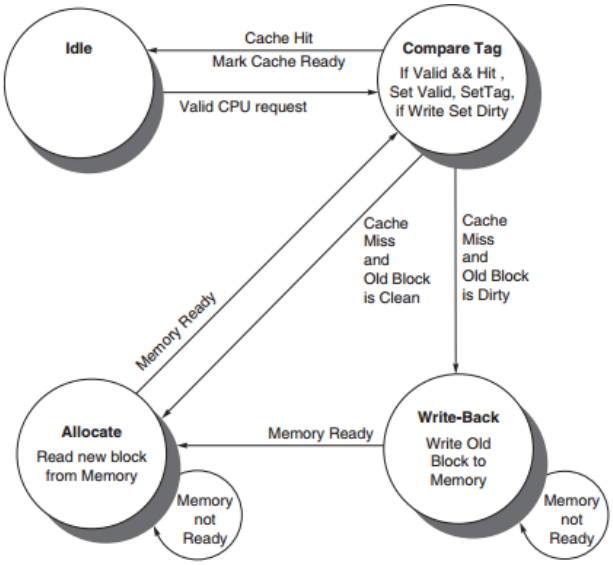


Figure 14. Finite-State Machine of a Simple Four State Direct-mapped Cache

Source : [19] D. A. Patterson and J. L. Hennessy, "Large and fast: Exploiting memory hierarchy," in *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Burlington, MA: Morgan Kaufmann, 2014, pp. 372–398

Transactions in the L1 cache on the CFTP follow the standard cache algorithm. The processor initiates a transaction with the L1 cache by providing an address, a control

signal to either read or write, and, on a write, the data to be written. Address bits twelve through seven are used to index into all four of the L1 cache RAMs. The retrieved tag from the tag RAM is compared to address bits thirty through thirteen, and if they match and the indexed entry in the valid RAM return a valid entry, a hit in the L1 cache occurred; otherwise, a miss occurred. In either case, the signal indicating a hit or miss is routed back to the processor to determine whether the processor stalls. The thirty-two bits of data, located at the offset as determined by address bits six down to two, in the block of data at the indexed entry of the data RAM, are also returned. Reads on a cache hit require no more actions. Writes, however, require the enable signals for the data and dirty RAM to transition high for one clock cycle. This causes the data RAM to store the value passed from the processor at the appropriate indexed and offset locations and the dirty RAM to record that the indexed location has been modified. On cache misses, the requested data needs to be retrieved from the L2 cache. If the indexed memory location that the requested data is supposed to populate is valid and dirty, the data block currently occupying that location needs to be written to the L2 cache. If the indexed memory location is not valid or dirty or following a write-back of a data block, a request is sent to the L2 cache to retrieve a block of data containing the requested address. Due to uncertainty in the amount of time required to access the L2 cache and because the L2 cache controller operates on a different clock domain, we implemented a four-way handshake between the L1 and L2 cache controllers. This necessitated the addition of acknowledgement states following the Allocate and Write-Back states. Upon acknowledgement from the L2 cache that the requested block of data has been fetched, the data block is filled into the data RAM of the requesting L1 cache and the valid, dirty, and tag rams are updated accordingly. We then transition back to the Request state, which generates a hit.

Finally, to maintain the capability to reprogram the main memory utilizing a direct connection from the UART interface to memory, we needed to add the ability to flush the entire contents of the L1 cache. When the UART interface finishes copying data to the L1 cache during reprogramming, all or some of the data will not have been written to the SD cards. To ensure that this happens, the UART interface sends a flush signal

causing the L1 cache to transition to the flush state. Because all data written to the L1 cache by the UART is dirty, we search the entire Dirty Ram for dirty entries and write those to the L2 cache. Once all L1 memory locations have been searched and all dirty data written to the L2 cache, a flush signal is sent to the L2 cache. When the L2 cache acknowledges that it has completed the flush, the L1 cache acknowledges to the UART that all data has been flushed to the SD cards and proceeds to initialize the L1 cache memories. The final L1 cache state machine with all conditions for state transitions is shown in Figure 15.

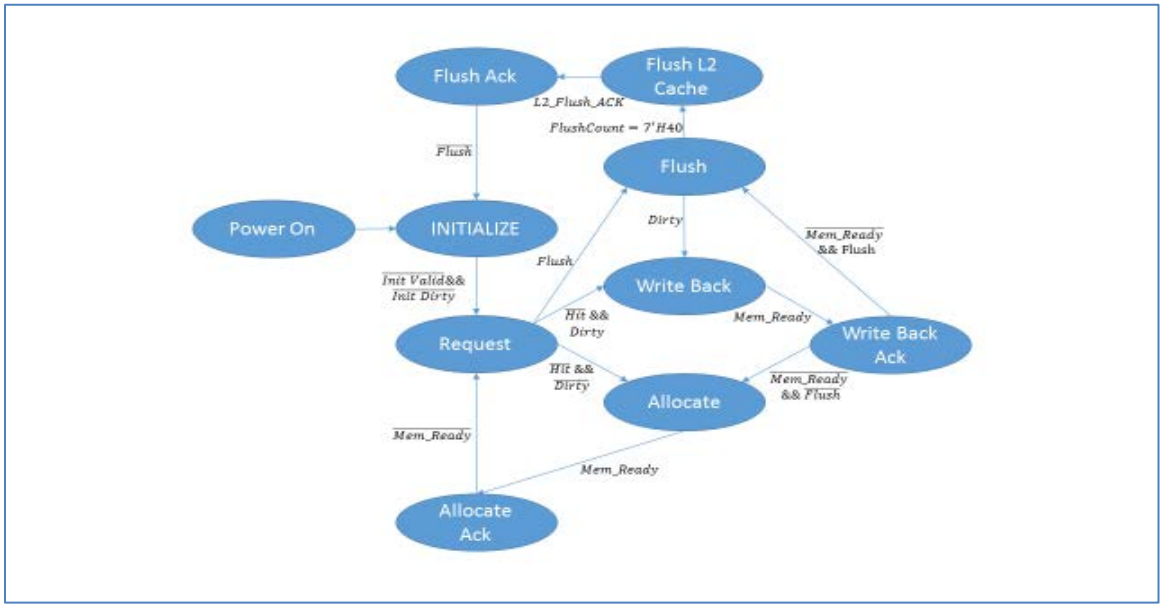


Figure 15. Flow Diagram of the CFTP L1 Cache Controller FSM

b. Bus Arbiter

The bus arbiter is a simple module. It utilizes a lock to provide access to the L2 Cache. When either the L1 Data Cache or L1 Instruction cache sends a request, the lock is checked to see if it is already claimed. If it is not claimed, the requesting cache claims it, and their input and output lines are directly connected to the L2 cache. Upon acknowledgment of a completed transaction between an L1 cache and the L2 cache, the lock is released. In the case of simultaneous requests, the data cache takes precedence because data requests happen further down the processor pipeline than instructions.

c. Level 2 Cache Controller

The overall behavior of the L2 cache is very similar to the L1 cache; however, it is structurally very different. The valid, tag, and dirty RAMs are all still located on the FPGA; however, the data RAM exists external to the FPGA on three 512-MB external DDR3 memories. A DDR3 Interface module was created to communicate between the external DDR3 memories and L2 cache controller. The difference in the structure and the process of interfacing with the DDR3 memories necessitated additional states to be added to the L2 cache controller FSM.

The single Request state used in the L1 cache was expanded to four states in the L2 cache: Idle, Hit Detection, DDR3 Request, and Request Acknowledge. This was due in part to the large sizes of the DDR3 memories and small block sizes selected for the L2 cache, requiring 20 index bits to completely address the DDR3 memory as shown in Table 3. BRAM was used to hold the tag, valid, and dirty bits, since distributed RAM resources were limited and had previously been used to implement the L1 cache. Unfortunately, BRAM has a minimal read clock latency of one clock cycle and forced us to add an additional wait state to determine if a hit or miss in the L2 cache occurred. If a cache hit was determined, the DDR3 had to be accessed to either write in new data or return the requested data. Because access times for the DDR3 memory are variable depending on the type of transaction requested and lost cycles due to memory refreshing, a four-way handshake was utilized between the L2 cache controller and the DDR3 interface. When the DDR3 interface signals that the requested transaction is complete, the L2 cache sends an acknowledgement to the L1 cache.

Cache misses in the L2 behave similarly to those of the L1 cache. In the event of a cache miss and the requested indexed memory slot being dirty, the data currently stored in that location must first be read back from the DDR3 memory and then written to the SD card. If a cache miss occurred and the requested indexed slot was not dirty or following a write-back, the requested data is retrieved from the SD card and written to the requested memory index in DDR3 memory. Similar to how the Request state had been expanded for DDR3 accessed, additional states had to be added for write-back and allocate states.

Finally, the flush states behave identically between the L1 and L2 caches with two small exceptions. First, the L2 cache has more memory to search. Second, once the L2 has completed a flush of its memories, it does not need to initiate any additional flush signals as all data now resides in non-volatile memory on the SD cards. After completion of a flush of its data, the L2 cache immediately acknowledges the flush request form the L1 cache. The full FSM for the L2 cache is shown in Figure 16.

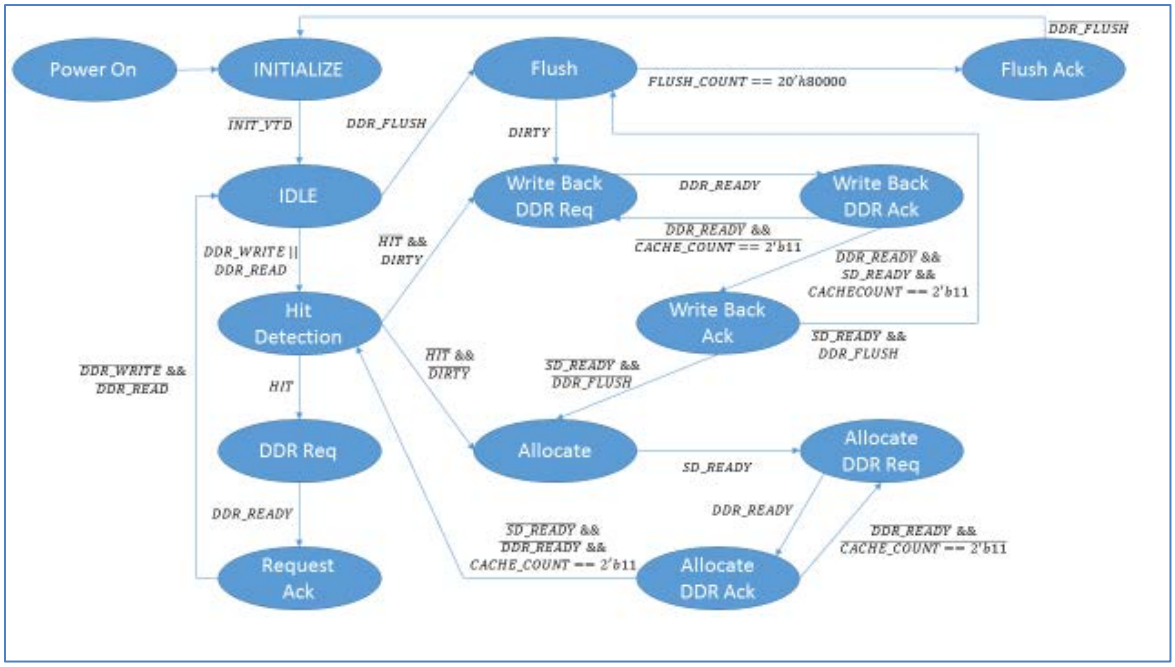


Figure 16. Flow Diagram of the CFTP L2 Cache Controller FSM

d. SD Card Interface

A SD card is used as our non-volatile memory storage in the CFTP and is at the tail end of our memory hierarchy. The interface uses the SPI protocol vice the SD protocol as it is simpler to implement. The SD interface is composed of the following modules: SD controller, SD incoming shift register, and SD outgoing FIFO, of which the last two are very simple. The SD incoming shift register maintains a copy of the most recent eight bits received. The outgoing FIFO takes in 8-bit words and when enabled issues a single bit at a time on the negative edge of the SD clock. Like the caches, most of

the complexity is in the controller, which provides four major functions: initialization, reads, writes, and alignment.

Each system has SD card controllers for each of the three SD cards for a total of nine SD card controllers. The outputs and registers of each separate controller in each system are voted on against the corresponding controller in the other two systems to maintain our GTMR architecture. The SD card controllers were implemented in this way to allow each physical SD card to be able to operate independently of the others. This is important because a SEFI can occur in any individual SD card and cause its read or write transaction to not process. When this happens, the individual cards need to be able to be reset and re-initialized. To compensate for individual cards being able to act independently, checkpoints have to be added into the state machine to align the controllers before a read or write operation is acknowledged.

The complete FSM for the CFTP is shown in Figure 17. After power on, the initialization sequence begins. Seventy-four clocks are passed to the SD card with the chip select line held high. Following that CMD0 is sent to force the SD card into SPI mode. If a valid response is received, CMD55 followed by ACMD41 is repeatedly sent until the SD card completes its initialization and transitions to its idle state.

Only two commands are implemented in the CFTP that can be transitioned to from the idle state: a single block read and a single block write. In a single block read, the L2 cache raises its read flag and provides a block address to the SD card. CMD17 is sent from the SD controller to the SD card with the address of the block to be read. The SD card then sends several responses. The first is a response that a valid read command was received. After a long access time, a start token is sent immediately followed by the data of the block requested and a 16-bit CRC. The data is registered in each SD card controller and an alignment signal is raised. Once all controllers are aligned, the read request is acknowledged to the L2 cache. In a single block write, the L2 cache raises its write flag and provides a block address and corresponding data to the SD card. CMD24 is sent from the SD controller to the SD card with the address of the block to be written. The SD card responds that a valid write request was sent. After delaying for eight clock cycles, the SD

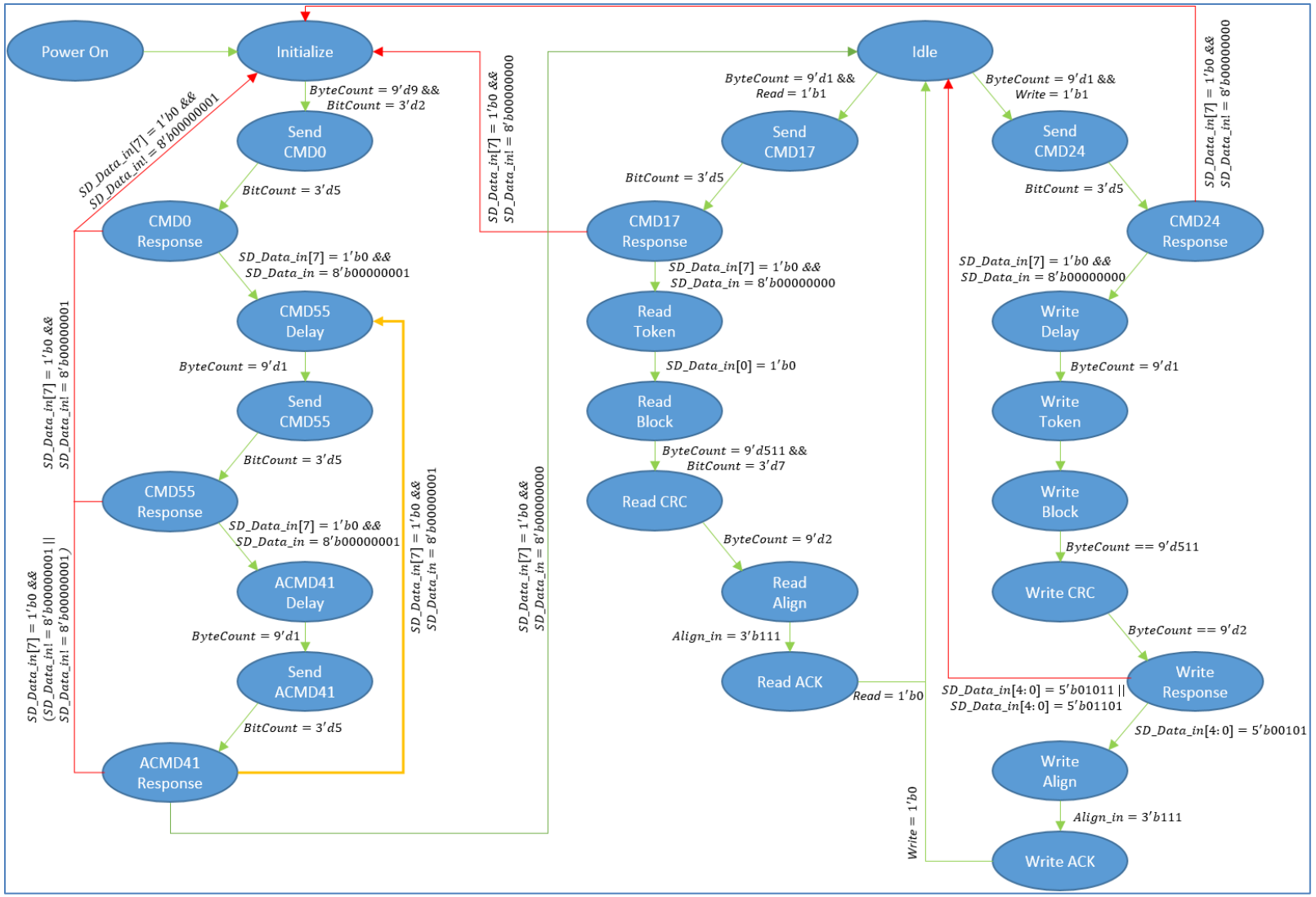


Figure 17. Flow Diagram of the CFTP SD Card Controller FSM

controller sends a start token, the data to be stored, and a 16-bit CRC. If the data is received correctly, the SD card issues a valid response and the SD controller raises its align flag. Once all controllers are aligned, the write request is acknowledged to the L2 cache.

4. NPSAT-1 PC-104 Interface

The primary purpose of the PC-104 interface module is to handle the actual transmission and reception of data between the C&DH processor and the experiments running on CFTP as well as to perform synchronization between clock domains. Communication between the C&DH processor and CFTP occurs on a bus architecture running through the PC-104 connectors of the C&DH stack. The bus architecture is a master/slave configuration with an 11-bit address bus, a 16-bit bi-directional data bus, and five control lines: bus clock (BCLK), output enable (OE), write enable (WE), chip select (CS), and interrupt request (IRQ). The C&DH processor acts as master and reads from and writes to the CFTP PC-104 interface module. CFTP is assigned an address range of 340h to 34Fh; however, only the even addresses are used since the ARM uses byte addressing and all transactions are 16-bit. Bus transactions are synchronous with the 51 MHz BCLK; however, due to other modules on the satellite utilizing CS3, read and write transaction take significantly longer with each containing 16 wait states. The PC-104 module is able to provide this communication link and synchronization by utilizing two FSMs interacting with two FIFOs.

The complete FSMs for the PC-104 interface are depicted in Figure 18. After power on, both the FSMs transition to their idle state. Four types of transactions can be processed. The FSM on the left hand side handles transactions between the PC-104 interface and the incoming and outgoing FIFOs, while the FSM on the right handles transactions between the CFTP processors and the incoming and outgoing FIFOs.

The BCLK drives transactions between the C&DH processor and the incoming and outgoing FIFOs. When an address is received that is in the CFTP's addressable range and CS3 goes low, the CFTP flight board is being accessed. The OE control line drops low when the CFTP is being read from, and the WE control line drops low when the

CFTP is being written to. In the case of a write, we first transition to the Read C&DH state where the write enable line of the incoming FIFO is held high for one clock cycle to capture the data on the PC-104 data bus. To prevent multiple captures, we transition to the Read Wait C&DH state on the next clock cycle, returning the write enable line of the incoming FIFO to a low level. Once any of the control signals transitions high or the address bus transitions out of range, we return to the idle state. Read transactions occur in a similar fashion except that we first transition to a Hi-Z C&DH state. In this state, we transition to polarity of the data bus to be able to write to it and send a signal to the bus transceivers the change their direction as well. The modified direction of the data bus persists throughout the read transaction. In addition, instead of accessing the write enable line of the incoming FIFO, we access the read enable line of the outgoing FIFO to retrieve 16-bits of data to drive on the data bus.

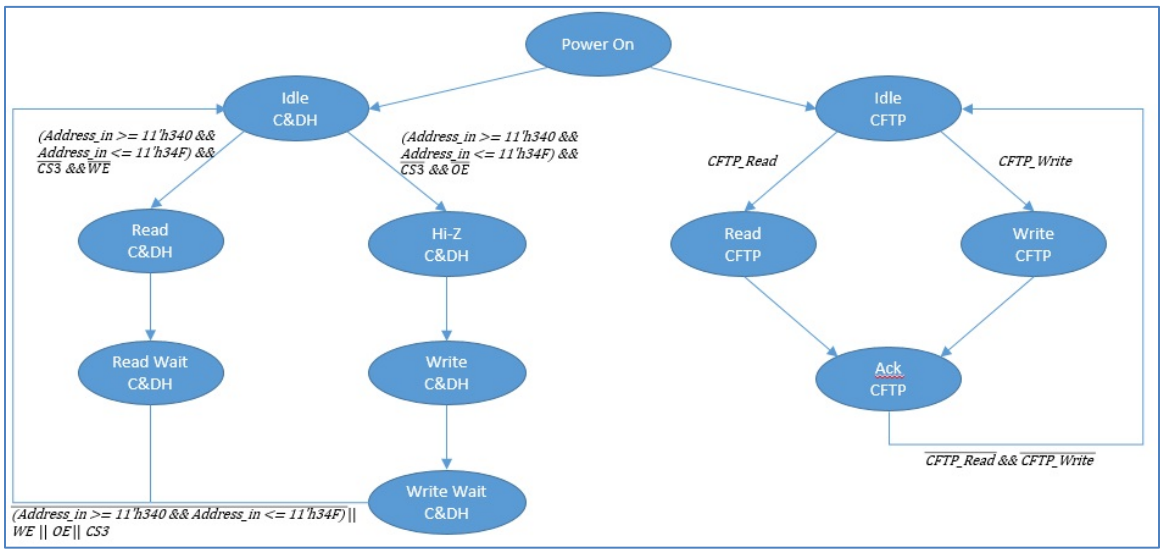


Figure 18. Flow Diagram of the CFTP PC-104 Interface FSM

The BCLK drives transactions between the C&DH processor and the incoming and outgoing FIFOs. When an address is received that is in the CFTP's addressable range and CS3 goes low, the CFTP flight board is being accessed. The OE control line drops low when the CFTP is being read from and the WE control line drops low when the CFTP is being written to. In the case of a write, we first transition to the Read C&DH

state where the write enable line of the incoming FIFO is held high for one clock cycle to capture the data on the PC-104 data bus. To prevent multiple captures, we transition to the Read Wait C&DH state on the next clock cycle, returning the write enable line of the incoming FIFO to a low level. Once any of the control signals transitions high or the address bus transitions out of range, we return to the idle state. Read transactions occur in a similar fashion except that we first transition to a Hi-Z C&DH state. In this state, we transition to polarity of the data bus to be able to write to it and send a signal to the bus transceivers the change their direction as well. The modified direction of the data bus persists throughout the read transaction. In addition, instead of accessing the write enable line of the incoming FIFO, we access the read enable line of the outgoing FIFO to retrieve 16 bits of data to drive on the data bus.

The FPGA system clocks drive transactions between the CFTP processors and the incoming and outgoing FIFOs. A standard four-way handshake is used to move data from the processors to the outgoing FIFO and to the processors from the incoming FIFO. From the idle state, either a read or write control signal is raised from the CFTP processors, and we transition to the read or write state, respectively. In the read state, 32 bits of data are clocked out from the incoming FIFO and placed on the processor data lines, while in the write state, the 32 bits of data on the processors data lines are clocked into the outgoing FIFO. On the following clock cycle from either the read or write state, we transition to the acknowledge state and signal to the processors that the transaction is complete. When the processor lowers its read or write signal, for whichever operation it had initially initiated, the FSM returns to the idle state.

The FIFO are constructed to allow for cross-clock domain synchronization. Both the read and write ports are clock independent. The synchronization is achieved by passing through an eight-stage synchronization circuit at the input to the FIFO. While this does cause a delay for the outputs to be available at the end of the FIFO, the delay is negligible when compared to the 16 wait states required for every bus transaction. The FIFO also act as buffers. By utilizing the empty flags on each of the FIFOs, we are able to produce an interrupt-based system, freeing processor clock cycles since neither

processor must constantly poll the interface and waste clock cycles when no data is available for transfer.

5. Configuration Memory Scrubbing

The final module implemented into the CFTP is the Xilinx Soft Error Mitigation IP solution. Operating in repair mode, this module is able to perform a complete scrub of the entire configuration memory every 23.5 ms and is able to correct single bit errors in each of the 28,320; 3,232-bit frames for an additional 630 us per error corrected [20]. Since the SEM is built in the user design space, it is susceptible to SEU itself and, therefore, has been implemented using BTMR since the internal registers are not available using this proprietary softcore. While a SEM is instantiated within each of the three systems, the SEM itself is a standalone module and does not interact with the rest of the user design.

The SEM and TMR schemes of CFTP work together to provide a reliable system. SEEs in the configuration memory can fundamentally change the user design and cause errors in individual systems; however, the systems in the user design work together to mask all errors, including those induced by configuration errors, through the use of a TMR structure. Over time, if the buildup of errors in the configuration memory is not corrected, the design will fail; however, the SEM solves this issue by preventing the buildup of errors by scrubbing the configuration memory.

B. HARDWARE DEVELOPMENT

The development time allotted to the hardware for this iteration of the CFTP was just over one month. This was in part due to a late start while it was determined how new hardware would be funded and in part to meet integration deadlines set by NPS's Space System Academic Group (SSAG). The author also had to learn how to properly use Altium Designer 15 for schematic entry and printed circuit board design.

The primary focus during the overhaul of the CFTP hardware was to simplify the design while also improving upon the capabilities of the original flight board with regard to SEE mitigation. A secondary objective was to provide a platform to observe the

radiation effects on several types of advanced memories. A full schematic of the CFTP flight board is available in Appendix C.

1. Part Selection

As a result of the shortened development cycle, in-depth research into the radiation effects on each specific components used on the flight board was not conducted. Our primary focus during part selection of the hardware was total dose and single event latch-up (SEL) effects that could be experienced within the FPGA. For the memories, it was assumed that should a failure occur, the design could potentially be modified to bypass the affected chips. No further research was conducted into the remainder of the components other than for providing a specific functionality to the circuit and temperature grading.

A prefabricated module, the Mercury KX-1, featuring a Kintex-7 325T FPGA, was selected to house the CFTP softcore. This particular model of the Kintex-7 series of FPGAs has undergone extensive radiation testing for use as both space flight hardware [12] and inside of the ATLAS Liquid Argon Calorimeter in the Large Hadron Collider [21]. Both note, to be effective, a minimum of a DTMR architecture and memory scrubbing should be applied to the softcore. SEL testing, conducted by former, confirmed a latch-up condition on the VCCAUX power rail existed; however, this latch-up condition is not well understood and has not been shown to cause damage to the FPGA itself [22]. TID testing, conducted by the latter, exposed the FPGA to protons with energies of 180 MeV and resulted in one failed FPGA at 340 krad and one still functional having received a dose of 446 krad. The use of the Mercury KX-1 module greatly simplified the design process as it included onboard DDR3 memories, a flash memory for configuration, and power conditioning integrated circuits (IC). Finally, all components on the daughter board were industrial grade components, allowing for the greater temperature variance the board will be exposed to in space.

A non-volatile memory device was still needed to store software programs to be executed by the CFTP. To maintain the TMR architecture on the flight board, three industrial grade Delkin SD cards were integrated into the hardware. These particular SD

cards were selected based on their TID that is in excess of 24 krad regardless of the type of biasing [23].

2. Printed Circuit Board Design

This iteration of the CFTP printed circuit board is a much simpler design than the previous version. With the exception of the board dimensions, layering, and positioning of the PC-104 connector, which had to be maintained to ensure the flight board fit properly into the C&DH stack, the board was completely redesigned. The design was initially entered in schematically into Altium Designer 15 and the PCB populated. All copper traces were hand drawn in Altium. Finally, a design check was run to ensure all manufacturer specifications were met before sending the design the Advanced Circuits to be fabricated.

a. Interconnect to PC-104 Bus

The interconnect between the PC-104 connectors and the Mercury KX-1 was straightforward. Since pin mappings on the FPGA can be constrained to any signal in the softcore design, pins for specific signals were chosen such that the complexity of routing on the printed circuit board was minimized. With the exception of power enable signal, power and ground pins, all signals between the PC-104 connectors and the FPGA are routed through octal bus transceivers. This is done to provide protection to the FPGA when powered down since all pins on the octal transceiver are tri-stated, preventing the flow of electricity into the IO lines of the FPGA when VCCINT is not powered. The octal transceivers, controlled by the FPGA, also provides an additional layer of protection to the FPGA by preventing the data lines to the FPGA from being able to be driven from both the C&DH processor and the CFTP at the same time.

b. Interconnect to SD Cards

Connection of the SD Cards was based on the design implemented on Digilent's Genesys 2 Development Board [24], the primary difference being that the pull-up resistor values were swapped to the manufacturer recommended values for all of the signal lines of the industrial grade Delkin SD Cards. This design was used because it allows for the

SD Cards to be powered down without the need for the entire system to be powered down. This allows for SEL and SEFI type errors, if detected and determined to originate from the SD cards, to be corrected without having to power down the entire system.

c. Configuration Memory

Configurations for the CFTP are currently stored in only two places. One copy is stored in flash memory on the KX-1 Mercury module, and a second copy is stored by the C&DH. When the Mercury KX-1 module is powered on, it attempts to retrieve the FPGA configuration from the onboard flash. This occurs because the boot mode pin of the Mercury KX-1 module is grounded. Should the configuration in flash memory become corrupted, the configuration can be loaded directly from the C&DH through the JTAG connections between the C&DH and Mercury KX-1 module.

d. Power Management

Power management was a key concern in the development of the hardware to alleviate certain SEE. SEL and SEFI effects can be corrected through power cycling. In fact, SEL can only be corrected through power cycling, while SEFI can also be correcting through internal reset circuitry. The primary interface to control power to the CFTP flight board is through the power enable pin on the Mercury KX-1 module. This pin is connected through an inverting buffer to the CFTP PWR pin on the PC-104 bus. When CFTP PWR is high, all voltage regulators on the Mercury KX-1, with the exception of the 3.3-V regulators, are disabled. The load power switches on the back of the flight board, which allow the 3.3 V generated by the Mercury KX-1 to power each of the p-channel metal–oxide–semiconductor field-effect transistors feeding the SD Cards and provides VCCIO power to the FPGA, are also disabled. This effectively powers down the Mercury KX-1 module and SD cards on the flight board. When CFTP PWR is low, this enables both the voltage regulators on the Mercury KX-1 module and the load power switches on the flight board. Regardless of the state of CFTP PWR, 5 V is provided to the board via three PC-104 connections. This 5-V power supply is directly connected to the Mercury KX-1 module as well another 3.3-V voltage regulator located on the flight board. The 3.3-V voltage regulator on the flight board provides constant power to the

octal transceivers so that they can be placed into tri-state when the mercury KX-1 module is powered down.

C. CHAPTER SUMMARY

In this chapter, both the softcore and hardware implementations were discussed. Of primary importance in the softcore design was the methodology by which GTMR was applied, the clock distribution, and implementation of the SEM module. These three aspects of the softcore provide the CFTP reliability. Though the hardware design cycle was limited to only one month, we were able to simplify the design and include some important features to combat SEE, namely the ability to cycle power on the CFTP flight board. In the next chapter, we discuss the performance of the softcore and hardware of the flight board.

IV. TESTING AND EVALUATION

A. OUT-OF-SYSTEM TESTING

A majority of testing occurred out of system and was done to verify functionality of the GTMR structure and memory interfaces. Exhaustive testing of all XUM softcore instructions was not conducted as it was assumed to function properly since no major changes were applied to it. Instead, the primary test program used was a slightly modified version of the UART demo program provided in the original source code for the XUM softcore that would output a character every half second and echo back any characters received. To minimize hardware differences between the flight board and development board, the Genesys 2 development board was used. This board features a KINTEX-7 FPGA of the same size and speed grade as that of the flight hardware, and so its performance should be similar. It also contains Flash, DDR3 and SD Card memories on board so we could verify those interfaces. Because a majority of the testing occurred on the Genesys development board and not flight hardware, the communications interface primarily used was XUM's UART module directly connected to a personal computer (PC) vice the NPSAT PC-104 interface. Internal logic analyzers, which communicate via the Joint Test Action Group (JTAG) signal lines between the PC and FPGA, were used extensively to both debug and verify the design.

1. Fault Injection Testing

Fault testing was performed by manipulation of the system clock lines. By forcing one of the clocks to run at a different speed from the other two, we observed the effect that a SET on an individual system clock would have on the softcore. Two results can possibly occur from a SET on a system clock: the affected system progresses to its next state earlier than it should or parts of the affected system can slip into a metastable state and progress to the next state or miss several clock cycles. Testing of SEUs on individual registers is not necessary as a system-wide effect essentially causes an SEU in every register of the affected system.

The case where an SET causes the affected system to progress to the next state is shown in Figure 19. This is simulated by running one of the system clocks at twice the frequency of the other two system clocks. As can be seen at sample-time two, all three systems are synchronized. There is a slight propagation delay in the incoming instruction seen in systems zero and one, however, by sample-time three those outputs become stable. At sample-time four, system clock zero experiences its SET. We see that system zero has transitioned to its next state based on its vote for the program counter (PC). While systems one and two maintain that the PC should be 0000000Ch, system zero's vote has changed to 0000020Ch. Though system zero's vote changed, the majority vote of all PCs, which drive follow-on combinational logic in all three systems, remains 0000000Ch. At sample-time six, we see all the systems resynchronize and agree that the PC should be 0000020Ch; system zero having been fully restored.

The case where an SET causes the affected system's PC to go into a simulated *metastable state* and not progress is shown in Figure 20. This is accomplished by running one of the system clocks at one quarter the speed of the other two system clocks. It can be seen at the rising edge of system clocks zero and one that the votes for the PC are updated, and since they are the same, the true vote being sent to all follow on logic in all three systems is the correct value. Due to the simulated *metastable state* of system two's PC, its vote cannot be relied upon, and in this example is frozen at the value where it entered this simulated *metastable state*. This does not necessarily have to be the case, and in a true metastability, the register could capture new values sporadically. Regardless, at sample-time 268, system two's PC registers can be seen to have recovered, and it can be seen that system two is once again synchronized with the other two systems.

Fault injection into configuration memory was not tested. The primary method for verifying the fault tolerance in the configuration memory would have been to subject the device to radiation testing. Once again, due to the extremely limited development time, we were not able to conduct radiation testing, and this needs to be verified on orbit. Since the SEM module does produce error signals, we will be able to differentiate errors resulting from configuration memory upsets form those originating in user memory.

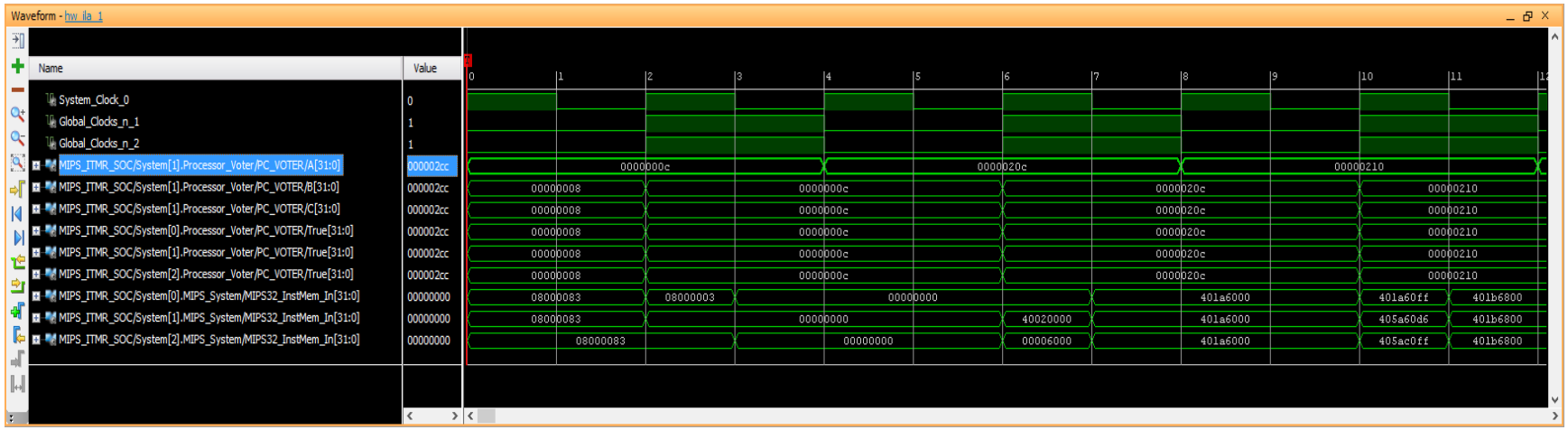


Figure 19. Waveform Capture Demonstrating GTMR Correction of a Fault Induced by a Fast Clock

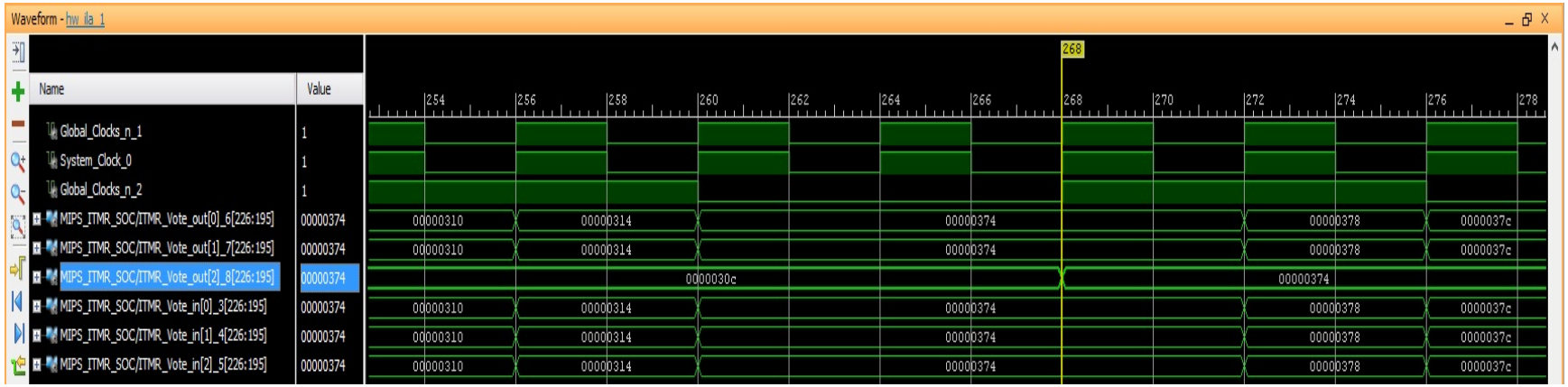


Figure 20. Waveform Capture Demonstrating GTMR Correction of a Fault Induced by a Slow Clock

2. Performance Testing

Periodically, throughout development, waveforms were captured to determine how quickly data are being passed between the various levels of memory in an effort to determine where the softcore could be optimized. Shown previously in Figures 19 and 20, we see that we were able to provide the XUM processors with new instructions every single clock cycle from the L1 cache, which is the optimal case. The penalty incurred for a L1 cache miss is depicted in Figure 21. By counting the number of samples Inst_Ready was low and dividing by four since the system clock runs at 50 MHz, we determined the number of CPU clock cycles the processor was stalled. Inst_Ready was low for samples 376 to 442, 76 total samples. This translates to a miss penalty of 19 processor clock cycles. During these 19 clock cycles, we transferred 1024 bits of data, or 32 instructions, to the L1 cache. Finally, the miss penalty for the L2 cache is show in Figure 22. Similar to the L1 cache, we determined the total miss penalty by observing the number of samples Inst_Ready is low, which in this case is exactly the miss penalty due to the decreased sample rate of 50 MHz. We see the total L2 cache miss time is approximately 26,500 clock cycles, a majority of which is the SD card either internally locating and preparing the data, CurrentState = 0Ch, or the actual shifting of the data block to the FPGA, CurrentState = 0Dh. The L2 miss penalty could be further reduced if the 4-bit SD protocol were used vice the 1-bit SPI protocol; however, this only reduces the portion of the penalty incurred during CurrentState = 0Dh and saves approximately 3,000 clock cycles. In the case where a write-back must be performed prior to allocation of new data, these penalties essentially double.

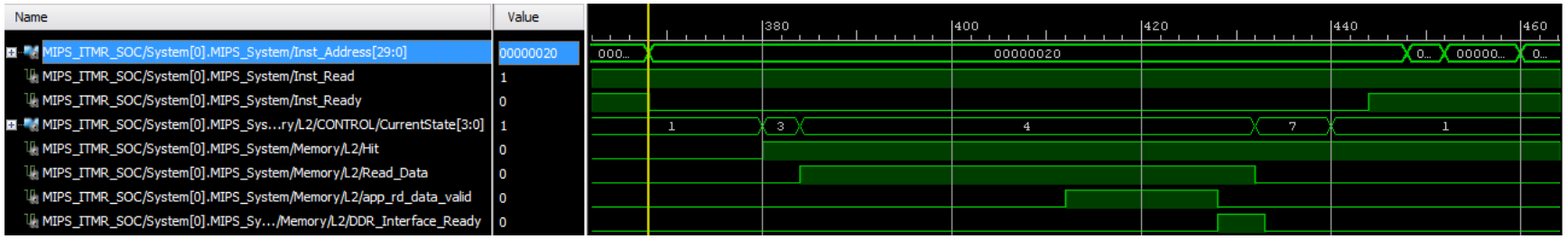


Figure 21. Waveform Capture of a L1 Cache Miss with a Sample Rate of 200MHz

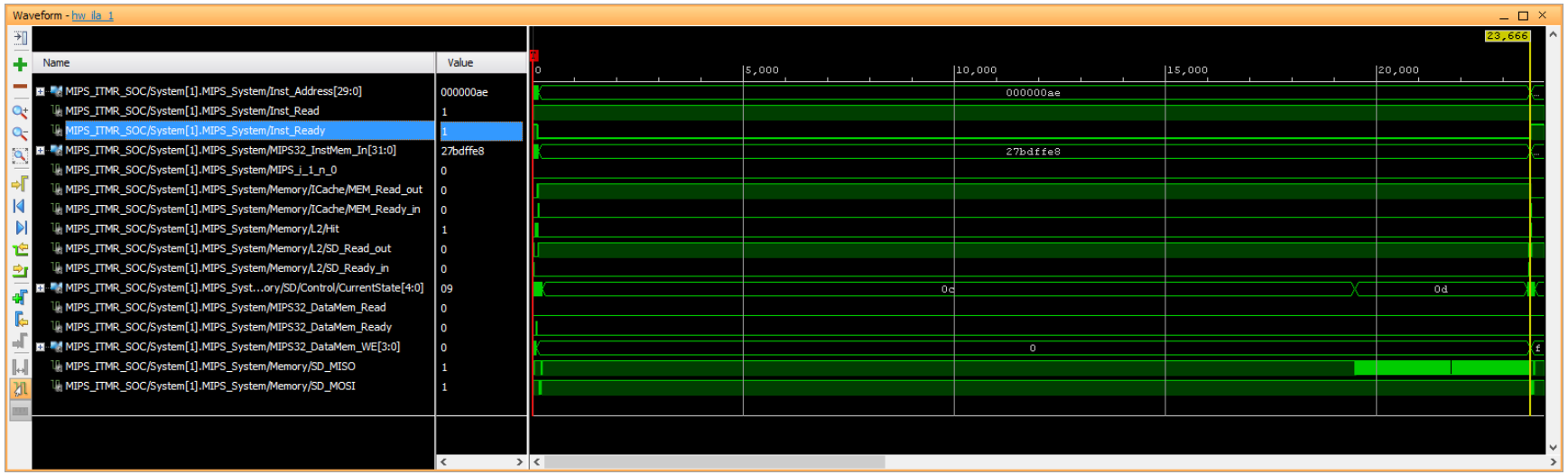


Figure 22. Waveform Capture of a L2 Cache Miss with a Sample Rate of 50MHz

B. IN-SYSTEM TESTING

The amount of time we had to perform testing with the flight board was limited due to the timing between production of the flight hardware and delivery to the SSAG for integration into the satellite bus. As such, testing was limited to verify functionality of the hardware itself and not of the softcore design. When another CFTP board becomes available, in-system testing of the softcore can commence.

1. Power Management Test

The power management test was used to verify that the ARM could turn power on and off to the CFTP flight board so that in the event a SEFI or SEL were detected those faults could be cleared. Prior to conducting the test, a simple program was loaded into the configuration flash of the CFTP flight board so that when power was applied all of the user defined light emitting diodes (LEDs) on the Mercury KX-1 board would light up. Power was then applied to the entire C&DH stack, and the ARM was loaded into monitor mode. After approximately 30 s, all of the user defined LEDs on the Mercury board illuminated. From the monitor mode command line, the command *poke32 FFFDD000 00000002* was issued. This command sets bit two of the C&DH processor's general purpose input output register E that is connected to CFTP's power enable pin through the PC-104 bus and a buffer. When the power enable pin to the CFTP flight board transitioned high, the power was disabled, and all of the user defined LEDs on the Mercury board were observed to extinguish. The command *poke32 FFFDD000 00000000* was issued to restore power. Approximately 30 s after issuing the command, the user defined LEDs on the Mercury board illuminated.

2. Communications Test

The PC-104 data, address, and control line connections were validated through a simple communications test. The CFTP flight board was loaded with a softcore that consisted of the PC-104 interface and an *echo* module that mimicked the XUM processors four-way handshake. When the incoming FIFO flagged that it had data, the *echo* module read the incoming FIFO and wrote the outgoing FIFO with what it had

retrieved. The ARM processor on the C&DH stack was loaded into monitor mode. Peek and Poke bus transactions were used to probe the status of the PC-104 interface as well as send and receive 32 bits of data. The results of two 16-bit write transactions is shown in Figure 23. After the second 16-bit transaction, we see the *echo* module read the incoming FIFO and then write back the data to the outgoing FIFO. In Figure 24, the two 16-bit read transactions are shown returning the two values previously sent in the same order they were received.

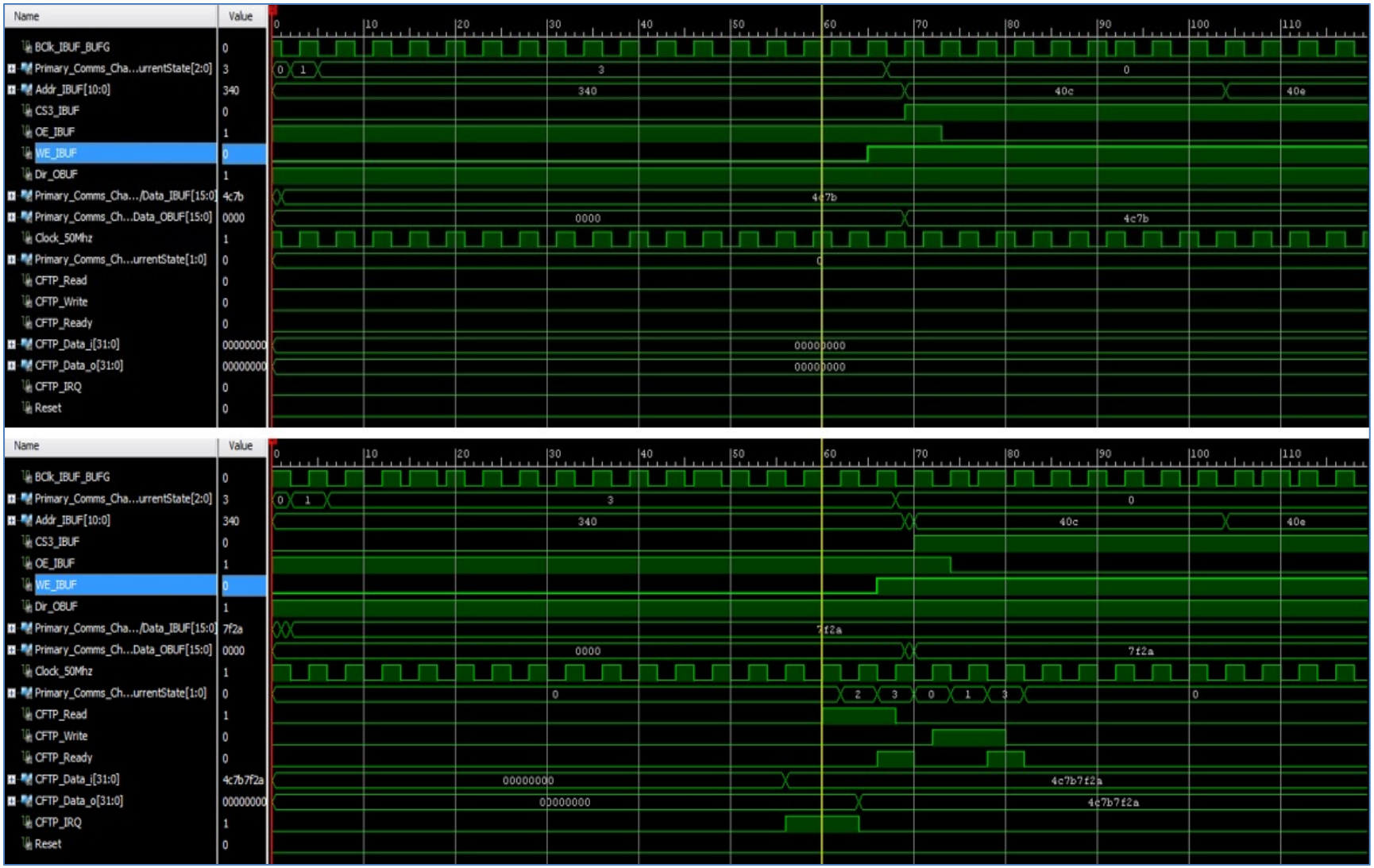


Figure 23. Two 16-bit Write Transactions to the CFTP over the PC-104 Bus

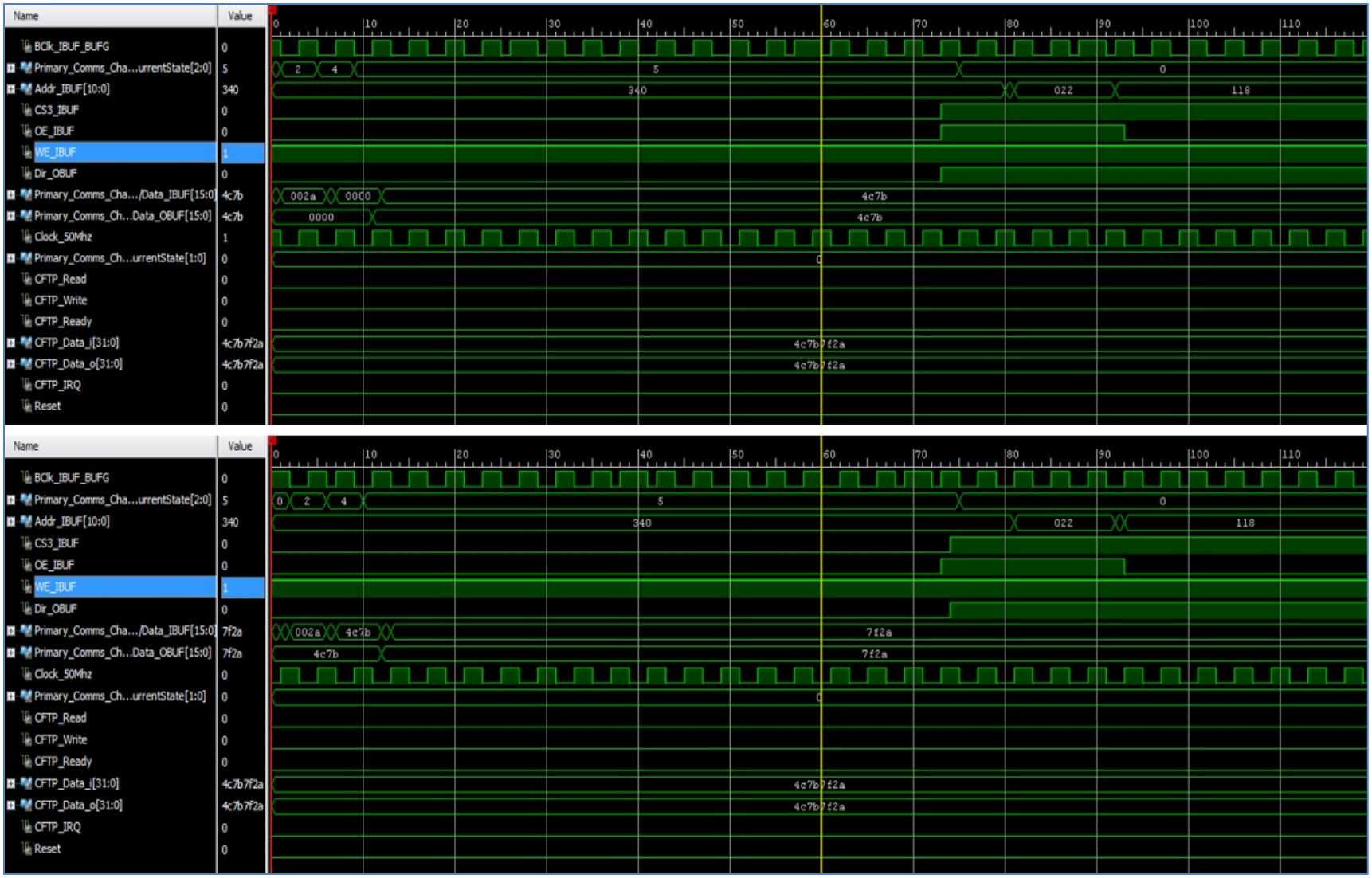


Figure 24. Two 16-bit Read Transactions from the CFTP over the PC-104 Bus

3. Programming Test

Prior to handover of the CFTP flight board to the SSAG, we needed to be able to program the FPGA utilizing the ARM processor in the C&DH to control the JTAG lines connected between them as it would be the only means of updating the softcore once integrated into the satellite. This was accomplished by generating a serial vector format (SVF) file of a bitstream to be loaded in into the FPGA. An excerpt of this file is shown in Figure 25. The portions of the file that were primarily of interest were the header and trailer sections, which initially prepare the FPGA to receive the bitstream and then start execute the startup sequence after the configuration is sent. Utilizing these portions of the SVF file and the finite-state machine the test access port (TAP) controller of the JTAG protocol shown in Figure 26, we built a C program to manually toggle the four JTAG lines to configure the FPGA. We verified the program executed correctly and were able to load multiple different configurations to the FPGA. The source for the JTAG program is located in appendix B.

```
STATE TEST-LOGIC-RESET;
//Loading device with 'idcode' instruction.
STATE RUN-TEST/IDLEIDLE;
STATE DR-SCAN;
STATE IR-SCAN;
STATE CAPTURE-IR;
STATE SHIFT-IR;
SIR 6 TDI (09) SMASK (3f) ;
STATE EXIT1-IR;
STATE UPDATE-IR;
STATE SELECT-DR-SCAN;
STATE CAPTURE-DR;
STATE SHIFT-DR;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f3651093) MASK (0fffffff) ;
```

Figure 25. Serial Vector Format File Used to Retrieve Kintex 325 Device ID

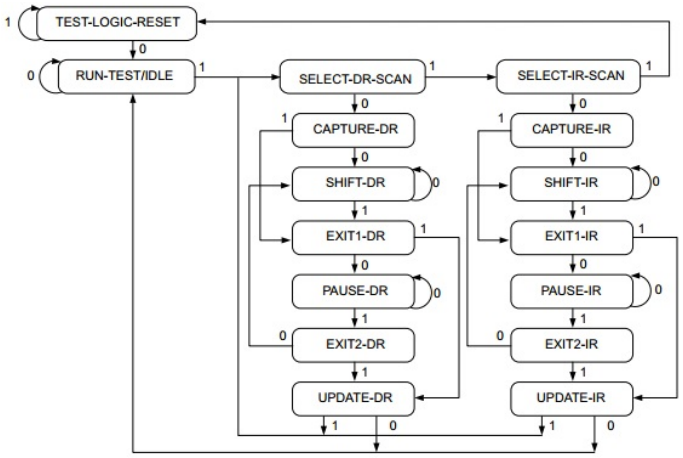


Figure 26. FSM of the TAP controller

Source: [25] Xilinx, Inc., 7 Series FPGAs Configuration User Guide v1.10, Jun. 2015.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

The work in this thesis constitutes a minimum baseline for what was desired to be implemented in the CFTP on NPSAT-1. We were able to meet our goals of complete overhaul of the original CFTP board following our primary objective that the system be reconfigurable, reliable, COTS based and within budget. Furthermore, we were able to apply the results of research that had occurred following the design of the original CFTP, providing configuration memory scrubbing and a more advanced internal triple-modular redundancy technique to the entire design. The design exceeded our initial goals through the development of a robust memory architecture; however, this implementation increased its utility but did not necessarily increase the reliability of the design.

While the initial goals were met, the CFTP is far from complete and can certainly be improved upon. Several areas of the softcore can be improved upon to further increase reliability of the system to include user data memory scrubbing, development of a flash memory controller, and re-coding the SEM and DDR3 IP modules to fully implements GTMR. The software has been largely overlooked in this project, and specific experiments to run on the processor and implementation of an actual operating system, such as Linux, could be investigated. Finally, a method for collecting metrics still needs to be implemented, and on-orbit analysis of the system should be conducted.

A. RECOMMENDATIONS FOR FUTURE WORK

Currently, the softcore provides little protection to the buildup of errors over time in memories of the user design. While data is compared and majority voted during the process of being read from the SD cards, DDR3 ICs, and the internal L1 cache, any data corrected is not then re-written to these memories. The only way to correct these memories is to force a cache fault, which is not ideal. A method for scrubbing user memories should be implemented.

The configuration capabilities of the CFTP should be greatly expanded upon. Currently, new configurations transmitted to the satellite must utilize the JTAG program to be written into the FPGA. A flash memory controller should be developed to allow the

FPGA to write to its own configuration memory. Furthermore, the Series 7 FPGAs supports a technology known as multi-boot. Multiple, different configurations could be stored in the flash memory and, upon booting into a default softcore design, another softcore design, located in the flash, could be selected and the system reloaded into it.

The DDR3 and SEM IP cores were used in this design for ease of use in initial implementation; however, since they are proprietary softcores, there is no way to access the internal registers. This opens the designs to SEU and, especially, SEFI errors that could occur within these modules. While, to some extent, the SEM is protected through BTMR, no TMR technique can be applied to the DDR3 IP core. This is because, in addition to not having access to the internal registers, we also do not have access to the tri-state signal that drives the tri-state buffers on the data lines between the CFTP and DDR3 ICs. Without the tri-state signal, we are unable to do a majority vote on the DDR3 data lines as it forces them to become uni-directional signals. Development of an in-house solution for these IP cores would alleviate these issues.

The only software used during the development of CFTP was that provided to test the XUM softcore. Software should be developed to fully test the entire CFTP system. Furthermore, software can be developed to run specific experiments on data collected organically on the satellite to further prove the viability of the system. Finally, for ease of use to the user, installation of a Linux-based operating system can be explored.

A final recommendation for future work is post-launch analysis of the CFTP system. During the course of this thesis research, it was found that there was a large gap of knowledge in the actual performance on orbit of the CFTP flown in 2006. Real data must be collected and analyzed to help direct future adjustments to the hardware, softcore, and software of this project. To properly conduct this analysis, a method for collating and transmitting all errors developed in the CFTP system must first be developed.

B. CLOSING REMARKS

In conclusion, this iteration of the CFTP successfully completed all design goals. The softcore proved to be both reliable and robust, able to survive a variety SEE. Flight

hardware was built and tested, passing all functional checks before being delivered and accepted by the SSAG for integration into NPSAT-1. Due to the re-configurability inherently a part of the CFTP through the use of an FPGA and its robust memory architecture, the design can be further improved upon up to launch and beyond to provide additional capabilities such a user side memory scrubbing and increasing the redundancy of configuration methods. In short, the CFTP will survive.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MODIFIED XUM SOURCE CODE

The complete source code for the CFTP is contained in this appendix. The source code for the softcore is in Section A, and the source code for the software running on the CFTP is in Section B.

The source code for the eXtensible Utah Multicore is licensed under the GNU Lesser General Public License version 3 (LGPLv3). The source code for the CFTP, being a modified and combined work, is also licensed under GNU LGPLv3. A copy of the GNU LGPLv3 follows.

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.
As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version.”

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.
You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.
If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.
3. Object Code Incorporating Material from Library Header Files.
The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the object code with a copy of the GNU GPL and this license document.
4. Combined Works.
You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
d) Do one of the following:
0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)
5. Combined Libraries.
You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
6. Revised Versions of the GNU Lesser General Public License.
The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

```
If the Library as you received it specifies that a proxy can decide whether future
versions of the GNU Lesser General Public License shall apply, that proxy's public
statement of acceptance of any version is permanent authorization for you to choose that
version for the Library.
```

A. CFTP SOFTCORE SOURCE CODE

1. Top Level

This subsection contains everything that exists on the top level of CFTP. This includes everything that could not be triplicated in addition to the CMTs.

(1) CFTP.v

```
`timescale 1ns / 1ps
`default_nettype none
/*
 * File         : CFTP.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   01-Mar-2016  ASJ       Release version.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   Top level module of the CFTP project, contains all modules which could not
 *   be triplicated, clock management tiles, and an instance of the CFTP TMR
 *   system.
 */

module CFTP(
   input wire clock_200MHz_p,
   input wire clock_200MHz_n,
   input wire reset_n,
   // I/O
   input wire UART_Rx,
   output wire UART_Tx,
   output wire Heartbeat_out,
   //Memory
   inout wire [31:0] ddr3_dq,
   inout wire [3:0] ddr3_dqs_p,
   inout wire [3:0] ddr3_dqs_n,
   output wire [14:0] ddr3_addr,
   output wire [2:0] ddr3_ba,
   output wire ddr3_ras_n,
   output wire ddr3_cas_n,
   output wire ddr3_we_n,
   output wire ddr3_reset_n,
   output wire [0:0] ddr3_ck_p,
   output wire [0:0] ddr3_ck_n,
   output wire [0:0] ddr3_cke,
   output wire [0:0] ddr3_cs_n,
   output wire [3:0] ddr3_dm,
   output wire [0:0] ddr3_odt,
   // SD Card Signals
   input wire SD_MISO,
   output wire SD_MOSI,
   output wire SD_CS,
```

```
output wire SD_SClk,
output wire SD_PWR
);

// Clock and Reset signals
wire Master_Clock_in, Master_Clock, DDR3_Clock;
wire [2:0] System_Clock, SD_Clock, SEM_Clock;
wire PLL_Locked, Select_SD_Clock_Speed;
reg reset;
wire RESET_in;

//DDR Signals
wire [255:0] app_rd_data;
wire app_rd_data_valid;
wire app_rdy;
wire app_wdf_rdy;
wire ui_clk;
wire ui_clk_sync_rst;
wire init_calib_complete;
wire [28:0] app_addr;
wire [2:0] app_cmd;
wire app_en;
wire [255:0] app_wdf_data;
wire app_wdf_end;
wire app_wdf_wren;

// SD Signals - Comment out for actual CFTP Board
wire [2:0] SD_MISO_in, SD_MOSI_out, SD_CS_out, SD_SClk_out;
assign SD_MISO_in[0] = SD_MISO;
assign SD_MISO_in[1] = SD_MISO;
assign SD_MISO_in[2] = SD_MISO;

//SEM Signals
wire CRCERROR;
wire ECCERROR;
wire ECCERRORSINGLE;
wire [25:0] FAR;
wire [4:0] SYNBIT;
wire[ 12:0] SYNDROME;
wire SYNDROMEVALID;
wire [6:0] SYNWORD;
wire [31:0] O;
wire CSIB;
wire [31:0] I;
wire RDWRB;

// Clock and Reset Generation

always @(posedge System_Clock[0]) begin
    reset = ~reset_n | ~PLL_Locked;
end

/*BUFG Global_Reset (
   .O(RESET_in),
   .I(reset)
);*/

IBUFGDS #(
   .DIFF_TERM(“FALSE”),
   .IBUF_LOW_PWR(“TRUE”),
   .IOSTANDARD(“DEFAULT”)
) IBUFGDS_inst (
   .O(Master_Clock_in),
   .I(clock_200MHz_p),
   .IB(clock_200MHz_n)
);

BUFG Global_Clock (
   .O(Master_Clock),
```

```
   .I(Master_Clock_in)
);

Master_Clock_Divider Global_Clocks(
    .clk_in1    (Master_Clock),
    .clk_out1   (System_Clock[0]),
    .clk_out2   (System_Clock[1]),
    .clk_out3   (System_Clock[2]),
    .clk_out4   (SD_Clock[0]),
    .clk_out5   (SD_Clock[1]),
    .clk_out6   (SD_Clock[2]),
    .clk_out7   (DDR3_Clock),
    .reset      (1'b0),
    .locked     (PLL_Locked));

SEM_CLK SEM_Clocks(
    .clk_in1    (Master_Clock),
    .clk_out1   (SEM_Clock[0]),
    .clk_out2   (SEM_Clock[1]),
    .clk_out3   (SEM_Clock[2]),
    .reset      (1'b0),
    .locked     ());

ITMR_MIPS_SOC MIPS_ITMR_SOC(
    .System_Clock_in        (System_Clock),
    .Reset_in               (/*RESET_in*/reset),
    //DDR U/I Signals
    .app_rd_data            (app_rd_data),
    .app_rd_data_valid      (app_rd_data_valid),
    .app_rdy                (app_rdy),
    .app_wdf_rdy            (app_wdf_rdy),
    .ui_clk                 (ui_clk),
    .ui_clk_sync_rst        (ui_clk_sync_rst),
    .init_calib_complete    (init_calib_complete),
    .app_addr               (app_addr),
    .app_cmd                (app_cmd),
    .app_en                 (app_en),
    .app_wdf_data           (app_wdf_data),
    .app_wdf_end            (app_wdf_end),
    .app_wdf_wren           (app_wdf_wren),
    //SD Signals
    .SD_Clock_in            (SD_Clock),
    .SD_MISO                (SD_MISO_in),
    .SD_MOSI                (SD_MOSI_out),
    .SD_CS                  (SD_CS_out),
    .SD_SClk                (SD_SClk_out),
    //UART Signals
    .UART_Rx_in             (UART_Rx),
    .UART_Tx_out            (UART_Tx),
    //SEM Singals
    .O_in                   (O),
    .SEM_Clock_in           (SEM_Clock),
    .CRCERROR_in            (CRCERROR),
    .ECCERROR_in            (ECCERROR),
    .ECCERRORSINGLE_in      (ECCERRORSINGLE),
    .SYNDROMEVALID_in       (SYNDROMEVALID),
    .SYNDROME_in            (SYNDROME),
    .FAR_in                 (FAR),
    .SYNBIT_in              (SYNBIT),
    .SYNWORD_in             (SYNWORD),
    .status_heartbeat       (Heartbeat_out),
    .icap_csib              (CSIB),
    .icap_rdwrb             (RDWRB),
    .icap_i                 (I)
);

assign SD_PWR = 1'b0;

mig_7series_0 DDR_Control(
```

```
    // Inouts
    .ddr3_dq        (ddr3_dq),
    .ddr3_dqs_n     (ddr3_dqs_n),
    .ddr3_dqs_p     (ddr3_dqs_p),
    // Outputs
    .ddr3_addr      (ddr3_addr),
    .ddr3_ba        (ddr3_ba),
    .ddr3_ras_n     (ddr3_ras_n),
    .ddr3_cas_n     (ddr3_cas_n),
    .ddr3_we_n      (ddr3_we_n),
    .ddr3_reset_n   (ddr3_reset_n),
    .ddr3_ck_p      (ddr3_ck_p),
    .ddr3_ck_n      (ddr3_ck_n),
    .ddr3_cke       (ddr3_cke),
    .ddr3_cs_n      (ddr3_cs_n),
    .ddr3_dm        (ddr3_dm),
    .ddr3_odt       (ddr3_odt),
    // Inputs
    // Single-ended system clock
    .sys_clk_i              (DDR3_Clock),
    // user interface signals
    .app_addr               (app_addr),
    .app_cmd                (app_cmd),
    .app_en                 (app_en),
    .app_wdf_data           (app_wdf_data),
    .app_wdf_end            (app_wdf_end),
    .app_wdf_mask           (32'h00000000), //Data is never masked.
    .app_wdf_wren           (app_wdf_wren),
    .app_rd_data            (app_rd_data),
    .app_rd_data_end        (),
    .app_rd_data_valid      (app_rd_data_valid),
    .app_rdy                (app_rdy),
    .app_wdf_rdy            (app_wdf_rdy),
    .app_sr_req             (1'b0),
    .app_ref_req            (1'b0),
    .app_zq_req             (1'b0),
    .app_sr_active          (),
    .app_ref_ack            (),
    .app_zq_ack             (),
    .ui_clk                 (ui_clk),
    .ui_clk_sync_rst        (ui_clk_sync_rst),
    .init_calib_complete    (init_calib_complete),
    .sys_rst                (/*RESET_in*/reset)
);

//Instantiates FrameECC primitive to loacate errors in configuration memory
FRAME_ECCE2 #(
    .FARSRC("EFAR"),
    .FRAME_RBT_IN_FILENAME("None")
)
FRAME_ECCE2_inst (
    .CRCERROR(CRCERROR),
    .ECCERROR(ECCERROR),
    .ECCERRORSINGLE(ECCERRORSINGLE),
    .FAR(FAR),
    .SYNBIT(SYNBIT),
    .SYNDROME(SYNDROME),
    .SYNDROMEVALID(SYNDROMEVALID),
    .SYNWORD(SYNWORD)
);

//Instantiates ICAP to scrub configuration memory.
ICAPE2 #(
    .DEVICE_ID(0'h3651093),
    .ICAP_WIDTH("X32"),
    .SIM_CFG_FILE_NAME("None")
)
ICAPE2_inst (
    .O  (O),
```

```
        .CLK(SEM_Clock[0]),
        .CSIB(CSIB),
        .I(I),
        .RDWRB(RDWRB)
    );

    // The below is not necessary for CFTP, but only for the Gensys 2 Dev board
    Voter #(.WIDTH(1)) Output_SD_MOSI (
        .A      (SD_MOSI_out[0]),
        .B      (SD_MOSI_out[1]),
        .C      (SD_MOSI_out[2]),
        .True   (SD_MOSI)
    );
    Voter #(.WIDTH(1)) Output_SD_CS (
        .A      (SD_CS_out[0]),
        .B      (SD_CS_out[1]),
        .C      (SD_CS_out[2]),
        .True   (SD_CS)
    );
    Voter #(.WIDTH(1)) Output_SD_SClk (
        .A      (SD_SClk_out[0]),
        .B      (SD_SClk_out[1]),
        .C      (SD_SClk_out[2]),
        .True   (SD_SClk)
    );

endmodule
`default_nettype wire
```

(2) Clock Management Tiles

The System, SD, and DDR Clocks were implemented using XILINX's clocking wizard with the following settings differing from default: primary input frequency 200MHz, clock_out1 – clock_out3: 50MHZ with BUFG; clock_out4 – clock_out6: 50MHZ without BUFG; clock_out7: 200MHZ with BUFG.

The SEM Clocks were implemented using XILINX's clocking wizard with the following settings differing from default: primary input frequency 200MHz, clock_out1 – clock_out3: 100MHZ with BUFG.

2. TMR Level (ITMR_MIPS_SOC.v)

```
`timescale 1ns / 1ps
`default_nettype none
/*
 * File         : ITMR_MIPS_SOC.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   01-Mar-2016  ASJ       Release version.
 *
 * Standards/Formatting:
 *   Verilog 2001.
 *
 * Description:
 *   TMR level of the CFTP project. This modules instantiates the three identical
```

```
 *   systems and three identical voting systems. Votes for all registers internal
 *   to each of the three systems are forwarded to each of the three voters. A
 *   majority vote is conducted and each of the three voters forwards the true
 *   results to one of the three systems. An output voter for signals leaving the
 *   TMR level to the top layer is instantiated to produce a single output.
 */

module ITMR_MIPS_SOC(
   input wire [2:0] System_Clock_in,
   input wire Reset_in,
   //DDR U/I Signals
   input wire [255:0] app_rd_data,
   input wire app_rd_data_valid,
   input wire app_rdy,
   input wire app_wdf_rdy,
   input wire ui_clk,
   input wire ui_clk_sync_rst,
   input wire init_calib_complete,
   output wire [28:0] app_addr,
   output wire [2:0] app_cmd,
   output wire app_en,
   output wire [255:0] app_wdf_data,
   output wire app_wdf_end,
   output wire app_wdf_wren,
   //SD Signals
   input wire [2:0] SD_Clock_in,
   input wire [2:0] SD_MISO,
   output wire [2:0] SD_MOSI,
   output wire [2:0] SD_CS,
(* DONT_TOUCH = "TRUE" *)output wire [2:0] SD_SClk,
   //UART Signals
   input wire UART_Rx_in,
   output wire UART_Tx_out,
   //SEM Singals
   input wire [31:0] O_in,
   input wire [2:0] SEM_Clock_in,
   input wire CRCERROR_in,
   input wire ECCERROR_in,
   input wire ECCERRORSINGLE_in,
   input wire SYNDROMEVALID_in,
   input wire [12:0] SYNDROME_in,
   input wire [25:0] FAR_in,
   input wire [4:0] SYNBIT_in,
   input wire [6:0] SYNWORD_in,
   output wire status_heartbeat,
   output wire icap_csib,
   output wire icap_rdwrb,
   output wire [31:0] icap_i
   );

genvar i;
parameter N = 3;

/*** Voting Signals ***/
(* DONT_TOUCH = "TRUE" *)wire [1853:0] ITMR_Vote_in [N-1:0];
(* DONT_TOUCH = "TRUE" *)wire [1853:0] ITMR_Vote_out [N-1:0];
(* DONT_TOUCH = "TRUE" *)wire [333:0] TMR_Vote_out [N-1:0];
wire SD_SClk_out;

/*** MIPS Processors ***/

generate
   for (i = 0; i < N; i = i + 1) begin : System

(* DONT_TOUCH = "TRUE" *)MIPS_SOC MIPS_System(
      .System_Clock_in        (System_Clock_in[i]),
      .Reset_in               (Reset_in),
      //DDR U/I Signals
      .app_rd_data            (app_rd_data),
```

```
      .app_rd_data_valid      (app_rd_data_valid),
      .app_rdy                (app_rdy),
      .app_wdf_rdy            (app_wdf_rdy),
      .ui_clk                 (ui_clk),
      .ui_clk_sync_rst        (ui_clk_sync_rst),
      .init_calib_complete    (init_calib_complete),
      //SD Signals
      .SD_Clock_in            (SD_Clock_in[i]),
      .SD_MISO                (SD_MISO),
      //UART Signals
      .UART_Rx_in             (UART_Rx_in),
      //SEM Singals
      .O_in                   (O_in),
      .SEM_Clock_in           (SEM_Clock_in[i]),
      .CRCERROR_in            (CRCERROR_in),
      .ECCERROR_in            (ECCERROR_in),
      .ECCERRORSINGLE_in      (ECCERRORSINGLE_in),
      .SYNDROMEVALID_in       (SYNDROMEVALID_in),
      .SYNDROME_in            (SYNDROME_in),
      .FAR_in                 (FAR_in),
      .SYNBIT_in              (SYNBIT_in),
      .SYNWORD_in             (SYNWORD_in),
      //Voting Singals
      .ITMR_Vote_in           (ITMR_Vote_in[i]),
      .ITMR_Vote_out          (ITMR_Vote_out[i]),
      .TMR_Vote_out           (TMR_Vote_out[i])
   );

/*** Processor Internal Voters ***/

(* DONT_TOUCH = “TRUE” *)Processor_Voter Processor_Voter (
      .Vote_0_in              (ITMR_Vote_out[0]),
      .Vote_1_in              (ITMR_Vote_out[1]),
      .Vote_2_in              (ITMR_Vote_out[2]),
      .Vote_out               (ITMR_Vote_in[i])
   );

       end
endgenerate

/*** Output Voter ***/

   Output_Voter Output_Voter (
      .Vote_SOC_A             (TMR_Vote_out[N-1]),
      .Vote_SOC_B             (TMR_Vote_out[N-2]),
      .Vote_SOC_C             (TMR_Vote_out[N-3]),
      .app_addr               (app_addr),
      .app_cmd                (app_cmd),
      .app_en                 (app_en),
      .app_wdf_data           (app_wdf_data),
      .app_wdf_end            (app_wdf_end),
      .app_wdf_wren           (app_wdf_wren),
      .UART_Tx_out            (UART_Tx_out),
      .status_heartbeat       (status_heartbeat),
      .icap_csib              (icap_csib),
      .icap_rdwrb             (icap_rdwrb),
      .icap_i                 (icap_i),
      .SD_MOSI                (SD_MOSI),
      .SD_CS                  (SD_CS),
      .SD_SClk                (SD_SClk_out)
    );

assign SD_SClk = {SD_SClk_out,SD_SClk_out,SD_SClk_out};

endmodule
`default_nettype wire
```

3. System Level (MIPS_SOC.v)

```
`timescale 1ns / 1ps
`default_nettype none
/*
 * File         : MIPS_SOC.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)

 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   01-Mar-2016  ASJ       Release version.
 *
 * Standards/Formatting:
 *   Verilog 2001.
 *
 * Description:
 *    System level of the CFTP project. This modules instantiates the major
 * components that composes a full system: MIPS Processor, Memory, IO interface,
 * SEM, and a MMU.
 */

module MIPS_SOC(
   input wire System_Clock_in,
   input wire Reset_in,
   //DDR U/I Signals
   input wire [255:0] app_rd_data,
   input wire app_rd_data_valid,
   input wire app_rdy,
   input wire app_wdf_rdy,
   input wire ui_clk,
   input wire ui_clk_sync_rst,
   input wire init_calib_complete,
   //SD Signals
   input wire SD_Clock_in,
   input wire [2:0] SD_MISO,
   //UART Signals
   input wire UART_Rx_in,
   //SEM Singals
   input wire [31:0] O_in,
   input wire SEM_Clock_in,
   input wire CRCERROR_in,
   input wire ECCERROR_in,
   input wire ECCERRORSINGLE_in,
   input wire SYNDROMEVALID_in,
   input wire [12:0] SYNDROME_in,
   input wire [25:0] FAR_in,
   input wire [4:0] SYNBIT_in,
   input wire [6:0] SYNWORD_in,
   //Voting Singals
   input wire [1853:0] ITMR_Vote_in,
   output wire [1853:0] ITMR_Vote_out,
   output wire [333:0] TMR_Vote_out
   );

   // MIPS Processor Signals
   wire [31:0] MIPS32_DataMem_In;
   wire [31:0] MIPS32_DataMem_Out, MIPS32_InstMem_In;
   wire [29:0] MIPS32_DataMem_Address, MIPS32_InstMem_Address;
   wire [3:0]  MIPS32_DataMem_WE;
   wire        MIPS32_DataMem_Read, MIPS32_InstMem_Read;
   wire        MIPS32_DataMem_Ready;
   wire [4:0]  MIPS32_Interrupts;
   wire        MIPS32_NMI;

   // Memory Signals
   wire  [3:0] Inst_Write;
   wire  Inst_Read;
```

```
wire  [29:0] Inst_Address;
wire  [31:0] Inst_Data_in;
wire Inst_Ready;
wire Flush;
wire Flush_Complete;
wire Data_Read;
wire [3:0] Data_Write;
wire [31:0] Data_Data_out;
wire Data_Ready;
wire Init_L1_Mem, Init_L2_Mem;
wire Select_SD_Clock_Speed;
wire SD_SClk;

// UART Bootloader Signals
wire UART_RE;
wire UART_WE;
wire [16:0] UART_DOUT;
wire UART_Ack;
wire UART_Interrupt;
wire UART_BootResetCPU;
wire [29:0] UART_BootAddress;
wire [31:0] UART_BootData;
wire UART_BootWriteMem_pre;
wire [3:0] UART_BootWriteMem;

Processor MIPS(
   .clock                     (System_Clock_in),
   .reset                     (UART_BootResetCPU||Init_L1_Mem||Init_L2_Mem),
   .Interrupts                (MIPS32_Interrupts),
   .NMI                       (MIPS32_NMI),
   .DataMem_In                (MIPS32_DataMem_In),
   .DataMem_Ready             (MIPS32_DataMem_Ready),
   .DataMem_Read              (MIPS32_DataMem_Read),
   .DataMem_Write             (MIPS32_DataMem_WE),
   .DataMem_Address           (MIPS32_DataMem_Address),
   .DataMem_Out               (MIPS32_DataMem_Out),
   .InstMem_In                (MIPS32_InstMem_In),
   .InstMem_Address           (MIPS32_InstMem_Address),
   .InstMem_Ready             (Inst_Ready),
   .InstMem_Read              (MIPS32_InstMem_Read),
   // Register Voting Signals
   .Vote_in                   (ITMR_Vote_in[1853:0]),
   .Vote_out                  (ITMR_Vote_out[1853:0])
);

System_Memory Memory(
   .clock                     (System_Clock_in),
   .reset                     (Reset_in),
   .Inst_Read_in              (Inst_Read),
   .Inst_Write_in             (Inst_Write),
   .Inst_Address_in           (Inst_Address),
   .Inst_Data_in              (Inst_Data_in),
   .Inst_Data_out             (MIPS32_InstMem_In),
   .Inst_Ready_out            (Inst_Ready),
   .flush                     (Flush),
   .flushcomplete             (Flush_Complete),
   .Data_Read_in              (Data_Read),
   .Data_Write_in             (Data_Write),
   .Data_Address_in           (MIPS32_DataMem_Address),
   .Data_Data_in              (MIPS32_DataMem_Out),
   .Data_Data_out             (Data_Data_out),
   .Data_Ready_out            (Data_Ready),
   .Init_L1_Mem_out           (Init_L1_Mem),
   .Init_L2_Mem_out           (Init_L2_Mem),
   // DDR U/I Signals
   .app_addr                  (TMR_Vote_out[28:0]),
   .app_cmd                   (TMR_Vote_out[31:29]),
   .app_en                    (TMR_Vote_out[32]),
   .app_wdf_data              (TMR_Vote_out[288:33]),
```

```
   .app_wdf_end               (TMR_Vote_out[289]),
   .app_wdf_wren              (TMR_Vote_out[290]),
   .app_rd_data               (app_rd_data),
   .app_rd_data_valid         (app_rd_data_valid),
   .app_rdy                   (app_rdy),
   .app_wdf_rdy               (app_wdf_rdy),
   .ui_clk                    (ui_clk),
   .ui_clk_sync_rst           (ui_clk_sync_rst),
   .init_calib_complete       (init_calib_complete),
   // SD Card Signals
   .SD_MISO                   (SD_MISO),
   .SD_MOSI                   (TMR_Vote_out[329:327]),
   .SD_CS                     (TMR_Vote_out[332:330]),
   .SD_SClk                   (TMR_Vote_out[333]),
   .Select_SD_Clock_Speed     (Select_SD_Clock_Speed)
);

SD_Clock_Gen SD_Clock(
   .clk_in                    (SD_Clock_in),
   .select                    (Select_SD_Clock_Speed),
   .clk_out                   (TMR_Vote_out[333])
);

uart_bootloader UART(
   .clock                     (System_Clock_in),
   .reset                     (Reset_in),
   .Read                      (UART_RE),
   .Write                     (UART_WE),
   .DataIn                    (MIPS32_DataMem_Out[8:0]),
   .DataOut                   (UART_DOUT),
   .Ack                       (UART_Ack),
   .DataReady                 (UART_Interrupt),
   .BootResetCPU              (UART_BootResetCPU),
   .BootWriteMem              (UART_BootWriteMem_pre),
   .BootAddr                  (UART_BootAddress),
   .BootData                  (UART_BootData),
   .RxD                       (UART_Rx_in),
   .TxD                       (TMR_Vote_out[291]),
   .Inst_Ready                (Inst_Ready),
   .Flush_out                 (Flush),
   .Flush_Complete_in         (Flush_Complete)
);

MMU Memory_Mapping_Unit(
   .MIPS32_InstMem_Read       (MIPS32_InstMem_Read),
   .MIPS32_DataMem_Read       (MIPS32_DataMem_Read),
   .MIPS32_DataMem_WE         (MIPS32_DataMem_WE),
   .MIPS32_InstMem_Address    (MIPS32_InstMem_Address),
   .MIPS32_DataMem_Address    (MIPS32_DataMem_Address[29:26]),
   .UART_Interrupt            (UART_Interrupt),
   .UART_Ack                  (UART_Ack),
   .UART_BootResetCPU         (UART_BootResetCPU),
   .UART_BootWriteMem_pre     (UART_BootWriteMem_pre),
   .UART_DOUT                 (UART_DOUT),
   .UART_BootAddress          (UART_BootAddress),
   .UART_BootData             (UART_BootData),
   .Data_Ready                (Data_Ready),
   .Data_Data_out             (Data_Data_out),
   .MIPS32_DataMem_Ready      (MIPS32_DataMem_Ready),
   .MIPS32_NMI                (MIPS32_NMI),
   .MIPS32_Interrupts         (MIPS32_Interrupts),
   .MIPS32_DataMem_In         (MIPS32_DataMem_In),
   .UART_WE                   (UART_WE),
   .UART_RE                   (UART_RE),
   .Inst_Read                 (Inst_Read),
   .Data_Read                 (Data_Read),
   .Inst_Write                (Inst_Write),
   .Data_Write                (Data_Write),
   .Inst_Address              (Inst_Address),
```

```
      .Inst_Data_in              (Inst_Data_in)
   );

   sem_0 SEM(
      .status_heartbeat          (TMR_Vote_out[292]),
      .monitor_txfull            (1'B0),
      .monitor_rxdata            (8'H00),
      .monitor_rxempty           (1'B1),
      .icap_o                    (O_in),
      .icap_csib                 (TMR_Vote_out[293]),
      .icap_rdwrb                (TMR_Vote_out[294]),
      .icap_i                    (TMR_Vote_out[326:295]),
      .icap_clk                  (SEM_Clock_in),
      .icap_grant                (1'B1),
      .fecc_crcerr               (CRCERROR_in),
      .fecc_eccerr               (ECCERROR_in),
      .fecc_eccerrsingle         (ECCERRORSINGLE_in),
      .fecc_syndromevalid        (SYNDROMEVALID_in),
      .fecc_syndrome             (SYNDROME_in),
      .fecc_far                  (FAR_in),
      .fecc_synbit               (SYNBIT_in),
      .fecc_synword              (SYNWORD_in)
   );

endmodule
`default_nettype wire
```

a. MIPS32R1 Processor

Subsections one through nineteen contain all the Verilog files to implement the MIPS32R1 processor.

(1) MIPS_Parameters.v

```
/*
 * File         : MIPS_Parameters.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   26-May-2012  GEA       Release version.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Provides a language abstraction for the MIPS32-specific op-codes and
 *   the processor-specific datapath, hazard, and exception bits which
 *   control the processor. These definitions are used extensively
 *   throughout the processor HDL modules.
 */

/*** Exception Vector Locations ***

     When the CPU powers up or is reset, it will begin execution at
'EXC_Vector_Base_Reset'.
     All other exceptions are the sum of a base address and offset:
      - The base address is either a bootstrap or normal value. It is controlled by the
'BEV' bit in the CP0 'Status' register. Both base addresses can be mapped to the same
location.
      - The offset address is either a standard offset (which is always used for
non-interrupt general exceptions in this processor because it lacks TLB Refill and Cache
errors), or a special interrupt-only offset for interrupts, which is enabled with the
'IV' bit in the CP0 'Cause' register.
```

```
     Current Setup:
        General exceptions go to 0x0. Interrupts go to 0x8. Booting starts at 0x10.
*/

`define EXC_Vector_Base_Reset        32'h0000_0010 // MIPS Standard is 0xBFC0_0000
`define EXC_Vector_Base_Other_NoBoot 32'h0000_0000 // MIPS Standard is 0x8000_0000
`define EXC_Vector_Base_Other_Boot   32'h0000_0000 // MIPS Standard is 0xBFC0_0200
`define EXC_Vector_Offset_General    32'h0000_0000 // MIPS Standard is 0x0000_0180
`define EXC_Vector_Offset_Special    32'h0000_0008 // MIPS Standard is 0x0000_0200

/*** Kernel/User Memory Areas ***

     Kernel memory starts at address 0x0. User memory starts at 'UMem_Lower' and extends
to the end of the address space.
     A distinction is made to protect against accesses to kernel memory while the
processor is in user mode. Lacking MMU hardware, these addresses are physical, not
virtual.
     This simple two-part division of the address space can be extended almost
arbitrarily in the Data Memory Controller. Note that there is currently no user/kernel
space check for the Instruction Memory, because it is assumed that instructions are in
the kernel space.
*/

`define UMem_Lower 32'h08000000

/*** Processor Endianness ***

     The MIPS Configuration Register (CP0 Register 16 Select 0) specifies the processor's
endianness. A processor in user mode may switch to reverse endianness, which will be the
opposite of this `define.
*/

`define Big_Endian 1'b1

/*** Encodings for MIPS32 Release 1 Architecture ***/

/* Op Code Categories */
`define Op_Type_R   6'b00_0000  // Standard R-Type instructions
`define Op_Type_R2  6'b01_1100  // Extended R-Like instructions
`define Op_Type_BI  6'b00_0001  // Branch/Trap extended instructions
`define Op_Type_CP0 6'b01_0000  // Coprocessor 0 instructions
`define Op_Type_CP1 6'b01_0001  // Coprocessor 1 instructions (not implemented)
`define Op_Type_CP2 6'b01_0010  // Coprocessor 2 instructions (not implemented)
`define Op_Type_CP3 6'b01_0011  // Coprocessor 3 instructions (not implemented)
// -------------------------------------
`define Op_Add      `Op_Type_R
`define Op_Addi     6'b00_1000
`define Op_Addiu    6'b00_1001
`define Op_Addu     `Op_Type_R
`define Op_And      `Op_Type_R
`define Op_Andi     6'b00_1100
`define Op_Beq      6'b00_0100
`define Op_Bgez     `Op_Type_BI
`define Op_Bgezal   `Op_Type_BI
`define Op_Bgtz     6'b00_0111
`define Op_Blez     6'b00_0110
`define Op_Bltz     `Op_Type_BI
`define Op_Bltzal   `Op_Type_BI
`define Op_Bne      6'b00_0101
`define Op_Break    `Op_Type_R
`define Op_Clo      `Op_Type_R2
`define Op_Clz      `Op_Type_R2
`define Op_Div      `Op_Type_R
`define Op_Divu     `Op_Type_R
`define Op_Eret     `Op_Type_CP0
`define Op_J        6'b00_0010
`define Op_Jal      6'b00_0011
`define Op_Jalr     `Op_Type_R
```

```
`define Op_Jr       `Op_Type_R
`define Op_Lb       6'b10_0000
`define Op_Lbu      6'b10_0100
`define Op_Lh       6'b10_0001
`define Op_Lhu      6'b10_0101
`define Op_Ll       6'b11_0000
`define Op_Lui      6'b00_1111
`define Op_Lw       6'b10_0011
`define Op_Lwl      6'b10_0010
`define Op_Lwr      6'b10_0110
`define Op_Madd     `Op_Type_R2
`define Op_Maddu    `Op_Type_R2
`define Op_Mfc0     `Op_Type_CP0
`define Op_Mfhi     `Op_Type_R
`define Op_Mflo     `Op_Type_R
`define Op_Movn     `Op_Type_R
`define Op_Movz     `Op_Type_R
`define Op_Msub     `Op_Type_R2
`define Op_Msubu    `Op_Type_R2
`define Op_Mtc0     `Op_Type_CP0
`define Op_Mthi     `Op_Type_R
`define Op_Mtlo     `Op_Type_R
`define Op_Mul      `Op_Type_R2
`define Op_Mult     `Op_Type_R
`define Op_Multu    `Op_Type_R
`define Op_Nor      `Op_Type_R
`define Op_Or       `Op_Type_R
`define Op_Ori      6'b00_1101
`define Op_Pref     6'b11_0011  // Prefetch does nothing in this implementation.
`define Op_Sb       6'b10_1000
`define Op_Sc       6'b11_1000
`define Op_Sh       6'b10_1001
`define Op_Sll      `Op_Type_R
`define Op_Sllv     `Op_Type_R
`define Op_Slt      `Op_Type_R
`define Op_Slti     6'b00_1010
`define Op_Sltiu    6'b00_1011
`define Op_Sltu     `Op_Type_R
`define Op_Sra      `Op_Type_R
`define Op_Srav     `Op_Type_R
`define Op_Srl      `Op_Type_R
`define Op_Srlv     `Op_Type_R
`define Op_Sub      `Op_Type_R
`define Op_Subu     `Op_Type_R
`define Op_Sw       6'b10_1011
`define Op_Swl      6'b10_1010
`define Op_Swr      6'b10_1110
`define Op_Syscall  `Op_Type_R
`define Op_Teq      `Op_Type_R
`define Op_Teqi     `Op_Type_BI
`define Op_Tge      `Op_Type_R
`define Op_Tgei     `Op_Type_BI
`define Op_Tgeiu    `Op_Type_BI
`define Op_Tgeu     `Op_Type_R
`define Op_Tlt      `Op_Type_R
`define Op_Tlti     `Op_Type_BI
`define Op_Tltiu    `Op_Type_BI
`define Op_Tltu     `Op_Type_R
`define Op_Tne      `Op_Type_R
`define Op_Tnei     `Op_Type_BI
`define Op_Xor      `Op_Type_R
`define Op_Xori     6'b00_1110

/* Op Code Rt fields for Branches & Traps */
`define OpRt_Bgez   5'b00001
`define OpRt_Bgezal 5'b10001
`define OpRt_Bltz   5'b00000
`define OpRt_Bltzal 5'b10000
`define OpRt_Teqi   5'b01100
```

```
`define OpRt_Tgei   5'b01000
`define OpRt_Tgeiu  5'b01001
`define OpRt_Tlti   5'b01010
`define OpRt_Tltiu  5'b01011
`define OpRt_Tnei   5'b01110

/* Op Code Rs fields for Coprocessors */
`define OpRs_MF     5'b00000
`define OpRs_MT     5'b00100

/* Special handling for ERET */
`define OpRs_ERET   5'b10000
`define Funct_ERET  6'b011000

/* Function Codes for R-Type Op Codes */
`define Funct_Add     6'b10_0000
`define Funct_Addu    6'b10_0001
`define Funct_And     6'b10_0100
`define Funct_Break   6'b00_1101
`define Funct_Clo     6'b10_0001    // same as Addu
`define Funct_Clz     6'b10_0000    // same as Add
`define Funct_Div     6'b01_1010
`define Funct_Divu    6'b01_1011
`define Funct_Jr      6'b00_1000
`define Funct_Jalr    6'b00_1001
`define Funct_Madd    6'b00_0000
`define Funct_Maddu   6'b00_0001
`define Funct_Mfhi    6'b01_0000
`define Funct_Mflo    6'b01_0010
`define Funct_Movn    6'b00_1011
`define Funct_Movz    6'b00_1010
`define Funct_Msub    6'b00_0100    // same as Sllv
`define Funct_Msubu   6'b00_0101
`define Funct_Mthi    6'b01_0001
`define Funct_Mtlo    6'b01_0011
`define Funct_Mul     6'b00_0010    // same as Srl
`define Funct_Mult    6'b01_1000
`define Funct_Multu   6'b01_1001
`define Funct_Nor     6'b10_0111
`define Funct_Or      6'b10_0101
`define Funct_Sll     6'b00_0000
`define Funct_Sllv    6'b00_0100
`define Funct_Slt     6'b10_1010
`define Funct_Sltu    6'b10_1011
`define Funct_Sra     6'b00_0011
`define Funct_Srav    6'b00_0111
`define Funct_Srl     6'b00_0010
`define Funct_Srlv    6'b00_0110
`define Funct_Sub     6'b10_0010
`define Funct_Subu    6'b10_0011
`define Funct_Syscall 6'b00_1100
`define Funct_Teq     6'b11_0100
`define Funct_Tge     6'b11_0000
`define Funct_Tgeu    6'b11_0001
`define Funct_Tlt     6'b11_0010
`define Funct_Tltu    6'b11_0011
`define Funct_Tne     6'b11_0110
`define Funct_Xor     6'b10_0110

/* ALU Operations (Implementation) */
`define AluOp_Add    5'd1
`define AluOp_Addu   5'd0
`define AluOp_And    5'd2
`define AluOp_Clo    5'd3
`define AluOp_Clz    5'd4
`define AluOp_Div    5'd5
`define AluOp_Divu   5'd6
`define AluOp_Madd   5'd7
`define AluOp_Maddu  5'd8
```

```
`define AluOp_Mfhi   5'd9
`define AluOp_Mflo   5'd10
`define AluOp_Msub   5'd13
`define AluOp_Msubu  5'd14
`define AluOp_Mthi   5'd11
`define AluOp_Mtlo   5'd12
`define AluOp_Mul    5'd15
`define AluOp_Mult   5'd16
`define AluOp_Multu  5'd17
`define AluOp_Nor    5'd18
`define AluOp_Or     5'd19
`define AluOp_Sll    5'd20
`define AluOp_Sllc   5'd21  // Move this if another AluOp is needed
`define AluOp_Sllv   5'd22
`define AluOp_Slt    5'd23
`define AluOp_Sltu   5'd24
`define AluOp_Sra    5'd25
`define AluOp_Srav   5'd26
`define AluOp_Srl    5'd27
`define AluOp_Srlv   5'd28
`define AluOp_Sub    5'd29
`define AluOp_Subu   5'd30
`define AluOp_Xor    5'd31


// Movc:10->11, Trap:9->10, TrapCond:8->9, RegDst:7->8

/*** Datapath ***

     All Signals are Active High. Branching and Jump signals (determined by "PCSrc"), as
well as ALU operation signals ("ALUOp") are handled by the controller and are not found
here.

     Bit  Name          Description
     ------------------------------
     15:  PCSrc         (Instruction Type)
     14:                   11: Instruction is Jump to Register
                           10: Instruction is Branch
                           01: Instruction is Jump to Immediate
                           00: Instruction does not branch nor jump
     13:  Link          (Link on Branch/Jump)
     ------------------------------
     12:  ALUSrc        (ALU Source) [0=ALU input B is 2nd register file output; 1
                              =Immediate value]
     11:  Movc          (Conditional Move)
     10:  Trap          (Trap Instruction)
     9 :  TrapCond      (Trap Condition) [0=ALU result is 0; 1=ALU result is not
                                  0]
     8 :  RegDst        (Register File Target) [0=Rt field; 1=Rd field]
     ------------------------------
     7 :  LLSC          (Load Linked or Store Conditional)
     6 :  MemRead       (Data Memory Read)
     5 :  MemWrite      (Data Memory Write)
     4 :  MemHalf       (Half Word Memory Access)
     3 :  MemByte       (Byte size Memory Access)
     2 :  MemSignExtend (Sign Extend Read Memory) [0=Zero Extend; 1=Sign Extend]
     ------------------------------
     1 :  RegWrite      (Register File Write)
     0 :  MemtoReg      (Memory to Register) [0=Register File write data is ALU
                                        output; 1=Is Data Memory]
     ------------------------------
*/
`define DP_None        16'b000_00000_000000_00      // Instructions which require
                                                     nothing of the main
                                                     datapath.
`define DP_RType       16'b000_00001_000000_10      // Standard R-Type
`define DP_IType       16'b000_10000_000000_10      // Standard I-Type
`define DP_Branch      16'b100_00000_000000_00      // Standard Branch
`define DP_BranchLink  16'b101_00000_000000_10      // Branch and Link
```

```
`define DP_HiLoWr      16'b000_00000_000000_00      // Write to Hi/Lo ALU register
                                                       (Div,Divu,Mult,Multu,Mthi,
                                                        Mtlo). Currently 'DP_None'.
`define DP_Jump        16'b010_00000_000000_00      // Standard Jump
`define DP_JumpLink    16'b011_00000_000000_10      // Jump and Link
`define DP_JumpLinkReg 16'b111_00000_000000_10      // Jump and Link Register
`define DP_JumpReg     16'b110_00000_000000_00      // Jump Register
`define DP_LoadByteS   16'b000_10000_010011_11      // Load Byte Signed
`define DP_LoadByteU   16'b000_10000_010010_11      // Load Byte Unsigned
`define DP_LoadHalfS   16'b000_10000_010101_11      // Load Half Signed
`define DP_LoadHalfU   16'b000_10000_010100_11      // Load Half Unsigned
`define DP_LoadWord    16'b000_10000_010000_11      // Load Word
`define DP_ExtWrRt     16'b000_00000_000000_10      // A DP-external write to Rt
`define DP_ExtWrRd     16'b000_00001_000000_10      // A DP-external write to Rd
`define DP_Movc        16'b000_01001_000000_10      // Conditional Move
`define DP_LoadLinked  16'b000_10000_110000_11      // Load Linked
`define DP_StoreCond   16'b000_10000_101000_11      // Store Conditional
`define DP_StoreByte   16'b000_10000_001010_00      // Store Byte
`define DP_StoreHalf   16'b000_10000_001100_00      // Store Half
`define DP_StoreWord   16'b000_10000_001000_00      // Store Word
`define DP_TrapRegCNZ  16'b000_00110_000000_00      // Trap using Rs and Rt,  non-zero
                                                       ALU (Tlt,Tltu,Tne)
`define DP_TrapRegCZ   16'b000_00100_000000_00      // Trap using RS and Rt, zero
                                                       ALU (Teq,Tge,Tgeu)
`define DP_TrapImmCNZ  16'b000_10110_000000_00      // Trap using Rs and Imm, non-zero
                                                       ALU (Tlti,Tltiu,Tnei)
`define DP_TrapImmCZ   16'b000_10100_000000_00      // Trap using Rs and Imm, zero
                                                       ALU (Teqi,Tgei,Tgeiu)
//------------------------------------------------------------
`define DP_Add     `DP_RType
`define DP_Addi    `DP_IType
`define DP_Addiu   `DP_IType
`define DP_Addu    `DP_RType
`define DP_And     `DP_RType
`define DP_Andi    `DP_IType
`define DP_Beq     `DP_Branch
`define DP_Bgez    `DP_Branch
`define DP_Bgezal  `DP_BranchLink
`define DP_Bgtz    `DP_Branch
`define DP_Blez    `DP_Branch
`define DP_Bltz    `DP_Branch
`define DP_Bltzal  `DP_BranchLink
`define DP_Bne     `DP_Branch
`define DP_Break   `DP_None
`define DP_Clo     `DP_RType
`define DP_Clz     `DP_RType
`define DP_Div     `DP_HiLoWr
`define DP_Divu    `DP_HiLoWr
`define DP_Eret    `DP_None
`define DP_J       `DP_Jump
`define DP_Jal     `DP_JumpLink
`define DP_Jalr    `DP_JumpLinkReg
`define DP_Jr      `DP_JumpReg
`define DP_Lb      `DP_LoadByteS
`define DP_Lbu     `DP_LoadByteU
`define DP_Lh      `DP_LoadHalfS
`define DP_Lhu     `DP_LoadHalfU
`define DP_Ll      `DP_LoadLinked
`define DP_Lui     `DP_IType
`define DP_Lw      `DP_LoadWord
`define DP_Lwl     `DP_LoadWord
`define DP_Lwr     `DP_LoadWord
`define DP_Madd    `DP_HiLoWr
`define DP_Maddu   `DP_HiLoWr
`define DP_Mfc0    `DP_ExtWrRt
`define DP_Mfhi    `DP_ExtWrRd
`define DP_Mflo    `DP_ExtWrRd
`define DP_Movn    `DP_Movc
`define DP_Movz    `DP_Movc
```

```
`define DP_Msub    `DP_HiLoWr
`define DP_Msubu   `DP_HiLoWr
`define DP_Mtc0    `DP_None
`define DP_Mthi    `DP_HiLoWr
`define DP_Mtlo    `DP_HiLoWr
`define DP_Mul     `DP_RType
`define DP_Mult    `DP_HiLoWr
`define DP_Multu   `DP_HiLoWr
`define DP_Nor     `DP_RType
`define DP_Or      `DP_RType
`define DP_Ori     `DP_IType
`define DP_Pref    `DP_None  // Not Implemented
`define DP_Sb      `DP_StoreByte
`define DP_Sc      `DP_StoreCond
`define DP_Sh      `DP_StoreHalf
`define DP_Sll     `DP_RType
`define DP_Sllv    `DP_RType
`define DP_Slt     `DP_RType
`define DP_Slti    `DP_IType
`define DP_Sltiu   `DP_IType
`define DP_Sltu    `DP_RType
`define DP_Sra     `DP_RType
`define DP_Srav    `DP_RType
`define DP_Srl     `DP_RType
`define DP_Srlv    `DP_RType
`define DP_Sub     `DP_RType
`define DP_Subu    `DP_RType
`define DP_Sw      `DP_StoreWord
`define DP_Swl     `DP_StoreWord
`define DP_Swr     `DP_StoreWord
`define DP_Syscall `DP_None
`define DP_Teq     `DP_TrapRegCZ
`define DP_Teqi    `DP_TrapImmCZ
`define DP_Tge     `DP_TrapRegCZ
`define DP_Tgei    `DP_TrapImmCZ
`define DP_Tgeiu   `DP_TrapImmCZ
`define DP_Tgeu    `DP_TrapRegCZ
`define DP_Tlt     `DP_TrapRegCNZ
`define DP_Tlti    `DP_TrapImmCNZ
`define DP_Tltiu   `DP_TrapImmCNZ
`define DP_Tltu    `DP_TrapRegCNZ
`define DP_Tne     `DP_TrapRegCNZ
`define DP_Tnei    `DP_TrapImmCNZ
`define DP_Xor     `DP_RType
`define DP_Xori    `DP_IType



/*** Exception Information ***

     All signals are Active High.

     Bit  Meaning
     ------------
     2:   Instruction can cause exceptions in ID
     1:   Instruction can cause exceptions in EX
     0:   Instruction can cause exceptions in MEM
*/
`define EXC_None 3'b000
`define EXC_ID   3'b100
`define EXC_EX   3'b010
`define EXC_MEM  3'b001
//--------------------------------
`define EXC_Add     `EXC_EX
`define EXC_Addi    `EXC_EX
`define EXC_Addiu   `EXC_None
`define EXC_Addu    `EXC_None
`define EXC_And     `EXC_None
```

```
`define EXC_Andi    `EXC_None
`define EXC_Beq     `EXC_None
`define EXC_Bgez    `EXC_None
`define EXC_Bgezal  `EXC_None
`define EXC_Bgtz    `EXC_None
`define EXC_Blez    `EXC_None
`define EXC_Bltz    `EXC_None
`define EXC_Bltzal  `EXC_None
`define EXC_Bne     `EXC_None
`define EXC_Break   `EXC_ID
`define EXC_Clo     `EXC_None
`define EXC_Clz     `EXC_None
`define EXC_Div     `EXC_None
`define EXC_Divu    `EXC_None
`define EXC_Eret    `EXC_ID
`define EXC_J       `EXC_None
`define EXC_Jal     `EXC_None
`define EXC_Jalr    `EXC_None
`define EXC_Jr      `EXC_None
`define EXC_Lb      `EXC_MEM
`define EXC_Lbu     `EXC_MEM
`define EXC_Lh      `EXC_MEM
`define EXC_Lhu     `EXC_MEM
`define EXC_Ll      `EXC_MEM
`define EXC_Lui     `EXC_None
`define EXC_Lw      `EXC_MEM
`define EXC_Lwl     `EXC_MEM
`define EXC_Lwr     `EXC_MEM
`define EXC_Madd    `EXC_None
`define EXC_Maddu   `EXC_None
`define EXC_Mfc0    `EXC_ID
`define EXC_Mfhi    `EXC_None
`define EXC_Mflo    `EXC_None
`define EXC_Movn    `EXC_None
`define EXC_Movz    `EXC_None
`define EXC_Msub    `EXC_None
`define EXC_Msubu   `EXC_None
`define EXC_Mtc0    `EXC_ID
`define EXC_Mthi    `EXC_None
`define EXC_Mtlo    `EXC_None
`define EXC_Mul     `EXC_None
`define EXC_Mult    `EXC_None
`define EXC_Multu   `EXC_None
`define EXC_Nor     `EXC_None
`define EXC_Or      `EXC_None
`define EXC_Ori     `EXC_None
`define EXC_Pref    `EXC_None    // XXX
`define EXC_Sb      `EXC_MEM
`define EXC_Sc      `EXC_MEM
`define EXC_Sh      `EXC_MEM
`define EXC_Sll     `EXC_None
`define EXC_Sllv    `EXC_None
`define EXC_Slt     `EXC_None
`define EXC_Slti    `EXC_None
`define EXC_Sltiu   `EXC_None
`define EXC_Sltu    `EXC_None
`define EXC_Sra     `EXC_None
`define EXC_Srav    `EXC_None
`define EXC_Srl     `EXC_None
`define EXC_Srlv    `EXC_None
`define EXC_Sub     `EXC_EX
`define EXC_Subu    `EXC_None
`define EXC_Sw      `EXC_MEM
`define EXC_Swl     `EXC_MEM
`define EXC_Swr     `EXC_MEM
`define EXC_Syscall `EXC_ID
`define EXC_Teq     `EXC_MEM
`define EXC_Teqi    `EXC_MEM
`define EXC_Tge     `EXC_MEM
```

```
`define EXC_Tgei    `EXC_MEM
`define EXC_Tgeiu   `EXC_MEM
`define EXC_Tgeu    `EXC_MEM
`define EXC_Tlt     `EXC_MEM
`define EXC_Tlti    `EXC_MEM
`define EXC_Tltiu   `EXC_MEM
`define EXC_Tltu    `EXC_MEM
`define EXC_Tne     `EXC_MEM
`define EXC_Tnei    `EXC_MEM
`define EXC_Xor     `EXC_None
`define EXC_Xori    `EXC_None



/*** Hazard & Forwarding Datapath ***

     All signals are Active High.

     Bit  Meaning
     ------------
     7:   Wants Rs by ID
     6:   Needs Rs by ID
     5:   Wants Rt by ID
     4:   Needs Rt by ID
     3:   Wants Rs by EX
     2:   Needs Rs by EX
     1:   Wants Rt by EX
     0:   Needs Rt by EX
*/
`define HAZ_Nothing  8'b00000000    // Jumps, Lui, Mfhi/lo, special, etc.
`define HAZ_IDRsIDRt 8'b11110000    // Beq, Bne, Traps
`define HAZ_IDRs     8'b11000000    // Most branches, Jumps to registers
`define HAZ_IDRt     8'b00110000    // Mtc0
`define HAZ_IDRtEXRs 8'b10111100    // Movn, Movz
`define HAZ_EXRsEXRt 8'b10101111    // Many R-Type ops
`define HAZ_EXRs     8'b10001100    // Immediates: Loads, Clo/z, Mthi/lo, etc.
`define HAZ_EXRsWRt  8'b10101110    // Stores
`define HAZ_EXRt     8'b00100011    // Shifts using Shamt field
//----------------------------------------
`define HAZ_Add     `HAZ_EXRsEXRt
`define HAZ_Addi    `HAZ_EXRs
`define HAZ_Addiu   `HAZ_EXRs
`define HAZ_Addu    `HAZ_EXRsEXRt
`define HAZ_And     `HAZ_EXRsEXRt
`define HAZ_Andi    `HAZ_EXRs
`define HAZ_Beq     `HAZ_IDRsIDRt
`define HAZ_Bgez    `HAZ_IDRs
`define HAZ_Bgezal  `HAZ_IDRs
`define HAZ_Bgtz    `HAZ_IDRs
`define HAZ_Blez    `HAZ_IDRs
`define HAZ_Bltz    `HAZ_IDRs
`define HAZ_Bltzal  `HAZ_IDRs
`define HAZ_Bne     `HAZ_IDRsIDRt
`define HAZ_Break   `HAZ_Nothing
`define HAZ_Clo     `HAZ_EXRs
`define HAZ_Clz     `HAZ_EXRs
`define HAZ_Div     `HAZ_EXRsEXRt
`define HAZ_Divu    `HAZ_EXRsEXRt
`define HAZ_Eret    `HAZ_Nothing
`define HAZ_J       `HAZ_Nothing
`define HAZ_Jal     `HAZ_Nothing
`define HAZ_Jalr    `HAZ_IDRs
`define HAZ_Jr      `HAZ_IDRs
`define HAZ_Lb      `HAZ_EXRs
`define HAZ_Lbu     `HAZ_EXRs
`define HAZ_Lh      `HAZ_EXRs
`define HAZ_Lhu     `HAZ_EXRs
`define HAZ_Ll      `HAZ_EXRs
```

```
`define HAZ_Lui     `HAZ_Nothing
`define HAZ_Lw      `HAZ_EXRs
`define HAZ_Lwl     `HAZ_EXRsEXRt
`define HAZ_Lwr     `HAZ_EXRsEXRt
`define HAZ_Madd    `HAZ_EXRsEXRt
`define HAZ_Maddu   `HAZ_EXRsEXRt
`define HAZ_Mfc0    `HAZ_Nothing
`define HAZ_Mfhi    `HAZ_Nothing
`define HAZ_Mflo    `HAZ_Nothing
`define HAZ_Movn    `HAZ_IDRtEXRs
`define HAZ_Movz    `HAZ_IDRtEXRs
`define HAZ_Msub    `HAZ_EXRsEXRt
`define HAZ_Msubu   `HAZ_EXRsEXRt
`define HAZ_Mtc0    `HAZ_IDRt
`define HAZ_Mthi    `HAZ_EXRs
`define HAZ_Mtlo    `HAZ_EXRs
`define HAZ_Mul     `HAZ_EXRsEXRt
`define HAZ_Mult    `HAZ_EXRsEXRt
`define HAZ_Multu   `HAZ_EXRsEXRt
`define HAZ_Nor     `HAZ_EXRsEXRt
`define HAZ_Or      `HAZ_EXRsEXRt
`define HAZ_Ori     `HAZ_EXRs
`define HAZ_Pref    `HAZ_Nothing // XXX
`define HAZ_Sb      `HAZ_EXRsWRt
`define HAZ_Sc      `HAZ_EXRsWRt
`define HAZ_Sh      `HAZ_EXRsWRt
`define HAZ_Sll     `HAZ_EXRt
`define HAZ_Sllv    `HAZ_EXRsEXRt
`define HAZ_Slt     `HAZ_EXRsEXRt
`define HAZ_Slti    `HAZ_EXRs
`define HAZ_Sltiu   `HAZ_EXRs
`define HAZ_Sltu    `HAZ_EXRsEXRt
`define HAZ_Sra     `HAZ_EXRt
`define HAZ_Srav    `HAZ_EXRsEXRt
`define HAZ_Srl     `HAZ_EXRt
`define HAZ_Srlv    `HAZ_EXRsEXRt
`define HAZ_Sub     `HAZ_EXRsEXRt
`define HAZ_Subu    `HAZ_EXRsEXRt
`define HAZ_Sw      `HAZ_EXRsWRt
`define HAZ_Swl     `HAZ_EXRsWRt
`define HAZ_Swr     `HAZ_EXRsWRt
`define HAZ_Syscall `HAZ_Nothing
`define HAZ_Teq     `HAZ_EXRsEXRt
`define HAZ_Teqi    `HAZ_EXRs
`define HAZ_Tge     `HAZ_EXRsEXRt
`define HAZ_Tgei    `HAZ_EXRs
`define HAZ_Tgeiu   `HAZ_EXRs
`define HAZ_Tgeu    `HAZ_EXRsEXRt
`define HAZ_Tlt     `HAZ_EXRsEXRt
`define HAZ_Tlti    `HAZ_EXRs
`define HAZ_Tltiu   `HAZ_EXRs
`define HAZ_Tltu    `HAZ_EXRsEXRt
`define HAZ_Tne     `HAZ_EXRsEXRt
`define HAZ_Tnei    `HAZ_EXRs
`define HAZ_Xor     `HAZ_EXRsEXRt
`define HAZ_Xori    `HAZ_EXRs
```

(2) Processor.v

```
`timescale 1ns / 1ps
/*
 * File         : Processor.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   23-Jul-2011  GEA       Initial design.
```

```
 *   2.0   26-May-2012  GEA       Release version with CP0.
 *   2.01   1-Nov-2012  GEA       Fixed issue with Jal.
 *   3.0   26-Feb-2016  ASJ       Registers modified to accomodate GTMR.
 *                                Voting signals added.
 *                                Memory interface protocol adjusted.
 *                                Fixed issue with UART
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   The top-level MIPS32 Processor. This file is mostly the instantiation
 *   and wiring of the building blocks of the processor according to the
 *   hardware design diagram. It contains very little logic itself.
 */
module Processor(
   input  clock,
   input  reset,
   input  [4:0] Interrupts,            // 5 general-purpose hardware interrupts
   input  NMI,                         // Non-maskable interrupt
   // Data Memory Interface
   input  [31:0] DataMem_In,
   input  DataMem_Ready,
   output DataMem_Read,
   output [3:0]  DataMem_Write,        // 4-bit Write, one for each byte in word.
   output [29:0] DataMem_Address,      // Addresses are words, not bytes.
   output [31:0] DataMem_Out,
   // Instruction Memory Interface
   input  [31:0] InstMem_In,
   output [29:0] InstMem_Address,      // Addresses are words, not bytes.
   input  InstMem_Ready,
   output InstMem_Read,
   // Register Voting Signals
   input  [1853:0] Vote_in,
   output [1853:0] Vote_out
);

`include “MIPS_Parameters.v”


/*** MIPS Instruction and Components (ID Stage) ***/
wire [31:0] Instruction;
wire [5:0]  OpCode = Instruction[31:26];
wire [4:0]  Rs = Instruction[25:21];
wire [4:0]  Rt = Instruction[20:16];
wire [4:0]  Rd = Instruction[15:11];
wire [5:0]  Funct = Instruction[5:0];
wire [15:0] Immediate = Instruction[15:0];
wire [25:0] JumpAddress = Instruction[25:0];
wire [2:0]  Cp0_Sel = Instruction[2:0];

/*** IF (Instruction Fetch) Signals ***/
wire IF_Stall, IF_Flush;
wire IF_EXC_AdIF;
wire IF_Exception_Stall;
wire IF_Exception_Flush;
wire IF_IsBDS;
wire [31:0] IF_PCAdd4, IF_PC_PreExc, IF_PCIn, IF_PCOut, IF_Instruction;

/*** ID (Instruction Decode) Signals ***/
wire ID_Stall;
wire [1:0] ID_PCSrc;
wire [1:0] ID_RsFwdSel, ID_RtFwdSel;
wire ID_Link, ID_Movn, ID_Movz;
wire ID_SignExtend;
wire ID_LLSC;
wire ID_RegDst, ID_ALUSrcImm, ID_MemWrite, ID_MemRead, ID_MemByte, ID_MemHalf,
     ID_MemSignExtend, ID_RegWrite, ID_MemtoReg;
wire [4:0] ID_ALUOp;
```

```
wire ID_Mfc0, ID_Mtc0, ID_Eret;
wire ID_NextIsDelay;
wire ID_CanErr, ID_ID_CanErr, ID_EX_CanErr, ID_M_CanErr;
wire ID_KernelMode;
wire ID_ReverseEndian;
wire ID_Trap, ID_TrapCond;
wire ID_EXC_Sys, ID_EXC_Bp, ID_EXC_RI;
wire ID_Exception_Stall;
wire ID_Exception_Flush;
wire ID_PCSrc_Exc;
wire [31:0] ID_ExceptionPC;
wire ID_CP1, ID_CP2, ID_CP3;
wire [31:0] ID_PCAdd4;
wire [31:0] ID_ReadData1_RF, ID_ReadData1_End;
wire [31:0] ID_ReadData2_RF, ID_ReadData2_End;
wire [31:0] CP0_RegOut;
wire ID_CmpEQ, ID_CmpGZ, ID_CmpLZ, ID_CmpGEZ, ID_CmpLEZ;
wire [29:0] ID_SignExtImm = (ID_SignExtend) ? {{14{Immediate[15]}}, Immediate} :
                            {14'h0000, Immediate}; //More Explicit Sign Extend
wire [31:0] ID_ImmLeftShift2 = {ID_SignExtImm[29:0], 2'b00};
wire [31:0] ID_JumpAddress = {ID_PCAdd4[31:28], JumpAddress[25:0], 2'b00};
wire [31:0] ID_BranchAddress;
wire [31:0] ID_RestartPC;
wire ID_IsBDS;
wire ID_Left, ID_Right;
wire ID_IsFlushed;

/*** EX (Execute) Signals ***/
wire EX_ALU_Stall, EX_Stall;
wire [1:0] EX_RsFwdSel, EX_RtFwdSel;
wire EX_Link;
wire [1:0] EX_LinkRegDst;
wire EX_ALUSrcImm;
wire [4:0] EX_ALUOp;
wire EX_Movn, EX_Movz;
wire EX_LLSC;
wire EX_MemRead, EX_MemWrite, EX_MemByte, EX_MemHalf, EX_MemSignExtend, EX_RegWrite,
     EX_MemtoReg;
wire [4:0] EX_Rs, EX_Rt;
wire EX_WantRsByEX, EX_NeedRsByEX, EX_WantRtByEX, EX_NeedRtByEX;
wire EX_Trap, EX_TrapCond;
wire EX_CanErr, EX_EX_CanErr, EX_M_CanErr;
wire EX_KernelMode;
wire EX_ReverseEndian;
wire EX_Exception_Stall;
wire EX_Exception_Flush;
wire [31:0] EX_ReadData1_PR, EX_ReadData1_Fwd, EX_ReadData2_PR, EX_ReadData2_Fwd,
            EX_ReadData2_Imm;
wire [31:0] EX_SignExtImm;
wire [4:0] EX_Rd, EX_RtRd, EX_Shamt;
wire [31:0] EX_ALUResult;
wire EX_BZero;
wire EX_EXC_Ov;
wire [31:0] EX_RestartPC;
wire EX_IsBDS;
wire EX_Left, EX_Right;

/*** MEM (Memory) Signals ***/
wire M_Stall, M_Stall_Controller;
wire M_LLSC;
wire M_MemRead, M_MemWrite, M_MemByte, M_MemHalf, M_MemSignExtend;
wire M_RegWrite, M_MemtoReg;
wire M_WriteDataFwdSel;
wire M_EXC_AdEL, M_EXC_AdES;
wire M_M_CanErr;
wire M_KernelMode;
wire M_ReverseEndian;
wire M_Trap, M_TrapCond;
wire M_EXC_Tr;
```

```
wire M_Exception_Flush;
wire [31:0] M_ALUResult, M_ReadData2_PR;
wire [4:0] M_RtRd;
wire [31:0] M_MemReadData;
wire [31:0] M_RestartPC;
wire M_IsBDS;
wire [31:0] M_WriteData_Pre;
wire M_Left, M_Right;
wire M_Exception_Stall;

/*** WB (Writeback) Signals ***/
wire WB_Stall, WB_RegWrite, WB_MemtoReg;
wire [31:0] WB_ReadData, WB_ALUResult;
wire [4:0]  WB_RtRd;
wire [31:0] WB_WriteData;

/*** Other Signals ***/
wire [7:0] ID_DP_Hazards, HAZ_DP_Hazards;

/*** Assignments ***/
assign IF_Instruction = (IF_Stall) ? 32'h00000000 : InstMem_In;
assign IF_IsBDS = ID_NextIsDelay;
assign HAZ_DP_Hazards = {ID_DP_Hazards[7:4], EX_WantRsByEX, EX_NeedRsByEX, EX_WantRtByEX,
                         EX_NeedRtByEX};
assign IF_EXC_AdIF = IF_PCOut[1] | IF_PCOut[0];
assign ID_CanErr = ID_ID_CanErr | ID_EX_CanErr | ID_M_CanErr;
assign EX_CanErr = EX_EX_CanErr | EX_M_CanErr;

// External Memory Interface
assign InstMem_Address = IF_PCOut[31:2];
assign DataMem_Address = M_ALUResult[31:2];
assign InstMem_Read = 1'b1;

/*** Datapath Controller ***/
Control Controller (
   .ID_Stall       (ID_Stall),
   .OpCode         (OpCode),
   .Funct          (Funct),
   .Rs             (Rs),
   .Rt             (Rt),
   .Cmp_EQ         (ID_CmpEQ),
   .Cmp_GZ         (ID_CmpGZ),
   .Cmp_GEZ        (ID_CmpGEZ),
   .Cmp_LZ         (ID_CmpLZ),
   .Cmp_LEZ        (ID_CmpLEZ),
   .IF_Flush       (IF_Flush),
   .DP_Hazards     (ID_DP_Hazards),
   .PCSrc          (ID_PCSrc),
   .SignExtend     (ID_SignExtend),
   .Link           (ID_Link),
   .Movn           (ID_Movn),
   .Movz           (ID_Movz),
   .Mfc0           (ID_Mfc0),
   .Mtc0           (ID_Mtc0),
   .CP1            (ID_CP1),
   .CP2            (ID_CP2),
   .CP3            (ID_CP3),
   .Eret           (ID_Eret),
   .Trap           (ID_Trap),
   .TrapCond       (ID_TrapCond),
   .EXC_Sys        (ID_EXC_Sys),
   .EXC_Bp         (ID_EXC_Bp),
   .EXC_RI         (ID_EXC_RI),
   .ID_CanErr      (ID_ID_CanErr),
   .EX_CanErr      (ID_EX_CanErr),
   .M_CanErr       (ID_M_CanErr),
   .NextIsDelay    (ID_NextIsDelay),
   .RegDst         (ID_RegDst),
   .ALUSrcImm      (ID_ALUSrcImm),
```

```
   .ALUOp          (ID_ALUOp),
   .LLSC           (ID_LLSC),
   .MemWrite       (ID_MemWrite),
   .MemRead        (ID_MemRead),
   .MemByte        (ID_MemByte),
   .MemHalf        (ID_MemHalf),
   .MemSignExtend  (ID_MemSignExtend),
   .Left           (ID_Left),
   .Right          (ID_Right),
   .RegWrite       (ID_RegWrite),
   .MemtoReg       (ID_MemtoReg)
);

/*** Hazard and Forward Control Unit ***/
Hazard_Detection HazardControl (
   .DP_Hazards          (HAZ_DP_Hazards),
   .ID_Rs               (Rs),
   .ID_Rt               (Rt),
   .EX_Rs               (EX_Rs),
   .EX_Rt               (EX_Rt),
   .EX_RtRd             (EX_RtRd),
   .MEM_RtRd            (M_RtRd),
   .WB_RtRd             (WB_RtRd),
   .EX_Link             (EX_Link),
   .EX_RegWrite         (EX_RegWrite),
   .MEM_RegWrite        (M_RegWrite),
   .WB_RegWrite         (WB_RegWrite),
   .MEM_MemRead         (M_MemRead),
   .MEM_MemWrite        (M_MemWrite),
   .InstMem_Read        (InstMem_Read),
   .InstMem_Ready       (InstMem_Ready),
   .Mfc0                (ID_Mfc0),
   .IF_Exception_Stall  (IF_Exception_Stall),
   .ID_Exception_Stall  (ID_Exception_Stall),
   .EX_Exception_Stall  (EX_Exception_Stall),
   .EX_ALU_Stall        (EX_ALU_Stall),
   .M_Stall_Controller  (M_Stall_Controller),
   .IF_Stall            (IF_Stall),
   .ID_Stall            (ID_Stall),
   .EX_Stall            (EX_Stall),
   .M_Stall             (M_Stall),
   .WB_Stall            (WB_Stall),
   .ID_RsFwdSel         (ID_RsFwdSel),
   .ID_RtFwdSel         (ID_RtFwdSel),
   .EX_RsFwdSel         (EX_RsFwdSel),
   .EX_RtFwdSel         (EX_RtFwdSel),
   .M_WriteDataFwdSel   (M_WriteDataFwdSel)
);

/*** Coprocessor 0: Exceptions and Interrupts ***/
CPZero CP0 (
   .clock                    (clock),
   .Mfc0                     (ID_Mfc0),
   .Mtc0                     (ID_Mtc0),
   .IF_Stall                 (IF_Stall),
   .ID_Stall                 (ID_Stall),
   .COP1                     (ID_CP1),
   .COP2                     (ID_CP2),
   .COP3                     (ID_CP3),
   .ERET                     (ID_Eret),
   .Rd                       (Rd),
   .Sel                      (Cp0_Sel),
   .Reg_In                   (ID_ReadData2_End),
   .Reg_Out                  (CP0_RegOut),
   .KernelMode               (ID_KernelMode),
   .ReverseEndian            (ID_ReverseEndian),
   .Int                      (Interrupts),
   .reset                    (reset),
   .EXC_NMI                  (NMI),
```

```
.EXC_AdIF                (IF_EXC_AdIF),
.EXC_AdEL                (M_EXC_AdEL),
.EXC_AdES                (M_EXC_AdES),
.EXC_Ov                  (EX_EXC_Ov),
.EXC_Tr                  (M_EXC_Tr),
.EXC_Sys                 (ID_EXC_Sys),
.EXC_Bp                  (ID_EXC_Bp),
.EXC_RI                  (ID_EXC_RI),
.ID_RestartPC            (ID_RestartPC),
.EX_RestartPC            (EX_RestartPC),
.M_RestartPC             (M_RestartPC),
.ID_IsFlushed            (ID_IsFlushed),
.IF_IsBD                 (IF_IsBDS),
.ID_IsBD                 (ID_IsBDS),
.EX_IsBD                 (EX_IsBDS),
.M_IsBD                  (M_IsBDS),
.BadAddr_M               (M_ALUResult),
.BadAddr_IF              (IF_PCOut),
.ID_CanErr               (ID_CanErr),
.EX_CanErr               (EX_CanErr),
.M_CanErr                (M_M_CanErr),
.IF_Exception_Stall      (IF_Exception_Stall),
.ID_Exception_Stall      (ID_Exception_Stall),
.EX_Exception_Stall      (EX_Exception_Stall),
.M_Exception_Stall       (M_Exception_Stall),
.IF_Exception_Flush      (IF_Exception_Flush),
.ID_Exception_Flush      (ID_Exception_Flush),
.EX_Exception_Flush      (EX_Exception_Flush),
.M_Exception_Flush       (M_Exception_Flush),
.Exc_PC_Sel              (ID_PCSrc_Exc),
.Exc_PC_Out              (ID_ExceptionPC),
.reset_r                 (Vote_in[2]),
.Status_BEV              (Vote_in[3]),
.Status_NMI              (Vote_in[4]),
.Status_ERL              (Vote_in[5]),
.ErrorEPC                (Vote_in[37:6]),
.Count                   (Vote_in[69:38]),
.Compare                 (Vote_in[101:70]),
.Status_CU_0             (Vote_in[102]),
.Status_RE               (Vote_in[103]),
.Status_IM               (Vote_in[111:104]),
.Status_UM               (Vote_in[112]),
.Status_IE               (Vote_in[113]),
.Cause_IV                (Vote_in[114]),
.Cause_IP                (Vote_in[122:115]),
.Cause_BD                (Vote_in[123]),
.Cause_CE                (Vote_in[125:124]),
.Cause_ExcCode30         (Vote_in[129:126]),
.Status_EXL              (Vote_in[130]),
.EPC                     (Vote_in[162:131]),
.BadVAddr                (Vote_in[194:163]),
.vote_reset_r            (Vote_out[2]),
.vote_Status_BEV         (Vote_out[3]),
.vote_Status_NMI         (Vote_out[4]),
.vote_Status_ERL         (Vote_out[5]),
.vote_ErrorEPC           (Vote_out[37:6]),
.vote_Count              (Vote_out[69:38]),
.vote_Compare            (Vote_out[101:70]),
.vote_Status_CU_0        (Vote_out[102]),
.vote_Status_RE          (Vote_out[103]),
.vote_Status_IM          (Vote_out[111:104]),
.vote_Status_UM          (Vote_out[112]),
.vote_Status_IE          (Vote_out[113]),
.vote_Cause_IV           (Vote_out[114]),
.vote_Cause_IP           (Vote_out[122:115]),
.vote_Cause_BD           (Vote_out[123]),
.vote_Cause_CE           (Vote_out[125:124]),
.vote_Cause_ExcCode30    (Vote_out[129:126]),
.vote_Status_EXL         (Vote_out[130]),
```

```
   .vote_EPC                  (Vote_out[162:131]),
   .vote_BadVAddr             (Vote_out[194:163])
);

/*** PC Source Non-Exception Mux ***/
Mux4 #(.WIDTH(32)) PCSrcStd_Mux (
   .sel  (ID_PCSrc),
   .in0  (IF_PCAdd4),
   .in1  (ID_JumpAddress),
   .in2  (ID_BranchAddress),
   .in3  (ID_ReadData1_End),
   .out  (IF_PC_PreExc)
);

/*** PC Source Exception Mux ***/
Mux2 #(.WIDTH(32)) PCSrcExc_Mux (
   .sel  (ID_PCSrc_Exc),
   .in0  (IF_PC_PreExc),
   .in1  (ID_ExceptionPC),
   .out  (IF_PCIn)
);

/*** Program Counter (MIPS spec is 0xBFC00000 starting address) ***/
assign IF_PCOut = Vote_in[226:195];
Register #(.WIDTH(32), .INIT(`EXC_Vector_Base_Reset)) PC (
   .clock   (clock),
   .reset   (reset),
   .enable  (~(IF_Stall | ID_Stall)),
   .D       (IF_PCIn),
   .Q       (IF_PCOut),
   .vote_Q  (Vote_out[226:195])
);

/*** PC +4 Adder ***/
Add PC_Add4 (
   .A  (IF_PCOut),
   .B  (32'h00000004),
   .C  (IF_PCAdd4)
);

/*** Instruction Fetch -> Instruction Decode Stage Register ***/
assign Instruction = Vote_in[258:227];
assign ID_PCAdd4 = Vote_in[290:259];
assign ID_IsBDS = Vote_in[291];
sign ID_RestartPC = Vote_in[323:292];
assign ID_IsFlushed = Vote_in[324];
IFID_Stage IFID (
   .clock                     (clock),
   .reset                     (reset),
   .IF_Flush                  (IF_Exception_Flush | IF_Flush),
   .IF_Stall                  (IF_Stall),
   .ID_Stall                  (ID_Stall),
   .IF_Instruction            (IF_Instruction),
   .IF_PCAdd4                 (IF_PCAdd4),
   .IF_PC                     (IF_PCOut),
   .IF_IsBDS                  (IF_IsBDS),
   .ID_Instruction            (Instruction),
   .ID_PCAdd4                 (ID_PCAdd4),
   .ID_IsBDS                  (ID_IsBDS),
   .ID_RestartPC              (ID_RestartPC),
   .ID_IsFlushed              (ID_IsFlushed),
   .vote_ID_Instruction       (Vote_out[258:227]),
   .vote_ID_PCAdd4            (Vote_out[290:259]),
   .vote_ID_IsBDS             (Vote_out[291]),
   .vote_ID_RestartPC         (Vote_out[323:292]),
   .vote_ID_IsFlushed         (Vote_out[324])
);

/*** Register File ***/
```

```
RegisterFile RegisterFile (
   .clock             (clock),
   .reset             (reset),
   .ReadReg1          (Rs),
   .ReadReg2          (Rt),
   .WriteReg          (WB_RtRd),
   .WriteData         (WB_WriteData),
   .RegWrite          (WB_RegWrite),
   .ReadData1         (ID_ReadData1_RF),
   .ReadData2         (ID_ReadData2_RF),
   .registers_in      (Vote_in[1316:325]),
   .registers_out     (Vote_out[1316:325])
);

/*** ID Rs Forwarding/Link Mux ***/
Mux4 #(.WIDTH(32)) IDRsFwd_Mux (
   .sel  (ID_RsFwdSel),
   .in0  (ID_ReadData1_RF),
   .in1  (M_ALUResult),
   .in2  (WB_WriteData),
   .in3  (32'h00000000),
   .out  (ID_ReadData1_End)
);

/*** ID Rt Forwarding/CP0 Mfc0 Mux ***/
Mux4 #(.WIDTH(32)) IDRtFwd_Mux (
   .sel  (ID_RtFwdSel),
   .in0  (ID_ReadData2_RF),
   .in1  (M_ALUResult),
   .in2  (WB_WriteData),
   .in3  (CP0_RegOut),
   .out  (ID_ReadData2_End)
);

/*** Condition Compare Unit ***/
Compare Compare (
   .A    (ID_ReadData1_End),
   .B    (ID_ReadData2_End),
   .EQ   (ID_CmpEQ),
   .GZ   (ID_CmpGZ),
   .LZ   (ID_CmpLZ),
   .GEZ  (ID_CmpGEZ),
   .LEZ  (ID_CmpLEZ)
);

/*** Branch Address Adder ***/
Add BranchAddress_Add (
   .A  (ID_PCAdd4),
   .B  (ID_ImmLeftShift2),
   .C  (ID_BranchAddress)
);

/*** Instruction Decode -> Execute Pipeline Stage ***/
assign EX_Link = Vote_in[1317];
assign EX_ALUSrcImm = Vote_in[1319];
assign EX_ALUOp = Vote_in[1324:1320];
assign EX_Movn = Vote_in[1325];
assign EX_Movz = Vote_in[1326];
assign EX_LLSC = Vote_in[1327];
assign EX_MemRead = Vote_in[1328];
assign EX_MemWrite = Vote_in[1329];
assign EX_MemByte = Vote_in[1330];
assign EX_MemHalf = Vote_in[1331];
assign EX_MemSignExtend = Vote_in[1332];
assign EX_Left = Vote_in[1333];
assign EX_Right = Vote_in[1334];
assign EX_RegWrite = Vote_in[1335];
assign EX_MemtoReg = Vote_in[1336];
assign EX_ReverseEndian = Vote_in[1337];
```

```
assign EX_RestartPC = Vote_in[1369:1338];
assign EX_IsBDS = Vote_in[1370];
assign EX_Trap = Vote_in[1371];
assign EX_TrapCond = Vote_in[1372];
assign EX_EX_CanErr = Vote_in[1373];
assign EX_M_CanErr = Vote_in[1374];
assign EX_ReadData1_PR = Vote_in[1406:1375];
assign EX_ReadData2_PR = Vote_in[1438:1407];
assign EX_Rs = Vote_in[1460:1456];
assign EX_Rt = Vote_in[1465:1461];
assign EX_WantRsByEX = Vote_in[1466];
assign EX_NeedRsByEX = Vote_in[1467];
assign EX_WantRtByEX = Vote_in[1468];
assign EX_NeedRtByEX = Vote_in[1469];
assign EX_KernelMode = Vote_in[1470];

IDEX_Stage IDEX (
   .clock                    (clock),
   .reset                    (reset),
   .ID_Flush                 (ID_Exception_Flush),
   .ID_Stall                 (ID_Stall),
   .EX_Stall                 (EX_Stall),
   .ID_Link                  (ID_Link),
   .ID_RegDst                (ID_RegDst),
   .ID_ALUSrcImm             (ID_ALUSrcImm),
   .ID_ALUOp                 (ID_ALUOp),
   .ID_Movn                  (ID_Movn),
   .ID_Movz                  (ID_Movz),
   .ID_LLSC                  (ID_LLSC),
   .ID_MemRead               (ID_MemRead),
   .ID_MemWrite              (ID_MemWrite),
   .ID_MemByte               (ID_MemByte),
   .ID_MemHalf               (ID_MemHalf),
   .ID_MemSignExtend         (ID_MemSignExtend),
   .ID_Left                  (ID_Left),
   .ID_Right                 (ID_Right),
   .ID_RegWrite              (ID_RegWrite),
   .ID_MemtoReg              (ID_MemtoReg),
   .ID_ReverseEndian         (ID_ReverseEndian),
   .ID_Rs                    (Rs),
   .ID_Rt                    (Rt),
   .ID_WantRsByEX            (ID_DP_Hazards[3]),
   .ID_NeedRsByEX            (ID_DP_Hazards[2]),
   .ID_WantRtByEX            (ID_DP_Hazards[1]),
   .ID_NeedRtByEX            (ID_DP_Hazards[0]),
   .ID_KernelMode            (ID_KernelMode),
   .ID_RestartPC             (ID_RestartPC),
   .ID_IsBDS                 (ID_IsBDS),
   .ID_Trap                  (ID_Trap),
   .ID_TrapCond              (ID_TrapCond),
   .ID_EX_CanErr             (ID_EX_CanErr),
   .ID_M_CanErr              (ID_M_CanErr),
   .ID_ReadData1             (ID_ReadData1_End),
   .ID_ReadData2             (ID_ReadData2_End),
   .ID_SignExtImm            (ID_SignExtImm[16:0]),
   .EX_Link                  (EX_Link),
   .EX_RegDst                (Vote_in[1318]),
   .EX_LinkRegDst            (EX_LinkRegDst),
   .EX_ALUSrcImm             (EX_ALUSrcImm),
   .EX_ALUOp                 (EX_ALUOp),
   .EX_Movn                  (EX_Movn),
   .EX_Movz                  (EX_Movz),
   .EX_LLSC                  (EX_LLSC),
   .EX_MemRead               (EX_MemRead),
   .EX_MemWrite              (EX_MemWrite),
   .EX_MemByte               (EX_MemByte),
   .EX_MemHalf               (EX_MemHalf),
   .EX_MemSignExtend         (EX_MemSignExtend),
   .EX_Left                  (EX_Left),
```

```
   .EX_Right                (EX_Right),
   .EX_RegWrite             (EX_RegWrite),
   .EX_MemtoReg             (EX_MemtoReg),
   .EX_ReverseEndian        (EX_ReverseEndian),
   .EX_RestartPC            (EX_RestartPC),
   .EX_IsBDS                (EX_IsBDS),
   .EX_Trap                 (EX_Trap),
   .EX_TrapCond             (EX_TrapCond),
   .EX_EX_CanErr            (EX_EX_CanErr),
   .EX_M_CanErr             (EX_M_CanErr),
   .EX_ReadData1            (EX_ReadData1_PR),
   .EX_ReadData2            (EX_ReadData2_PR),
   .EX_SignExtImm_pre       (Vote_in[1455:1439]),
   .EX_Rs                   (EX_Rs),
   .EX_Rt                   (EX_Rt),
   .EX_WantRsByEX           (EX_WantRsByEX),
   .EX_NeedRsByEX           (EX_NeedRsByEX),
   .EX_WantRtByEX           (EX_WantRtByEX),
   .EX_NeedRtByEX           (EX_NeedRtByEX),
   .EX_KernelMode           (EX_KernelMode),
   .EX_SignExtImm           (EX_SignExtImm),
   .EX_Rd                   (EX_Rd),
   .EX_Shamt                (EX_Shamt),
   .vote_EX_Link            (Vote_out[1317]),
   .vote_EX_RegDst          (Vote_out[1318]),
   .vote_EX_ALUSrcImm       (Vote_out[1319]),
   .vote_EX_ALUOp           (Vote_out[1324:1320]),
   .vote_EX_Movn            (Vote_out[1325]),
   .vote_EX_Movz            (Vote_out[1326]),
   .vote_EX_LLSC            (Vote_out[1327]),
   .vote_EX_MemRead         (Vote_out[1328]),
   .vote_EX_MemWrite        (Vote_out[1329]),
   .vote_EX_MemByte         (Vote_out[1330]),
   .vote_EX_MemHalf         (Vote_out[1331]),
   .vote_EX_MemSignExtend   (Vote_out[1332]),
   .vote_EX_Left            (Vote_out[1333]),
   .vote_EX_Right           (Vote_out[1334]),
   .vote_EX_RegWrite        (Vote_out[1335]),
   .vote_EX_MemtoReg        (Vote_out[1336]),
   .vote_EX_ReverseEndian   (Vote_out[1337]),
   .vote_EX_RestartPC       (Vote_out[1369:1338]),
   .vote_EX_IsBDS           (Vote_out[1370]),
   .vote_EX_Trap            (Vote_out[1371]),
   .vote_EX_TrapCond        (Vote_out[1372]),
   .vote_EX_EX_CanErr       (Vote_out[1373]),
   .vote_EX_M_CanErr        (Vote_out[1374]),
   .vote_EX_ReadData1       (Vote_out[1406:1375]),
   .vote_EX_ReadData2       (Vote_out[1438:1407]),
   .vote_EX_SignExtImm_pre  (Vote_out[1455:1439]),
   .vote_EX_Rs              (Vote_out[1460:1456]),
   .vote_EX_Rt              (Vote_out[1465:1461]),
   .vote_EX_WantRsByEX      (Vote_out[1466]),
   .vote_EX_NeedRsByEX      (Vote_out[1467]),
   .vote_EX_WantRtByEX      (Vote_out[1468]),
   .vote_EX_NeedRtByEX      (Vote_out[1469]),
   .vote_EX_KernelMode      (Vote_out[1470])
);

/*** EX Rs Forwarding Mux ***/
Mux4 #(.WIDTH(32)) EXRsFwd_Mux (
   .sel  (EX_RsFwdSel),
   .in0  (EX_ReadData1_PR),
   .in1  (M_ALUResult),
   .in2  (WB_WriteData),
   .in3  (EX_RestartPC),
   .out  (EX_ReadData1_Fwd)
);

/*** EX Rt Forwarding / Link Mux ***/
```

```
Mux4 #(.WIDTH(32)) EXRtFwdLnk_Mux (
   .sel  (EX_RtFwdSel),
   .in0  (EX_ReadData2_PR),
   .in1  (M_ALUResult),
   .in2  (WB_WriteData),
   .in3  (32'h00000008),
   .out  (EX_ReadData2_Fwd)
);

/*** EX ALU Immediate Mux ***/
Mux2 #(.WIDTH(32)) EXALUImm_Mux (
   .sel  (EX_ALUSrcImm),
   .in0  (EX_ReadData2_Fwd),
   .in1  (EX_SignExtImm),
   .out  (EX_ReadData2_Imm)
);

/*** EX RtRd / Link Mux ***/
Mux4 #(.WIDTH(5)) EXRtRdLnk_Mux (
   .sel  (EX_LinkRegDst),
   .in0  (EX_Rt),
   .in1  (EX_Rd),
   .in2  (5'b11111),
   .in3  (5'b00000),
   .out  (EX_RtRd)
);

/*** Arithmetic Logic Unit ***/
ALU ALU (
   .clock             (clock),
   .reset             (reset),
   .EX_Stall          (EX_Stall),
   .EX_Flush          (EX_Exception_Flush),
   .A                 (EX_ReadData1_Fwd),
   .B                 (EX_ReadData2_Imm),
   .Operation         (EX_ALUOp),
   .Shamt             (EX_Shamt),
   .Result            (EX_ALUResult),
   .BZero             (EX_BZero),
   .EXC_Ov            (EX_EXC_Ov),
   .ALU_Stall         (EX_ALU_Stall),
   .HILO              (Vote_in[1534:1471]),
   .div_fsm           (Vote_in[1535]),
   .vote_HILO         (Vote_out[1534:1471]),
   .vote_div_fsm      (Vote_out[1535]),
   .active            (Vote_in[1536]),
   .neg               (Vote_in[1537]),
   .div_result        (Vote_in[1569:1538]),
   .denom             (Vote_in[1601:1570]),
   .work              (Vote_in[1633:1602]),
   .vote_active       (Vote_out[1536]),
   .vote_neg          (Vote_out[1537]),
   .vote_div_result   (Vote_out[1569:1538]),
   .vote_denom        (Vote_out[1601:1570]),
   .vote_work         (Vote_out[1633:1602])
);

/*** Execute -> Memory Pipeline Stage ***/
assign M_RegWrite = Vote_in[1634];
assign M_MemtoReg = Vote_in[1635];
assign M_ReverseEndian = Vote_in[1636];
assign M_LLSC = Vote_in[1637];
assign M_MemRead = Vote_in[1638];
assign M_MemWrite = Vote_in[1639];
assign M_MemByte = Vote_in[1640];
assign M_MemHalf = Vote_in[1641];
assign M_MemSignExtend = Vote_in[1642];
assign M_Left = Vote_in[1643];
assign M_Right = Vote_in[1644];
```

```
assign M_KernelMode = Vote_in[1645];
assign M_RestartPC = Vote_in[1677:1646];
assign M_IsBDS = Vote_in[1678];
assign M_Trap = Vote_in[1679];
assign M_TrapCond = Vote_in[1680];
assign M_M_CanErr = Vote_in[1681];
assign M_ALUResult = Vote_in[1713:1682];
assign M_ReadData2_PR = Vote_in[1745:1714];
assign M_RtRd = Vote_in[1750:1746];

EXMEM_Stage EXMEM (
   .clock                    (clock),
   .reset                    (reset),
   .EX_Flush                 (EX_Exception_Flush),
   .EX_Stall                 (EX_Stall),
   .M_Stall                  (M_Stall),
   .EX_Movn                  (EX_Movn),
   .EX_Movz                  (EX_Movz),
   .EX_BZero                 (EX_BZero),
   .EX_RegWrite              (EX_RegWrite),
   .EX_MemtoReg              (EX_MemtoReg),
   .EX_ReverseEndian         (EX_ReverseEndian),
   .EX_LLSC                  (EX_LLSC),
   .EX_MemRead               (EX_MemRead),
   .EX_MemWrite              (EX_MemWrite),
   .EX_MemByte               (EX_MemByte),
   .EX_MemHalf               (EX_MemHalf),
   .EX_MemSignExtend         (EX_MemSignExtend),
   .EX_Left                  (EX_Left),
   .EX_Right                 (EX_Right),
   .EX_KernelMode            (EX_KernelMode),
   .EX_RestartPC             (EX_RestartPC),
   .EX_IsBDS                 (EX_IsBDS),
   .EX_Trap                  (EX_Trap),
   .EX_TrapCond              (EX_TrapCond),
   .EX_M_CanErr              (EX_M_CanErr),
   .EX_ALU_Result            (EX_ALUResult),
   .EX_ReadData2             (EX_ReadData2_Fwd),
   .EX_RtRd                  (EX_RtRd),
   .M_RegWrite               (M_RegWrite),
   .M_MemtoReg               (M_MemtoReg),
   .M_ReverseEndian          (M_ReverseEndian),
   .M_LLSC                   (M_LLSC),
   .M_MemRead                (M_MemRead),
   .M_MemWrite               (M_MemWrite),
   .M_MemByte                (M_MemByte),
   .M_MemHalf                (M_MemHalf),
   .M_MemSignExtend          (M_MemSignExtend),
   .M_Left                   (M_Left),
   .M_Right                  (M_Right),
   .M_KernelMode             (M_KernelMode),
   .M_RestartPC              (M_RestartPC),
   .M_IsBDS                  (M_IsBDS),
   .M_Trap                   (M_Trap),
   .M_TrapCond               (M_TrapCond),
   .M_M_CanErr               (M_M_CanErr),
   .M_ALU_Result             (M_ALUResult),
   .M_ReadData2              (M_ReadData2_PR),
   .M_RtRd                   (M_RtRd),
   .vote_M_RegWrite          (Vote_out[1634]),
   .vote_M_MemtoReg          (Vote_out[1635]),
   .vote_M_ReverseEndian     (Vote_out[1636]),
   .vote_M_LLSC              (Vote_out[1637]),
   .vote_M_MemRead           (Vote_out[1638]),
   .vote_M_MemWrite          (Vote_out[1639]),
   .vote_M_MemByte           (Vote_out[1640]),
   .vote_M_MemHalf           (Vote_out[1641]),
   .vote_M_MemSignExtend     (Vote_out[1642]),
   .vote_M_Left              (Vote_out[1643]),
```

```
   .vote_M_Right              (Vote_out[1644]),
   .vote_M_KernelMode         (Vote_out[1645]),
   .vote_M_RestartPC          (Vote_out[1677:1646]),
   .vote_M_IsBDS              (Vote_out[1678]),
   .vote_M_Trap               (Vote_out[1679]),
   .vote_M_TrapCond           (Vote_out[1680]),
   .vote_M_M_CanErr           (Vote_out[1681]),
   .vote_M_ALU_Result         (Vote_out[1713:1682]),
   .vote_M_ReadData2          (Vote_out[1745:1714]),
   .vote_M_RtRd               (Vote_out[1750:1746])
);

/*** Trap Detection Unit ***/
TrapDetect TrapDetect (
   .Trap       (M_Trap),
   .TrapCond   (M_TrapCond),
   .ALUResult  (M_ALUResult),
   .EXC_Tr     (M_EXC_Tr)
);

/*** MEM Write Data Mux ***/
Mux2 #(.WIDTH(32)) MWriteData_Mux (
   .sel  (M_WriteDataFwdSel),
   .in0  (M_ReadData2_PR),
   .in1  (WB_WriteData),
   .out  (M_WriteData_Pre)
);

/*** Data Memory Controller ***/
MemControl DataMem_Controller (
   .clock              (clock),
   .reset              (reset),
   .DataIn             (M_WriteData_Pre),
   .Address            (M_ALUResult),
   .MReadData          (DataMem_In),
   .MemRead            (M_MemRead),
   .MemWrite           (M_MemWrite),
   .DataMem_Ready      (DataMem_Ready),
   .Byte               (M_MemByte),
   .Half               (M_MemHalf),
   .SignExtend         (M_MemSignExtend),
   .KernelMode         (M_KernelMode),
   .ReverseEndian      (M_ReverseEndian),
   .LLSC               (M_LLSC),
   .ERET               (ID_Eret),
   .Left               (M_Left),
   .Right              (M_Right),
   .M_Exception_Stall  (M_Exception_Stall),
   .IF_Stall           (IF_Stall),
   .DataOut            (M_MemReadData),
   .MWriteData         (DataMem_Out),
   .WriteEnable        (DataMem_Write),
   .ReadEnable         (DataMem_Read),
   .M_Stall            (M_Stall_Controller),
   .EXC_AdEL           (M_EXC_AdEL),
   .EXC_AdES           (M_EXC_AdES),
   .LLSC_Address       (Vote_in[1780:1751]),
   .LLSC_Atomic        (Vote_in[1781]),
   .RW_Mask            (Vote_in[1782]),
   .vote_LLSC_Address  (Vote_out[1780:1751]),
   .vote_LLSC_Atomic   (Vote_out[1781])
);

/*** Memory -> Writeback Pipeline Stage ***/
assign WB_RegWrite = Vote_in[1783];
assign WB_MemtoReg = Vote_in[1784];
assign WB_ReadData = Vote_in[1816:1785];
assign WB_ALUResult = Vote_in[1848:1817];
assign WB_RtRd = Vote_in[1853:1849];
```

```
MEMWB_Stage MEMWB (
   .clock             (clock),
   .reset             (reset),
   .M_Flush           (M_Exception_Flush),
   .M_Stall           (M_Stall),
   .WB_Stall          (WB_Stall),
   .M_RegWrite        (M_RegWrite),
   .M_MemtoReg        (M_MemtoReg),
   .M_ReadData        (M_MemReadData),
   .M_ALU_Result      (M_ALUResult),
   .M_RtRd            (M_RtRd),
   .WB_RegWrite       (WB_RegWrite),
   .WB_MemtoReg       (WB_MemtoReg),
   .WB_ReadData       (WB_ReadData),
   .WB_ALU_Result     (WB_ALUResult),
   .WB_RtRd           (WB_RtRd),
   .vote_WB_RegWrite  (Vote_out[1783]),
   .vote_WB_MemtoReg  (Vote_out[1784]),
   .vote_WB_ReadData  (Vote_out[1816:1785]),
   .vote_WB_ALU_Result (Vote_out[1848:1817]),
   .vote_WB_RtRd      (Vote_out[1853:1849])
   );

/*** WB MemtoReg Mux ***/
Mux2 #(.WIDTH(32)) WBMemtoReg_Mux (
   .sel  (WB_MemtoReg),
   .in0  (WB_ALUResult),
   .in1  (WB_ReadData),
   .out  (WB_WriteData)
);

endmodule
```

(3) Control.v

```
`timescale 1ns / 1ps
/*
 * File         : Control.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0    7-Jun-2011  GEA       Initial design.
 *   2.0   26-May-2012  GEA       Release version with CP0.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   The Datapath Controller. This module sets the datapath control
 *   bits for an incoming instruction. These control bits follow the
 *   instruction through each pipeline stage as needed, and constitute
 *   the effective operation of the processor through each pipeline stage.
 */
module Control(
   input  ID_Stall,
   input  [5:0] OpCode,
   input  [5:0] Funct,
   input  [4:0] Rs,        // used to differentiate mfc0 and mtc0
   input  [4:0] Rt,        // used to differentiate bgez,bgezal,bltz,
                           // bltzal,teqi,tgei,tgeiu,tlti,tltiu,tnei
   input  Cmp_EQ,
   input  Cmp_GZ,
   input  Cmp_GEZ,
   input  Cmp_LZ,
   input  Cmp_LEZ,
   //------------
```

```
output IF_Flush,
output reg [7:0] DP_Hazards,
output [1:0] PCSrc,
output SignExtend,
output Link,
output Movn,
output Movz,
output Mfc0,
output Mtc0,
output CP1,
output CP2,
output CP3,
output Eret,
output Trap,
output TrapCond,
output EXC_Sys,
output EXC_Bp,
output EXC_RI,
output ID_CanErr,
output EX_CanErr,
output M_CanErr,
output NextIsDelay,
output RegDst,
output ALUSrcImm,
output reg [4:0] ALUOp,
output LLSC,
output MemWrite,
output MemRead,
output MemByte,
output MemHalf,
output MemSignExtend,
output Left,
output Right,
output RegWrite,
output MemtoReg
);

`include “MIPS_Parameters.v”

wire Movc;
wire Branch, Branch_EQ, Branch_GTZ, Branch_LEZ, Branch_NEQ, Branch_GEZ,
     Branch_LTZ;
wire Unaligned_Mem;
reg [15:0] Datapath;
assign PCSrc[0]      = Datapath[14];
assign Link          = Datapath[13];
assign ALUSrcImm     = Datapath[12];
assign Movc          = Datapath[11];
assign Trap          = Datapath[10];
assign TrapCond      = Datapath[9];
assign RegDst        = Datapath[8];
assign LLSC          = Datapath[7];
assign MemRead       = Datapath[6];
assign MemWrite      = Datapath[5];
assign MemHalf       = Datapath[4];
assign MemByte       = Datapath[3];
assign MemSignExtend = Datapath[2];
assign RegWrite      = Datapath[1];
assign MemtoReg      = Datapath[0];
reg [2:0] DP_Exceptions;
assign ID_CanErr = DP_Exceptions[2];
assign EX_CanErr = DP_Exceptions[1];
assign  M_CanErr = DP_Exceptions[0];
// Set the main datapath control signals based on the Op Code
always @(*) begin
   if (ID_Stall)
      Datapath <= `DP_None;
   else begin
      Case (OpCode)
```

```
// R-Type
`Op_Type_R : begin
   case (Funct)
      `Funct_Add     : Datapath <= `DP_Add;
      `Funct_Addu    : Datapath <= `DP_Addu;
      `Funct_And     : Datapath <= `DP_And;
      `Funct_Break   : Datapath <= `DP_Break;
      `Funct_Div     : Datapath <= `DP_Div;
      `Funct_Divu    : Datapath <= `DP_Divu;
      `Funct_Jalr    : Datapath <= `DP_Jalr;
      `Funct_Jr      : Datapath <= `DP_Jr;
      `Funct_Mfhi    : Datapath <= `DP_Mfhi;
      `Funct_Mflo    : Datapath <= `DP_Mflo;
      `Funct_Movn    : Datapath <= `DP_Movn;
      `Funct_Movz    : Datapath <= `DP_Movz;
      `Funct_Mthi    : Datapath <= `DP_Mthi;
      `Funct_Mtlo    : Datapath <= `DP_Mtlo;
      `Funct_Mult    : Datapath <= `DP_Mult;
      `Funct_Multu   : Datapath <= `DP_Multu;
      `Funct_Nor     : Datapath <= `DP_Nor;
      `Funct_Or      : Datapath <= `DP_Or;
      `Funct_Sll     : Datapath <= `DP_Sll;
      `Funct_Sllv    : Datapath <= `DP_Sllv;
      `Funct_Slt     : Datapath <= `DP_Slt;
      `Funct_Sltu    : Datapath <= `DP_Sltu;
      `Funct_Sra     : Datapath <= `DP_Sra;
      `Funct_Srav    : Datapath <= `DP_Srav;
      `Funct_Srl     : Datapath <= `DP_Srl;
      `Funct_Srlv    : Datapath <= `DP_Srlv;
      `Funct_Sub     : Datapath <= `DP_Sub;
      `Funct_Subu    : Datapath <= `DP_Subu;
      `Funct_Syscall : Datapath <= `DP_Syscall;
      `Funct_Teq     : Datapath <= `DP_Teq;
      `Funct_Tge     : Datapath <= `DP_Tge;
      `Funct_Tgeu    : Datapath <= `DP_Tgeu;
      `Funct_Tlt     : Datapath <= `DP_Tlt;
      `Funct_Tltu    : Datapath <= `DP_Tltu;
      `Funct_Tne     : Datapath <= `DP_Tne;
      `Funct_Xor     : Datapath <= `DP_Xor;
      default        : Datapath <= `DP_None;
   endcase
end
// R2-Type
`Op_Type_R2 : begin
   case (Funct)
      `Funct_Clo   : Datapath <= `DP_Clo;
      `Funct_Clz   : Datapath <= `DP_Clz;
      `Funct_Madd  : Datapath <= `DP_Madd;
      `Funct_Maddu : Datapath <= `DP_Maddu;
      `Funct_Msub  : Datapath <= `DP_Msub;
      `Funct_Msubu : Datapath <= `DP_Msubu;
      `Funct_Mul   : Datapath <= `DP_Mul;
      default      : Datapath <= `DP_None;
   endcase
end
// I-Type
`Op_Addi    : Datapath <= `DP_Addi;
`Op_Addiu   : Datapath <= `DP_Addiu;
`Op_Andi    : Datapath <= `DP_Andi;
`Op_Ori     : Datapath <= `DP_Ori;
`Op_Pref    : Datapath <= `DP_Pref;
`Op_Slti    : Datapath <= `DP_Slti;
`Op_Sltiu   : Datapath <= `DP_Sltiu;
`Op_Xori    : Datapath <= `DP_Xori;
// Jumps (using immediates)
`Op_J       : Datapath <= `DP_J;
`Op_Jal     : Datapath <= `DP_Jal;
// Branches and Traps
`Op_Type_BI : begin
```

```
            case (Rt)
               `OpRt_Bgez   : Datapath <= `DP_Bgez;
               `OpRt_Bgezal : Datapath <= `DP_Bgezal;
               `OpRt_Bltz   : Datapath <= `DP_Bltz;
               `OpRt_Bltzal : Datapath <= `DP_Bltzal;
               `OpRt_Teqi   : Datapath <= `DP_Teqi;
               `OpRt_Tgei   : Datapath <= `DP_Tgei;
               `OpRt_Tgeiu  : Datapath <= `DP_Tgeiu;
               `OpRt_Tlti   : Datapath <= `DP_Tlti;
               `OpRt_Tltiu  : Datapath <= `DP_Tltiu;
               `OpRt_Tnei   : Datapath <= `DP_Tnei;
               default      : Datapath <= `DP_None;
            endcase
         end
         `Op_Beq     : Datapath <= `DP_Beq;
         `Op_Bgtz    : Datapath <= `DP_Bgtz;
         `Op_Blez    : Datapath <= `DP_Blez;
         `Op_Bne     : Datapath <= `DP_Bne;
         // Coprocessor 0
         `Op_Type_CP0 : begin
            case (Rs)
               `OpRs_MF   : Datapath <= `DP_Mfc0;
               `OpRs_MT   : Datapath <= `DP_Mtc0;
               `OpRs_ERET : Datapath <= (Funct == `Funct_ERET) ? `DP_Eret :
                                        `DP_None;
               default    : Datapath <= `DP_None;
            endcase
         end
         // Memory
         `Op_Lb   : Datapath <= `DP_Lb;
         `Op_Lbu  : Datapath <= `DP_Lbu;
         `Op_Lh   : Datapath <= `DP_Lh;
         `Op_Lhu  : Datapath <= `DP_Lhu;
         `Op_Ll   : Datapath <= `DP_Ll;
         `Op_Lui  : Datapath <= `DP_Lui;
         `Op_Lw   : Datapath <= `DP_Lw;
         `Op_Lwl  : Datapath <= `DP_Lwl;
         `Op_Lwr  : Datapath <= `DP_Lwr;
         `Op_Sb   : Datapath <= `DP_Sb;
         `Op_Sc   : Datapath <= `DP_Sc;
         `Op_Sh   : Datapath <= `DP_Sh;
         `Op_Sw   : Datapath <= `DP_Sw;
         `Op_Swl  : Datapath <= `DP_Swl;
         `Op_Swr  : Datapath <= `DP_Swr;
         default  : Datapath <= `DP_None;
      endcase
   end
end

// Set the Hazard Control Signals and Exception Indicators based on the Op Code
   always @(*) begin
      case (OpCode)
         // R-Type
         `Op_Type_R : begin
            case (Funct)
               `Funct_Add : begin
                  DP_Hazards <= `HAZ_Add;
                  DP_Exceptions <= `EXC_Add;
               end
               `Funct_Addu : begin
                  DP_Hazards <= `HAZ_Addu;
                  DP_Exceptions <= `EXC_Addu;
               end
               `Funct_And : begin
                  DP_Hazards <= `HAZ_And;
                  DP_Exceptions <= `EXC_And;
               end
               `Funct_Break : begin
                  DP_Hazards <= `HAZ_Break;
```

```
   DP_Exceptions <= `EXC_Break;
end
`Funct_Div : begin
   DP_Hazards <= `HAZ_Div;
   DP_Exceptions <= `EXC_Div;
end
`Funct_Divu : begin
   DP_Hazards <= `HAZ_Divu;
   DP_Exceptions <= `EXC_Divu;
end
`Funct_Jalr : begin
   DP_Hazards <= `HAZ_Jalr;
   DP_Exceptions <= `EXC_Jalr;
end
`Funct_Jr : begin
   DP_Hazards <= `HAZ_Jr;
   DP_Exceptions <= `EXC_Jr;
end
`Funct_Mfhi : begin
   DP_Hazards <= `HAZ_Mfhi;
   DP_Exceptions <= `EXC_Mfhi;
end
`Funct_Mflo : begin
   DP_Hazards <= `HAZ_Mflo;
   DP_Exceptions <= `EXC_Mflo;
end
`Funct_Movn : begin
   DP_Hazards <= `HAZ_Movn;
   DP_Exceptions <= `EXC_Movn;
end
`Funct_Movz : begin
   DP_Hazards <= `HAZ_Movz;
   DP_Exceptions <= `EXC_Movz;
end
`Funct_Mthi : begin
   DP_Hazards <= `HAZ_Mthi;
   DP_Exceptions <= `EXC_Mthi;
end
`Funct_Mtlo : begin
   DP_Hazards <= `HAZ_Mtlo;
   DP_Exceptions <= `EXC_Mtlo;
end
‘Funct_Mult : begin
   DP_Hazards <= `HAZ_Mult;
   DP_Exceptions <= `EXC_Mult;
end
`Funct_Multu : begin
   DP_Hazards <= `HAZ_Multu;
   DP_Exceptions <= `EXC_Multu;
end
`Funct_Nor : begin
   DP_Hazards <= `HAZ_Nor;
   DP_Exceptions <= `EXC_Nor;
end
`Funct_Or : begin
   DP_Hazards <= `HAZ_Or;
   DP_Exceptions <= `EXC_Or;
end
`Funct_Sll : begin
   DP_Hazards <= `HAZ_Sll;
   DP_Exceptions <= `EXC_Sll;
 end
`Funct_Sllv : begin
   DP_Hazards <= `HAZ_Sllv;
   DP_Exceptions <= `EXC_Sllv;
end
`Funct_Slt : begin
   DP_Hazards <= `HAZ_Slt;
   DP_Exceptions <= `EXC_Slt;
```

```
      end
      `Funct_Sltu : begin
         DP_Hazards <= `HAZ_Sltu;
         DP_Exceptions <= `EXC_Sltu;
      end
      `Funct_Sra : begin
         DP_Hazards <= `HAZ_Sra;
         DP_Exceptions <= `EXC_Sra;
      end
      `Funct_Srav : begin
         DP_Hazards <= `HAZ_Srav;
         DP_Exceptions <= `EXC_Srav;
      end
      `Funct_Srl : begin
         DP_Hazards <= `HAZ_Srl;
         DP_Exceptions <= `EXC_Srl;
      end
      `Funct_Srlv : begin
         DP_Hazards <= `HAZ_Srlv;
         DP_Exceptions <= `EXC_Srlv;
      end
      `Funct_Sub : begin
         DP_Hazards <= `HAZ_Sub;
         DP_Exceptions <= `EXC_Sub;
      end
      `Funct_Subu : begin
         DP_Hazards <= `HAZ_Subu;
         DP_Exceptions <= `EXC_Subu;
      end
      `Funct_Syscall : begin
         DP_Hazards <= `HAZ_Syscall;
         DP_Exceptions <= `EXC_Syscall;
      end
      `Funct_Teq : begin
         DP_Hazards <= `HAZ_Teq;
         DP_Exceptions <= `EXC_Teq;
      end
      `Funct_Tge : begin
         DP_Hazards <= `HAZ_Tge;
         DP_Exceptions <= `EXC_Tge;
      end
      `Funct_Tgeu : begin
         DP_Hazards <= `HAZ_Tgeu;
         DP_Exceptions <= `EXC_Tgeu;
      end
      `Funct_Tlt : begin
         DP_Hazards <= `HAZ_Tlt;
         DP_Exceptions <= `EXC_Tlt;
      end
      `Funct_Tltu : begin
         DP_Hazards <= `HAZ_Tltu;
         DP_Exceptions <= `EXC_Tltu;
      end
      `Funct_Tne : begin
         DP_Hazards <= `HAZ_Tne;
         DP_Exceptions <= `EXC_Tne;
      end
      `Funct_Xor : begin
         DP_Hazards <= `HAZ_Xor;
         DP_Exceptions <= `EXC_Xor;
      end
      default : begin
         DP_Hazards <= 8'hxx;
         DP_Exceptions <= 3'bxxx;
      end
   endcase
end
// R2-Type
`Op_Type_R2 : begin
```

```
   case (Funct)
      `Funct_Clo : begin
         DP_Hazards <= `HAZ_Clo;
         DP_Exceptions <= `EXC_Clo;
      end
      `Funct_Clz : begin
         DP_Hazards <= `HAZ_Clz;
         DP_Exceptions <= `EXC_Clz;
      end
      `Funct_Madd : begin
         DP_Hazards <= `HAZ_Madd;
         DP_Exceptions <= `EXC_Madd;
      end
      `Funct_Maddu : begin
         DP_Hazards <= `HAZ_Maddu;
         DP_Exceptions <= `EXC_Maddu;
      end
      `Funct_Msub : begin
         DP_Hazards <= `HAZ_Msub;
         DP_Exceptions <= `EXC_Msub;
      end
      `Funct_Msubu : begin
         DP_Hazards <= `HAZ_Msubu;
         DP_Exceptions <= `EXC_Msubu;
      end
      `Funct_Mul : begin
         DP_Hazards <= `HAZ_Mul;
         DP_Exceptions <= `EXC_Mul;
      end
      default : begin
         DP_Hazards <= 8'hxx;
         DP_Exceptions <= 3'bxxx;
      end
   endcase
end
// I-Type
`Op_Addi : begin
   DP_Hazards <= `HAZ_Addi;
   DP_Exceptions <= `EXC_Addi;
end
`Op_Addiu : begin
   DP_Hazards <= `HAZ_Addiu;
   DP_Exceptions <= `EXC_Addiu;
end
`Op_Andi : begin
   DP_Hazards <= `HAZ_Andi;
   DP_Exceptions <= `EXC_Andi;
end
`Op_Ori : begin
   DP_Hazards <= `HAZ_Ori;
   DP_Exceptions <= `EXC_Ori;
end
`Op_Pref : begin
   DP_Hazards <= `HAZ_Pref;
   DP_Exceptions <= `EXC_Pref;
end
`Op_Slti : begin
   DP_Hazards <= `HAZ_Slti;
   DP_Exceptions <= `EXC_Slti;
end
`Op_Sltiu : begin
   DP_Hazards <= `HAZ_Sltiu;
   DP_Exceptions <= `EXC_Sltiu;
end
`Op_Xori : begin
   DP_Hazards <= `HAZ_Xori;
   DP_Exceptions <= `EXC_Xori;
end
// Jumps
```

```
`Op_J : begin
   DP_Hazards <= `HAZ_J;
   DP_Exceptions <= `EXC_J;
end
`Op_Jal : begin
   DP_Hazards <= `HAZ_Jal;
   DP_Exceptions <= `EXC_Jal;
end
// Branches and Traps
`Op_Type_BI : begin
   case (Rt)
      `OpRt_Bgez : begin
         DP_Hazards <= `HAZ_Bgez;
         DP_Exceptions <= `EXC_Bgez;
      end
      `OpRt_Bgezal : begin
         DP_Hazards <= `HAZ_Bgezal;
         DP_Exceptions <= `EXC_Bgezal;
      end
      `OpRt_Bltz : begin
         DP_Hazards <= `HAZ_Bltz;
         DP_Exceptions <= `EXC_Bltz;
      end
      `OpRt_Bltzal : begin
         DP_Hazards <= `HAZ_Bltzal;
         DP_Exceptions <= `EXC_Bltzal;
      end
      `OpRt_Teqi : begin
         DP_Hazards <= `HAZ_Teqi;
         DP_Exceptions <= `EXC_Teqi;
      end
      `OpRt_Tgei : begin
         DP_Hazards <= `HAZ_Tgei;
         DP_Exceptions <= `EXC_Tgei;
      end
      `OpRt_Tgeiu : begin
         DP_Hazards <= `HAZ_Tgeiu;
         DP_Exceptions <= `EXC_Tgeiu;
      end
      `OpRt_Tlti : begin
         DP_Hazards <= `HAZ_Tlti;
         DP_Exceptions <= `EXC_Tlti;
      end
      `OpRt_Tltiu : begin
         DP_Hazards <= `HAZ_Tltiu;
         DP_Exceptions <= `EXC_Tltiu;
      end
      `OpRt_Tnei : begin
         DP_Hazards <= `HAZ_Tnei;
         DP_Exceptions <= `EXC_Tnei;
      end
      default : begin
         DP_Hazards <= 8'hxx;
         DP_Exceptions <= 3'bxxx;
      end
   endcase
end
`Op_Beq : begin
   DP_Hazards <= `HAZ_Beq;
   DP_Exceptions <= `EXC_Beq;
end
`Op_Bgtz : begin
   DP_Hazards <= `HAZ_Bgtz;
   DP_Exceptions <= `EXC_Bgtz;
end
`Op_Blez : begin
   DP_Hazards <= `HAZ_Blez;
   DP_Exceptions <= `EXC_Blez;
end
```

```
`Op_Bne : begin
   DP_Hazards <= `HAZ_Bne;
   DP_Exceptions <= `EXC_Bne;
end
// Coprocessor 0
`Op_Type_CP0 : begin
   case (Rs)
      `OpRs_MF : begin
         DP_Hazards <= `HAZ_Mfc0;
         DP_Exceptions <= `EXC_Mfc0;
      end
      `OpRs_MT : begin
         DP_Hazards <= `HAZ_Mtc0;
         DP_Exceptions <= `EXC_Mtc0;
      end
      `OpRs_ERET : begin
         DP_Hazards <= (Funct == `Funct_ERET) ? `HAZ_Eret : 8'hxx;
         DP_Exceptions <= `EXC_Eret;
      end
      default : begin
         DP_Hazards <= 8'hxx;
         DP_Exceptions <= 3'bxxx;
      end
   endcase
end
// Memory
`Op_Lb : begin
   DP_Hazards <= `HAZ_Lb;
   DP_Exceptions <= `EXC_Lb;
end
`Op_Lbu : begin
   DP_Hazards <= `HAZ_Lbu;
   DP_Exceptions <= `EXC_Lbu;
end
`Op_Lh : begin
   DP_Hazards <= `HAZ_Lh;
   DP_Exceptions <= `EXC_Lh;
end
`Op_Lhu : begin
   DP_Hazards <= `HAZ_Lhu;
   DP_Exceptions <= `EXC_Lhu;
end
`Op_Ll : begin
   DP_Hazards <= `HAZ_Ll;
   DP_Exceptions <= `EXC_Ll;
end
`Op_Lui : begin
   DP_Hazards <= `HAZ_Lui;
   DP_Exceptions <= `EXC_Lui;
end
`Op_Lw : begin
   DP_Hazards <= `HAZ_Lw;
   DP_Exceptions <= `EXC_Lw;
end
`Op_Lwl : begin
   DP_Hazards <= `HAZ_Lwl;
   DP_Exceptions <= `EXC_Lwl;
end
`Op_Lwr : begin
   DP_Hazards <= `HAZ_Lwr;
   DP_Exceptions <= `EXC_Lwr;
end
`Op_Sb : begin
   DP_Hazards <= `HAZ_Sb;
   DP_Exceptions <= `EXC_Sb;
end
`Op_Sc : begin
   DP_Hazards <= `HAZ_Sc;
   DP_Exceptions <= `EXC_Sc;
```

```
      end
      `Op_Sh : begin
         DP_Hazards <= `HAZ_Sh;
         DP_Exceptions <= `EXC_Sh;
      end
      `Op_Sw : begin
         DP_Hazards <= `HAZ_Sw;
         DP_Exceptions <= `EXC_Sw;
      end
      `Op_Swl : begin
         DP_Hazards <= `HAZ_Swl;
         DP_Exceptions <= `EXC_Swl;
      end
      `Op_Swr : begin
         DP_Hazards <= `HAZ_Swr;
         DP_Exceptions <= `EXC_Swr;
      end
      default : begin
         DP_Hazards <= 8'hxx;
         DP_Exceptions <= 3'bxxx;
      end
   endcase
end

// ALU Assignment
always @(*) begin
   if (ID_Stall)
      ALUOp <= `AluOp_Addu;//Any Op that doesn't write HILO or cause exceptions
   else begin
      case (OpCode)
         `Op_Type_R : begin
            case (Funct)
               `Funct_Add     : ALUOp <= `AluOp_Add;
               `Funct_Addu    : ALUOp <= `AluOp_Addu;
               `Funct_And     : ALUOp <= `AluOp_And;
               `Funct_Div     : ALUOp <= `AluOp_Div;
               `Funct_Divu    : ALUOp <= `AluOp_Divu;
               `Funct_Jalr    : ALUOp <= `AluOp_Addu;
               `Funct_Mfhi    : ALUOp <= `AluOp_Mfhi;
               `Funct_Mflo    : ALUOp <= `AluOp_Mflo;
               `Funct_Movn    : ALUOp <= `AluOp_Addu;
               `Funct_Movz    : ALUOp <= `AluOp_Addu;
               `Funct_Mthi    : ALUOp <= `AluOp_Mthi;
               `Funct_Mtlo    : ALUOp <= `AluOp_Mtlo;
               `Funct_Mult    : ALUOp <= `AluOp_Mult;
               `Funct_Multu   : ALUOp <= `AluOp_Multu;
               `Funct_Nor     : ALUOp <= `AluOp_Nor;
               `Funct_Or      : ALUOp <= `AluOp_Or;
               `Funct_Sll     : ALUOp <= `AluOp_Sll;
               `Funct_Sllv    : ALUOp <= `AluOp_Sllv;
               `Funct_Slt     : ALUOp <= `AluOp_Slt;
               `Funct_Sltu    : ALUOp <= `AluOp_Sltu;
               `Funct_Sra     : ALUOp <= `AluOp_Sra;
               `Funct_Srav    : ALUOp <= `AluOp_Srav;
               `Funct_Srl     : ALUOp <= `AluOp_Srl;
               `Funct_Srlv    : ALUOp <= `AluOp_Srlv;
               `Funct_Sub     : ALUOp <= `AluOp_Sub;
               `Funct_Subu    : ALUOp <= `AluOp_Subu;
               `Funct_Syscall : ALUOp <= `AluOp_Addu;
               `Funct_Teq     : ALUOp <= `AluOp_Subu;
               `Funct_Tge     : ALUOp <= `AluOp_Slt;
               `Funct_Tgeu    : ALUOp <= `AluOp_Sltu;
               `Funct_Tlt     : ALUOp <= `AluOp_Slt;
               `Funct_Tltu    : ALUOp <= `AluOp_Sltu;
               `Funct_Tne     : ALUOp <= `AluOp_Subu;
               `Funct_Xor     : ALUOp <= `AluOp_Xor;
               default        : ALUOp <= `AluOp_Addu;
            endcase
         end
```

```
          `Op_Type_R2 : begin
             case (Funct)
                `Funct_Clo   : ALUOp <= `AluOp_Clo;
                `Funct_Clz   : ALUOp <= `AluOp_Clz;
                `Funct_Madd  : ALUOp <= `AluOp_Madd;
                `Funct_Maddu : ALUOp <= `AluOp_Maddu;
                `Funct_Msub  : ALUOp <= `AluOp_Msub;
                `Funct_Msubu : ALUOp <= `AluOp_Msubu;
                `Funct_Mul   : ALUOp <= `AluOp_Mul;
                default      : ALUOp <= `AluOp_Addu;
             endcase
          end
          `Op_Type_BI : begin
             case (Rt)
                `OpRt_Teqi   : ALUOp <= `AluOp_Subu;
                `OpRt_Tgei   : ALUOp <= `AluOp_Slt;
                `OpRt_Tgeiu  : ALUOp <= `AluOp_Sltu;
                `OpRt_Tlti   : ALUOp <= `AluOp_Slt;
                `OpRt_Tltiu  : ALUOp <= `AluOp_Sltu;
                `OpRt_Tnei   : ALUOp <= `AluOp_Subu;
                default      : ALUOp <= `AluOp_Addu;  // Branches don't matter.
             endcase
          end
          `Op_Type_CP0 : ALUOp <= `AluOp_Addu;
          `Op_Addi     : ALUOp <= `AluOp_Add;
          `Op_Addiu    : ALUOp <= `AluOp_Addu;
          `Op_Andi     : ALUOp <= `AluOp_And;
          `Op_Jal      : ALUOp <= `AluOp_Addu;
          `Op_Lb       : ALUOp <= `AluOp_Addu;
          `Op_Lbu      : ALUOp <= `AluOp_Addu;
          `Op_Lh       : ALUOp <= `AluOp_Addu;
          `Op_Lhu      : ALUOp <= `AluOp_Addu;
          `Op_Ll       : ALUOp <= `AluOp_Addu;
          `Op_Lui      : ALUOp <= `AluOp_Sllc;
          `Op_Lw       : ALUOp <= `AluOp_Addu;
          `Op_Lwl      : ALUOp <= `AluOp_Addu;
          `Op_Lwr      : ALUOp <= `AluOp_Addu;
          `Op_Ori      : ALUOp <= `AluOp_Or;
          `Op_Sb       : ALUOp <= `AluOp_Addu;
          `Op_Sc       : ALUOp <= `AluOp_Addu;  // XXX Needs HW implement
          `Op_Sh       : ALUOp <= `AluOp_Addu;
          `Op_Slti     : ALUOp <= `AluOp_Slt;
          `Op_Sltiu    : ALUOp <= `AluOp_Sltu;
          `Op_Sw       : ALUOp <= `AluOp_Addu;
          `Op_Swl      : ALUOp <= `AluOp_Addu;
          `Op_Swr      : ALUOp <= `AluOp_Addu;
          `Op_Xori     : ALUOp <= `AluOp_Xor;
          default      : ALUOp <= `AluOp_Addu;
       endcase
    end
end

/***
 These remaining options cover portions of the datapath that are not
 controlled directly by the datapath bits. Note that some refer to bits of
 the opcode or other fields, which breaks the otherwise fully-abstracted view
 of instruction encodings. Make sure when adding custom instructions that
 no false positives/negatives are generated here.
 ***/

// Branch Detection: Options are mutually exclusive.
assign Branch_EQ  =  OpCode[2] & ~OpCode[1] & ~OpCode[0] &  Cmp_EQ;
assign Branch_GTZ =  OpCode[2] &  OpCode[1] &  OpCode[0] &  Cmp_GZ;
assign Branch_LEZ =  OpCode[2] &  OpCode[1] & ~OpCode[0] &  Cmp_LEZ;
assign Branch_NEQ =  OpCode[2] & ~OpCode[1] &  OpCode[0] & ~Cmp_EQ;
assign Branch_GEZ = ~OpCode[2] &  Rt[0] & Cmp_GEZ;
assign Branch_LTZ = ~OpCode[2] & ~Rt[0] & Cmp_LZ;

assign Branch = Branch_EQ | Branch_GTZ | Branch_LEZ | Branch_NEQ | Branch_GEZ |
```

```
                  Branch_LTZ;
   assign PCSrc[1] = (Datapath[15] & ~Datapath[14]) ? Branch : Datapath[15];

/* In MIPS32, all Branch and Jump operations execute the Branch Delay Slot,
 * or next instruction, regardless if the branch is taken or not. The exception
 * is the “Branch Likely” instruction group. These are deprecated, however, and
 * not implemented here. “IF_Flush” is defined to allow for the cancelation of a
 * Branch Delay Slot should these be implemented later.
 */
   assign IF_Flush = 0;

   // Indicator that next instruction is a Branch Delay Slot.
   assign NextIsDelay = Datapath[15] | Datapath[14];

   // Sign- or Zero-Extension Control. The only ops that require zero-extension
   // are Andi, Ori, and Xori. The following also zero-extends ‘lui’, however it
   // does not alter the effect of lui.
   assign SignExtend = (OpCode[5:2] != 4’b0011);

   // Move Conditional
   assign Movn = Movc &  Funct[0];
   assign Movz = Movc & ~Funct[0];

   // Coprocessor 0 (Mfc0, Mtc0) control signals.
   assign Mfc0 = ((OpCode == `Op_Type_CP0) && (Rs == `OpRs_MF));
   assign Mtc0 = ((OpCode == `Op_Type_CP0) && (Rs == `OpRs_MT));
   assign Eret = ((OpCode == `Op_Type_CP0) && (Rs == `OpRs_ERET) && (Funct ==
                 `Funct_ERET));

   // Coprocessor 1,2,3 accesses (not implemented)
   assign CP1 = (OpCode == `Op_Type_CP1);
   assign CP2 = (OpCode == `Op_Type_CP2);
   assign CP3 = (OpCode == `Op_Type_CP3);

   // Exceptions found in ID
   assign EXC_Sys = ((OpCode == `Op_Type_R) && (Funct == `Funct_Syscall));
   assign EXC_Bp  = ((OpCode == `Op_Type_R) && (Funct == `Funct_Break));

   // Unaligned Memory Accesses (lwl, lwr, swl, swr)
   assign Unaligned_Mem = OpCode[5] & ~OpCode[4] & OpCode[1] & ~OpCode[0];
   assign Left  = Unaligned_Mem & ~OpCode[2];
   assign Right = Unaligned_Mem &  OpCode[2];

   // TODO: Reserved Instruction Exception must still be implemented
   assign EXC_RI  = 0;

endmodule
```

(4) Hazard_Detection.v

```
`timescale 1ns / 1ps
/*
 * File         : Hazard_Detection.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   23-Jul-2011  GEA       Initial design.
 *   2.0   26-May-2012  GEA       Release version with CP0.
 *   2.01   1-Nov-2012  GEA       Fixed issue with Jal.
 *   3.0    1-Mar-2016  ASJ       Modified IF_Stall for L1 Cache
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Hazard Detection and Forward Control. This is the glue that allows a
```

```
 *   pipelined processor to operate efficiently and correctly in the presence
 *   of data, structural, and control hazards. For each pipeline stage, it
 *   detects whether that stage requires data that is still in the pipeline,
 *   and whether that data may be forwarded or if the pipeline must be stalled.
 *
 *   This module is heavily commented. Read below for more information.
 */
module Hazard_Detection(
    input  [7:0] DP_Hazards,
    input  [4:0] ID_Rs,
    input  [4:0] ID_Rt,
    input  [4:0] EX_Rs,
    input  [4:0] EX_Rt,
    input  [4:0] EX_RtRd,
    input  [4:0] MEM_RtRd,
    input  [4:0] WB_RtRd,
    input  EX_Link,
    input  EX_RegWrite,
    input  MEM_RegWrite,
    input  WB_RegWrite,
    input  MEM_MemRead,
    input  MEM_MemWrite,// Needed for Store Conditional which writes to a register
    input  InstMem_Read,
    input  InstMem_Ready,
    input  Mfc0,                // Using fwd mux; not part of haz/fwd.
    input  IF_Exception_Stall,
    input  ID_Exception_Stall,
    input  EX_Exception_Stall,
    input  EX_ALU_Stall,
    input  M_Stall_Controller,  // Determined by data memory controller
    output IF_Stall,
    output ID_Stall,
    output EX_Stall,
    output M_Stall,
    output WB_Stall,
    output [1:0] ID_RsFwdSel,
    output [1:0] ID_RtFwdSel,
    output [1:0] EX_RsFwdSel,
    output [1:0] EX_RtFwdSel,
    output M_WriteDataFwdSel
    );

/* Hazard and Forward Detection
 *
 * Most instructions read from one or more registers. Normally this occurs in
 * the ID stage. However, frequently the register file in the ID stage is stale
 * when one or more forward stages in the pipeline (EX, MEM, or WB) contains
 * an instruction which will eventually update it but has not yet done so.
 *
 * A hazard condition is created when a forward pipeline stage is set to write
 * the same register that a current pipeline stage (e.g., in ID) needs to read.
 * The solution is to stall the current stage (and effectively all stages behind
 * it) or bypass (forward) the data from forward stages. Fortunately forwarding
 * works for most combinations of instructions.
 *
 * Hazard and Forward conditions are handled based on two simple rules:
 * “Wants” and “Needs.” If an instruction “wants” data in a certain pipeline
 * stage, and that data is available further along in the pipeline, it will
 * be forwarded. If it “needs” data and the data is not yet available for
 * forwarding, the pipeline stage stalls. If it does not want or need data in a
 * certain stage, forwarding is disabled and a stall will not occur. This is
 * important for instructions which insert custom data, such as jal or movz.
 *
 * Currently, “Want” and “Need” conditions are defined for both Rs data and Rt
 * data (the two read registers in MIPS), and these conditions exist in the
 * ID and EX pipeline stages. This is a total of eight condition bits.
 *
 * A unique exception exists with Store instructions, which don’t need the
 * “Rt” data until the MEM stage. Because data doesn’t change in WB, and WB
```

```
* is the only stage following MEM, forwarding is *always* possible from
* WB to Mem. This unit handles this situation, and a condition bit is not
* needed.
*
* When data is needed from the MEM stage by a previous stage (ID or EX), the
* decision to forward or stall is based on whether MEM is accessing memory
* (stall) or not (forward). Normally store instructions don't write to registers
* and thus are never needed for a data dependence, so the signal 'MEM_MemRead'
* is sufficient to determine. Because of the Store Conditional instruction,
* however, 'MEM_MemWrite' must also be considered because it writes to a
* register.
*/

  wire WantRsByID, NeedRsByID, WantRtByID, NeedRtByID, WantRsByEX, NeedRsByEX,
       WantRtByEX, NeedRtByEX;
  assign WantRsByID = DP_Hazards[7];
  assign NeedRsByID = DP_Hazards[6];
  assign WantRtByID = DP_Hazards[5];
  assign NeedRtByID = DP_Hazards[4];
  assign WantRsByEX = DP_Hazards[3];
  assign NeedRsByEX = DP_Hazards[2];
  assign WantRtByEX = DP_Hazards[1];
  assign NeedRtByEX = DP_Hazards[0];

  // Trick allowed by RegDst = 0 which gives Rt. MEM_Rt is only used on Data
  // Memory write operations (stores), and RegWrite is always 0 in this case.
  wire [4:0] MEM_Rt = MEM_RtRd;

  // Forwarding should not happen when the src/dst register is $zero
  wire EX_RtRd_NZ  = (EX_RtRd  != 5'b00000);
  wire MEM_RtRd_NZ = (MEM_RtRd != 5'b00000);
  wire WB_RtRd_NZ  = (WB_RtRd  != 5'b00000);

  // ID Dependencies
  wire Rs_IDEX_Match  = (ID_Rs == EX_RtRd)  & EX_RtRd_NZ  & (WantRsByID |
                        NeedRsByID) & EX_RegWrite;
  wire Rt_IDEX_Match  = (ID_Rt == EX_RtRd)  & EX_RtRd_NZ  & (WantRtByID |
                        NeedRtByID) & EX_RegWrite;
  wire Rs_IDMEM_Match = (ID_Rs == MEM_RtRd) & MEM_RtRd_NZ & (WantRsByID |
                        NeedRsByID) & MEM_RegWrite;
  wire Rt_IDMEM_Match = (ID_Rt == MEM_RtRd) & MEM_RtRd_NZ & (WantRtByID |
                        NeedRtByID) & MEM_RegWrite;
  wire Rs_IDWB_Match  = (ID_Rs == WB_RtRd)  & WB_RtRd_NZ  & (WantRsByID |
                        NeedRsByID) & WB_RegWrite;
  wire Rt_IDWB_Match  = (ID_Rt == WB_RtRd)  & WB_RtRd_NZ  & (WantRtByID |
                        NeedRtByID) & WB_RegWrite;
  // EX Dependencies
  wire Rs_EXMEM_Match = (EX_Rs == MEM_RtRd) & MEM_RtRd_NZ & (WantRsByEX |
                        NeedRsByEX) & MEM_RegWrite;
  wire Rt_EXMEM_Match = (EX_Rt == MEM_RtRd) & MEM_RtRd_NZ & (WantRtByEX |
                        NeedRtByEX) & MEM_RegWrite;
  wire Rs_EXWB_Match  = (EX_Rs == WB_RtRd)  & WB_RtRd_NZ  & (WantRsByEX |
                        NeedRsByEX) & WB_RegWrite;
  wire Rt_EXWB_Match  = (EX_Rt == WB_RtRd)  & WB_RtRd_NZ  & (WantRtByEX |
                        NeedRtByEX) & WB_RegWrite;
  // MEM Dependencies
  wire Rt_MEMWB_Match = (MEM_Rt == WB_RtRd) & WB_RtRd_NZ  & WB_RegWrite;


  // ID needs data from EX  : Stall
  wire ID_Stall_1 = (Rs_IDEX_Match  &  NeedRsByID);
  wire ID_Stall_2 = (Rt_IDEX_Match  &  NeedRtByID);
  // ID needs data from MEM : Stall if mem access
  wire ID_Stall_3 = (Rs_IDMEM_Match &  (MEM_MemRead | MEM_MemWrite) &
                    NeedRsByID);
  wire ID_Stall_4 = (Rt_IDMEM_Match &  (MEM_MemRead | MEM_MemWrite) &
                    NeedRtByID);
  // ID wants data from MEM : Forward if not mem access
  wire ID_Fwd_1   = (Rs_IDMEM_Match & ~(MEM_MemRead | MEM_MemWrite));
```

```
   wire ID_Fwd_2   = (Rt_IDMEM_Match & ~(MEM_MemRead | MEM_MemWrite));
   // ID wants/needs data from WB  : Forward
   wire ID_Fwd_3   = (Rs_IDWB_Match);
   wire ID_Fwd_4   = (Rt_IDWB_Match);
   // EX needs data from MEM : Stall if mem access
   wire EX_Stall_1 = (Rs_EXMEM_Match &  (MEM_MemRead | MEM_MemWrite) &
                      NeedRsByEX);
   wire EX_Stall_2 = (Rt_EXMEM_Match &  (MEM_MemRead | MEM_MemWrite) &
                      NeedRtByEX);
   // EX wants data from MEM : Forward if not mem access
   wire EX_Fwd_1   = (Rs_EXMEM_Match & ~(MEM_MemRead | MEM_MemWrite));
   wire EX_Fwd_2   = (Rt_EXMEM_Match & ~(MEM_MemRead | MEM_MemWrite));
   // EX wants/needs data from WB  : Forward
   wire EX_Fwd_3   = (Rs_EXWB_Match);
   wire EX_Fwd_4   = (Rt_EXWB_Match);
   // MEM needs data from WB : Forward
   wire MEM_Fwd_1  = (Rt_MEMWB_Match);


   // Stalls and Control Flow Final Assignments
   assign WB_Stall = M_Stall;
   assign  M_Stall = IF_Stall | M_Stall_Controller;
   assign EX_Stall = (EX_Stall_1 | EX_Stall_2 | EX_Exception_Stall) | EX_ALU_Stall
                     | M_Stall;
   assign ID_Stall = (ID_Stall_1 | ID_Stall_2 | ID_Stall_3 | ID_Stall_4 |
                      ID_Exception_Stall) | EX_Stall;
   assign IF_Stall = /*InstMem_Read |*/ ~InstMem_Ready | IF_Exception_Stall;

   // Forwarding Control Final Assignments
   assign ID_RsFwdSel = (ID_Fwd_1) ? 2'b01 : ((ID_Fwd_3) ? 2'b10 : 2'b00);
   assign ID_RtFwdSel = (Mfc0) ? 2'b11 : ((ID_Fwd_2) ? 2'b01 : ((ID_Fwd_4) ?
                         2'b10 : 2'b00));
   assign EX_RsFwdSel = (EX_Link) ? 2'b11 : ((EX_Fwd_1) ? 2'b01 : ((EX_Fwd_3) ?
                         2'b10 : 2'b00));
   assign EX_RtFwdSel = (EX_Link)  ? 2'b11 : ((EX_Fwd_2) ? 2'b01 : ((EX_Fwd_4) ?
                         2'b10 : 2'b00));
   assign M_WriteDataFwdSel = MEM_Fwd_1;

endmodule
```

(5) CPZero.v

```
`timescale 1ns / 1ps
/*
 * File         : CPZero.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   16-Sep-2011  GEA       Initial design.
 *   2.0   14-May-2012  GEA       Complete rework.
 *   3.0    1-Mar-2016  ASJ       Voting signals added to all registers
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   The MIPS-32 Coprocessor 0 (CP0). This is the processor management unit that
 *   allows interrupts, traps, system calls, and other exceptions. It
 *   distinguishes user and kernel modes, provides status information, and can
 *   override program flow. This processor is designed for "bare metal" memory
 *   access and thus does not have virtual memory hardware as a part of it.
 *   However, the subset of CP0 is MIPS-32-compliant.
 */
module CPZero(
   input  clock,
   //-- CP0 Functionality --//
```

```
    input  Mfc0,                 // CPU instruction is Mfc0
    input  Mtc0,                 // CPU instruction is Mtc0
    input  IF_Stall,
    input  ID_Stall,             // Commits are not made during stalls
    input  COP1,                 // Instruction for Coprocessor 1
    input  COP2,                 // Instruction for Coprocessor 2
    input  COP3,                 // Instruction for Coprocessor 3
    input  ERET,                 // Instruction is ERET (Exception Return)
    input  [4:0] Rd,             // Specifies Cp0 register
    input  [2:0] Sel,            // Specifies Cp0 'select'
    input  [31:0] Reg_In,        // Data from GP register to replace CP0register
    output reg [31:0] Reg_Out,  // Data from CP0 register for GP register
    output KernelMode,          // Kernel mode indicator for pipeline transit
    output ReverseEndian,       // Reverse Endian memory indicator for User Mode
    //-- Hw Interrupts --//
    input  [4:0] Int,           // Five hardware interrupts external to the
                                // processor
    //-- Exceptions --//
    input  reset,               // Cold Reset (EXC_Reset)
//  input  EXC_SReset,          // Soft Reset (not implemented)
    input  EXC_NMI,             // Non-Maskable Interrupt
    input  EXC_AdIF,            // Address Error Exception from i-fetch (mapped to
                                // AdEL)
    input  EXC_AdEL,            // Address Error Exception from data memory load
    input  EXC_AdES,            // Address Error Exception from data memory store
    input  EXC_Ov,              // Integer Overflow Exception
    input  EXC_Tr,              // Trap Exception
    input  EXC_Sys,             // System Call Exception
    input  EXC_Bp,              // Breakpoint Exception
    input  EXC_RI,              // Reserved Instruction Exception
    //-- Exception Data --//
    input  [31:0] ID_RestartPC, // PC for exception, whether PC of instruction or
                                // of branch (PC-4) if BDS
    input  [31:0] EX_RestartPC, // Same as 'ID_RestartPC' but in EX stage
    input  [31:0] M_RestartPC,  // Same as 'ID_RestartPC' but in MEM stage
    input  ID_IsFlushed,
    input  IF_IsBD,             // Indicator of IF exception being a branch delay
                                // slot instruction
    input  ID_IsBD,             // Indicator of ID exception being a branch delay
                                // slot instruction
    input  EX_IsBD,             // Indicator of EX exception being a branch delay
                                // slot instruction
    input  M_IsBD,              // Indicator of M  exception being a branch delay
                                // slot instruction
    input  [31:0] BadAddr_M,    // Bad 'Virtual' Address for exceptions AdEL, AdES
                                // in MEM stage
    input  [31:0] BadAddr_IF,   // Bad 'Virtual' Address for AdIF (i.e. AdEL) in
                                // IF stage
    input  ID_CanErr,           // Cumulative signal, i.e. (ID_ID_CanErr |
                                // ID_EX_CanErr | ID_M_CanErr)
    input  EX_CanErr,           // Cumulative signal, i.e. (EX_EX_CanErr |
                                // EX_M_CanErr)
    input  M_CanErr,            // Memory stage can error (i.e., cause exception)
    //-- Exception Control Flow --/
    output IF_Exception_Stall,
    output ID_Exception_Stall,
    output EX_Exception_Stall,
    output M_Exception_Stall,
    output IF_Exception_Flush,
    output ID_Exception_Flush,
    output EX_Exception_Flush,
    output M_Exception_Flush,
    output Exc_PC_Sel,              // Mux selector for exception PC override
    output reg [31:0] Exc_PC_Out,   // Address for PC at the beginning of an
                                    // exception
    //Voter Signals for Registers
    input  reset_r,
    input  Status_BEV,
    input  Status_NMI,
```

```
   input  Status_ERL,
   input  [31:0] ErrorEPC,
   input  [31:0] Count,
   input  [31:0] Compare,
   input  Status_CU_0,
   input  Status_RE,
   input  [7:0] Status_IM,
   input  Status_UM,
   input  Status_IE,
   input  Cause_IV,
   input  [7:0] Cause_IP,
   input  Cause_BD,
   input  [1:0] Cause_CE,
   input  [3:0] Cause_ExcCode30,
   input  Status_EXL,
   input  [31:0] EPC,
   input  [31:0] BadVAddr,
   output reg vote_reset_r,
   output reg vote_Status_BEV,
   output reg vote_Status_NMI,
   output reg vote_Status_ERL,
   output reg [31:0] vote_ErrorEPC,
   output reg [31:0] vote_Count,
   output reg [31:0] vote_Compare,
   output reg vote_Status_CU_0,
   output reg vote_Status_RE,
   output reg [7:0] vote_Status_IM,
   output reg vote_Status_UM,
   output reg vote_Status_IE,
   output reg vote_Cause_IV,
   output reg [7:0] vote_Cause_IP,
   output reg vote_Cause_BD,
   output reg [1:0] vote_Cause_CE,
   output reg [3:0] vote_Cause_ExcCode30,
   output reg vote_Status_EXL,
   output reg [31:0] vote_EPC,
   output reg [31:0] vote_BadVAddr
   );

   `include “MIPS_Parameters.v”


/***
Exception Control Flow Notes

   - Exceptions can occur in every pipeline stage. This implies that more than one
exception can be raised in a single cycle. When this occurs, only the forward-most
exception (i.e. MEM over EX) is handled. This and the following note guarantee program
order.

   - An exception in any pipeline stage must stall that stage until all following stages
are exception-free. This is because it only makes sense for exceptions to occur in
program order.

   - A pipeline stage which causes an exception must flush, i.e., prevent any commits it
would have normally made and convert itself to a NOP for the next pipeline stage.
Furthermore, it must flush all previous pipeline stages as well in order to retain
program order.

   - Instructions reading CP0 (mtc0) read in ID without further action. Writes to CP0
(mtc0, eret) also write in ID, but only after forward pipeline stages have been cleared
of possible exceptions. This prevents many insidious bugs, such as switching to User Mode
in ID when a legitimate memory access in kernel mode is processing in MEM, or conversely
a switch to Kernel Mode in ID when an instruction in User Mode is attempting a kernel
region memory access (when a kernel mode signal does not propagate through the pipeline).

   - Commits occur in ID (CP0), EX (HILO), MEM, and WB (registers).
```

- Hardware interrupts are detected and inserted in the ID stage, but only when there are no other possible exceptions in the pipeline. Because they appear ‘asynchronous’ to the processor, the remaining instructions in forward stages (EX, MEM, WB) can either be flushed or completed. It is simplest to have them complete to avoid restarts, but the interrupt latency is higher if e.g., the MEM stage stalls on a memory access (this would be unavoidable on single-cycle processors). This implementation allows all forward instructions to complete, for a greater instruction throughput but higher interrupt latency.

- Software interrupts should appear synchronous in the program order, meaning that all instructions previous to them should complete and no instructions after them should start until the interrupts has been processed.

```
   Exception Name             Short Name          Pipeline Stage
      Address Error Ex         (AdEL, AdES)        MEM, IF
      Integer Overflow Ex      (Ov)                EX
      Trap Ex                  (Tr)                MEM
      Syscall                  (Sys)               ID
      Breakpoint               (Bp)                ID
      Reserved Instruction     (RI)                ID
      Coprocessor Unusable     (CpU)               ID
      Interrupt                (Int)               ID
      Reset, SReset, NMI                           ID
***/


   // Exceptions Generated Internally
   wire EXC_CpU;

   // Hardware Interrupt #5, caused by Timer/Perf counter
   wire Int5;

   // Top-level Authoritative Interrupt Signal
   wire EXC_Int;

   // General Exception detection (all but Interrupts, Reset, Soft Reset, and NMI)
   wire EXC_General = EXC_AdIF | EXC_AdEL | EXC_AdES | EXC_Ov | EXC_Tr | EXC_Sys |
                      EXC_Bp | EXC_RI | EXC_CpU;

   // Misc
   wire CP0_WriteCond;
   reg  [3:0] Cause_ExcCode_bits;

   always @(posedge clock) begin
       vote_reset_r <= reset;
   end

   /***
    MIPS-32 COPROCESSOR 0 (Cp0) REGISTERS

    These are defined in “MIPS32 Architecture for Programmers Volume III:
    The MIPS32 Privileged Resource Architecture” from MIPS Technologies, Inc.

    Optional registers are omitted. Changes to the processor (such as adding
    an MMU/TLB, etc., must be reflected in these registers.
   */

   // BadVAddr Register (Register 8, Select 0)


   // Count Register (Register 9, Select 0)


   // Compare Register (Register 11, Select 0)


   // Status Register (Register 12, Select 0)
   wire [2:0] Status_CU_321 = 3’b000;
   // Access Control to CPs, [2]->Cp3, ..[0].->Cp0
```

```
wire Status_RP = 0;
wire Status_FR = 0;
//   Status_RE         // Reverse Endian Memory for User Mode
wire Status_MX = 0;
wire Status_PX = 0;
//   Status_BEV;        // Exception vector locations (0->Norm, 1->Bootstrap)
wire Status_TS = 0;
wire Status_SR = 0;     // Soft reset not implemented
//   Status_NMI;        // Non-Maskable Interrupt
wire Status_RES = 0;
wire [1:0] Status_Custom = 2'b00;
//   Status_IM;    // Interrupt mask
wire Status_KX = 0;
wire Status_SX = 0;
wire Status_UX = 0;
//   Status_UM;         // Base operating mode (0->Kernel, 1->User)
wire Status_R0 = 0;
//   Status_ERL;        // Error Level     (0->Normal, 1->Error (reset, NMI))
//   Status_EXL;        // Exception level (0->Normal, 1->Exception)
//   Status_IE;         // Interrupt Enable
wire [31:0] Status = {Status_CU_321, Status_CU_0, Status_RP, Status_FR,
                      Status_RE, Status_MX, Status_PX, Status_BEV, Status_TS,
                      Status_SR, Status_NMI, Status_RES, Status_Custom,
                      Status_IM, Status_KX, Status_SX, Status_UX, Status_UM,
                      Status_R0, Status_ERL, Status_EXL, Status_IE};

// Cause Register (Register 13, Select 0)
//   Cause_BD;           // Exception occured in Branch Delay
//   Cause_CE;           // CP number for CP Unusable exception
//   Cause_IV;           // Indicator of general IV (0->0x180) or special IV
                         // (1->0x200)
wire Cause_WP = 0;
//   Cause_IP;           // Pending HW Interrupt indicator.
wire Cause_ExcCode4 = 0; // Can be made into a register when this bit is
                         // needed.
//   Cause_ExcCode30;    //Description of Exception (only lower 4 bits
                         // currently used; see above)
wire [31:0] Cause  = {Cause_BD, 1'b0, Cause_CE, 4'b0000, Cause_IV, Cause_WP,
                      6'b000000, Cause_IP, 1'b0, Cause_ExcCode4,
                      Cause_ExcCode30, 2'b00};

// Exception Program Counter (Register 14, Select 0)

// Processor Identification (Register 15, Select 0)
wire [7:0] ID_Options = 8'b0000_0000;
wire [7:0] ID_CID = 8'b0000_0000;
wire [7:0] ID_PID = 8'b0000_0000;
wire [7:0] ID_Rev = 8'b0000_0001;
wire [31:0] PRId = {ID_Options, ID_CID, ID_PID, ID_Rev};

// Configuration Register (Register 16, Select 0)
wire Config_M = 1;
wire [14:0] Config_Impl = 15'b000_0000_0000_0000;
wire Config_BE = `Big_Endian;
wire [1:0] Config_AT = 2'b00;
wire [2:0] Config_AR = 3'b000;
wire [2:0] Config_MT = 3'b000;
wire [2:0] Config_K0 = 3'b000;
wire [31:0] Config = {Config_M, Config_Impl, Config_BE, Config_AT, Config_AR,
                      Config_MT, 4'b0000, Config_K0};

// Configuration Register 1 (Register 16, Select 1)
wire Config1_M = 0;
wire [5:0] Config1_MMU = 6'b000000;
wire [2:0] Config1_IS = 3'b000;
wire [2:0] Config1_IL = 3'b000;
wire [2:0] Config1_IA = 3'b000;
wire [2:0] Config1_DS = 3'b000;
wire [2:0] Config1_DL = 3'b000;
```

```
   wire [2:0] Config1_DA = 3'b000;
   wire Config1_C2 = 0;
   wire Config1_MD = 0;
   wire Config1_PC = 0;    // XXX Performance Counters
   wire Config1_WR = 0;    // XXX Watch Registers
   wire Config1_CA = 0;
   wire Config1_EP = 0;
   wire Config1_FP = 0;
   wire [31:0] Config1 = {Config1_M, Config1_MMU, Config1_IS, Config1_IL,
                          Config1_IA, Config1_DS, Config1_DL, Config1_DA,
                          Config1_C2, Config1_MD, Config1_PC, Config1_WR,
                          Config1_CA, Config1_EP, Config1_FP};

   // Performance Counter Register (Register 25) XXX TODO

   // ErrorEPC Register (Register 30, Select 0)

   // Exception Detection and Processing
   wire M_Exception_Detect, EX_Exception_Detect, ID_Exception_Detect,
        IF_Exception_Detect;
   wire M_Exception_Mask, EX_Exception_Mask, ID_Exception_Mask, IF_Exception_Mask;
   wire M_Exception_Ready, EX_Exception_Ready, ID_Exception_Ready,
        IF_Exception_Ready;

   /*** Coprocessor Unusable Exception ***/
   assign EXC_CpU = COP1 | COP2 | COP3 | ((Mtc0 | Mfc0 | ERET) & ~(Status_CU_0 |
                    KernelMode));

   /*** Kernel Mode Signal ***/
   assign KernelMode = ~Status_UM | Status_EXL | Status_ERL;

   /*** Reverse Endian for User Mode ***/
   assign ReverseEndian = Status_RE;

   /*** Interrupts ***/
   assign Int5 = (Count == Compare);
   wire   Enabled_Interrupt = EXC_NMI | (Status_IE & ((Cause_IP[7:0] &
                              Status_IM[7:0]) != 8'h00));
   assign EXC_Int = Enabled_Interrupt & ~Status_EXL & ~Status_ERL & ~ID_IsFlushed;

   assign CP0_WriteCond = (Status_CU_0 | KernelMode) & Mtc0 & ~ID_Stall;


/***
Exception Hazard Flow Control Explanation:

   - An exception at any time in any stage causes its own and any previous stages
     to flush (clear own commits, NOPS to fwd stages).
   - An exception in a stage can also stall that stage (and inherently all
     previous stages) if and only if:
        1. A forward stage is capable of causing an exception AND
        2. A forward stage is not currently causing an exception.
   - An exception is ready to process when it is detected and not stalled in a
     stage.

   Flush specifics per pipeline stage:
      MEM: Mask 'MemWrite' and 'MemRead' (for performance) after EX/M and before
           data memory. NOPs to M/WB.
      EX : Mask writes to HI/LO. NOPs to EX/M.
      ID : Mask writes (reads?) to CP0. NOPs to ID/EX.
      IF : NOP to IF/ID.
***/

   /*** Exceptions grouped by pipeline stage ***/
   assign   M_Exception_Detect = EXC_AdEL | EXC_AdES | EXC_Tr;
   assign  EX_Exception_Detect = EXC_Ov;
   assign  ID_Exception_Detect = EXC_Sys | EXC_Bp | EXC_RI | EXC_CpU | EXC_Int;
   assign  IF_Exception_Detect = EXC_AdIF;
```

```
/*** Exception mask conditions ***/

// A potential bug would occur if e.g. EX stalls, MEM has data, but MEM is not
// stalled and finishes going through the pipeline so forwarding would fail.
// This is not a problem however because EX would not need data since it would
// flush on an exception.
assign   M_Exception_Mask = IF_Stall;
assign  EX_Exception_Mask = IF_Stall | M_CanErr;
assign  ID_Exception_Mask = IF_Stall | M_CanErr | EX_CanErr;
assign  IF_Exception_Mask = M_CanErr | EX_CanErr | ID_CanErr | EXC_Int;

/***
Exceptions which must wait for forward stages. A stage will not stall if a
forward stage has an exception. These stalls must be inserted as stall
conditions in the hazard unit so that it will take care of chaining.
All writes to CP0 must also wait for forward hazard conditions to clear.
***/
assign  M_Exception_Stall  =  M_Exception_Detect  & M_Exception_Mask;
assign  EX_Exception_Stall =  EX_Exception_Detect & EX_Exception_Mask &
                              ~M_Exception_Detect;
assign  ID_Exception_Stall = (ID_Exception_Detect | ERET | Mtc0) &
                              ID_Exception_Mask & ~(EX_Exception_Detect |
                              M_Exception_Detect);
assign  IF_Exception_Stall =  IF_Exception_Detect & IF_Exception_Mask &
                              ~(ID_Exception_Detect | EX_Exception_Detect |
                              M_Exception_Detect);


/*** Exceptions which are ready to process (mutually exclusive) ***/
// XXX can remove ~ID_Stall since in mask now (?)
assign   M_Exception_Ready =  ~ID_Stall & M_Exception_Detect  &
                              ~M_Exception_Mask;
assign  EX_Exception_Ready =  ~ID_Stall & EX_Exception_Detect &
                              ~EX_Exception_Mask;
assign  ID_Exception_Ready =  ~ID_Stall & ID_Exception_Detect &
                              ~ID_Exception_Mask;
assign  IF_Exception_Ready =  ~ID_Stall & IF_Exception_Detect &
                              ~IF_Exception_Mask;

/***
Flushes. A flush clears a pipeline stage's control signals and prevents the
stage from committing any changes. Data such as 'RestartPC' and the detected
exception must remain.
***/
assign   M_Exception_Flush = M_Exception_Detect;
assign  EX_Exception_Flush = M_Exception_Detect | EX_Exception_Detect;
assign  ID_Exception_Flush = M_Exception_Detect | EX_Exception_Detect |
                             ID_Exception_Detect;
assign  IF_Exception_Flush = M_Exception_Detect | EX_Exception_Detect |
                             ID_Exception_Detect | IF_Exception_Detect | (ERET
                             & ~ID_Stall) | reset_r;


/*** Software reads of CP0 Registers ***/
always @(*) begin
   if (Mfc0 & (Status_CU_0 | KernelMode)) begin
      case (Rd)
         5'd8  : Reg_Out <= BadVAddr;
         5'd9  : Reg_Out <= Count;
         5'd11 : Reg_Out <= Compare;
         5'd12 : Reg_Out <= Status;
         5'd13 : Reg_Out <= Cause;
         5'd14 : Reg_Out <= EPC;
         5'd15 : Reg_Out <= PRId;
         5'd16 : Reg_Out <= (Sel == 3'b000) ? Config : Config1;
         5'd30 : Reg_Out <= ErrorEPC;
         default : Reg_Out <= 32'h0000_0000;
      endcase
   end
```

```
      else begin
         Reg_Out <= 32'h0000_0000;
      end
   end

/* Cp0 Register Assignments: Non-general exceptions (Reset, Soft Reset, NMI...) */
   always @(posedge clock) begin
      if (reset) begin
         vote_Status_BEV <= 1;
         vote_Status_NMI <= 0;
         vote_Status_ERL <= 1;
         vote_ErrorEPC   <= 32'b0;
      end
      else if (ID_Exception_Ready & EXC_NMI) begin
         vote_Status_BEV <= 1;
         vote_Status_NMI <= 1;
         vote_Status_ERL <= 1;
         vote_ErrorEPC   <= ID_RestartPC;
      end
      else begin
         vote_Status_BEV    <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[22] : Status_BEV;
         vote_Status_NMI    <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[19] : Status_NMI;
         vote_Status_ERL    <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[2] : ((Status_ERL & ERET & ~ID_Stall) ?
                               1'b0 : Status_ERL);
         vote_ErrorEPC      <= (CP0_WriteCond & (Rd == 5'd30) & (Sel == 3'b000)) ?
                               Reg_In : ErrorEPC;
      end
   end

   /*** Cp0 Register Assignments: All other registers ***/
   always @(posedge clock) begin
      if (reset) begin
         vote_Count         <= 32'b0;
         vote_Compare       <= 32'b0;
         vote_Status_CU_0   <= 0;
         vote_Status_RE     <= 0;
         vote_Status_IM     <= 8'b0;
         vote_Status_UM     <= 0;
         vote_Status_IE     <= 0;
         vote_Cause_IV      <= 0;
         vote_Cause_IP      <= 8'b0;
      end
      else begin
         vote_Count         <= (CP0_WriteCond & (Rd == 5'd9 ) & (Sel == 3'b000)) ?
                               Reg_In : Count + 1;
         vote_Compare       <= (CP0_WriteCond & (Rd == 5'd11) & (Sel == 3'b000)) ?
                               Reg_In : Compare;
         vote_Status_CU_0   <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[28] : Status_CU_0;
         vote_Status_RE     <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[25] : Status_RE;
         vote_Status_IM     <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[15:8] : Status_IM;
         vote_Status_UM     <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[4] : Status_UM;
         vote_Status_IE     <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                               Reg_In[0] : Status_IE;
         vote_Cause_IV      <= (CP0_WriteCond & (Rd == 5'd13) & (Sel == 3'b000)) ?
                               Reg_In[23]: Cause_IV;
         /* Cause_IP indicates 8 interrupts:
               [7]   is set by the timer comparison (Compare == Count), and
                     cleared by writing to Compare.
               [6:2] are set and cleared by external hardware.
               [1:0] are set and cleared by software.
          */
         vote_Cause_IP[7]   <= (CP0_WriteCond & (Rd == 5'd11) & (Sel == 3'b000)) ?
```

```
                           1'b0 : ((Int5) ? 1'b1 : Cause_IP[7]);
      vote_Cause_IP[6:2] <= Int[4:0];
      vote_Cause_IP[1:0] <= (CP0_WriteCond & (Rd == 5'd13) & (Sel == 3'b000)) ?
                           Reg_In[9:8] : Cause_IP[1:0];
   end
end

/*** Cp0 Register Assignments: General Exception and Interrupt Processing ***/
always @(posedge clock) begin
   if (reset) begin
      vote_Cause_BD <= 0;
      vote_Cause_CE <= 2'b00;
      vote_Cause_ExcCode30 <= 4'b0000;
      vote_Status_EXL <= 0;
      vote_EPC <= 32'h0;
      vote_BadVAddr <= 32'h0;
   end
   else begin
      // MEM stage
      if (M_Exception_Ready) begin
         vote_Cause_BD <= (Status_EXL) ? Cause_BD : M_IsBD;
         vote_Cause_CE <= (COP3) ? 2'b11 : ((COP2) ? 2'b10 : ((COP1) ? 2'b01 :
                          2'b00));
         vote_Cause_ExcCode30 <= Cause_ExcCode_bits;
         vote_Status_EXL <= 1;
         vote_EPC <= (Status_EXL) ? EPC : M_RestartPC;
         vote_BadVAddr <= BadAddr_M;
      end
      // EX stage
      else if (EX_Exception_Ready) begin
         vote_Cause_BD <= (Status_EXL) ? Cause_BD : EX_IsBD;
         vote_Cause_CE <= (COP3) ? 2'b11 : ((COP2) ? 2'b10 : ((COP1) ? 2'b01 :
                          2'b00));
         vote_Cause_ExcCode30 <= Cause_ExcCode_bits;
         vote_Status_EXL <= 1;
         vote_EPC <= (Status_EXL) ? EPC : EX_RestartPC;
         vote_BadVAddr <= BadVAddr;
      end
      // ID stage
      else if (ID_Exception_Ready) begin
         vote_Cause_BD <= (Status_EXL) ? Cause_BD : ID_IsBD;
         vote_Cause_CE <= (COP3) ? 2'b11 : ((COP2) ? 2'b10 : ((COP1) ? 2'b01 :
                          2'b00));
         vote_Cause_ExcCode30 <= Cause_ExcCode_bits;
         vote_Status_EXL <= 1;
         vote_EPC <= (Status_EXL) ? EPC : ID_RestartPC;
         vote_BadVAddr <= BadVAddr;
      end
      // IF stage
      else if (IF_Exception_Ready) begin
         vote_Cause_BD <= (Status_EXL) ? Cause_BD : IF_IsBD;
         vote_Cause_CE <= (COP3) ? 2'b11 : ((COP2) ? 2'b10 : ((COP1) ? 2'b01 :
                          2'b00));
         vote_Cause_ExcCode30 <= Cause_ExcCode_bits;
         vote_Status_EXL <= 1;
         vote_EPC <= (Status_EXL) ? EPC : BadAddr_IF;
         vote_BadVAddr <= BadAddr_IF;
      end
      // No exceptions this cycle
      else begin
         vote_Cause_BD <= 1'b0;
         vote_Cause_CE <= Cause_CE;
         vote_Cause_ExcCode30 <= Cause_ExcCode30;
         // Without new exceptions, 'Status_EXL' is set by software or cleared
         // by ERET.
         vote_Status_EXL <= (CP0_WriteCond & (Rd == 5'd12) & (Sel == 3'b000)) ?
                          Reg_In[1] : ((Status_EXL & ERET & ~ID_Stall) ?
                          1'b0 : Status_EXL);
         // The EPC is also writable by software
```

```
            vote_EPC <= (CP0_WriteCond & (Rd == 5'd14) & (Sel == 3'b000)) ?
                        Reg_In : EPC;
            vote_BadVAddr <= BadVAddr;
         end
      end
   end


   /*** Program Counter for all Exceptions/Interrupts ***/
   always @(*) begin
      // Following is redundant since PC has initial value now.
      if (reset) begin
         Exc_PC_Out <= `EXC_Vector_Base_Reset;
      end
      else if (ERET & ~ID_Stall) begin
         Exc_PC_Out <= (Status_ERL) ? ErrorEPC : EPC;
      end
      else if (EXC_General) begin
         Exc_PC_Out <= (Status_BEV) ? (`EXC_Vector_Base_Other_Boot +
                       `EXC_Vector_Offset_General) :
                       (`EXC_Vector_Base_Other_NoBoot +
                       `EXC_Vector_Offset_General);
      end
      else if (EXC_NMI) begin
         Exc_PC_Out <= `EXC_Vector_Base_Reset;
      end
      else if (EXC_Int & Cause_IV) begin
         Exc_PC_Out <= (Status_BEV) ? (`EXC_Vector_Base_Other_Boot +
                       `EXC_Vector_Offset_Special) :
                       (`EXC_Vector_Base_Other_NoBoot +
                       `EXC_Vector_Offset_Special);
      end
      else begin
         Exc_PC_Out <= (Status_BEV) ? (`EXC_Vector_Base_Other_Boot +
                       `EXC_Vector_Offset_General) :
                       (`EXC_Vector_Base_Other_NoBoot +
                       `EXC_Vector_Offset_General);
      end
   end

   //assign Exc_PC_Sel = (reset | (ERET & ~ID_Stall) | EXC_General | EXC_Int);
   assign Exc_PC_Sel = reset | (ERET & ~ID_Stall) | IF_Exception_Ready |
                       ID_Exception_Ready | EX_Exception_Ready |
                       M_Exception_Ready;

   /*** Cause Register ExcCode Field ***/
   always @(*) begin
      // Ordered by Pipeline Stage with Interrupts last
      if      (EXC_AdEL) Cause_ExcCode_bits <= 4'h4;     // 00100
      else if (EXC_AdES) Cause_ExcCode_bits <= 4'h5;     // 00101
      else if (EXC_Tr)   Cause_ExcCode_bits <= 4'hd;     // 01101
      else if (EXC_Ov)   Cause_ExcCode_bits <= 4'hc;     // 01100
      else if (EXC_Sys)  Cause_ExcCode_bits <= 4'h8;     // 01000
      else if (EXC_Bp)   Cause_ExcCode_bits <= 4'h9;     // 01001
      else if (EXC_RI)   Cause_ExcCode_bits <= 4'ha;     // 01010
      else if (EXC_CpU)  Cause_ExcCode_bits <= 4'hb;     // 01011
      else if (EXC_AdIF) Cause_ExcCode_bits <= 4'h4;     // 00100
      else if (EXC_Int)  Cause_ExcCode_bits <= 4'h0;     // 00000 // OK that NMI
                                                                  // writes this.
      else Cause_ExcCode_bits <= 4'bxxxx;
   end

endmodule
```

(6) Mux4.v

```
`timescale 1ns / 1ps
/*
```

```
 * File         : Mux4.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   7-Jun-2011   GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A 4-input Mux of variable width, defaulting to 32-bit width.
 */
module Mux4 #(parameter WIDTH = 32)(
   input  [1:0] sel,
   input  [(WIDTH-1):0] in0, in1, in2, in3,
   output reg [(WIDTH-1):0] out
   );

   always @(*) begin
      case (sel)
         2'b00 : out <= in0;
         2'b01 : out <= in1;
         2'b10 : out <= in2;
         2'b11 : out <= in3;
      endcase
   end

endmodule
```

(7) Mux2.v

```
`timescale 1ns / 1ps
/*
 * File         : Mux2.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   7-Jun-2011   GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A 2-input Mux of variable width, defaulting to 32-bit width.
 */
module Mux2 #(parameter WIDTH = 32)(
   input  sel,
   input  [(WIDTH-1):0] in0, in1,
   output [(WIDTH-1):0] out
   );

   assign out = (sel) ? in1 : in0;

endmodule
```

(8) Register.v

```
`timescale 1ns / 1ps
/*
 * File         : Register.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
```

```
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   7-Jun-2011   GEA       Initial design.
 *   2.0   1-Mar-2016   ASJ       Voting signals added to all register
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A variable-width register (d flip-flop) with configurable initial
 *   value. Default is 32-bit width and 0s for initial value.
 */
module Register #(parameter WIDTH = 32, INIT = 0)(
   input  clock,
   input  reset,
   input  enable,
   input  [(WIDTH-1):0] D,
   input  [(WIDTH-1):0] Q,
   output reg [(WIDTH-1):0] vote_Q
   );

   initial
      vote_Q = INIT;

   always @(posedge clock) begin
      vote_Q <= (reset) ? INIT : ((enable) ? D : Q);
   end

endmodule
```

(9) Add.v

```
`timescale 1ns / 1ps
/*
 * File         : Add.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   7-Jun-2011   GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A simple 32-bit 2-input adder.
 */
module Add(
   input  [31:0] A,
   input  [31:0] B,
   output [31:0] C
   );

   assign C = (A + B);

endmodule
```

(10) IFID_Stage.v

```
`timescale 1ns / 1ps
/*
 * File         : IFID_Stage.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
```

```
 *   Rev   Date         Initials  Description of Change
 *   1.0   9-Jun-2011   GEA       Initial design.
 *   2.0  26-Jul-2012   GEA       Many updates have been made.
 *   3.0   1-Mar-2016   ASJ       Voting signals added to all registers
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   The Pipeline Register to bridge the Instruction Fetch
 *   and Instruction Decode stages.
 */
module IFID_Stage(
   input  clock,
   input  reset,
   input  IF_Flush,
   input  IF_Stall,
   input  ID_Stall,
   // Control Signals
   input  [31:0] IF_Instruction,
   // Data Signals
   input  [31:0] IF_PCAdd4,
   input  [31:0] IF_PC,
   input  IF_IsBDS,
   // Voter Signals for Registers
   input [31:0] ID_Instruction,
   input [31:0] ID_PCAdd4,
   input [31:0] ID_RestartPC,
   input ID_IsBDS,
   input ID_IsFlushed,
   output reg [31:0] vote_ID_Instruction,
   output reg [31:0] vote_ID_PCAdd4,
   output reg [31:0] vote_ID_RestartPC,
   output reg vote_ID_IsBDS,
   output reg vote_ID_IsFlushed
   );

/***
    The purpose of a pipeline register is to capture data from one pipeline stage and
provide it to the next pipeline stage. This creates at least one clock cycle of delay,
but reduces the combinatorial path length of signals which allows for higher clock
speeds.

    All pipeline registers update unless the forward stage is stalled. When this occurs
or when the current stage is being flushed, the forward stage will receive data that is
effectively a NOP and causes nothing to happen throughout the remaining pipeline
traversal. In other words:

    A stall masks all control signals to forward stages. A flush permanently clears
control signals to forward stages (but not certain data for exception purposes).
***/


/***
    The signal 'ID_IsFlushed' is needed because of interrupts. Normally, a flushed
instruction is a NOP which will never cause an exception and thus its restart PC will
never be needed or used. However, interrupts are detected in ID and may occur when any
instruction, flushed or not, is in the ID stage. It is an error to save the restart PC of
a flushed instruction since it was never supposed to execute (such as the "delay slot"
after ERET or the branch delay slot after a canceled Branch Likely instruction). A simple
way to prevent this is to pass a signal to ID indicating that its instruction was
flushed. Interrupt detection is then masked when this signal is high, and the interrupt
will trigger on the next instruction load to ID.
***/

   always @(posedge clock) begin
      vote_ID_Instruction <= (reset) ? 32'b0 : ((ID_Stall) ? ID_Instruction :
                             ((IF_Stall | IF_Flush) ? 32'b0 : IF_Instruction));
      vote_ID_PCAdd4      <= (reset) ? 32'b0 : ((ID_Stall) ?
```

```
                              ID_PCAdd4 : IF_PCAdd4);
      vote_ID_IsBDS       <= (reset) ? 1'b0  : ((ID_Stall) ? ID_IsBDS : IF_IsBDS);
      vote_ID_RestartPC   <= (reset) ? 32'b0 : ((ID_Stall | IF_IsBDS) ?
                              ID_RestartPC : IF_PC);
      vote_ID_IsFlushed   <= (reset) ? 1'b0  : ((ID_Stall) ? ID_IsFlushed :
                              IF_Flush);
   end

endmodule
```

(11) RegisterFile.v

```
`timescale 1ns / 1ps
/*
 * File         : RegisterFile.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   7-Jun-2011   GEA       Initial design.
 *   2.0   1-Mar-2016   ASJ       Voting signals added to all registers
 *                                Feedback path added for stalls
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A Register File for a MIPS processor. Contains 32 general-purpose
 *   32-bit wide registers and two read ports. Register 0 always reads
 *   as zero.
 */
module RegisterFile(
   input  clock,
   input  reset,
   input  [4:0]  ReadReg1, ReadReg2, WriteReg,
   input  [31:0] WriteData,
   input  RegWrite,
   output [31:0] ReadData1, ReadData2,
   // Voter Signals for Registers
   input  [991:0] registers_in,
   output [991:0] registers_out
   );

   // Register file of 32 32-bit registers. Register 0 is hardwired to 0s
   wire [31:0] registers [1:31];
   reg [31:0] vote_registers [1:31];

   // Sequential (clocked) write.
   // 'WriteReg' is the register index to write. 'RegWrite' is the command.
   integer i;

   always @(posedge clock) begin
      for (i=1; i<32; i=i+1) begin
         vote_registers[i] <= (reset) ? 0 : (RegWrite && (WriteReg==i)) ?
                              WriteData : registers[i];
      end
   end

   genvar j;

   generate
      for (j=1; j<32; j=j+1) begin : Voter_Signals
         assign registers[j] = registers_in[((32*j)-1):(32*(j-1))];
         assign registers_out[((32*j)-1):(32*(j-1))] = vote_registers[j];
      end
   endgenerate
```

```
   // Combinatorial Read. Register 0 is all 0s.
   assign ReadData1 = (ReadReg1 == 0) ? 32'h00000000 : registers[ReadReg1];
   assign ReadData2 = (ReadReg2 == 0) ? 32'h00000000 : registers[ReadReg2];

endmodule
```

(12) Compare.v

```
`timescale 1ns / 1ps
/*
 * File         : Compare.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   15-Jun-2011  GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Compares two 32-bit values and outputs the following information about them:
 *      EQ  : A and B are equal
 *      GZ  : A is greater than zero
 *      LZ  : A is less than zero
 *      GEZ : A is greater than or equal to zero
 *      LEZ : A is less than or equal to zero
 */
module Compare(
   input  [31:0] A,
   input  [31:0] B,
   output EQ,
   output GZ,
   output LZ,
   output GEZ,
   output LEZ
   );

   wire   ZeroA = (A == 32'b0);

   assign EQ  = ( A == B);
   assign GZ  = (~A[31] & ~ZeroA);
   assign LZ  =   A[31];
   assign GEZ =  ~A[31];
   assign LEZ = ( A[31] |  ZeroA);

endmodule
```

(13) IDEX_Stage.v

```
`timescale 1ns / 1ps
/*
 * File         : IDEX_Stage.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   9-Jun-2011   GEA       Initial design.
 *   2.0  26-Jul-2012   GEA       Many updates have been made.
 *   3.0   1-Mar-2016   ASJ       Voting signal added to all registers.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
```

```
 * Description:
 *   The Pipeline Register to bridge the Instruction Decode
 *   and Execute stages.
 */
module IDEX_Stage(
   input  clock,
   input  reset,
   input  ID_Flush,
   input  ID_Stall,
   input  EX_Stall,
   // Control Signals
   input  ID_Link,
   input  ID_RegDst,
   input  ID_ALUSrcImm,
   input  [4:0] ID_ALUOp,
   input  ID_Movn,
   input  ID_Movz,
   input  ID_LLSC,
   input  ID_MemRead,
   input  ID_MemWrite,
   input  ID_MemByte,
   input  ID_MemHalf,
   input  ID_MemSignExtend,
   input  ID_Left,
   input  ID_Right,
   input  ID_RegWrite,
   input  ID_MemtoReg,
   input  ID_ReverseEndian,
   // Hazard & Forwarding
   input  [4:0] ID_Rs,
   input  [4:0] ID_Rt,
   input  ID_WantRsByEX,
   input  ID_NeedRsByEX,
   input  ID_WantRtByEX,
   input  ID_NeedRtByEX,
   // Exception Control/Info
   input  ID_KernelMode,
   input  [31:0] ID_RestartPC,
   input  ID_IsBDS,
   input  ID_Trap,
   input  ID_TrapCond,
   input  ID_EX_CanErr,
   input  ID_M_CanErr,
   // Data Signals
   input  [31:0] ID_ReadData1,
   input  [31:0] ID_ReadData2,
   input  [16:0] ID_SignExtImm, // ID_Rd, ID_Shamt included here
   // ----------------
   output [4:0] EX_Rd,
   output [4:0] EX_Shamt,
   output [1:0] EX_LinkRegDst,
   output [31:0] EX_SignExtImm,
   // Voter Signals for Registers
   input [16:0] EX_SignExtImm_pre,
   input EX_RegDst,
   input EX_Link,
   input EX_ALUSrcImm,
   input [4:0] EX_ALUOp,
   input EX_Movn,
   input EX_Movz,
   input EX_LLSC,
   input EX_MemRead,
   input EX_MemWrite,
   input EX_MemByte,
   input EX_MemHalf,
   input EX_MemSignExtend,
   input EX_Left,
   input EX_Right,
   input EX_RegWrite,
```

```
   input EX_MemtoReg,
   input EX_ReverseEndian,
   input [4:0]  EX_Rs,
   input [4:0]  EX_Rt,
   input EX_WantRsByEX,
   input EX_NeedRsByEX,
   input EX_WantRtByEX,
   input EX_NeedRtByEX,
   input EX_KernelMode,
   input [31:0] EX_RestartPC,
   input EX_IsBDS,
   input EX_Trap,
   input EX_TrapCond,
   input EX_EX_CanErr,
   input EX_M_CanErr,
   input [31:0] EX_ReadData1,
   input [31:0] EX_ReadData2,
   output reg [16:0] vote_EX_SignExtImm_pre,
   output reg vote_EX_RegDst,
   output reg vote_EX_Link,
   output reg vote_EX_ALUSrcImm,
   output reg [4:0] vote_EX_ALUOp,
   output reg vote_EX_Movn,
   output reg vote_EX_Movz,
   output reg vote_EX_LLSC,
   output reg vote_EX_MemRead,
   output reg vote_EX_MemWrite,
   output reg vote_EX_MemByte,
   output reg vote_EX_MemHalf,
   output reg vote_EX_MemSignExtend,
   output reg vote_EX_Left,
   output reg vote_EX_Right,
   output reg vote_EX_RegWrite,
   output reg vote_EX_MemtoReg,
   output reg vote_EX_ReverseEndian,
   output reg [4:0]  vote_EX_Rs,
   output reg [4:0]  vote_EX_Rt,
   output reg vote_EX_WantRsByEX,
   output reg vote_EX_NeedRsByEX,
   output reg vote_EX_WantRtByEX,
   output reg vote_EX_NeedRtByEX,
   output reg vote_EX_KernelMode,
   output reg [31:0] vote_EX_RestartPC,
   output reg vote_EX_IsBDS,
   output reg vote_EX_Trap,
   output reg vote_EX_TrapCond,
   output reg vote_EX_EX_CanErr,
   output reg vote_EX_M_CanErr,
   output reg [31:0] vote_EX_ReadData1,
   output reg [31:0] vote_EX_ReadData2
   );

/***
   The purpose of a pipeline register is to capture data from one pipeline stage     and
provide it to the next pipeline stage. This creates at least one clock cycle     of
delay, but reduces the combinatorial path length of signals which allows for     higher
clock speeds.

   All pipeline registers update unless the forward stage is stalled. When this occurs or
when the current stage is being flushed, the forward stage will receive data that is
effectively a NOP and causes nothing to happen throughout the remaining pipeline
traversal. In other words:

   A stall masks all control signals to forward stages. A flush permanently clears
control signals to forward stages (but not certain data for exception purposes).
***/

   assign EX_LinkRegDst = (EX_Link) ? 2'b10 : ((EX_RegDst) ? 2'b01 : 2'b00);
   assign EX_Rd = EX_SignExtImm[15:11];
```

```
   assign EX_Shamt = EX_SignExtImm[10:6];
   assign EX_SignExtImm = (EX_SignExtImm_pre[16]) ? {15'h7fff,
                          EX_SignExtImm_pre[16:0]} : {15'h0000,
                          EX_SignExtImm_pre[16:0]};

   always @(posedge clock) begin
      vote_EX_Link <= (reset) ? 1'b0 : ((EX_Stall) ? EX_Link : ID_Link);
      vote_EX_RegDst <= (reset) ? 1'b0 : ((EX_Stall) ? EX_RegDst : ID_RegDst);
      vote_EX_ALUSrcImm <= (reset) ? 1'b0  : ((EX_Stall) ? EX_ALUSrcImm :
                           ID_ALUSrcImm);
      vote_EX_ALUOp <= (reset) ? 5'b0 : ((EX_Stall) ? EX_ALUOp : ((ID_Stall |
                       ID_Flush) ? 5'b0 : ID_ALUOp));
      vote_EX_Movn <= (reset) ? 1'b0  : ((EX_Stall) ? EX_Movn : ID_Movn);
      vote_EX_Movz <= (reset) ? 1'b0  : ((EX_Stall) ? EX_Movz : ID_Movz);
      vote_EX_LLSC <= (reset) ? 1'b0  : ((EX_Stall) ? EX_LLSC : ID_LLSC);
      vote_EX_MemRead <= (reset) ? 1'b0  : ((EX_Stall) ? EX_MemRead : ((ID_Stall |
                         ID_Flush) ? 1'b0 : ID_MemRead));
      vote_EX_MemWrite <= (reset) ? 1'b0 : ((EX_Stall) ? EX_MemWrite : ((ID_Stall
                          | ID_Flush) ? 1'b0 : ID_MemWrite));
      vote_EX_MemByte <= (reset) ? 1'b0 : ((EX_Stall) ? EX_MemByte : ID_MemByte);
      vote_EX_MemHalf <= (reset) ? 1'b0  : ((EX_Stall) ? EX_MemHalf : ID_MemHalf);
      vote_EX_MemSignExtend <= (reset) ? 1'b0 : ((EX_Stall) ? EX_MemSignExtend :
                               ID_MemSignExtend);
      vote_EX_Left <= (reset) ? 1'b0 : ((EX_Stall) ? EX_Left : ID_Left);
      vote_EX_Right <= (reset) ? 1'b0  : ((EX_Stall) ? EX_Right : ID_Right);
      vote_EX_RegWrite <= (reset) ? 1'b0  : ((EX_Stall) ? EX_RegWrite : ((ID_Stall
                          | ID_Flush) ? 1'b0 : ID_RegWrite));
      vote_EX_MemtoReg <= (reset) ? 1'b0 : ((EX_Stall) ? EX_MemtoReg :
                          ID_MemtoReg);
      vote_EX_ReverseEndian  <= (reset) ? 1'b0 : ((EX_Stall) ? EX_ReverseEndian :
                                ID_ReverseEndian);
      vote_EX_RestartPC <= (reset) ? 32'b0 : ((EX_Stall) ? EX_RestartPC :
                           ID_RestartPC);
      vote_EX_IsBDS <= (reset) ? 1'b0 : ((EX_Stall) ? EX_IsBDS : ID_IsBDS);
      vote_EX_Trap <= (reset) ? 1'b0  : ((EX_Stall) ? EX_Trap : ((ID_Stall |
                      ID_Flush) ? 1'b0 : ID_Trap));
      vote_EX_TrapCond <= (reset) ? 1'b0 : ((EX_Stall) ? EX_TrapCond :
                          ID_TrapCond);
      vote_EX_EX_CanErr <= (reset) ? 1'b0 : ((EX_Stall) ? EX_EX_CanErr :
                           ((ID_Stall | ID_Flush) ? 1'b0 : ID_EX_CanErr));
      vote_EX_M_CanErr <= (reset) ? 1'b0 : ((EX_Stall) ? EX_M_CanErr : ((ID_Stall
                          | ID_Flush) ? 1'b0 : ID_M_CanErr));
      vote_EX_ReadData1 <= (reset) ? 32'b0 : ((EX_Stall) ? EX_ReadData1 :
                           ID_ReadData1);
      vote_EX_ReadData2 <= (reset) ? 32'b0 : ((EX_Stall) ? EX_ReadData2 :
                           ID_ReadData2);
      vote_EX_SignExtImm_pre <= (reset) ? 17'b0 : ((EX_Stall) ?
                                EX_SignExtImm_pre : ID_SignExtImm);
      vote_EX_Rs <= (reset) ? 5'b0  : ((EX_Stall) ? EX_Rs : ID_Rs);
      vote_EX_Rt <= (reset) ? 5'b0  : ((EX_Stall) ? EX_Rt : ID_Rt);
      vote_EX_WantRsByEX <= (reset) ? 1'b0 : ((EX_Stall) ? EX_WantRsByEX :
                            ((ID_Stall | ID_Flush) ? 1'b0 : ID_WantRsByEX));
      vote_EX_NeedRsByEX <= (reset) ? 1'b0 : ((EX_Stall) ? EX_NeedRsByEX :
                            ((ID_Stall | ID_Flush) ? 1'b0 : ID_NeedRsByEX));
      vote_EX_WantRtByEX <= (reset) ? 1'b0 : ((EX_Stall) ? EX_WantRtByEX :
                            ((ID_Stall | ID_Flush) ? 1'b0 : ID_WantRtByEX));
      vote_EX_NeedRtByEX <= (reset) ? 1'b0 : ((EX_Stall) ? EX_NeedRtByEX :
                            ((ID_Stall | ID_Flush) ? 1'b0 : ID_NeedRtByEX));
      vote_EX_KernelMode <= (reset) ? 1'b0 : ((EX_Stall) ? EX_KernelMode :
                            ID_KernelMode);
   end

endmodule
```

(14) ALU.v

```
`timescale 1ns / 1ps
/*
```

```
 * File         : ALU.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   7-Jun-2011   GEA       Initial design.
 *   2.0   26-Jul-2012  GEA       Many changes have been made.
 *   3.0   1-Mar-2016   ASJ       Voting Signals added to all registers.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   An Arithmetic Logic Unit for a MIPS32 processor. This module computes all
 *   arithmetic operations, including the following:
 *
 *   Add, Subtract, Multiply, And, Or, Nor, Xor, Shift, Count leading 1s/0s.
 *
 *   Multiply implemented using XILINX IP core with the following settings:
 *   Data Type Both Signed or Unsigned, Width 32, Use Multipliers, 0 pipeline stages.
 */
module ALU(
   input  clock,
   input  reset,
   input  EX_Stall,
   input  EX_Flush,
   input  [31:0] A, B,
   input  [4:0]  Operation,
   input  signed [4:0] Shamt,
   output reg signed [31:0] Result,
   output BZero,           // Used for Movc
   output reg EXC_Ov,
   output ALU_Stall,       // Stalls due to long ALU operations
   //Voter Signals for Registers
   input [63:0] HILO,
   input div_fsm,
   output reg [63:0] vote_HILO,
   output reg vote_div_fsm,
   input  active,
   input  neg,
   input  [31:0] div_result,
   input  [31:0] denom,
   input  [31:0] work,
   output vote_active,
   output vote_neg,
   output [31:0] vote_div_result,
   output [31:0] vote_denom,
   output [31:0] vote_work
   );

   `include “MIPS_Parameters.v”

/***
Performance Notes:
   The ALU is the longest delay path in the Execute stage, and one of the longest     in
the entire processor. This path varies based on the logic blocks that are chosen to
implement various functions, but there is certainly room to improve the speed of
arithmetic operations. The ALU could also be placed in a separate pipeline stage after
the Execute forwarding has completed.
***/

/***
Divider Logic:
  The hardware divider requires 32 cycles to complete. Because it writes its results to
HILO and not to the pipeline, the pipeline can proceed without stalling. When a later
instruction tries to access HILO, the pipeline will stall if the divide operation has not
yet completed.
***/
```

```
// Internal state registers
reg  HILO_Access;                   // Behavioral; not DFFs
reg  [5:0] CLO_Result, CLZ_Result;  // Behavioral; not DFFs

// Internal signals
wire [31:0] HI, LO;
wire HILO_Commit;
wire signed [31:0] As, Bs;
wire AddSub_Add;
wire signed [31:0] AddSub_Result;
wire signed [63:0] Mult_Result;
wire [63:0] Multu_Result;
wire [31:0] Quotient;
wire [31:0] Remainder;
wire Div_Stall;
wire Div_Start, Divu_Start;
wire DivOp;
wire Div_Commit;

// Assignments
assign HI = HILO[63:32];
assign LO = HILO[31:0];
assign HILO_Commit = ~(EX_Stall | EX_Flush);
assign As = A;
assign Bs = B;
assign AddSub_Add = ((Operation == `AluOp_Add) | (Operation == `AluOp_Addu));
assign AddSub_Result = (AddSub_Add) ? (A + B) : (A - B);
Signed_Mult Signed_Multiplier (
   .a      (As),
   .b      (Bs),
   .p      (Mult_Result));
Unsigned_Mult Unsigned_Multiplier (
   .a      (A),
   .b      (B),
   .p      (Multu_Result));
assign BZero = (B == 32'h00000000);
assign DivOp = (Operation == `AluOp_Div) || (Operation == `AluOp_Divu);
assign Div_Commit = (div_fsm == 1'b1) && (Div_Stall == 1'b0);
assign Div_Start = (div_fsm == 1'b0) && (Operation == `AluOp_Div)  &&
                   (HILO_Commit == 1'b1);
assign Divu_Start = (div_fsm == 1'b0) && (Operation == `AluOp_Divu) &&
                    (HILO_Commit == 1'b1);
assign ALU_Stall = (div_fsm == 1'b1) && (HILO_Access == 1'b1);

always @(*) begin
   case (Operation)
      `AluOp_Add  : Result <= AddSub_Result;
      `AluOp_Addu : Result <= AddSub_Result;
      `AluOp_And  : Result <= A & B;
      `AluOp_Clo  : Result <= {26'b0, CLO_Result};
      `AluOp_Clz  : Result <= {26'b0, CLZ_Result};
      `AluOp_Mfhi : Result <= HI;
      `AluOp_Mflo : Result <= LO;
      `AluOp_Mul  : Result <= Mult_Result[31:0];
      `AluOp_Nor  : Result <= ~(A | B);
      `AluOp_Or   : Result <= A | B;
      `AluOp_Sll  : Result <= B << Shamt;
      `AluOp_Sllc : Result <= {B[15:0], 16'b0};
      `AluOp_Sllv : Result <= B << A[4:0];
      `AluOp_Slt  : Result <= (As < Bs) ? 32'h00000001 : 32'h00000000;
      `AluOp_Sltu : Result <= (A < B)   ? 32'h00000001 : 32'h00000000;
      `AluOp_Sra  : Result <= Bs >>> Shamt;
      `AluOp_Srav : Result <= Bs >>> As[4:0];
      `AluOp_Srl  : Result <= B >> Shamt;
      `AluOp_Srlv : Result <= B >> A[4:0];
      `AluOp_Sub  : Result <= AddSub_Result;
      `AluOp_Subu : Result <= AddSub_Result;
```

```
      `AluOp_Xor  : Result <= A ^ B;
      default     : Result <= 32'hxxxx_xxxx;
   endcase
end


always @(posedge clock) begin
   if (reset) begin
      vote_HILO <= 64'h00000000_00000000;
   end
   else if (Div_Commit) begin
      vote_HILO <= {Remainder, Quotient};
   end
   else if (HILO_Commit) begin
      case (Operation)
         `AluOp_Mult  : vote_HILO <= Mult_Result;
         `AluOp_Multu : vote_HILO <= Multu_Result;
         `AluOp_Madd  : vote_HILO <= HILO + Mult_Result;
         `AluOp_Maddu : vote_HILO <= HILO + Multu_Result;
         `AluOp_Msub  : vote_HILO <= HILO - Mult_Result;
         `AluOp_Msubu : vote_HILO <= HILO - Multu_Result;
         `AluOp_Mthi  : vote_HILO <= {A, LO};
         `AluOp_Mtlo  : vote_HILO <= {HI, B};
         default      : vote_HILO <= HILO;
      endcase
   end
   else begin
      vote_HILO <= HILO;
   end
end

// Detect accesses to HILO. RAW and WAW hazards are possible while a
// divide operation is computing, so reads and writes to HILO must stall
// while the divider is busy.
// (This logic could be put into an earlier pipeline stage or into the
// datapath bits to improve timing.)
always @(Operation) begin
   case (Operation)
      `AluOp_Div   : HILO_Access <= 1;
      `AluOp_Divu  : HILO_Access <= 1;
      `AluOp_Mfhi  : HILO_Access <= 1;
      `AluOp_Mflo  : HILO_Access <= 1;
      `AluOp_Mult  : HILO_Access <= 1;
      `AluOp_Multu : HILO_Access <= 1;
      `AluOp_Madd  : HILO_Access <= 1;
      `AluOp_Maddu : HILO_Access <= 1;
      `AluOp_Msub  : HILO_Access <= 1;
      `AluOp_Msubu : HILO_Access <= 1;
      `AluOp_Mthi  : HILO_Access <= 1;
      `AluOp_Mtlo  : HILO_Access <= 1;
      default      : HILO_Access <= 0;
   endcase
end

// Divider FSM: The divide unit is either available or busy.
always @(posedge clock) begin
   if (reset) begin
      vote_div_fsm <= 1'd0;
   end
   else begin
      case (div_fsm)
         1'd0 : vote_div_fsm <= (DivOp & HILO_Commit) ? 1'd1 : 1'd0;
         1'd1 : vote_div_fsm <= (~Div_Stall) ? 1'd0 : 1'd1;
      endcase
   end
end

// Detect overflow for signed operations. Note that MIPS32 has no overflow
// detection for multiplication/division operations.
```

```
always @(*) begin
   case (Operation)
      `AluOp_Add : EXC_Ov <= ((A[31] ~^ B[31]) & (A[31] ^ AddSub_Result[31]));
      `AluOp_Sub : EXC_Ov <= ((A[31]  ^ B[31]) & (A[31] ^ AddSub_Result[31]));
      default    : EXC_Ov <= 0;
   endcase
end

// Count Leading Ones
always @(A) begin
   casex (A)
      32'b0xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd0;
      32'b10xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd1;
      32'b110x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd2;
      32'b1110_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd3;
      32'b1111_0xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd4;
      32'b1111_10xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd5;
      32'b1111_110x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd6;
      32'b1111_1110_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd7;
      32'b1111_1111_0xxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd8;
      32'b1111_1111_10xx_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd9;
      32'b1111_1111_110x_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd10;
      32'b1111_1111_1110_xxxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd11;
      32'b1111_1111_1111_0xxx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd12;
      32'b1111_1111_1111_10xx_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd13;
      32'b1111_1111_1111_110x_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd14;
      32'b1111_1111_1111_1110_xxxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd15;
      32'b1111_1111_1111_1111_0xxx_xxxx_xxxx_xxxx : CLO_Result <= 6'd16;
      32'b1111_1111_1111_1111_10xx_xxxx_xxxx_xxxx : CLO_Result <= 6'd17;
      32'b1111_1111_1111_1111_110x_xxxx_xxxx_xxxx : CLO_Result <= 6'd18;
      32'b1111_1111_1111_1111_1110_xxxx_xxxx_xxxx : CLO_Result <= 6'd19;
      32'b1111_1111_1111_1111_1111_0xxx_xxxx_xxxx : CLO_Result <= 6'd20;
      32'b1111_1111_1111_1111_1111_10xx_xxxx_xxxx : CLO_Result <= 6'd21;
      32'b1111_1111_1111_1111_1111_110x_xxxx_xxxx : CLO_Result <= 6'd22;
      32'b1111_1111_1111_1111_1111_1110_xxxx_xxxx : CLO_Result <= 6'd23;
      32'b1111_1111_1111_1111_1111_1111_0xxx_xxxx : CLO_Result <= 6'd24;
      32'b1111_1111_1111_1111_1111_1111_10xx_xxxx : CLO_Result <= 6'd25;
      32'b1111_1111_1111_1111_1111_1111_110x_xxxx : CLO_Result <= 6'd26;
      32'b1111_1111_1111_1111_1111_1111_1110_xxxx : CLO_Result <= 6'd27;
      32'b1111_1111_1111_1111_1111_1111_1111_0xxx : CLO_Result <= 6'd28;
      32'b1111_1111_1111_1111_1111_1111_1111_10xx : CLO_Result <= 6'd29;
      32'b1111_1111_1111_1111_1111_1111_1111_110x : CLO_Result <= 6'd30;
      32'b1111_1111_1111_1111_1111_1111_1111_1110 : CLO_Result <= 6'd31;
      32'b1111_1111_1111_1111_1111_1111_1111_1111 : CLO_Result <= 6'd32;
      default : CLO_Result <= 6'd0;
   endcase
end

// Count Leading Zeros
always @(A) begin
   casex (A)
      32'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd0;
      32'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd1;
      32'b001x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd2;
      32'b0001_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd3;
      32'b0000_1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd4;
      32'b0000_01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd5;
      32'b0000_001x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd6;
      32'b0000_0001_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd7;
      32'b0000_0000_1xxx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd8;
      32'b0000_0000_01xx_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd9;
      32'b0000_0000_001x_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd10;
      32'b0000_0000_0001_xxxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd11;
      32'b0000_0000_0000_1xxx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd12;
      32'b0000_0000_0000_01xx_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd13;
      32'b0000_0000_0000_001x_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd14;
      32'b0000_0000_0000_0001_xxxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd15;
      32'b0000_0000_0000_0000_1xxx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd16;
      32'b0000_0000_0000_0000_01xx_xxxx_xxxx_xxxx : CLZ_Result <= 6'd17;
```

```
         32'b0000_0000_0000_0000_001x_xxxx_xxxx_xxxx : CLZ_Result <= 6'd18;
         32'b0000_0000_0000_0000_0001_xxxx_xxxx_xxxx : CLZ_Result <= 6'd19;
         32'b0000_0000_0000_0000_0000_1xxx_xxxx_xxxx : CLZ_Result <= 6'd20;
         32'b0000_0000_0000_0000_0000_01xx_xxxx_xxxx : CLZ_Result <= 6'd21;
         32'b0000_0000_0000_0000_0000_001x_xxxx_xxxx : CLZ_Result <= 6'd22;
         32'b0000_0000_0000_0000_0000_0001_xxxx_xxxx : CLZ_Result <= 6'd23;
         32'b0000_0000_0000_0000_0000_0000_1xxx_xxxx : CLZ_Result <= 6'd24;
         32'b0000_0000_0000_0000_0000_0000_01xx_xxxx : CLZ_Result <= 6'd25;
         32'b0000_0000_0000_0000_0000_0000_001x_xxxx : CLZ_Result <= 6'd26;
         32'b0000_0000_0000_0000_0000_0000_0001_xxxx : CLZ_Result <= 6'd27;
         32'b0000_0000_0000_0000_0000_0000_0000_1xxx : CLZ_Result <= 6'd28;
         32'b0000_0000_0000_0000_0000_0000_0000_01xx : CLZ_Result <= 6'd29;
         32'b0000_0000_0000_0000_0000_0000_0000_001x : CLZ_Result <= 6'd30;
         32'b0000_0000_0000_0000_0000_0000_0000_0001 : CLZ_Result <= 6'd31;
         32'b0000_0000_0000_0000_0000_0000_0000_0000 : CLZ_Result <= 6'd32;
         default : CLZ_Result <= 6'd0;
      endcase
   end

   // Multicycle divide unit
   Divide Divider (
      .clock       (clock),
      .reset       (reset),
      .OP_div      (Div_Start),
      .OP_divu     (Divu_Start),
      .Dividend    (A),
      .Divisor     (B),
      .Quotient    (Quotient),
      .Remainder   (Remainder),
      .Stall       (Div_Stall),
      .active      (active),
      .neg         (neg),
      .result      (div_result),
      .denom       (denom),
      .work        (work),
      .vote_active (vote_active),
      .vote_neg    (vote_neg),
      .vote_result (vote_div_result),
      .vote_denom  (vote_denom),
      .vote_work   (vote_work));

endmodule
```

(15) Divide.v

```
`timescale 1ns / 1ns
/*
 * File         : Divide.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Neil Russell
 *
 * Modification History:
 *  Rev   Date         Initials  Description of Change
 *  1.0   6-Nov-2012   NJR       Initial design.
 *  2.0   1-Mar-2016   ASJ       Voting signals added to all registers
 *
 * Description:
 *  A multi-cycle 32-bit divider.
 *
 *  On any cycle that one of OP_div or OP_divu are true, the Dividend and
 *  Divisor will be captured and a multi-cycle divide operation initiated.
 *  Stall will go true on the next cycle and the first cycle of the divide
 *  operation completed. After some time (about 32 cycles), Stall will go
 *  false on the same cycle that the result becomes valid. OP_div or OP_divu
 *  will abort any currently running divide operation and initiate a new one.
 */
```

```
module Divide(
   input  clock,
   input  reset,
   input  OP_div,               // True to initiate a signed divide
   input  OP_divu,              // True to initiate an unsigned divide
   input  [31:0] Dividend,
   input  [31:0] Divisor,
   output [31:0] Quotient,
   output [31:0] Remainder,
   output Stall,                        // True while calculating
   //Voter Signals for Registers
   input  active,               // True if the divider is running
   input  neg,                  // True if the result will be negative
   input  [31:0] result,            // Begin with dividend, end with quotient
   input  [31:0] denom,             // Divisor
   input  [31:0] work,
   output reg  vote_active,         // True if the divider is running
   output reg  vote_neg,            // True if the result will be negative
   output reg  [31:0] vote_result,  // Begin with dividend, end with quotient
   output reg  [31:0] vote_denom,   // Divisor
   output reg  [31:0] vote_work
   );

   reg [4:0] cycle;      // Number of cycles to go

   // Calculate the current digit
   wire [32:0] sub = { work[30:0], result[31] } - denom;

   // Send the results to our master
   assign Quotient = !neg ? result : -result;
   assign Remainder = work;
   assign Stall = active;

   // The state machine
   always @(posedge clock) begin
      if (reset) begin
         vote_active <= 0;
         vote_neg <= 0;
         cycle <= 0;
         vote_result <= 0;
         vote_denom <= 0;
         vote_work <= 0;
      end
      else begin
         if (OP_div) begin
            // Set up for a signed divide. Remember the resulting sign,
            // and make the operands positive.
            cycle <= 5'd31;
            vote_result <= (Dividend[31] == 0) ? Dividend : -Dividend;
            vote_denom <= (Divisor[31] == 0) ? Divisor : -Divisor;
            vote_work <= 32'b0;
            vote_neg <= Dividend[31] ^ Divisor[31];
            vote_active <= 1;
         end
         else if (OP_divu) begin
            // Set up for an unsigned divide.
            cycle <= 5'd31;
            vote_result <= Dividend;
            vote_denom <= Divisor;
            vote_work <= 32'b0;
            vote_neg <= 0;
            vote_active <= 1;
         end
         else if (active) begin
            // Run an iteration of the divide.
            if (sub[32] == 0) begin
               vote_work <= sub[31:0];
               vote_result <= {result[30:0], 1'b1};
            end
```

```
            else begin
               vote_work <= {work[30:0], result[31]};
               vote_result <= {result[30:0], 1'b0};
            end
            if (cycle == 0) begin
               vote_active <= 0;
            end
            cycle <= cycle - 5'd1;
         end
      end
   end
endmodule
```

(16) EXMEM_Stage

```
`timescale 1ns / 1ps
/*
 * File         : EXMEM_Stage.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   9-Jun-2011   GEA       Initial design.
 *   2.0  26-Jul-2012   GEA       Many updates have been made.
 *   3.0   1-Mar-2016   ASJ       Voting signals added to all registers
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   The Pipeline Register to bridge the Execute and Memory stages.
 */
module EXMEM_Stage(
   input  clock,
   input  reset,
   input  EX_Flush,
   input  EX_Stall,
   input  M_Stall,
   // Control Signals
   input  EX_Movn,
   input  EX_Movz,
   input  EX_BZero,
   input  EX_RegWrite,  // Future Control to WB
   input  EX_MemtoReg,  // Future Control to WB
   input  EX_ReverseEndian,
   input  EX_LLSC,
   input  EX_MemRead,
   input  EX_MemWrite,
   input  EX_MemByte,
   input  EX_MemHalf,
   input  EX_MemSignExtend,
   input  EX_Left,
   input  EX_Right,
   // Exception Control/Info
   input  EX_KernelMode,
   input  [31:0] EX_RestartPC,
   input  EX_IsBDS,
   input  EX_Trap,
   input  EX_TrapCond,
   input  EX_M_CanErr,
   // Data Signals
   input  [31:0] EX_ALU_Result,
   input  [31:0] EX_ReadData2,
   input  [4:0]  EX_RtRd,
   // Voter Signals for Registers
   input M_RegWrite,
   input M_MemtoReg,
```

```
   input M_ReverseEndian,
   input M_LLSC,
   input M_MemRead,
   input M_MemWrite,
   input M_MemByte,
   input M_MemHalf,
   input M_MemSignExtend,
   input M_Left,
   input M_Right,
   input M_KernelMode,
   input [31:0] M_RestartPC,
   input M_IsBDS,
   input M_Trap,
   input M_TrapCond,
   input M_M_CanErr,
   input [31:0] M_ALU_Result,
   input [31:0] M_ReadData2,
   input [4:0]  M_RtRd,
   output reg vote_M_RegWrite,
   output reg vote_M_MemtoReg,
   output reg vote_M_ReverseEndian,
   output reg vote_M_LLSC,
   output reg vote_M_MemRead,
   output reg vote_M_MemWrite,
   output reg vote_M_MemByte,
   output reg vote_M_MemHalf,
   output reg vote_M_MemSignExtend,
   output reg vote_M_Left,
   output reg vote_M_Right,
   output reg vote_M_KernelMode,
   output reg [31:0] vote_M_RestartPC,
   output reg vote_M_IsBDS,
   output reg vote_M_Trap,
   output reg vote_M_TrapCond,
   output reg vote_M_M_CanErr,
   output reg [31:0] vote_M_ALU_Result,
   output reg [31:0] vote_M_ReadData2,
   output reg [4:0]  vote_M_RtRd
   );

/***
   The purpose of a pipeline register is to capture data from one pipeline stage and
provide it to the next pipeline stage. This creates at least one clock cycle of delay,
but reduces the combinatorial path length of signals which allows for higher clock
speeds.

   All pipeline registers update unless the forward stage is stalled. When this occurs or
when the current stage is being flushed, the forward stage will receive data that is
effectively a NOP and causes nothing to happen throughout the remaining pipeline
traversal. In other words:

     A stall masks all control signals to forward stages. A flush permanently clears
control signals to forward stages (but not certain data for exception purposes).
***/

   // Mask of RegWrite if a Move Conditional failed.
   wire MovcRegWrite = (EX_Movn & ~EX_BZero) | (EX_Movz & EX_BZero);

   always @(posedge clock) begin
      vote_M_RegWrite <= (reset) ? 1'b0 : ((M_Stall) ? M_RegWrite : ((EX_Stall |
                         EX_Flush) ? 1'b0 : ((EX_Movn | EX_Movz) ? MovcRegWrite :
                         EX_RegWrite)));
      vote_M_MemtoReg <= (reset) ? 1'b0 : ((M_Stall) ? M_MemtoReg : EX_MemtoReg);
      vote_M_ReverseEndian <= (reset) ? 1'b0 : ((M_Stall) ? M_ReverseEndian :
                              EX_ReverseEndian);
      vote_M_LLSC <= (reset) ? 1'b0 : ((M_Stall) ? M_LLSC : EX_LLSC);
      vote_M_MemRead <= (reset) ? 1'b0 : ((M_Stall) ? M_MemRead : ((EX_Stall |
                        EX_Flush) ? 1'b0 : EX_MemRead));
      vote_M_MemWrite <= (reset) ? 1'b0 : ((M_Stall) ? M_MemWrite : ((EX_Stall |
```

```
                        EX_Flush) ? 1'b0 : EX_MemWrite));
      vote_M_MemByte <= (reset) ? 1'b0 : ((M_Stall) ? M_MemByte : EX_MemByte);
      vote_M_MemHalf <= (reset) ? 1'b0 : ((M_Stall) ? M_MemHalf : EX_MemHalf);
      vote_M_MemSignExtend <= (reset) ? 1'b0 : ((M_Stall) ? M_MemSignExtend :
                              EX_MemSignExtend);
      vote_M_Left <= (reset) ? 1'b0 : ((M_Stall) ? M_Left : EX_Left);
      vote_M_Right <= (reset) ? 1'b0 : ((M_Stall) ? M_Right : EX_Right);
      vote_M_KernelMode <= (reset) ? 1'b0 : ((M_Stall) ? M_KernelMode :
                           EX_KernelMode);
      vote_M_RestartPC <= (reset) ? 32'b0 : ((M_Stall) ? M_RestartPC :
                          EX_RestartPC);
      vote_M_IsBDS <= (reset) ? 1'b0 : ((M_Stall) ? M_IsBDS : EX_IsBDS);
      vote_M_Trap <= (reset) ? 1'b0 : ((M_Stall) ? M_Trap : ((EX_Stall |
                     EX_Flush) ? 1'b0 : EX_Trap));
      vote_M_TrapCond <= (reset) ? 1'b0 : ((M_Stall) ? M_TrapCond : EX_TrapCond);
      vote_M_M_CanErr <= (reset) ? 1'b0 : ((M_Stall) ? M_M_CanErr : ((EX_Stall |
                         EX_Flush) ? 1'b0 : EX_M_CanErr));
      vote_M_ALU_Result <= (reset) ? 32'b0 : ((M_Stall) ? M_ALU_Result :
                           EX_ALU_Result);
      vote_M_ReadData2 <= (reset) ? 32'b0 : ((M_Stall) ? M_ReadData2 :
                          EX_ReadData2);
      vote_M_RtRd <= (reset) ? 5'b0 : ((M_Stall) ? M_RtRd : EX_RtRd);
   end

endmodule
```

(17) TrapDetect.v

```
`timescale 1ns / 1ps
/*
 * File         : TrapDetect.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   15-May-2012  GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Detects a Trap Exception in the pipeline.
 */
module TrapDetect(
   input  Trap,
   input  TrapCond,
   input  [31:0] ALUResult,
   output EXC_Tr
   );

   wire ALUZero = (ALUResult == 32'h00000000);
   assign EXC_Tr = Trap & (TrapCond ^ ALUZero);

endmodule
```

(18) MemControl.v

```
`timescale 1ns / 1ps
/*
 * File         : MemControl.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   24-Jun-2011  GEA       Initial design.
```

```
 *   2.0   28-Jun-2012  GEA       Expanded from a simple byte/half/word unit to
 *                                An advanced data memory controller capable of
 *                                handling big/little endian, atomic and unaligned
 *                                memory accesses.
 *   3.0    1-Mar-2016  ASJ       Voting signals added to all registers.
 *                                M_Stall modified for L1 cache access.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A Data Memory Controller which handles all read and write requests from the
 *   processor to data memory. All data accesses--whether big endian, little
 *   endian, byte, half, word, or unaligned transfers--are transformed into a
 *   simple read and write command to data memory over a 32-bit data bus, where
 *   the read command is one bit and the write command is 4 bits, one for each
 *   byte in the 32-bit word.
 */

module MemControl(
   input  clock,
   input  reset,
   input  [31:0] DataIn,           // Data from CPU
   input  [31:0] Address,          // From CPU
   input  [31:0] MReadData,        // Data from Memory
   input  MemRead,                 // Memory Read command from CPU
   input  MemWrite,                // Memory Write command from CPU
   input  DataMem_Ready,           // Ready signal from Memory
   input  Byte,                    // Load/Store is Byte (8-bit)
   input  Half,                    // Load/Store is Half (16-bit)
   input  SignExtend,              // Sub-word load should be sign extended
   input  KernelMode,              // (Exception logic)
   input  ReverseEndian,           // Reverse Endian Memory for User Mode
   input  LLSC,                    // (LLSC logic)
   input  ERET,                    // (LLSC logic)
   input  Left,                    // Unaligned Load/Store Word Left
   input  Right,                   // Unaligned Load/Store Word Right
   input  M_Exception_Stall,
   input  IF_Stall,                // XXX Clean this up between this module and
                                   // HAZ/FWD
   output reg [31:0] DataOut,      // Data to CPU
   output [31:0] MWriteData,       // Data to Memory
   output reg [3:0] WriteEnable,   // Write Enable to Memory for each of 4 bytes
                                   // of Memory
   output ReadEnable,              // Read Enable to Memory
   output M_Stall,
   output EXC_AdEL,                // Load Exception
   output EXC_AdES,                // Store Exception
   // Voter Signals for Registers
   input  [29:0] LLSC_Address,
   input  LLSC_Atomic,
   input  RW_Mask,
   output reg [29:0] vote_LLSC_Address,
   output reg vote_LLSC_Atomic
   );

   `include “MIPS_Parameters.v”

   /*** Reverse Endian Mode
   Normal memory accesses in the processor are Big Endian. The endianness can be
   reversed to Little Endian in User Mode only.
   ***/
   wire BE = KernelMode | ~ReverseEndian;

   /*** Indicator that the current memory reference must be word-aligned ***/
   wire Word = ~(Half | Byte | Left | Right);

   // Exception Detection
   wire EXC_KernelMem = ~KernelMode & (Address < `UMem_Lower);
```

```
   wire EXC_Word = Word & (Address[1] | Address[0]);
   wire EXC_Half = Half & Address[0];
   assign EXC_AdEL = MemRead  & (EXC_KernelMem | EXC_Word | EXC_Half);
   assign EXC_AdES = MemWrite & (EXC_KernelMem | EXC_Word | EXC_Half);

   /*** Load Linked and Store Conditional logic ***

   A 32-bit register keeps track of the address for atomic Load Linked / Store
Conditional operations. This register can be updated during stalls since it is not
visible to forward stages. It does not need to be flushed during exceptions, since ERET
destroys the atomicity condition and there are no detrimental effects in an exception
handler.

   The atomic condition is set with a Load Linked instruction, and cleared on an ERET
instruction or when any store instruction writes to one or more bytes covered by the word
address register. It does not update on a stall condition.

   The MIPS32 spec states that an ERET instruction between LL and SC will cause the
atomicity condition to fail. This implementation uses the ERET signal from the ID stage,
which means instruction sequences such as “LL SC” could appear to have an ERET
instruction between them even though they don’t. One way to fix this is to pass the ERET
signal through the pipeline to the MEM stage. However, because of the nature of LL/SC
operations (they occur in a loop which checks the result at each iteration), an ERET will
normally never be inserted into the pipeline programmatically until the LL/SC sequence
has completed (exceptions such as interrupts can still cause ERET, but they can still
cause them in the LL SC sequence as well). In other words, by not passing ERET through
the pipeline, the only possible effect is a performance penalty. Also this may be
irrelevant since currently ERET stalls for forward stages which can cause exceptions,
which includes LL and SC.
*/

   wire LLSC_MemWrite_Mask;

   always @(posedge clock) begin
      vote_LLSC_Address <= (reset) ? 30'b0 : (MemRead & LLSC) ? Address[31:2] :
                           LLSC_Address;
   end

   always @(posedge clock) begin
      if (reset) begin
         vote_LLSC_Atomic <= 1'b0;
      end
      else if (MemRead) begin
         vote_LLSC_Atomic <= (LLSC) ? 1'b1 : LLSC_Atomic;
      end
      else if (ERET | (~M_Stall & ~IF_Stall & MemWrite & (Address[31:2] ==
               LLSC_Address))) begin
         vote_LLSC_Atomic <= 1'b0;
      end
      else begin
         vote_LLSC_Atomic <= LLSC_Atomic;
      end
   end
   assign LLSC_MemWrite_Mask = (LLSC & MemWrite & (~LLSC_Atomic | (Address[31:2]
                               != LLSC_Address)));

   wire WriteCondition = MemWrite & ~(EXC_KernelMem | EXC_Word | EXC_Half) &
                         ~LLSC_MemWrite_Mask;
   wire ReadCondition  = MemRead  & ~(EXC_KernelMem | EXC_Word | EXC_Half);

   assign M_Stall = ((ReadEnable | (WriteEnable != 4'b0000)) && ~DataMem_Ready) |
                    M_Exception_Stall;
   assign ReadEnable  = ReadCondition;

   wire Half_Access_L  = (Address[1] ^  BE);
   wire Half_Access_R  = (Address[1] ~^ BE);
   wire Byte_Access_LL = Half_Access_L & (Address[1] ~^ Address[0]);
   wire Byte_Access_LM = Half_Access_L & (Address[0] ~^ BE);
   wire Byte_Access_RM = Half_Access_R & (Address[0] ^  BE);
```

```
wire Byte_Access_RR = Half_Access_R & (Address[1] ~^ Address[0]);

// Write-Enable Signals to Memory
always @(*) begin
   if (WriteCondition) begin
      if (Byte) begin
         WriteEnable[3] <= Byte_Access_LL;
         WriteEnable[2] <= Byte_Access_LM;
         WriteEnable[1] <= Byte_Access_RM;
         WriteEnable[0] <= Byte_Access_RR;
      end
      else if (Half) begin
         WriteEnable[3] <= Half_Access_L;
         WriteEnable[2] <= Half_Access_L;
         WriteEnable[1] <= Half_Access_R;
         WriteEnable[0] <= Half_Access_R;
      end
      else if (Left) begin
         case (Address[1:0])
            2'b00 : WriteEnable <= (BE) ? 4'b1111 : 4'b0001;
            2'b01 : WriteEnable <= (BE) ? 4'b0111 : 4'b0011;
            2'b10 : WriteEnable <= (BE) ? 4'b0011 : 4'b0111;
            2'b11 : WriteEnable <= (BE) ? 4'b0001 : 4'b1111;
         endcase
      end
      else if (Right) begin
         case (Address[1:0])
            2'b00 : WriteEnable <= (BE) ? 4'b1000 : 4'b1111;
            2'b01 : WriteEnable <= (BE) ? 4'b1100 : 4'b1110;
            2'b10 : WriteEnable <= (BE) ? 4'b1110 : 4'b1100;
            2'b11 : WriteEnable <= (BE) ? 4'b1111 : 4'b1000;
         endcase
      end
      else begin
         WriteEnable <= 4'b1111;
      end
   end
   else begin
      WriteEnable <= 4'b0000;
   end
end

// Data Going to Memory
assign MWriteData[31:24] = (Byte) ? DataIn[7:0] : ((Half) ? DataIn[15:8] :
                           DataIn[31:24]);
assign MWriteData[23:16] = (Byte | Half) ? DataIn[7:0] : DataIn[23:16];
assign MWriteData[15:8]  = (Byte) ? DataIn[7:0] : DataIn[15:8];
assign MWriteData[7:0]   = DataIn[7:0];

// Data Read from Memory
always @(*) begin
   if (Byte) begin
      if (Byte_Access_LL) begin
         DataOut <= (SignExtend & MReadData[31]) ? {24'hFFFFFF,
                    MReadData[31:24]} : {24'h000000, MReadData[31:24]};
      end
      else if (Byte_Access_LM) begin
         DataOut <= (SignExtend & MReadData[23]) ? {24'hFFFFFF,
                    MReadData[23:16]} : {24'h000000, MReadData[23:16]};
      end
      else if (Byte_Access_RM) begin
         DataOut <= (SignExtend & MReadData[15]) ? {24'hFFFFFF,
         MReadData[15:8]} : {24'h000000, MReadData[15:8]};
      end
      else begin
         DataOut <= (SignExtend & MReadData[7]) ? {24'hFFFFFF,
                    MReadData[7:0]} : {24'h000000, MReadData[7:0]};
      end
   end
```

```
      else if (Half) begin
         if (Half_Access_L) begin
            DataOut <= (SignExtend & MReadData[31]) ? {16'hFFFF,
                       MReadData[31:16]} : {16'h0000, MReadData[31:16]};
         end
         else begin
            DataOut <= (SignExtend & MReadData[15]) ? {16'hFFFF,
            MReadData[15:0]}  : {16'h0000, MReadData[15:0]};
         end
      end
      else if (LLSC & MemWrite) begin
         DataOut <= (LLSC_Atomic & (Address[31:2] == LLSC_Address)) ?
                    32'h0000_0001 : 32'h0000_0000;
      end
      else if (Left) begin
         case (Address[1:0])
            2'b00 : DataOut <= (BE) ? MReadData : {MReadData[7:0],  DataIn[23:0]};
            2'b01 : DataOut <= (BE) ? {MReadData[23:0], DataIn[7:0]} :
                    {MReadData[15:0], DataIn[15:0]};
            2'b10 : DataOut <= (BE) ? {MReadData[15:0], DataIn[15:0]} :
                    {MReadData[23:0], DataIn[7:0]};
            2'b11 : DataOut <= (BE) ? {MReadData[7:0],  DataIn[23:0]} :
                    MReadData;
         endcase
      end
      else if (Right) begin
         case (Address[1:0])
            2'b00 : DataOut <= (BE) ? {DataIn[31:8],  MReadData[31:24]} :
                    MReadData;
            2'b01 : DataOut <= (BE) ? {DataIn[31:16], MReadData[31:16]} :
                    {DataIn[31:24], MReadData[31:8]};
            2'b10 : DataOut <= (BE) ? {DataIn[31:24], MReadData[31:8]}  :
                    {DataIn[31:16], MReadData[31:16]};
            2'b11 : DataOut <= (BE) ? MReadData : {DataIn[31:8],
                    MReadData[31:24]};
         endcase
      end
      else begin
         DataOut <= MReadData;
      end
   end

endmodule
```

(19) MEMWB_Stage.v

```
`timescale 1ns / 1ps
/*
 * File         : MEMWB_Stage.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   9-Jun-2011   GEA       Initial design.
 *   2.0  26-Jul-2012   GEA       Many updates have been made.
 *   3.0   1-Mar-2016   ASJ       Voting signals added to all registers
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   The Pipeline Register to bridge the Memory and Writeback stages.
 */
module MEMWB_Stage(
   input  clock,
   input  reset,
   input  M_Flush,
```

```
   input  M_Stall,
   input  WB_Stall,
   // Control Signals
   input  M_RegWrite,
   input  M_MemtoReg,
   // Data Signals
   input  [31:0] M_ReadData,
   input  [31:0] M_ALU_Result,
   input  [4:0]  M_RtRd,
   // Voter Signals for Registers
   input WB_RegWrite,
   input WB_MemtoReg,
   input [31:0] WB_ReadData,
   input [31:0] WB_ALU_Result,
   input [4:0]  WB_RtRd,
   output reg vote_WB_RegWrite,
   output reg vote_WB_MemtoReg,
   output reg [31:0] vote_WB_ReadData,
   output reg [31:0] vote_WB_ALU_Result,
   output reg [4:0]  vote_WB_RtRd
   );


/***
   The purpose of a pipeline register is to capture data from one pipeline stage and
provide it to the next pipeline stage. This creates at least one clock cycle of delay,
but reduces the combinatorial path length of signals which allows for higher clock
speeds.

   All pipeline registers update unless the forward stage is stalled. When this occurs or
when the current stage is being flushed, the forward stage will receive data that is
effectively a NOP and causes nothing to happen throughout the remaining pipeline
traversal. In other words:

   A stall masks all control signals to forward stages. A flush permanently clears
control signals to forward stages (but not certain data for exception purposes).

   Since WB is the final stage in the pipeline, it would normally never stall. However,
because the MEM stage may be using data forwarded from WB, WB must stall when MEM is
stalled. If it didn't, the forward data would not be preserved. If the processor didn't
forward any data, a stall would not be needed.

   In practice, the only time WB stalls is when forwarding for a Lw->Sw sequence, since
MEM doesn't need the data until its stage, but it does not latch the forwarded data. This
means WB_Stall is probably identical to M_Stall. There is no speed difference by allowing
WB to stall.
***/

   always @(posedge clock) begin
      vote_WB_RegWrite <= (reset) ? 1'b0 : ((WB_Stall) ? WB_RegWrite : ((M_Stall |
                          M_Flush) ? 1'b0 : M_RegWrite));
      vote_WB_MemtoReg <= (reset) ? 1'b0 : ((WB_Stall) ? WB_MemtoReg :
                          M_MemtoReg);
      vote_WB_ReadData <= (reset) ? 32'b0 : ((WB_Stall) ? WB_ReadData :
                          M_ReadData);
      vote_WB_ALU_Result <= (reset) ? 32'b0 : ((WB_Stall) ? WB_ALU_Result :
                            M_ALU_Result);
      vote_WB_RtRd  <= (reset) ? 5'b0 : ((WB_Stall) ? WB_RtRd : M_RtRd);
   end

endmodule
```

b. Memory Hierarchy

Subsections one through twenty-three contain all the Verilog files to implement the entire memory hierarchy composed of the L1 caches, L2 cache controller, the bus arbiter, DDR controller, and SD Card controller.

(1) System_Memory.v

```
`timescale 1ns / 1ps
`default_nettype none
/*
 * File         : System_Memory.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module instantiates the major components of memory: L1 Data Cache,
 *   L1 Instruction Cache, Bus Arbiter, L2 Cache Controller and DDR Interface,
 *   and SD card controller.
 */

module System_Memory(
   input  wire clock,
   input  wire reset,
   input  wire Inst_Read_in,
   input  wire [3:0]  Inst_Write_in,
   input  wire [29:0] Inst_Address_in,
   input  wire [31:0] Inst_Data_in,
   output wire [31:0] Inst_Data_out,
   output wire Inst_Ready_out,
   input  wire flush,
   output wire flushcomplete,
   input  wire Data_Read_in,
   input  wire [3:0]  Data_Write_in,
   input  wire [29:0] Data_Address_in,
   input  wire [31:0] Data_Data_in,
   output wire [31:0] Data_Data_out,
   output wire Data_Ready_out,
   output wire Init_L1_Mem_out,
   output wire Init_L2_Mem_out,
   //DDR U/I Signals
   output wire [28:0] app_addr,
   output wire [2:0] app_cmd,
   output wire app_en,
   output wire [255:0] app_wdf_data,
   output wire app_wdf_end,
   output wire app_wdf_wren,
   input wire [255:0] app_rd_data,
   input wire app_rd_data_valid,
   input wire app_rdy,
   input wire app_wdf_rdy,
   input wire ui_clk,
   input wire ui_clk_sync_rst,
   input wire init_calib_complete,
```

```
   // SD Card Signals
   input wire [2:0] SD_MISO,
   output wire [2:0] SD_MOSI,
   output wire [2:0] SD_CS,
   input wire SD_SClk,
   output wire Select_SD_Clock_Speed
   );

parameter N = 3;
genvar i;

   wire [29:0] Inst_Address;
   wire Inst_Write, Inst_Read, Inst_Ready;
   wire [1023:0] Inst_Data_i, Inst_Data_o;

   wire [29:0] Data_Address;
   wire Data_Write, Data_Read, Data_Ready;
   wire [1023:0] Data_Data_i, Data_Data_o;

   wire [29:0] DDR_Address;
   wire DDR_Write, DDR_Read, DDR_Ready;
   wire [1023:0] DDR_Data_i, DDR_Data_o;

   wire [4095:0] SD_Data_o, SD_Data_i;
   wire SD_Write, SD_Read, SD_Ready;
   wire [29:0] SD_Address;

   wire L2_flush, L2_flushcomplete;

(* DONT_TOUCH = "TRUE" *) wire Vote_SD_Ready [N-1:0];
(* DONT_TOUCH = "TRUE" *) wire [4095:0] Vote_SD_Data_o [N-1:0];
(* DONT_TOUCH = "TRUE" *) wire Vote_Select_SD_Clock_Speed [N-1:0];

   wire Align [N-1:0];
   wire [2:0] Align_in;

   Level_1_Cache ICache(
      .Clock_in               (clock),
      .Reset_in               (reset),
      .CPU_Address_in         (Inst_Address_in),
      .CPU_Write_Data_in      (Inst_Write_in),
      .CPU_Data_in            (Inst_Data_in),
      .CPU_Data_out           (Inst_Data_out),
      .CPU_Read_in            (Inst_Read_in),
      .CPU_Ready_out          (Inst_Ready_out),
      .CPU_Flush_in           (flush),
      .CPU_Flush_Complete_out (flushcomplete),
      .MEM_Flush_out          (L2_flush),
      .MEM_Flush_Complete_in  (L2_flushcomplete),
      .MEM_Address_out        (Inst_Address),
      .MEM_Write_Data_out     (Inst_Write),
      .MEM_Data_in            (Inst_Data_i),
      .MEM_Data_out           (Inst_Data_o),
      .MEM_Read_out           (Inst_Read),
      .MEM_Ready_in           (Inst_Ready),
      .Init_Mem_out           (Init_L1_Mem_out)
   );

   Level_1_Cache DCache(
      .Clock_in               (clock),
      .Reset_in               (reset||flush),
      .CPU_Address_in         (Data_Address_in),
      .CPU_Write_Data_in      (Data_Write_in),
      .CPU_Data_in            (Data_Data_in),
      .CPU_Data_out           (Data_Data_out),
      .CPU_Read_in            (Data_Read_in),
      .CPU_Ready_out          (Data_Ready_out),
      .CPU_Flush_in           (1'b0),
      .CPU_Flush_Complete_out (),
```

```
   .MEM_Flush_out           (),
   .MEM_Flush_Complete_in   (1'b0),
   .MEM_Address_out         (Data_Address),
   .MEM_Write_Data_out      (Data_Write),
   .MEM_Data_in             (Data_Data_i),
   .MEM_Data_out            (Data_Data_o),
   .MEM_Read_out            (Data_Read),
   .MEM_Ready_in            (Data_Ready),
   .Init_Mem_out            ()
);

Bus_Arbiter BUS(
   .Clock_in         (clock),
   .Reset_in         (reset),
   .Inst_Read_in     (Inst_Read),
   .Inst_Write_in    (Inst_Write),
   .Inst_Address_in  (Inst_Address),
   .Inst_Data_in     (Inst_Data_o),
   .Inst_Ready_out   (Inst_Ready),
   .Inst_Data_out    (Inst_Data_i),
   .Data_Read_in     (Data_Read),
   .Data_Write_in    (Data_Write),
   .Data_Address_in  (Data_Address),
   .Data_Data_in     (Data_Data_o),
   .Data_Ready_out   (Data_Ready),
   .Data_Data_out    (Data_Data_i),
   .DDR_Read_in      (DDR_Read),
   .DDR_Write_in     (DDR_Write),
   .DDR_Address_in   (DDR_Address),
   .DDR_Data_in      (DDR_Data_i),
   .DDR_Ready_out    (DDR_Ready),
   .DDR_Data_out     (DDR_Data_o)
);

Level_2_Cache L2(
   .Clock_in               (clock),
   .Reset_in               (reset),
   .DDR_Address_in         (DDR_Address),
   .DDR_Write_Data_in      (DDR_Write),
   .DDR_Data_in            (DDR_Data_i),
   .DDR_Data_out           (DDR_Data_o),
   .DDR_Read_in            (DDR_Read),
   .DDR_Ready_out          (DDR_Ready),
   .DDR_Flush_in           (L2_flush),
   .DDR_Flush_Complete_out (L2_flushcomplete),
   .SD_Address_out         (SD_Address),
   .SD_Write_Data_out      (SD_Write),
   .SD_Data_in             (SD_Data_o),
   .SD_Data_out            (SD_Data_i),
   .SD_Read_out            (SD_Read),
   .SD_Ready_in            (SD_Ready),
   .Init_VTD               (Init_L2_Mem_out),
   // DDR U/I Signals
   .app_addr               (app_addr),
   .app_cmd                (app_cmd),
   .app_en                 (app_en),
   .app_wdf_data           (app_wdf_data),
   .app_wdf_end            (app_wdf_end),
   .app_wdf_wren           (app_wdf_wren),
   .app_rd_data            (app_rd_data),
   .app_rd_data_valid      (app_rd_data_valid),
   .app_rdy                (app_rdy),
   .app_wdf_rdy            (app_wdf_rdy),
   .ui_clk                 (ui_clk),
   .ui_clk_sync_rst        (ui_clk_sync_rst),
   .init_calib_complete    (init_calib_complete)
);

generate
```

```
      for (i = 0; i < N; i = i + 1) begin : System
(* DONT_TOUCH = "TRUE" *)SD_Module_Interface SD(
            .Reset                  (reset),
            .MISO_in                (SD_MISO[i]),
            .MOSI_out               (SD_MOSI[i]),
            .CS_out                 (SD_CS[i]),
            .SD_Clock               (SD_SClk),
            .Read_in                (SD_Read),
            .Write_in               (SD_Write),
            .Ready_out              (Vote_SD_Ready[i]),
            .Address_in             (SD_Address),
            .DDR_Data_out           (SD_Data_i),
            .DDR_Data_in            (Vote_SD_Data_o[i]),
            .Select_SD_Clock_Speed  (Vote_Select_SD_Clock_Speed[i]),
            .Align_in               (Align_in),
            .Align_out              (Align[i])
          );
       end
    endgenerate

    assign Align_in = {Align[N-1],Align[N-2],Align[N-3]};

    Voter #(.WIDTH(1)) SD_Rdy (
       .A       (Vote_SD_Ready[0]),
       .B       (Vote_SD_Ready[1]),
       .C       (Vote_SD_Ready[2]),
       .True   (SD_Ready)
    );

    Voter #(.WIDTH(4096)) SD_Dat (
       .A      (Vote_SD_Data_o[0]),
       .B      (Vote_SD_Data_o[1]),
       .C      (Vote_SD_Data_o[2]),
       .True   (SD_Data_o)
    );

    Voter #(.WIDTH(1)) SD_Clk_Spd (
       .A      (Vote_Select_SD_Clock_Speed[0]),
       .B      (Vote_Select_SD_Clock_Speed[1]),
       .C      (Vote_Select_SD_Clock_Speed[2]),
       .True   (Select_SD_Clock_Speed)
    );

endmodule

`default_nettype wire
```

(2) Level_1_Cache.v

```
`timescale 1ns / 1ps

/*
 * File         : Level_1_Cache.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module instantiates the major components of a L1 Cache: Controller,
 *   Tag RAM, Valid RAM, Dirty RAM, Hit Detector, and Counters. CFTP uses an 8KB
 *   direct mapped L1 Cache for each data and instructions.
 */
```

```
module Level_1_Cache(
   input Clock_in,
   input Reset_in,
   input [29:0] CPU_Address_in,
   input [3:0] CPU_Write_Data_in,
   input [31:0] CPU_Data_in,
   output [31:0] CPU_Data_out,
   input CPU_Read_in,
   output CPU_Ready_out,
   input CPU_Flush_in,
   output MEM_Flush_out,
   input MEM_Flush_Complete_in,
   output CPU_Flush_Complete_out,
   output [29:0] MEM_Address_out,
   output MEM_Write_Data_out,
   input [1023:0] MEM_Data_in,
   output [1023:0] MEM_Data_out,
   output MEM_Read_out,
   input MEM_Ready_in,
   output Init_Mem_out
   );

   wire [3:0] Write_Data;
   wire [4:0] Offset;
   wire [7:0] Flush_Counter, Counter;
   wire [5:0] Index;
   wire [16:0] Tag;

   Cache_Control CONTROL (
      .CPU_Address_in(CPU_Address_in),
      .CPU_Write_Data_in(CPU_Write_Data_in),
      .CPU_Read_in(CPU_Read_in),
      .CPU_Ready_out(CPU_Ready_out),
      .CPU_Flush_in(CPU_Flush_in),
      .CPU_Flush_out(CPU_Flush_Complete_out),
      .MEM_Flush_out(MEM_Flush_out),
      .MEM_Flush_Complete_in(MEM_Flush_Complete_in),
      .MEM_Address_out(MEM_Address_out),
      .MEM_Write_Data_out(MEM_Write_Data_out),
      .MEM_Read_out(MEM_Read_out),
      .MEM_Ready_in(MEM_Ready_in),
      .Index_out(Index),
      .Offset_out(Offset),
      .Write_Data_out(Write_Data),
      .Dirty_in(Dirty_o),
      .Dirty_out(Dirty_i),
      .Write_Dirty_out(Write_Dirty),
      .Init_Dirty_in(Init_Dirty),
      .Tag_in(Tag),
      .Write_Valid_Tag_out(Write_Valid_Tag),
      .Valid_out(Valid_i),
      .Init_Valid_in(Init_Mem_out),
      .Hit_in(Hit),
      .Counter_in(Counter),
      .Reset_Counter_out(Reset_Counter),
      .Counter_Enable_out(Counter_Enable),
      .Flush_Count_in(Flush_Counter),
      .Flush_Reset_Counter_out(Flush_Reset_Counter),
      .Flush_Counter_Enable_out(Flush_Counter_Enable),
      .Clock_in(Clock_in),
      .Reset_in(Reset_in),
      .Reset_Valid_Dirty_out(Reset_Valid_Dirty),
      .Fill_out(Fill)
   );

   Data_Ram DATA_RAM (
      .Offset_in(Offset),
      .Index_in(Index),
```

```
      .CPU_Data_in(CPU_Data_in),
      .MEM_Data_in(MEM_Data_in),
      .Clock_in(Clock_in),
      .Write_Data_in(Write_Data),
      .CPU_Data_out(CPU_Data_out),
      .MEM_Data_out(MEM_Data_out),
      .Fill_in(Fill)
   );

   Dirty_Ram DIRTY_RAM (
      .Clock_in(Clock_in),
      .Reset_in(Reset_Valid_Dirty || Reset_in),
      .Index_in(Index),
      .Dirty_in(Dirty_i),
      .Write_Dirty_in(Write_Dirty),
      .Dirty_out(Dirty_o),
      .Counter_in(Counter),
      .Init_Dirty_out(Init_Dirty)
   );

   Tag_Ram TAG_RAM (
      .Index_in(Index),
      .Tag_in(CPU_Address_in[27:11]),
      .Clock_in(Clock_in),
      .Write_Tag_in(Write_Valid_Tag),
      .Tag_out(Tag)
   );

   Valid_Ram VALID_RAM (
      .Clock_in(Clock_in),
      .Reset_in(Reset_Valid_Dirty || Reset_in),
      .Index_in(Index),
      .Valid_in(Valid_i),
      .Write_Valid_in(Write_Valid_Tag),
      .Valid_out(Valid_o),
      .Counter_in(Counter),
      .Init_Valid_out(Init_Mem_out)
   );

   Hit_Detection HIT (
      .CPU_Tag_in(CPU_Address_in[27:11]),
      .Cache_Tag_in(Tag),
      .Valid_Bit_in(Valid_o),
      .HIT_out(Hit)
   );

   Counter Cache_Count (
      .Clock_in(Clock_in),
      .Enable_in(Counter_Enable),
      .Reset_in(Reset_Counter),
      .Count_out(Counter)
   );

   Counter Flush_Count (
      .Clock_in(Clock_in),
      .Enable_in(Flush_Counter_Enable),
      .Reset_in(Flush_Reset_Counter),
      .Count_out(Flush_Counter)
   );

endmodule
```

(3) Cache_Control.v

```
`timescale 1ns / 1ps

/*
 * File         : Cache_Control.v
```

```
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The cache controller determine the state that the L1 cache is in. Depending
 *   on the state various output signals are set to write values to the Tag,
 *   Valid, Dirty, and Data RAMS, as well as to enable and reset the two counters.
 *   A flow diagram a the full state machine is available in my thesis.
 */

module Cache_Control(
   input [29:0] CPU_Address_in,
   input [3:0] CPU_Write_Data_in,
   input CPU_Read_in,
   output CPU_Ready_out,
   input CPU_Flush_in,
   output CPU_Flush_out,
   output MEM_Flush_out,
   input MEM_Flush_Complete_in,
   output [29:0] MEM_Address_out,
   output MEM_Write_Data_out,
   output MEM_Read_out,
   input MEM_Ready_in,
   output [5:0] Index_out,
   output [4:0] Offset_out,
   output [3:0] Write_Data_out,
   input Dirty_in,
   output Dirty_out,
   output Write_Dirty_out,
   input Init_Dirty_in,
   input [16:0] Tag_in,
   output Write_Valid_Tag_out,
   output Valid_out,
   input Init_Valid_in,
   input Hit_in,
   input [7:0] Counter_in,
   output Reset_Counter_out,
   output Counter_Enable_out,
   input [7:0] Flush_Count_in,
   output Flush_Reset_Counter_out,
   output Flush_Counter_Enable_out,
   input Clock_in,
   input Reset_in,
   output Reset_Valid_Dirty_out,
   output Fill_out
   );

// State Encoding
parameter STATE_CACHE_INIT          = 4'b0000;
parameter STATE_CACHE_IDLE          = 4'b0001;
parameter STATE_CACHE_REQUEST_ACK   = 4'b0010;
parameter STATE_CACHE_ALLOCATE      = 4'b0011;
parameter STATE_CACHE_ALLOCATE_ACK  = 4'b0100;
parameter STATE_CACHE_WRITEBACK     = 4'b0101;
parameter STATE_CACHE_WRITEBACK_ACK = 4'b0110;
parameter STATE_CACHE_FLUSH         = 4'b1000;
parameter STATE_CACHE_FLUSH_L2      = 4'b1001;
parameter STATE_CACHE_FLUSH_ACK     = 4'b1010;

// State reg Declarations
reg [3:0] CurrentState ;
reg [3:0] NextState ;
```

```
// Internal Signals
wire CPU_Request;
wire [16:0] Tag;

// Outputs

// Continuous Assignments
assign CPU_Request = CPU_Read_in || (|CPU_Write_Data_in) || CPU_Flush_in;
assign Tag = (CurrentState == STATE_CACHE_WRITEBACK || CurrentState ==
             STATE_CACHE_WRITEBACK_ACK) ? Tag_in : CPU_Address_in[27:11];

// State Outputs
//CPU Interface
assign CPU_Ready_out = Hit_in;
assign CPU_Flush_out = (CurrentState == STATE_CACHE_FLUSH_ACK) ? 1'b1 : 1'b0;

//MEM Interface
assign MEM_Address_out = {2'b0, Tag, Index_out, 5'd0};
assign MEM_Write_Data_out = (CurrentState == STATE_CACHE_WRITEBACK) ? 1'b1 : 1'b0;
assign MEM_Read_out = (CurrentState == STATE_CACHE_ALLOCATE) ? 1'b1 : 1'b0;
assign Fill_out = (CurrentState == STATE_CACHE_ALLOCATE && MEM_Ready_in);

//Data RAM Interface
assign Index_out = ((CurrentState == STATE_CACHE_FLUSH) || ((CurrentState ==
                   STATE_CACHE_WRITEBACK || CurrentState ==
                   STATE_CACHE_WRITEBACK_ACK) && CPU_Flush_in)) ? Flush_Count_in :
                   CPU_Address_in[10:5];
assign Offset_out = CPU_Address_in[4:0];
assign Write_Data_out = (CurrentState == STATE_CACHE_IDLE && Hit_in) ?
                        CPU_Write_Data_in : 4'b0000;

//Dirty RAM Interface
assign Dirty_out = (CurrentState == STATE_CACHE_IDLE && (|CPU_Write_Data_in)) ?
                   1'b1 : 1'b0;
assign Write_Dirty_out = ((CurrentState == STATE_CACHE_IDLE &&
                         (|CPU_Write_Data_in) && Hit_in) || (CurrentState ==
                         STATE_CACHE_ALLOCATE_ACK && ~MEM_Ready_in)) ? 1'b1 :
                         1'b0;

//Valid RAM Interface
assign Valid_out = (CurrentState == STATE_CACHE_ALLOCATE_ACK &&  ~MEM_Ready_in) ?
                   1'b1 : 1'b0;
assign Write_Valid_Tag_out = (CurrentState == STATE_CACHE_ALLOCATE_ACK &&
                             ~MEM_Ready_in) ? 1'b1 : 1'b0;

//Counter Interface
assign Reset_Counter_out = (Reset_in || CurrentState == STATE_CACHE_IDLE ||
                           ((CurrentState == STATE_CACHE_ALLOCATE_ACK ||
                           CurrentState == STATE_CACHE_WRITEBACK_ACK) &&
                           Counter_in == 8'b00001111 && ~MEM_Ready_in) ||
                           (CurrentState == STATE_CACHE_FLUSH_ACK)) ? 1'b1 : 1'b0;
assign Counter_Enable_out = (((CurrentState == STATE_CACHE_ALLOCATE_ACK ||
                            CurrentState == STATE_CACHE_WRITEBACK_ACK) &&
                            ~MEM_Ready_in) || (CurrentState ==
                            STATE_CACHE_INIT)) ? 1'b1 : 1'b0;

//Flush Counter Interface
assign Flush_Reset_Counter_out = (CurrentState == STATE_CACHE_INIT) ? 1'b1 : 1'b0;
assign Flush_Counter_Enable_out = ((CPU_Flush_in && CurrentState ==
                                  STATE_CACHE_WRITEBACK_ACK &&  ~MEM_Ready_in) ||
                                  (CurrentState == STATE_CACHE_FLUSH &&
                                  ~Dirty_in)) ? 1'b1 : 1'b0;
assign Reset_Valid_Dirty_out = (CurrentState == STATE_CACHE_FLUSH_ACK) ? 1'b1:
                               1'b0;

//L2 Flush
assign MEM_Flush_out = (CurrentState == STATE_CACHE_FLUSH_L2) ? 1'b1: 1'b0;
```

```
// Synchronous State - Transition
always@ ( posedge Clock_in ) CurrentState = NextState;

// Conditional State - Transition
always@ (*) begin
   NextState = CurrentState ;
      case ( CurrentState )
         STATE_CACHE_INIT : begin
            NextState = (~Init_Valid_in && ~Init_Dirty_in) ? STATE_CACHE_IDLE :
                        STATE_CACHE_INIT;
         end
         STATE_CACHE_IDLE : begin //Request
            NextState = (Reset_in) ? STATE_CACHE_INIT : (CPU_Request) ?
                        (CPU_Flush_in) ? STATE_CACHE_FLUSH : (Hit_in) ?
                        CurrentState : (Dirty_in) ? STATE_CACHE_WRITEBACK :
                        STATE_CACHE_ALLOCATE : CurrentState;
         end
         STATE_CACHE_REQUEST_ACK : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : CurrentState;
         end
         STATE_CACHE_ALLOCATE : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : ( MEM_Ready_in ) ?
                        STATE_CACHE_ALLOCATE_ACK : CurrentState;
         end
         STATE_CACHE_ALLOCATE_ACK : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~MEM_Ready_in ) ?
                        STATE_CACHE_IDLE : CurrentState;
         end
         STATE_CACHE_WRITEBACK : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : ( MEM_Ready_in ) ?
                        STATE_CACHE_WRITEBACK_ACK : CurrentState;
         end
         STATE_CACHE_WRITEBACK_ACK : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~MEM_Ready_in ) ?
                        (CPU_Flush_in) ? STATE_CACHE_FLUSH :
                        STATE_CACHE_ALLOCATE : CurrentState;
         end
         STATE_CACHE_FLUSH : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : ( Flush_Count_in ==
                        8'b01000000 ) ? STATE_CACHE_FLUSH_L2 : ( Dirty_in ) ?
                        STATE_CACHE_WRITEBACK : CurrentState ;
         end
         STATE_CACHE_FLUSH_L2 : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : (MEM_Flush_Complete_in) ?
                        STATE_CACHE_FLUSH_ACK : CurrentState ;
         end
         STATE_CACHE_FLUSH_ACK : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~CPU_Flush_in ) ?
                        STATE_CACHE_INIT : CurrentState;
         end
         default : begin
            NextState = (Reset_in) ? STATE_CACHE_INIT : STATE_CACHE_IDLE;
         end
      endcase
   end

endmodule
```

(4) Data_Ram.v

```
`timescale 1ns / 1ps

/*
 * File         : Data_Ram.v
 * Project      : Naval Postgraduate School, CFTP Project
```

```
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The Data Ram stores indexed blocks of data used in the L1 cache. During
 *   Allocate operations the fill signals will cause all offsets of data to be
 *   written. Regardless, an input address will provide the full block exposed to
 *   the L2 cache and only the requested offset exposed to the CPU side.
 */

module Data_Ram(
   input [4:0] Offset_in,
   input [5:0] Index_in,
   input [31:0] CPU_Data_in,
   input [1023:0] MEM_Data_in,
   input Clock_in,
   input [3:0] Write_Data_in,
   output [31:0] CPU_Data_out,
   output [1023:0] MEM_Data_out,
   input Fill_in
   );

   wire [31:0] Write_Data_Offset;
   wire [7:0] Data0 [31:0];
   wire [7:0] Data1 [31:0];
   wire [7:0] Data2 [31:0];
   wire [7:0] Data3 [31:0];

   genvar i,j;
   generate
      for (i=0; i<32; i=i+1) begin : DATA_RAM
         Distributed_RAM #(6,8,64) DATA_RAM_0(
            .Clock_in         (Clock_in),
            .Write_Enable_in  ((Fill_in) ? 1'b1 : (Write_Data_Offset[i] &&
                              Write_Data_in[0])),
            .Address_in       (Index_in),
            .Data_in          ((Fill_in) ? MEM_Data_in[(32*i)+:8] :
                              CPU_Data_in[0+:8]),
            .Data_out         (Data0 [i])
         );
         Distributed_RAM #(6,8,64) DATA_RAM_1(
            .Clock_in         (Clock_in),
            .Write_Enable_in  ((Fill_in) ? 1'b1 : (Write_Data_Offset[i] &&
                              Write_Data_in[1])),
            .Address_in       (Index_in),
            .Data_in          ((Fill_in) ? MEM_Data_in[((32*i)+8)+:8] :
                              CPU_Data_in[8+:8]),
            .Data_out         (Data1 [i])
         );
         Distributed_RAM #(6,8,64) DATA_RAM_2(
            .Clock_in         (Clock_in),
            .Write_Enable_in  ((Fill_in) ? 1'b1 : (Write_Data_Offset[i] &&
                              Write_Data_in[2])),
            .Address_in       (Index_in),
            .Data_in          ((Fill_in) ? MEM_Data_in[((32*i)+16)+:8] :
                              CPU_Data_in[16+:8]),
            .Data_out         (Data2 [i])
         );
         Distributed_RAM #(6,8,64) DATA_RAM_3(
            .Clock_in         (Clock_in),
            .Write_Enable_in  ((Fill_in) ? 1'b1 : (Write_Data_Offset[i] &&
                              Write_Data_in[3])),
            .Address_in       (Index_in),
```

```
            .Data_in          ((Fill_in) ? MEM_Data_in[((32*i)+24)+:8] :
                              CPU_Data_in[24+:8]),
            .Data_out         (Data3 [i])
         );
         assign MEM_Data_out[(32*i)+:32] = {Data3[i], Data2[i], Data1[i],
                                            Data0[i]};
      end
   endgenerate

   assign Write_Data_Offset = (1 << Offset_in);
   assign CPU_Data_out = {Data3[Offset_in], Data2[Offset_in], Data1[Offset_in],
                          Data0[Offset_in]};


endmodule
```

(5) Dirty_Ram.v

```
`timescale 1ns / 1ps

/*
 * File         : Dirty_Ram.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The Dirty Ram keeps track of which blocks in the cache have been written to
 *   by the CPU. If a block has been marked as dirty but its tag does not match
 *   index being requested, the block must first be written back to the L2 cache.
 *   Otherwise, data can be directly loaded into the indexed slot from the L2
 *   cache.
 */

module Dirty_Ram(
   input  Clock_in,
   input  Reset_in,
   input  [5:0] Index_in,
   input  Dirty_in,
   input  Write_Dirty_in,
   output Dirty_out,
   input [7:0] Counter_in,
   output reg Init_Dirty_out
   );

   Distributed_RAM #(6,1,64) DIRTY_RAM(
      .Clock_in         (Clock_in),
      .Write_Enable_in  ((Init_Dirty_out) ? 1'b1 : Write_Dirty_in),
      .Address_in       ((Init_Dirty_out) ? Counter_in[5:0] : Index_in),
      .Data_in          ((Init_Dirty_out) ? 1'b0 : Dirty_in),
      .Data_out         (Dirty_out)
   );

   always @ (posedge Clock_in) begin
      if (Reset_in) begin
         Init_Dirty_out = 1'b1;
      end
      else begin
         if (Counter_in == 8'b01000000) begin
            Init_Dirty_out = 1'b0;
         end
      end
```

```
   end

endmodule
```

(6) Valid_Ram.v

```
`timescale 1ns / 1ps

/*
 * File         : Valid_Ram.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The valid RAM keeps track of which blocks in the cache have retrieved data
 *   from the L2 cache. This prevents us from being able to return data to the
 *   CPU without having first read it from disk.
 */

module Valid_Ram(
input  Clock_in,
   input  Reset_in,
   input  [5:0] Index_in,
   input Valid_in,
   input  Write_Valid_in,
   output Valid_out,
   input [7:0] Counter_in,
   output reg Init_Valid_out
   );

   Distributed_RAM#(6,1,64) VALID_RAM(
      .Clock_in         (Clock_in),
      .Write_Enable_in  ((Init_Valid_out) ? 1'b1 : Write_Valid_in),
      .Address_in       ((Init_Valid_out) ? Counter_in[5:0] : Index_in),
      .Data_in          ((Init_Valid_out) ? 1'b0 : Valid_in),
      .Data_out         (Valid_out)
   );

   always @ (posedge Clock_in) begin
      if (Reset_in) begin
         Init_Valid_out = 1'b1;
      end
      else begin
         if (Counter_in == 8'b01000000) begin
            Init_Valid_out = 1'b0;
         end
      end
   end

endmodule
```

(7) Tag_Ram.v

```
`timescale 1ns / 1ps

/*
 * File         : Tag_Ram.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
```

```
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The tag RAM stores the upper addess bits of blocks stored in the cache. These
 *   bits are the physical address bits not including those used in the index and
 *   offset.
 */

module Tag_Ram(
   input [5:0] Index_in,
   input [16:0] Tag_in,
   input Clock_in,
   input Write_Tag_in,
   output [16:0] Tag_out
   );

   Distributed_RAM #(6,17,64) TAG_RAM(
      .Clock_in         (Clock_in),
      .Write_Enable_in  (Write_Tag_in),
      .Address_in       (Index_in),
      .Data_in          (Tag_in),
      .Data_out         (Tag_out)
   );

endmodule
```

(8) Distributed_RAM.v

```
`timescale 1ns / 1ps

/*
 * File         : Distributed_RAM.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module implements a chunk of distributed pipeline RAM as defined by the
 *   parameters passed into it. Pipeline distributed RAM is used with CFTP's L1
 *   cache due to it zero clock cycle read latency which allows memory to run at
 *   the same frequency as the CPU and still be able to return data every clock
 *   cycle.
 */

module Distributed_RAM(
   Clock_in,
   Write_Enable_in,
   Address_in,
   Data_in,
   Data_out
   );

   parameter AddressWidth = 8;
   parameter DataWidth = 8;
   parameter Depth = 256;

   input Clock_in;
```

```
   input Write_Enable_in;
   input [AddressWidth-1:0]Address_in;
   input [DataWidth-1:0]Data_in;
   output [DataWidth-1:0]Data_out;

   (* ram_style = “distributed” *)
   reg [DataWidth-1:0] RAM [Depth-1:0];

   integer i;
   initial begin
      for (i = 0; i < Depth; i = i + 1) begin
         RAM[i] = {DataWidth{1’b0}};
      end
   end

   always @ (posedge Clock_in) begin
      if (Write_Enable_in) begin
         RAM[Address_in]<=Data_in;
      end
   end

   assign Data_out = RAM[Address_in];

endmodule
```

(9) Hit_Detection.v

```
`timescale 1ns / 1ps

/*
 * File         : Hit_Detection.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module performs a comparison between the upper address bits of the
 *   requested address and those stroed in the tag RAM. If they match, the
 *   requested data is stored in the L1 cache.
 */

module Hit_Detection(
   input [16:0] CPU_Tag_in,
   input [16:0] Cache_Tag_in,
   input Valid_Bit_in,
   output HIT_out
   );

   wire Matching_Tag;

   assign Matching_Tag = (CPU_Tag_in == Cache_Tag_in) ? 1’b1 : 1’b0;
   assign HIT_out = Valid_Bit_in && Matching_Tag;

endmodule
```

(10) Counter.v

```
`timescale 1ns / 1ps

/*
 * File         : Counter.v
```

```
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module creates a counter of the width passed in. It also allows the
 *   counter to be reset and enabled or disabled.
 */

module Counter(
   Clock_in,
   Enable_in,
   Reset_in,
   Count_out
   );

   parameter CountWidth = 8;

   input Clock_in;
   input Enable_in;
   input Reset_in;
   output reg [CountWidth-1:0] Count_out;

   initial Count_out = 0;

   always @ (posedge Clock_in) begin
      if (Reset_in) begin
         Count_out = 0;
      end
      else begin
         if (Enable_in) begin
            Count_out = Count_out + 1;
         end
      end
   end

endmodule
```

(11) Bus_Arbiter.v

```
`timescale 1ns / 1ps

/*
 * File         : Bus_Arbiter.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The cache architecture in CFTP has seperate instruction and data caches at
 *   the L1 level. These are combined into a shared cache at the L2 level. The
 *   bus arbiter allows only one access to the L2 cache at a time. This is
 *   accomplished through the use of a locking mechanisim. While the either of
 *   the L1 instruction or data cache holds the lock, its control lines and data
 *   paths are directly connected to the L2 cache while the others are blocked.
 */
```

```
module Bus_Arbiter(
   input Clock_in,
   input Reset_in,
   input Inst_Read_in,
   input Inst_Write_in,
   input [29:0] Inst_Address_in,
   input [1023:0] Inst_Data_in,
   output Inst_Ready_out,
   output [1023:0] Inst_Data_out,
   input Data_Read_in,
   input Data_Write_in,
   input [29:0] Data_Address_in,
   input [1023:0] Data_Data_in,
   output Data_Ready_out,
   output [1023:0] Data_Data_out,
   output DDR_Read_in,
   output DDR_Write_in,
   output [29:0] DDR_Address_in,
   output [1023:0] DDR_Data_in,
   input DDR_Ready_out,
   input [1023:0] DDR_Data_out
   );

   reg [1:0] Bus_Locked;
   reg Bus_Select;

   assign Inst_Ready_out = (!Reset_in) ? (!Bus_Select) ? DDR_Ready_out : 1'b0 :
                           1'b0;
   assign Inst_Data_out = (!Reset_in) ? DDR_Data_out : 1024'd0;
   assign Data_Ready_out = (!Reset_in) ? (Bus_Select) ? DDR_Ready_out : 1'b0 :
                           1'b0;
   assign Data_Data_out = (!Reset_in) ? DDR_Data_out :1024'd0;
   assign DDR_Read_in = (!Reset_in) ? (Bus_Select) ? Data_Read_in : Inst_Read_in :
                        1'b0;
   assign DDR_Write_in = (!Reset_in) ? (Bus_Select) ? Data_Write_in :
                         Inst_Write_in : 1'b0;
   assign DDR_Address_in = (!Reset_in) ? (Bus_Select) ? Data_Address_in :
                           Inst_Address_in : 30'd0;
   assign DDR_Data_in = (!Reset_in) ? (Bus_Select) ? Data_Data_in : Inst_Data_in :
                        1024'd0;

   always@(posedge Clock_in) begin
      if (Reset_in)
         Bus_Locked = 2'b00;
      else begin
         if (Bus_Locked == 2'b00) begin
            if ((Inst_Read_in | Inst_Write_in) && !DDR_Ready_out)
               Bus_Locked = 2'b10;
            if ((Data_Read_in | Data_Write_in) && !DDR_Ready_out)
               Bus_Locked = 2'b01;
         end
         if (Bus_Locked == 2'b01) begin
            if (~(Data_Read_in | Data_Write_in) && !DDR_Ready_out &&
               (Inst_Read_in | Inst_Write_in))
               Bus_Locked = 2'b10;
            if (~(Data_Read_in | Data_Write_in) && !DDR_Ready_out &&
               ~(Inst_Read_in | Inst_Write_in))
               Bus_Locked = 2'b00;
         end
         if (Bus_Locked == 2'b10) begin
            if ((Data_Read_in | Data_Write_in) && !DDR_Ready_out &&
               ~(Inst_Read_in | Inst_Write_in))
               Bus_Locked = 2'b01;
            if (~(Data_Read_in | Data_Write_in) && !DDR_Ready_out &&
               ~(Inst_Read_in | Inst_Write_in))
               Bus_Locked = 2'b00;
         end
      end
```

```
      Bus_Select = (!Reset_in) ? (Bus_Locked == 2'b10) ? 1'b0 :
                   (Bus_Locked == 2'b01) ? 1'b1 : Bus_Select : 1'b0;
   end

endmodule
```

(12) Level_2_Cache.v

```
`timescale 1ns / 1ps

/*
 * File         : Level_2_Cache.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module instantiates the major components of CFTP's L2 Cache: Controller,
 *   Valid/Tag/Dirty RAM, Hit Detector, DDR Interface and Counters. CFTP uses a
 *   shared 512MB direct mapped L2 Cache.
 */

module Level_2_Cache(
   input Clock_in,
   input Reset_in,
   input [27:0] DDR_Address_in,
   input DDR_Write_Data_in,
   input [1023:0] DDR_Data_in,
   output [1023:0] DDR_Data_out,
   input DDR_Read_in,
   output DDR_Ready_out,
   input DDR_Flush_in,
   output DDR_Flush_Complete_out,
   output [27:0] SD_Address_out,
   output SD_Write_Data_out,
   input [4095:0] SD_Data_in,
   output [4095:0] SD_Data_out,
   output SD_Read_out,
   input SD_Ready_in,
   output Init_VTD,
   //DDR U/I Signals
   output [28:0] app_addr,
   output [2:0] app_cmd,
   output app_en,
   output [255:0] app_wdf_data,
   output app_wdf_end,
   output app_wdf_wren,
   input [255:0] app_rd_data,
   input app_rd_data_valid,
   input app_rdy,
   input app_wdf_rdy,
   input ui_clk,
   input ui_clk_sync_rst,
   input init_calib_complete
   );

   wire [1:0] Offset;
   wire Write_Data;
   wire [1:0] Offset_Count;
   wire [19:0] Flush_Count;
   wire [18:0] Index;
   wire [1023:0] Data_i, Data_o;
```

```
wire [1:0] Tag;

Cache_Control_2 CONTROL (
   //Interface to Level 1 Cache
   .DDR_Address_in          (DDR_Address_in),
   .DDR_Write_in            (DDR_Write_Data_in),
   .DDR_Read_in             (DDR_Read_in),
   .DDR_Data_in             (DDR_Data_in),
   .DDR_Data_out            (DDR_Data_out),
   .DDR_Ready_out           (DDR_Ready_out),
   .DDR_Flush_in            (DDR_Flush_in),
   .DDR_Flush_out           (DDR_Flush_Complete_out),
   //Interface to SD Card
   .SD_Address_out          (SD_Address_out),
   .SD_Write_out            (SD_Write_Data_out),
   .SD_Read_out             (SD_Read_out),
   .SD_Data_in              (SD_Data_in),
   .SD_Data_out             (SD_Data_out),
   .SD_Ready_in             (SD_Ready_in),
   //Cache Signals
   .Tag_in                  (Tag),
   .Index_out               (Index),
   .Offset_out              (Offset),
   .Data_out                (Data_o),
   .Data_in                 (Data_i),
   .Write_Data_out          (Write_Data),
   .Read_Data_out           (Read_Data),
   .Dirty_in                (Dirty_o),
   .Dirty_out               (Dirty_i),
   .Write_Dirty_out         (Write_Dirty),
   .Init_VTD_in             (Init_VTD),
   .Write_Valid_Tag_out     (Write_Valid_Tag),
   .Valid_out               (Valid_i),
   .Hit_in                  (Hit),
   .DDR_Interface_Ready_in  (DDR_Interface_Ready),
   .Reset_VTD               (Reset_VTD),
   //Counter Signals
   .Offset_Count_in         (Offset_Count),
   .Reset_Offset_Count_out  (Reset_Offset_Count),
   .Enable_Offset_Count_out (Enable_Offset_Count),
   .Flush_Count_in          (Flush_Count),
   .Reset_Flush_Count_out   (Reset_Flush_Count),
   .Enable_Flush_Count_out  (Enable_Flush_Count),
   //Clock and Reset
   .Clock_in                (Clock_in),
   .Reset_in                (Reset_in)
);

DDR_Interface DDR(
   .Address_in    ({Index,Offset,5'b00000}),
   .Data_in       (Data_o),
   .Data_out      (Data_i),
   .Read_in       (Read_Data),
   .Write_in      (Write_Data),
   .Ready_out     (DDR_Interface_Ready),
   // DDR U/I Signals
   .app_addr              (app_addr),
   .app_cmd               (app_cmd),
   .app_en                (app_en),
   .app_wdf_data          (app_wdf_data),
   .app_wdf_end           (app_wdf_end),
   .app_wdf_wren          (app_wdf_wren),
   .app_rd_data           (app_rd_data),
   .app_rd_data_valid     (app_rd_data_valid),
   .app_rdy               (app_rdy),
   .app_wdf_rdy           (app_wdf_rdy),
   .ui_clk                (ui_clk),
   .ui_clk_sync_rst       (ui_clk_sync_rst),
   .init_calib_complete   (init_calib_complete)
```

```
   );

   L2_VALID_TAG_DIRTY_RAM_Wrapper VTD_RAM (
      .Clock_in           (Clock_in),
      .Reset_in           (Reset_VTD||Reset_in),
      .Index_in           (Index),
      .Dirty_in           (Dirty_i),
      .Write_Dirty_in     (Write_Dirty),
      .Dirty_out          (Dirty_o),
      .Init_VTD_out       (Init_VTD),
      .Tag_in             (DDR_Address_in[27:26]),
      .Write_Valid_Tag_in (Write_Valid_Tag),
      .Tag_out            (Tag),
      .Valid_in           (Valid_i),
      .Valid_out          (Valid_o),
      .Counter_in         (Flush_Count)
   );

   Hit_Detection_2 HIT (
      .CPU_Tag_in     (DDR_Address_in[27:26]),
      .Cache_Tag_in   (Tag),
      .Valid_Bit_in   (Valid_o),
      .HIT_out        (Hit)
   );

   Counter #(2) O_Count (
      .Clock_in   (Clock_in),
      .Enable_in  (Enable_Offset_Count),
      .Reset_in   (Reset_Offset_Count),
      .Count_out  (Offset_Count)
   );

   Counter #(20) F_Count (
      .Clock_in   (Clock_in),
      .Enable_in  (Enable_Flush_Count),
      .Reset_in   (Reset_Flush_Count),
      .Count_out  (Flush_Count)
   );

endmodule
```

(13) Cache_Control_2.v

```
`timescale 1ns / 1ps
`default_nettype none

/*
 * File         : Cache_Control_2.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The cache controller determine the state that the L2 cache is in. Depending
 *   on the state various output signals are set to write values to the Tag,
 *   Valid, Dirty, and DDR Interface, as well as to enable and reset the counters.
 *   A flow diagram a the full state machine is available in my thesis
 */

module Cache_Control_2(
   //Interface to Level 1 Cache
   input wire [27:0] DDR_Address_in,
```

```
    input wire DDR_Write_in,
    input wire DDR_Read_in,
    input wire [1023:0] DDR_Data_in,
    output reg [1023:0] DDR_Data_out,
    output wire DDR_Ready_out,
    input wire DDR_Flush_in,
    output wire DDR_Flush_out,

    //Interface to SD Card
    output wire [27:0] SD_Address_out,
    output wire SD_Write_out,
    output wire SD_Read_out,
    input wire [4095:0] SD_Data_in,
    output reg [4095:0] SD_Data_out,
    input wire SD_Ready_in,

    //Cache Signals
    input wire [1:0] Tag_in,
    output wire [18:0] Index_out,
    output wire [1:0] Offset_out,
    output wire [1023:0] Data_out,
    input wire [1023:0] Data_in,
    output wire Write_Data_out,
    output wire Read_Data_out,
    input wire Dirty_in,
    output wire Dirty_out,
    output wire Write_Dirty_out,
    input wire Init_VTD_in,
    output wire Write_Valid_Tag_out,
    output wire Valid_out,
    input wire Hit_in,
    input wire DDR_Interface_Ready_in,
    output wire Reset_VTD,

    //Counter Signals
    input wire [1:0] Offset_Count_in,
    output wire Reset_Offset_Count_out,
    output wire Enable_Offset_Count_out,
    input wire [19:0] Flush_Count_in,
    output wire Reset_Flush_Count_out,
    output wire Enable_Flush_Count_out,

    //Clock and Reset
    input wire Clock_in,
    input wire Reset_in
    );


// State Encoding
parameter STATE_CACHE_INIT              = 4'b0000;
parameter STATE_CACHE_IDLE              = 4'b0001;
parameter STATE_CACHE_HIT_DETECTION     = 4'b0010;
parameter STATE_CACHE_DDR_REQ           = 4'b0011;
parameter STATE_CACHE_DDR_REQ_ACK       = 4'b0100;
parameter STATE_CACHE_ALLOCATE          = 4'b0101;
parameter STATE_CACHE_ALLOCATE_DDR_REQ  = 4'b0110;
parameter STATE_CACHE_ALLOCATE_DDR_ACK  = 4'b0111;
parameter STATE_CACHE_WRITEBACK_DDR_REQ = 4'b1000;
parameter STATE_CACHE_WRITEBACK_DDR_ACK = 4'b1001;
parameter STATE_CACHE_WRITEBACK_ACK     = 4'b1010;
parameter STATE_CACHE_FLUSH             = 4'b1011;
parameter STATE_CACHE_FLUSH_ACK         = 4'b1100;
parameter STATE_CACHE_ALLOCATE_DDR_WRITE_LATENCY = 4'b1101;

// State reg Declarations
reg [3:0] CurrentState;
reg [3:0] NextState;

// Internal Signals
```

```
wire [1:0] Tag;

// Outputs
// Continuous Assignments
assign Tag = (CurrentState == STATE_CACHE_WRITEBACK_DDR_REQ ||
              CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK ||
              CurrentState == STATE_CACHE_WRITEBACK_ACK) ? Tag_in :
              DDR_Address_in[27:26];

// State Outputs
//L1 Interface
assign DDR_Ready_out = (CurrentState == STATE_CACHE_DDR_REQ_ACK) ? 1'b1 : 1'b0;
assign DDR_Flush_out = (CurrentState == STATE_CACHE_FLUSH_ACK) ? 1'b1 : 1'b0;

//SD Interface
assign SD_Address_out = {Tag, Index_out, 7'b0};
assign SD_Write_out = (CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK &&
                      (~DDR_Interface_Ready_in ) && (Offset_Count_in == 2'b11)) ?
                      1'b1 : 1'b0;
assign SD_Read_out = (CurrentState == STATE_CACHE_ALLOCATE) ? 1'b1 : 1'b0;

//DDR Interface
assign Index_out = ((CurrentState == STATE_CACHE_FLUSH) ||
                   ((CurrentState == STATE_CACHE_WRITEBACK_DDR_REQ ||
                   CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK ||
                   CurrentState == STATE_CACHE_WRITEBACK_ACK) && DDR_Flush_in)) ?
                   Flush_Count_in : DDR_Address_in[25:7];
assign Offset_out = (CurrentState == STATE_CACHE_WRITEBACK_DDR_REQ ||
                    CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK ||
                    CurrentState == STATE_CACHE_ALLOCATE_DDR_REQ ||
                    CurrentState == STATE_CACHE_ALLOCATE_DDR_ACK) ?
                    Offset_Count_in : DDR_Address_in[6:5];
assign Data_out = (CurrentState == STATE_CACHE_ALLOCATE_DDR_REQ ||
                  CurrentState == STATE_CACHE_ALLOCATE_DDR_ACK) ?
                  (Offset_Count_in [1]) ? (Offset_Count_in [0]) ?
                  SD_Data_in[4095:3072] : SD_Data_in[3071:2048] :
                  (Offset_Count_in [0]) ? SD_Data_in[2047:1024] :
                  SD_Data_in[1023:0] : DDR_Data_in;
assign Write_Data_out = (CurrentState == STATE_CACHE_DDR_REQ) ? DDR_Write_in :
                        (CurrentState == STATE_CACHE_ALLOCATE_DDR_REQ) ? 1'b1 :
                        1'b0;
assign Read_Data_out = (CurrentState == STATE_CACHE_DDR_REQ) ? DDR_Read_in :
                       (CurrentState == STATE_CACHE_WRITEBACK_DDR_REQ) ? 1'b1 :
                       1'b0;

//VTD Interface
assign Dirty_out = (CurrentState == STATE_CACHE_DDR_REQ && DDR_Write_in) ? 1'b1 :
                   1'b0;
assign Write_Dirty_out = ((CurrentState == STATE_CACHE_DDR_REQ && DDR_Write_in) ||
                         (CurrentState == STATE_CACHE_ALLOCATE_DDR_ACK &&
                         ~DDR_Interface_Ready_in  && Offset_Count_in == 2'b11 &&
                         ~SD_Ready_in)) ? 1'b1 : 1'b0;
assign Valid_out = ((Dirty_out) || (CurrentState == STATE_CACHE_ALLOCATE_DDR_ACK
                  && ~DDR_Interface_Ready_in  && Offset_Count_in == 2'b11 &&
                  ~SD_Ready_in && ~DDR_Flush_in)) ? 1'b1 : 1'b0;
assign Write_Valid_Tag_out = (CurrentState == STATE_CACHE_ALLOCATE_DDR_ACK &&
                             ~DDR_Interface_Ready_in  && Offset_Count_in == 2'b11
                             && ~SD_Ready_in) ? 1'b1 : 1'b0;
assign Reset_VTD = (CurrentState == STATE_CACHE_FLUSH_ACK);

//Cache Counter Interface
assign Reset_Offset_Count_out = (Reset_in || CurrentState == STATE_CACHE_IDLE ||
                                CurrentState == STATE_CACHE_ALLOCATE ||
                                CurrentState == STATE_CACHE_FLUSH) ? 1'b1 : 1'b0;
assign Enable_Offset_Count_out = ((CurrentState == STATE_CACHE_ALLOCATE_DDR_ACK ||
                                 CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK) &&
                                 ~(DDR_Interface_Ready_in) && (Offset_Count_in ==
                                 2'b00 || Offset_Count_in == 2'b01 ||
                                 Offset_Count_in == 2'b10)) ? 1'b1 : 1'b0;
```

```
//Flush Counter Interface
assign Reset_Flush_Count_out = (Reset_in || CurrentState == STATE_CACHE_IDLE ||
                               CurrentState == STATE_CACHE_FLUSH_ACK) ? 1'b1 :
                               1'b0;
assign Enable_Flush_Count_out = (CurrentState == STATE_CACHE_INIT ||
                          CurrentState == STATE_CACHE_ALLOCATE_DDR_WRITE_LATENCY ||
                                 (DDR_Flush_in && CurrentState ==
                                 STATE_CACHE_WRITEBACK_ACK && ~SD_Ready_in) ||
                                 (CurrentState == STATE_CACHE_FLUSH &&
                                 ~Dirty_in)) ? 1'b1 : 1'b0;

// Synchronous State - Transition
always@ (posedge Clock_in or posedge Reset_in) begin
   if (Reset_in == 1'b1) begin
      CurrentState = 4'd0;
      DDR_Data_out = 1024'd0;
      SD_Data_out[4095:3072] = 1024'd0;
      SD_Data_out[3071:2048] = 1024'd0;
      SD_Data_out[2047:1024] = 1024'd0;
      SD_Data_out[1023:0] = 1024'd0;
   end
   else begin
      CurrentState = NextState;
      DDR_Data_out = ((CurrentState == STATE_CACHE_DDR_REQ || CurrentState ==
                     STATE_CACHE_DDR_REQ_ACK) && DDR_Interface_Ready_in) ?
                     Data_in : DDR_Data_out;
      SD_Data_out[4095:3072] = (CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK
                               && Offset_Count_in == 2'b11 &&
                               DDR_Interface_Ready_in) ? Data_in :
                               SD_Data_out[4095:3072];
      SD_Data_out[3071:2048] = (CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK
                               && Offset_Count_in == 2'b10 &&
                               DDR_Interface_Ready_in) ? Data_in :
                               SD_Data_out[3071:2048];
      SD_Data_out[2047:1024] = (CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK
                               && Offset_Count_in == 2'b01 &&
                               DDR_Interface_Ready_in) ? Data_in :
                               SD_Data_out[2047:1024];
      SD_Data_out[1023:0] = (CurrentState == STATE_CACHE_WRITEBACK_DDR_ACK &&
                            Offset_Count_in == 2'b00 &&
                            DDR_Interface_Ready_in) ? Data_in :
                            SD_Data_out[1023:0];
   end
end

// Conditional State - Transition
always@ (*) begin
   case ( CurrentState )
      STATE_CACHE_INIT : begin
         NextState = (~Init_VTD_in) ? STATE_CACHE_IDLE : STATE_CACHE_INIT;
      end
      STATE_CACHE_IDLE : begin //Request
         NextState = (Reset_in) ? STATE_CACHE_INIT : (DDR_Flush_in) ?
                     STATE_CACHE_FLUSH : (DDR_Write_in || DDR_Read_in) ?
                     STATE_CACHE_HIT_DETECTION : CurrentState;
      end
      STATE_CACHE_HIT_DETECTION : begin //Because Tag Data stored in BRAM, takes
                                        //one clock cycle to access
         NextState = (Reset_in) ? STATE_CACHE_INIT : (Hit_in) ?
                     STATE_CACHE_DDR_REQ : (Dirty_in) ?
                     STATE_CACHE_WRITEBACK_DDR_REQ : STATE_CACHE_ALLOCATE;
      end
      STATE_CACHE_DDR_REQ : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : (DDR_Interface_Ready_in) ?
                     STATE_CACHE_DDR_REQ_ACK : CurrentState;
      end
      STATE_CACHE_DDR_REQ_ACK : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : (~DDR_Write_in &&
```

```
                  ~DDR_Read_in) ? STATE_CACHE_IDLE : CurrentState;
      end
      STATE_CACHE_ALLOCATE : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( SD_Ready_in ) ?
                     STATE_CACHE_ALLOCATE_DDR_REQ : CurrentState;
      end
      STATE_CACHE_ALLOCATE_DDR_REQ : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : (DDR_Interface_Ready_in) ?
                     STATE_CACHE_ALLOCATE_DDR_ACK : CurrentState;
      end
      STATE_CACHE_ALLOCATE_DDR_ACK : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~DDR_Interface_Ready_in ) ?
                     (Offset_Count_in == 2'b11) ? (~SD_Ready_in) ?
                     STATE_CACHE_ALLOCATE_DDR_WRITE_LATENCY : CurrentState :
                     STATE_CACHE_ALLOCATE_DDR_REQ : CurrentState;
      end
      STATE_CACHE_ALLOCATE_DDR_WRITE_LATENCY : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( Flush_Count_in ==
                     20'h00004 ) ? STATE_CACHE_HIT_DETECTION : CurrentState;
      end
      STATE_CACHE_WRITEBACK_DDR_REQ : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( DDR_Interface_Ready_in ) ?
                     STATE_CACHE_WRITEBACK_DDR_ACK : CurrentState;
      end
      STATE_CACHE_WRITEBACK_DDR_ACK : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~DDR_Interface_Ready_in ) ?
                     (Offset_Count_in == 2'b11) ? (SD_Ready_in) ?
                     STATE_CACHE_WRITEBACK_ACK : CurrentState :
                     STATE_CACHE_WRITEBACK_DDR_REQ : CurrentState;
      end
      STATE_CACHE_WRITEBACK_ACK : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~SD_Ready_in ) ?
                     (DDR_Flush_in) ? STATE_CACHE_FLUSH : STATE_CACHE_ALLOCATE :
                     CurrentState;
      end
      STATE_CACHE_FLUSH : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( Flush_Count_in ==
                     20'h80000 ) ? STATE_CACHE_FLUSH_ACK : ( Dirty_in ) ?
                     STATE_CACHE_WRITEBACK_DDR_REQ : CurrentState ;
      end
      STATE_CACHE_FLUSH_ACK : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : ( ~DDR_Flush_in ) ?
                     STATE_CACHE_INIT : CurrentState;
      end
      default : begin
         NextState = (Reset_in) ? STATE_CACHE_INIT : STATE_CACHE_IDLE;
      end
   endcase
end

endmodule
`default_nettype wire
```

(14) L2_VALID_TAG_DIRTY_RAM_Wrapper.v

```
`timescale 1ns / 1ps

/*
 * File         : L2_VALID_TAG_DIRTY_RAM_Wrapper.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
```

```
 *
 * Description:
 *   The Valid, Tag, Dirty RAM function the same as in the L1 cache, however we
 *   have combined them in the L2 cache. The L2 cache utilizes BRAM for the
 *   storage of valid, tag and dirty bits.
 */

module L2_VALID_TAG_DIRTY_RAM_Wrapper(
   input Clock_in,
   input Reset_in,
   input [18:0] Index_in,
   input Dirty_in,
   input Write_Dirty_in,
   output Dirty_out,
   output reg Init_VTD_out,
   input [1:0] Tag_in,
   input Write_Valid_Tag_in,
   output [1:0] Tag_out,
   input Valid_in,
   output Valid_out,
   input [19:0] Counter_in);

   L2_VALID_TAG_DIRTY_RAM L2_VTD(
      .clka   (Clock_in),
      .rsta   (Init_VTD_out),
      .wea    ((Write_Valid_Tag_in)||((Init_VTD_out) ? 1'b1 : Write_Dirty_in)),
      .addra  ((Init_VTD_out) ? Counter_in[18:0] : Index_in),
      .dina   ((Init_VTD_out) ? 4'b0000 : {Valid_in, Dirty_in, Tag_in
      .douta  ({Valid_out, Dirty_out, Tag_out}),
      .clkb   (Clock_in),
      .web    (1'b0),
      .addrb  (19'd0),
      .dinb   (4'd0),
      .doutb  ());

   always @ (posedge Clock_in) begin
      if (Reset_in) begin
         Init_VTD_out = 1'b1;
      end
      else begin
         if (Counter_in == 20'h80000) begin
            Init_VTD_out = 1'b0;
         end
      end
   end

endmodule
```

(15) L2_VALID_TAG_DIRTY_RAM.v

```
`timescale 1ns / 1ps

/*
 * File         : L2_VALID_TAG_DIRTY_RAM.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module implement a chunk of block RAM to store the L2 valid, tag and
 *   dirty bits.
 */
```

```
module L2_VALID_TAG_DIRTY_RAM(
   input clka,
   input rsta,
   input wea,
   input [18:0] addra,
   input [3:0] dina,
   output reg [3:0] douta,
   input clkb,
   input rstb,
   input web,
   input [18:0] addrb,
   input [3:0] dinb,
   output reg [3:0] doutb
   );

   (* ram_style = “block” *)
   reg [3:0] RAM [0:524287];
   reg [3:0] dob,doa;

   always @(posedge clka) begin
      RAM[addra] <= (wea) ? dina: RAM[addra];
      doa <= (rsta) ? 4’h0 : (wea) ? dina: RAM[addra];
      douta <= doa;
   end

   always @(posedge clkb) begin
      RAM[addrb] <= (web) ? dinb : RAM[addrb];
      dob <= (rstb) ? 4’h0 : (web) ? dinb: RAM[addrb];
      doutb <= dob;
   end

endmodule
```

(16) Hit_Detection_2.v

```
`timescale 1ns / 1ps

/*
 * File         : Hit_Detection_2.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module compares the tag and upper 2 bits of addresses to determine a hit
 *   in the L2 cache.
 */

module Hit_Detection_2(
   input [1:0] CPU_Tag_in,
   input [1:0] Cache_Tag_in,
   input Valid_Bit_in,
   output HIT_out
   );

   wire Matching_Tag;

   assign Matching_Tag = (CPU_Tag_in == Cache_Tag_in) ? 1’b1 : 1’b0;
   assign HIT_out = Valid_Bit_in && Matching_Tag;

endmodule
```

(17) DDR_Interface.v

```
`timescale 1ns / 1ps
`default_nettype none

/*
 * File         : DDR_Interface.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module takes in data and address from the L2 cache and massages it to
 *   fit the format required for the DDR IP core. It also provides triplication
 *   of the data out to the DDR IC chips.
 *
 *   DDR3 Implemented using Xilinx MIG following guidance from either Enclustra or
 *   Digilent when implementing on the Mercury KX-1 or Genesys 2 respectively.
 */

module DDR_Interface(
   input wire [25:0] Address_in,
   input wire [1023:0] Data_in,
   output wire [1023:0] Data_out,
   input wire Read_in,
   input wire Write_in,
   output wire Ready_out,
   // DDR U/I Signals
   output wire [28:0] app_addr,
   output wire [2:0] app_cmd,
   output wire app_en,
   output wire [255:0] app_wdf_data,
   output wire app_wdf_end,
   output wire app_wdf_wren,
   input wire [255:0] app_rd_data,
   input wire app_rd_data_valid,
   input wire app_rdy,
   input wire app_wdf_rdy,
   input wire ui_clk,
   input wire ui_clk_sync_rst,
   input wire init_calib_complete
   );

   integer l;
   genvar i,j,k;

   parameter STATE_IDLE = 3'b000;
   parameter STATE_SET_COMMAND_READ = 3'b011;
   parameter STATE_SET_COMMAND_WRITE = 3'b100;
   parameter STATE_READ_VALID = 3'b101;
   parameter STATE_ACKNOWLEDGE = 3'b111;

   parameter COMMAND_WRITE = 3'b000;
   parameter COMMAND_READ = 3'b001;

   reg [2:0] CurrentState, NextState;
   reg [3:0] Cycle;
   reg [3:0] Read_Cycle;
   reg [4095:0] Read_Data;

   wire [2:0] Command; // command for current request
   wire Enable; // active-high strobe for 'cmd' and 'addr'
   wire Ready;
```

```
wire Write_Ready; // write data FIFO is ready to receive data
wire Write_Last; // active-high last ‘wdf_data’
wire Write_Enable;
wire [255:0] Data_o;
wire Valid_Read; // active-high ‘rd_data’ valid
wire Init_Complete; // active-high calibration complete
wire Clock, Reset;

assign app_addr = {Address_in[25:5], Cycle, 4’b0000};
assign app_cmd = Command;
assign app_en = Enable;
assign app_wdf_data = {Data_in[Cycle*64+56+:8], Data_in[Cycle*64+56+:8],
                       Data_in[Cycle*64+56+:8], Data_in[Cycle*64+56+:8],
                       Data_in[Cycle*64+48+:8], Data_in[Cycle*64+48+:8],
                       Data_in[Cycle*64+48+:8], Data_in[Cycle*64+48+:8],
                       Data_in[Cycle*64+40+:8], Data_in[Cycle*64+40+:8],
                       Data_in[Cycle*64+40+:8], Data_in[Cycle*64+40+:8],
                       Data_in[Cycle*64+32+:8], Data_in[Cycle*64+32+:8],
                       Data_in[Cycle*64+32+:8], Data_in[Cycle*64+32+:8],
                       Data_in[Cycle*64+24+:8], Data_in[Cycle*64+24+:8],
                       Data_in[Cycle*64+24+:8], Data_in[Cycle*64+24+:8],
                       Data_in[Cycle*64+16+:8], Data_in[Cycle*64+16+:8],
                       Data_in[Cycle*64+16+:8], Data_in[Cycle*64+16+:8],
                       Data_in[Cycle*64+8+:8], Data_in[Cycle*64+8+:8],
                       Data_in[Cycle*64+8+:8], Data_in[Cycle*64+8+:8],
                       Data_in[Cycle*64+:8], Data_in[Cycle*64+:8],
                       Data_in[Cycle*64+:8], Data_in[Cycle*64+:8]};
assign app_wdf_end = Write_Last;
assign app_wdf_wren = Write_Enable;
assign Data_o = app_rd_data;
assign Valid_Read = app_rd_data_valid;
assign Ready = app_rdy;
assign Write_Ready = app_wdf_rdy;
assign Clock = ui_clk;
assign Reset = ui_clk_sync_rst;
assign Init_Complete = init_calib_complete;

always@(posedge Clock) begin
   CurrentState = (Reset) ? STATE_IDLE : NextState ;
end

always@(*) begin
   case(CurrentState)
      STATE_IDLE:
         NextState = (Init_Complete) ? (Write_in) ? STATE_SET_COMMAND_WRITE :
                     (Read_in) ? STATE_SET_COMMAND_READ : STATE_IDLE :
                     STATE_IDLE;
      STATE_SET_COMMAND_READ:
         NextState = (Ready && Cycle == 4’b1111) ? STATE_READ_VALID :
                     STATE_SET_COMMAND_READ;
      STATE_SET_COMMAND_WRITE:
         NextState = (Write_Ready && Ready && Cycle == 4’b1111) ?
                     STATE_ACKNOWLEDGE : STATE_SET_COMMAND_WRITE;
      STATE_READ_VALID:
         NextState = (Valid_Read && Read_Cycle == 4’b1111) ?
                     STATE_ACKNOWLEDGE : STATE_READ_VALID;
      STATE_ACKNOWLEDGE:
         NextState = (~Write_in && ~Read_in) ? STATE_IDLE : STATE_ACKNOWLEDGE;
   endcase
end

assign Enable = (CurrentState == STATE_SET_COMMAND_READ || CurrentState ==
                STATE_SET_COMMAND_WRITE) ? 1’b1 : 1’b0;
assign Command = (CurrentState == STATE_SET_COMMAND_READ) ? COMMAND_READ :
                 COMMAND_WRITE;
assign Write_Enable = (CurrentState == STATE_SET_COMMAND_WRITE && Write_Ready
                       && Ready) ? 1’b1 : 1’b0;
assign Write_Last = (CurrentState == STATE_SET_COMMAND_WRITE && Write_Ready &&
                    Ready) ? 1’b1 : 1’b0;
```

```
   generate
      for (i=0; i<16; i=i+1) begin
         for (j=0; j<8; j=j+1) begin
            for (k=0; k<8; k=k+1) begin
               assign Data_out[i*64+j*8+k] = (Read_Data[i*256+j*32+k] &&
                                              Read_Data[i*256+j*32+k+8]) ||
                                              (Read_Data[i*256+j*32+k+8] &&
                                              Read_Data[i*256+j*32+k+16]) ||
                                              (Read_Data[i*256+j*32+k] &&
                                              Read_Data[i*256+j*32+k+16]);
            end
         end
      end
   endgenerate

   assign Ready_out = (CurrentState == STATE_ACKNOWLEDGE);

   always@(posedge Clock) begin
      for (l=0; l<16; l=l+1) begin
         Read_Data[l*256+:256] = (Reset) ? 256'd0 : (Read_Cycle == l &&
                                 Valid_Read) ? Data_o : Read_Data[l*256+:256];
      end
      Cycle = (Reset) ? 4'd0 : ((CurrentState == STATE_SET_COMMAND_READ && Ready)
              || (CurrentState == STATE_SET_COMMAND_WRITE && Write_Ready &&
              Ready)) ? Cycle + 1 : Cycle;
      Read_Cycle = (Reset) ? 4'd0 : (Valid_Read) ? Read_Cycle + 1 : Read_Cycle;
   end

endmodule
`default_nettype wire
```

(18) SD_Module_Interface.v

```
`timescale 1ns / 1ps
`default_nettype none

/*
 * File         : SD_Module_Interface.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module instantiates the major components for the SD interface, the
 *   controller and the SPI bus.
 */

module SD_Module_Interface(
   input wire Reset,
   input wire MISO_in,
   output wire MOSI_out,
   output wire CS_out,
   input wire SD_Clock,
   output wire [4:0] CurrentState,
   input wire Read_in,
   input wire Write_in,
   output wire Ready_out,
   input wire [29:0] Address_in,
   input wire [4095:0] DDR_Data_out,
   output wire [4095:0] DDR_Data_in,
   output wire Select_SD_Clock_Speed,
```

```
   // SD Card Alignment Signals
   input wire [2:0] Align_in,
   output wire Align_out
   );

   wire [7:0] Data_i, Data_o;
   wire Write, Clock_Speed, Data_Send_Complete;

   SD_Control Control(
      .SD_Clock           (SD_Clock),
      .Reset              (Reset),
      .SD_Data_in         (Data_i),
      .Write_out          (Write),
      .Clock_Speed        (Select_SD_Clock_Speed),
      .SD_Data_out        (Data_o),
      .CS_out             (CS_out),
      .Data_Send_Complete (Data_Send_Complete),
      .CurrentState       (CurrentState),
      .Read_in            (Read_in),
      .Write_in           (Write_in),
      .Ready_out          (Ready_out),
      .Address_in         (Address_in),
      .DDR_Data_out       (DDR_Data_out),
      .DDR_Data_in        (DDR_Data_in),
      // SD Card Alignment Signals
      .Align_in           (Align_in),
      .Align_out          (Align_out)
   );

   SPI_Interface Interface(
      .Reset              (Reset), //Active-High
      .Data_in            (Data_o),
      .Write_in           (Write), //Active-High
      .Data_out           (Data_i),
      .Data_Send_Complete (Data_Send_Complete),
      .SD_Clock           (SD_Clock),
      .MISO               (MISO_in),
      .MOSI               (MOSI_out)
   );

endmodule
`default_nettype wire
```

(19) SD_Control.v

```
`timescale 1ns / 1ps
`default_nettype none

/*
 * File         : SD_Control.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   The SD controller controls what state we are in to perform initialization,
 *   reads, and writes to the SD card. It provides appropriately formatted data
 *   to the output FIFO in the SPI interface and is constantly monitoring the
 *   input shift register of the SPI interface. During reads from the SD card it
 *   registers data to send to the L2 cache. Finally, it allows for the alignment
 *   of the three seperate SD cards at the tail end of any read or write cycle.
 */
```

```
module SD_Control(
   input wire SD_Clock,
   input wire Reset, //Active-High
   input wire [7:0] SD_Data_in,
   output reg Write_out, //Active-High
   output wire Clock_Speed, // 0 = 400kHz, 1 = 50MHz
   output reg [7:0] SD_Data_out,
   output wire CS_out,
   input wire Data_Send_Complete,
   output reg [4:0] CurrentState,
   input wire Read_in,
   input wire Write_in,
   output wire Ready_out,
   input wire [29:0] Address_in,
   input wire [4095:0] DDR_Data_out,
   output reg [4095:0] DDR_Data_in,
   // SD Card Alignment Signals
   input wire [2:0] Align_in,
   output wire Align_out
   );

   wire [2:0] BitCount;
   wire [8:0] ByteCount;
   reg [4:0] NextState;
   wire CountReset, BitEnable, ByteEnable;

   parameter SD_STATE_INIT             = 5'b00000;
   parameter SD_STATE_CMD0             = 5'b00001;
   parameter SD_STATE_CMD0_Response    = 5'b00010;
   parameter SD_STATE_DELAY1           = 5'b00011;
   parameter SD_STATE_CMD55            = 5'b00100;
   parameter SD_STATE_CMD55_Response   = 5'b00101;
   parameter SD_STATE_DELAY2           = 5'b00110;
   parameter SD_STATE_ACMD41           = 5'b00111;
   parameter SD_STATE_ACMD41_Response  = 5'b01000;
   parameter SD_STATE_IDLE             = 5'b01001;
   parameter SD_STATE_CMD17            = 5'b01010;
   parameter SD_STATE_CMD17_Response   = 5'b01011;
   parameter SD_STATE_READTOKEN        = 5'b01100;
   parameter SD_STATE_READBLOCK        = 5'b01101;
   parameter SD_STATE_READCRC          = 5'b01110;
   parameter SD_STATE_READALIGN        = 5'b11110;
   parameter SD_STATE_READACK          = 5'b01111;
   parameter SD_STATE_CMD24            = 5'b10000;
   parameter SD_STATE_CMD24_Response   = 5'b10001;
   parameter SD_STATE_WRITEDELAY       = 5'b10010;
   parameter SD_STATE_WRITETOKEN       = 5'b10011;
   parameter SD_STATE_WRITEBLOCK       = 5'b10100;
   parameter SD_STATE_WRITECRC         = 5'b10101;
   parameter SD_STATE_WRITERESPONSE    = 5'b10110;
   parameter SD_STATE_WRITEALIGN       = 5'b11111;
   parameter SD_STATE_WRITEACK         = 5'b10111;

   initial DDR_Data_in = 4095'd0;

   Counter #(.CountWidth(3)) Bit(
      .Clock_in   (SD_Clock),
      .Enable_in  (BitEnable),
      .Reset_in   (CountReset),
      .Count_out  (BitCount)
   );

   Counter #(.CountWidth(9)) Byte(
      .Clock_in   (SD_Clock),
      .Enable_in  (ByteEnable),
      .Reset_in   (CountReset),
      .Count_out  (ByteCount)
   );
```

```
// Assignments
assign Ready_out = (CurrentState == SD_STATE_READACK || CurrentState ==
                   SD_STATE_WRITEACK);
assign CS_out = (CurrentState == SD_STATE_INIT);
assign Clock_Speed = ~(CurrentState == SD_STATE_INIT || CurrentState ==
                     SD_STATE_CMD0 || CurrentState == SD_STATE_CMD0_Response ||
                     CurrentState == SD_STATE_DELAY1 || CurrentState ==
                     SD_STATE_CMD55 || CurrentState == SD_STATE_CMD55_Response
                     || CurrentState == SD_STATE_DELAY2 || CurrentState ==
                     SD_STATE_ACMD41 || CurrentState ==
                     SD_STATE_ACMD41_Response);
//Disable all counting while SD card is busy.
assign BitEnable = ~((CurrentState == SD_STATE_IDLE) && (SD_Data_in[0] ==
                   1'b0));
//Ensure atleast 1Byte delay between end of reads and writes.
assign ByteEnable = ((BitCount==3'd7)&&~(CurrentState == SD_STATE_IDLE &&
                     ByteCount == 9'd1))||(CurrentState == SD_STATE_WRITEBLOCK);
assign CountReset = (Reset) || (CurrentState != NextState);
assign Align_out = (CurrentState == SD_STATE_READALIGN || CurrentState ==
                   SD_STATE_WRITEALIGN);

always @ (posedge SD_Clock)
begin
   Write_out = ((CurrentState == SD_STATE_CMD0) || (CurrentState ==
               SD_STATE_CMD55) || (CurrentState == SD_STATE_ACMD41) ||
               (CurrentState == SD_STATE_CMD17) || (CurrentState ==
               SD_STATE_CMD24) ||(CurrentState == SD_STATE_WRITETOKEN) ||
               (CurrentState == SD_STATE_WRITEBLOCK)||(CurrentState ==
               SD_STATE_WRITECRC));
   //Data Out
   case (CurrentState)
      SD_STATE_CMD0:
         case (BitCount)
            3'd0 : SD_Data_out = 8'h40;
            3'd5 : SD_Data_out = 8'h95;
            default : SD_Data_out = 8'h00;
         endcase
      SD_STATE_CMD55:
         case (BitCount)
            3'd0 : SD_Data_out = 8'h77;
            default : SD_Data_out = 8'h00;
         endcase
      SD_STATE_ACMD41:
         case (BitCount)
            3'd0 : SD_Data_out = 8'h69;
            default : SD_Data_out = 8'h00;
         endcase
      SD_STATE_CMD17:
         case (BitCount)
            3'd0 : SD_Data_out = 8'h51;
            3'd1 : SD_Data_out = Address_in[29:22];
            3'd2 : SD_Data_out = Address_in[21:14];
            3'd3 : SD_Data_out = Address_in[13:6];
            3'd4 : SD_Data_out = {Address_in[5:0],2'b0};
            default : SD_Data_out = 8'h00;
         endcase
      SD_STATE_CMD24:
         case (BitCount)
            3'd0 : SD_Data_out = 8'h58;
            3'd1 : SD_Data_out = Address_in[29:22];
            3'd2 : SD_Data_out = Address_in[21:14];
            3'd3 : SD_Data_out = Address_in[13:6];
            3'd4 : SD_Data_out = {Address_in[5:0],2'b0};
            default : SD_Data_out = 8'h00;
         endcase
      SD_STATE_WRITETOKEN:
         SD_Data_out = 8'hfe;
```

```
      SD_STATE_WRITEBLOCK:
         SD_Data_out = DDR_Data_out[4095-(8*ByteCount)-:8];
      default :
         SD_Data_out = 8'hff;
   endcase
   //Data In
   if ((CurrentState == SD_STATE_READBLOCK) && (BitCount == 3'd7))
      DDR_Data_in[4095-(8*ByteCount)-:8] = SD_Data_in;
end

// Synchronous State Transistion
always @ (posedge SD_Clock) CurrentState = (Reset) ? SD_STATE_INIT : NextState;

// State Logic

always @ (*)
   case (CurrentState)
      SD_STATE_INIT :
         NextState = (ByteCount == 9'd9 && BitCount == 3'd2) ? SD_STATE_CMD0 :
                     SD_STATE_INIT;
      SD_STATE_CMD0 :
         NextState = (BitCount == 3'd5) ? SD_STATE_CMD0_Response :
                     SD_STATE_CMD0;
      SD_STATE_CMD0_Response :
         NextState = (~SD_Data_in[7]) ? (SD_Data_in == 8'h01) ?
                     SD_STATE_DELAY1 : SD_STATE_INIT : SD_STATE_CMD0_Response;
      SD_STATE_DELAY1 :
         NextState = (ByteCount == 9'd1) ? SD_STATE_CMD55 : SD_STATE_DELAY1;
      SD_STATE_CMD55 :
         NextState = (BitCount == 3'd5) ? SD_STATE_CMD55_Response :
                     SD_STATE_CMD55;
      SD_STATE_CMD55_Response :
         NextState = (~SD_Data_in[7]) ? (SD_Data_in == 8'h01) ?
                     SD_STATE_DELAY2 : SD_STATE_INIT : SD_STATE_CMD55_Response;
      SD_STATE_DELAY2 :
         NextState = (ByteCount == 9'd1) ? SD_STATE_ACMD41 : SD_STATE_DELAY2;
      SD_STATE_ACMD41 :
         NextState = (BitCount == 3'd5) ? SD_STATE_ACMD41_Response :
                     SD_STATE_ACMD41;
      SD_STATE_ACMD41_Response :
         NextState = (~SD_Data_in[7]) ? (SD_Data_in == 8'h00) ? SD_STATE_IDLE :
                     (SD_Data_in == 8'b01) ? SD_STATE_CMD55 : SD_STATE_INIT :
                     SD_STATE_ACMD41_Response;
      SD_STATE_IDLE :
         NextState = (ByteCount == 9'd1) ? (Write_in) ? SD_STATE_CMD24 :
                     (Read_in) ? SD_STATE_CMD17 : SD_STATE_IDLE :
                     SD_STATE_IDLE;
      SD_STATE_CMD17 :
         NextState = (BitCount == 3'd5) ? SD_STATE_CMD17_Response :
                     SD_STATE_CMD17;
      SD_STATE_CMD17_Response :
         NextState = (ByteCount == 9'd20) ? SD_STATE_CMD17 : (~SD_Data_in[7]) ?
                     (SD_Data_in == 8'h00) ? SD_STATE_READTOKEN : (SD_Data_in
                     == 8'h01) ? SD_STATE_INIT : SD_STATE_CMD24 :
                     SD_STATE_CMD17_Response;
      SD_STATE_READTOKEN :
         NextState = (~SD_Data_in[0]) ? SD_STATE_READBLOCK :
                     SD_STATE_READTOKEN;
      SD_STATE_READBLOCK :
         NextState = ((ByteCount == 9'd511) && (BitCount == 3'd7)) ?
                     SD_STATE_READCRC : SD_STATE_READBLOCK;
      SD_STATE_READCRC :
         NextState = (ByteCount == 9'd2) ? SD_STATE_READALIGN :
                     SD_STATE_READCRC;
      SD_STATE_READALIGN :
         NextState = (Align_in == 3'b111) ? SD_STATE_READACK :
                     SD_STATE_READALIGN;
      SD_STATE_READACK :
         NextState = (~Read_in) ? SD_STATE_IDLE : SD_STATE_READACK;
```

```
         SD_STATE_CMD24 :
            NextState = (BitCount == 3'd5) ? SD_STATE_CMD24_Response :
                        SD_STATE_CMD24;
         SD_STATE_CMD24_Response :
            NextState = (ByteCount == 9'd20) ? SD_STATE_CMD24 : (~SD_Data_in[7]) ?
                        (SD_Data_in == 8'h00) ? SD_STATE_WRITEDELAY : (SD_Data_in
                        == 8'h01) ? SD_STATE_INIT : SD_STATE_CMD24 :
                        SD_STATE_CMD24_Response;
         SD_STATE_WRITEDELAY :
            NextState = (ByteCount == 9'd1) ? SD_STATE_WRITETOKEN :
                        SD_STATE_WRITEDELAY;
         SD_STATE_WRITETOKEN :
            NextState = SD_STATE_WRITEBLOCK;
         SD_STATE_WRITEBLOCK :
            NextState = (ByteCount == 9'd511) ? SD_STATE_WRITECRC :
                        SD_STATE_WRITEBLOCK;
         SD_STATE_WRITECRC :
            NextState = (ByteCount == 9'd2) ? SD_STATE_WRITERESPONSE :
                        SD_STATE_WRITECRC;
         SD_STATE_WRITERESPONSE :
            NextState = (SD_Data_in[4:0]==5'b00101) ? SD_STATE_WRITEALIGN :
                        (SD_Data_in[4:0]==5'b01011 || SD_Data_in[4:0]==5'b01101) ?
                        SD_STATE_IDLE : SD_STATE_WRITERESPONSE;
         SD_STATE_WRITEALIGN :
            NextState = (Align_in == 3'b111) ? SD_STATE_WRITEACK :
                        SD_STATE_WRITEALIGN;
         SD_STATE_WRITEACK :
            NextState = (~Write_in) ? SD_STATE_IDLE : SD_STATE_WRITEACK;
         default :
            NextState = SD_STATE_INIT;
      endcase

endmodule
`default_nettype wire
```

(20) SPI_Interface.v

```
`timescale 1ns / 1ps

/*
 * File         : SPI_Interface.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module implements the SPI interface for the SD card. It operates in SPI
 *   mode 1.
 */

module SPI_Interface(
   input Reset, //Active-High
   input [7:0] Data_in,
   input Write_in, //Active-High
   output [7:0] Data_out,
   output Data_Send_Complete,
   input SD_Clock,
   input MISO,
   output MOSI
   );

   SD_Out_FIFO_Wrapper Outgoing(
```

```
      .clk    (SD_Clock),
      .srst   (Reset),
      .din    (Data_in),
      .wr_en  (Write_in),
      .dout   (MOSI),
      .empty  (Data_Send_Complete)
   );

   SD_In_ShiftRegister Incoming(
      .clk    (SD_Clock),
      .srst   (Reset),
      .din    (MISO),
      .dout   (Data_out)
   );

endmodule
```

(21) SD_Out_FIFO_Wrapper.v

```
`timescale 1ns / 1ps

/*
 * File         : SD_Out_FIFO_Wrapper.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   Wrapper for the Outgoing FIFO. The outgoing FIFO was generated using the
 *   XILINX IP generator. The FIFO settings are : Common Clock, BRAM, Write Width
 *   8, Write Depth 512, Read Width 1, First-word Fall-through, synchronous reset,
 *   and a read latency of 0.
 */

module SD_Out_FIFO_Wrapper(
   input clk,
   input srst,
   input [7:0] din,
   input wr_en,
   output reg dout,
   output empty
   );

   wire Temp_out;

   SD_Out_FIFO Outgoing(
      .clk    (clk),
      .srst   (srst),
      .din    (din),
      .wr_en  (wr_en),
      .rd_en  (1'b1),
      .dout   (Temp_out),
      .full   (),
      .empty  (empty)
   );

   always@(negedge clk) dout = (empty) ? 1'b1 : Temp_out;

endmodule
```

(22) SD_In_ShiftRegister.v

```
`timescale 1ns / 1ps

/*
 * File         : SD_In_ShiftRegister.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   Shift register for inputs from the SD card. Most recent 8 bits are stored
 *   for processing as data streams in.
 */

module SD_In_ShiftRegister(
   input clk,
   input srst,
   input din,
   output reg [7:0] dout);

   always@(posedge clk)
   begin
      if (srst) dout = 8'hff;
      else dout = {dout[6:0],din};
   end
endmodule
```

(23) SD_Clock_Gen.v

```
`timescale 1ns / 1ps

/*
 * File         : SD_Clock_Gen.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   SD cards require their initialization clock to be 400kHz max. The Delkin SD
 *   card we are using runs at 50MHz post initialization. This module produces a
 *   400kHz signal from an input 50MHz signal. A BUGMUX is used to both select
 *   the appropriate clock and globally buffer it.
 */

module SD_Clock_Gen(
   input clk_in,
   input select,
   output clk_out
   );

   reg clock400k;
   reg [8:0] count400k;

   initial clock400k = 1'b0;
```

```
   initial count400k = 1'd0;

   always@(posedge clk_in) clock400k = (count400k == 7'd124) ? ~clock400k :
                                       clock400k;
   always@(posedge clk_in) count400k = (count400k == 7'd124) ? 8'd0 :
                                       count400k + 1;

   BUFGMUX SD_Clock(
      .O(clk_out),   // 1-bit output: Clock output
      .I0(clock400k), // 1-bit input: Clock input (S=0)
      .I1(clk_in), // 1-bit input: Clock input (S=1)
      .S(select)    // 1-bit input: Clock select
   );

endmodule
```

c. Input / Output Interfaces

Subsections one through seven contain all the Verilog files to implement the IO interfaces. Subsections one through seven implements the UART interface, while subsection eight implements the PC-104 interface.

(1) uart_bootloader_v2.v

```
`timescale 1ns / 1ps

/*
 * File         : uart_bootloader_v2.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   24-May-2010  GEA       Initial design of standalone bootloader
 *   2.0    7-Jul-2012  GEA       Added data memory bus to allow for general-
 *                                purpose use.
 *   3.0    1-Mar-2016  ASJ       Fixed issue where BootSwEnabled would toggle
 *                                every time data was written to the UART.
 *                                Expanded Bootload address range to encompass the
 *                                entire address space.
 *                                Inserted Flush State to Flush cache memory
 *                                following a bootload.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   An RS-232 compatible UART coupled with the XUM bootloader.
 *
 *   The UART is general-purpose and capable of sending and receiving at a
 *   pre-determined BAUD rate (determined by the clocking module)
 *   with 8 data bits, 1 stop bit, and no parity. In other words it
 *   is 8N1 with only RxD and TxD signals. It uses two 256-byte FIFO
 *   buffers, one for receiving and the other for transmitting.
 *
 *   The XUM bootloader protocol is as follows:
 *
 *      1. Programmer sends 'XUM' ASCII bytes.
 *      2. Programmer sends a number indicating how many 32-bit data words
 *         it has to send, minus 1. (For example, if it has one 32-bit data word,
 *         this number would be 0.) The size of this number is 18 bits.
 *         This means the minimum transmission size is 1 word (32 bits), and
 *         the maximum transmission size is 262144 words, or exactly 1 MB.
```

```
 *         This 18-bit number is sent MSB first, in three bytes, with the six
 *         most-significant bits set to 0.
 *      3. The FPGA sends back the third size byte from the programmer, allowing
 *         the programmer to determine if the FPGA is listening and conforming
 *         to the XUM boot protocol.
 *      4. The programmer sends another 18-bit number indicating the starting
 *         offset in memory where the data should be placed. Normally this will
 *         be 0. This number is also sent in three bytes, and the six most-
 *         significant bits of the first byte are ignored.
 *      5. The programmer sends the data. A copy of each byte that it sends will
 *         be sent back to the programmer from the FPGA, allowing the programmer
 *         to determine if all of the data was transmitted successfully.
 *
 *   On reset, the bootloader is enabled by default. When the bootloader is
 *   enabled, the data memory bus will not see any incoming data. To configure the
 *   UART for general-purpose use, software must issue a write command to the UART
 *   over the data memory bus with bit 8 set. This disables the boot protocol
 *   until the UART is reset again and allows normal use. Note however that there
 *   is a 5-second guard time after reset during which the boot loader is
 *   enabled regardless of any software commands to disable it. After the 5 second
 *   time has lapsed after reset, the software state determines the operating mode
 *   of the UART.
 */
module uart_bootloader(
   input  clock,
   input  reset,
   input  Read,                // MMIO
   input  Write,               // MMIO
   input  [8:0] DataIn,        // MMIO
   output reg [16:0] DataOut,  // MMIO
   output Ack,                 // MMIO
   output DataReady,           // Can be used as an interrupt
   output BootResetCPU,        // XUM Boot Protocol: Reset CPU
   output BootWriteMem,        // XUM Boot Protocol: Write to CPU memory
   output reg [29:0] BootAddr, // XUM Boot Protocol
   output reg [31:0] BootData, // XUM Boot Protocol
   input RxD,                  // UART Rx Signal
   output TxD,                 // UART Tx Signal
   input Inst_Ready,           // Boot Protocol: Cache Busy/#Ready Signal
   output Flush_out,           // Boot Protocol: Flush Cache
   input Flush_Complete_in     // Boot Protocol: Acknowledge Cache Flushed
   );

   localparam [4:0] IDLE=0, WRITE=1, READ=2, BUSW=3, XHEAD1=4, XHEAD2=5, XHEAD3=6,
                    XSIZE1=7, XSIZE2=8, XSIZE3=9,XSIZE4=10, XOFST1=11, XOFST2=12,
                    XOFST3=13, XOFST4=14, XDATA1=15, XDATA2=16, XDATA3=17,
                    XDATA4=18, XWRITE=19, XADDRI=20, XFLUSH = 21;

   // UART module signals
   wire uart_write;
   reg  uart_read;
   wire uart_data_ready;
   wire [7:0] uart_data_in;
   wire [7:0] uart_data_out;
   wire [8:0] uart_rx_count;

   reg [8:0] DataIn_r;             // Latch for incoming data to improve timing
   wire DisableBoot = DataIn_r[8]; // Software boot disable command is bit 8
   reg [28:0] BootTimedEnable;     // Hardware override enabler for boot loader
                                   // after reset
   reg  BootSwEnabled;             // Software enabled/disabled state of
                                   // bootloader
   wire BootProtoEnabled;          // Master bootloader enabled signal
   reg [25:0] rx_count;            // Number of 32-bit words received (boot
                                   // loader)
   reg [25:0] rx_size;             // Number of 32-bit words to expect (boot
                                   // loader)
   reg  [4:0] state;
```

```
always @(posedge clock) begin
   if (reset) begin
      state <= IDLE;
   end
   else begin
      case (state)
         IDLE: begin
            if (Write) state <= WRITE;
            else if (Read) state <= READ;
            else if (BootProtoEnabled & uart_data_ready) state <= XHEAD1;
            else state <= IDLE;
         end
         WRITE: state <= BUSW;
         READ: state <= BUSW;
         BUSW: state <= ~(Read | Write) ? IDLE : BUSW;
         XHEAD1: state <= (uart_data_out == 8'h58) ? XHEAD2 : IDLE; // 'X'
         XHEAD2: state <= (uart_data_ready) ? ((uart_data_out == 8'h55) ?
                           XHEAD3 : IDLE) : XHEAD2;  // 'U'
         XHEAD3: state <= (uart_data_ready) ? ((uart_data_out == 8'h4D) ?
                          XSIZE1 : IDLE) : XHEAD3;  // 'M'
         XSIZE1: state <= (uart_data_ready) ? ((uart_data_out[7:2] ==
                          6'b000000) ? XSIZE2 : IDLE) : XSIZE1;
         XSIZE2: state <= (uart_data_ready) ? XSIZE3 : XSIZE2;
         XSIZE3: state <= (uart_data_ready) ? XSIZE4 : XSIZE3;
         XSIZE4: state <= (uart_data_ready) ? XOFST1 : XSIZE4;
         XOFST1: state <= (uart_data_ready) ? XOFST2 : XOFST1;
         XOFST2: state <= (uart_data_ready) ? XOFST3 : XOFST2;
         XOFST3: state <= (uart_data_ready) ? XOFST4 : XOFST3;
         XOFST4: state <= (uart_data_ready) ? XDATA1 : XOFST4;
         XDATA1: state <= (uart_data_ready) ? XDATA2 : XDATA1;
         XDATA2: state <= (uart_data_ready) ? XDATA3 : XDATA2;
         XDATA3: state <= (uart_data_ready) ? XDATA4 : XDATA3;
         XDATA4: state <= (uart_data_ready) ? XWRITE : XDATA4;
         XWRITE: state <= (Inst_Ready) ? XADDRI : XWRITE;
         XADDRI: state <= (rx_count == rx_size) ? XFLUSH : XDATA1;
         XFLUSH: state <= (Flush_Complete_in) ? IDLE : XFLUSH;
         default: state <= IDLE;
      endcase
   end
end

always @(*) begin
   case (state)
      IDLE: uart_read <= 0;
      WRITE: uart_read <= 0;
      READ: uart_read <= 1;
      BUSW: uart_read <= 0;
      XHEAD1: uart_read <= uart_data_ready;
      XHEAD2: uart_read <= uart_data_ready;
      XHEAD3: uart_read <= uart_data_ready;
      XSIZE1: uart_read <= uart_data_ready;
      XSIZE2: uart_read <= uart_data_ready;
      XSIZE3: uart_read <= uart_data_ready;
      XSIZE4: uart_read <= uart_data_ready;
      XOFST1: uart_read <= uart_data_ready;
      XOFST2: uart_read <= uart_data_ready;
      XOFST3: uart_read <= uart_data_ready;
      XOFST4: uart_read <= uart_data_ready;
      XDATA1: uart_read <= uart_data_ready;
      XDATA2: uart_read <= uart_data_ready;
      XDATA3: uart_read <= uart_data_ready;
      XDATA4: uart_read <= uart_data_ready;
      XWRITE: uart_read <= 0;
      XADDRI: uart_read <= 0;
      default: uart_read <= 0;
   endcase
end

always @(posedge clock) begin
```

```
   DataIn_r <= ((state == IDLE) & Write) ? DataIn : DataIn_r;
end

always @(posedge clock) begin
   DataOut <= (reset) ? 17'h00000 : ((state == READ) ? {uart_rx_count[8:0],
              uart_data_out[7:0]} : DataOut);
end

always @(posedge clock) begin
   // 5 sec @ 100 MHz
   BootTimedEnable <= (reset) ? 29'h00000000 : (BootTimedEnable !=
                      29'h1dcd6500) ? BootTimedEnable + 1 : BootTimedEnable;
   BootSwEnabled <= (reset) ? 1 : ((state == WRITE && DisableBoot) ? 1'b0 :
                    BootSwEnabled);
end

assign BootResetCPU = (state != IDLE) && (state != WRITE) && (state != READ) &&
                      (state != BUSW) && (state != XHEAD1) && (state != XHEAD2)
                      && (state != XHEAD3) && (state != XSIZE1);
assign BootWriteMem = (state == XWRITE);
assign uart_write = ((state == WRITE) & ~DisableBoot) | (uart_data_ready &
                    ((state == XSIZE4) | (state == XDATA1) | (state ==
                    XDATA2) | (state == XDATA3) | (state == XDATA4)));
assign uart_data_in = (state == WRITE) ? DataIn_r[7:0] : uart_data_out;
assign Ack = (state == BUSW);
assign DataReady = uart_data_ready;
assign BootProtoEnabled = BootSwEnabled | (BootTimedEnable != 29'h1dcd6500);
assign Flush_out = (state == XFLUSH);


// XUM Boot Protocol Logic
always @(posedge clock) begin
   BootData[31:24] <= (reset) ? 8'h00 : (((state == XDATA1) &
                      uart_data_ready) ? uart_data_out : BootData[31:24]);
   BootData[23:16] <= (reset) ? 8'h00 : (((state == XDATA2) &
                      uart_data_ready) ? uart_data_out : BootData[23:16]);
   BootData[15:8]  <= (reset) ? 8'h00 : (((state == XDATA3) &
                      uart_data_ready) ? uart_data_out : BootData[15:8]);
   BootData[7:0]   <= (reset) ? 8'h00 : (((state == XDATA4) &
                      uart_data_ready) ? uart_data_out : BootData[7:0]);
end

always @(posedge clock) begin
   if (reset) begin
      BootAddr <= 30'h00000000;
   end
   else if (state == XADDRI) begin
      BootAddr <= BootAddr + 1;
   end
   else begin
      BootAddr[29:24] <= ((state == XOFST1) & uart_data_ready) ?
                         uart_data_out[5:0] : BootAddr[29:24];
      BootAddr[23:16] <= ((state == XOFST2) & uart_data_ready) ?
                         uart_data_out[7:0] : BootAddr[23:16];
      BootAddr[15:8]  <= ((state == XOFST3) & uart_data_ready) ?
                         uart_data_out[7:0] : BootAddr[15:8];
      BootAddr[7:0]   <= ((state == XOFST4) & uart_data_ready) ?
                         uart_data_out[7:0] : BootAddr[7:0];
   end
end

always @(posedge clock) begin
   rx_count <= (state == IDLE) ? 18'h00000 : ((state == XADDRI) ? rx_count +
               1 : rx_count);
end

always @(posedge clock) begin
   rx_size[25:24] <= (reset) ? 2'b00 : (((state == XSIZE1) & uart_data_ready) ?
                     uart_data_out[1:0] : rx_size[25:24]);
```

```
      rx_size[23:16] <= (reset) ? 8'b00 : (((state == XSIZE2) & uart_data_ready) ?
                        uart_data_out[7:0] : rx_size[23:16]);
      rx_size[15:8]  <= (reset) ? 8'h00 : (((state == XSIZE3) & uart_data_ready) ?
                        uart_data_out[7:0] : rx_size[15:8]);
      rx_size[7:0]   <= (reset) ? 8'h00 : (((state == XSIZE4) & uart_data_ready) ?
                        uart_data_out[7:0] : rx_size[7:0]);
   end

   // UART Driver
   uart_min UART (
      .clock       (clock),
      .reset       (reset),
      .write       (uart_write),
      .data_in     (uart_data_in),
      .read        (uart_read),
      .data_out    (uart_data_out),
      .data_ready  (uart_data_ready),
      .rx_count    (uart_rx_count),
      .RxD         (RxD),
      .TxD         (TxD)
   );

endmodule
```

(2) uart-min.v

```
`timescale 1ns / 1ps
/*
 * File         : uart-min.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   24-May-2010  GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   115200 baud 8-N-1 serial port, using only Tx and Rx.
 *   (8 data bits, no parity, 1 stop bit, no flow control.)
 *   Configurable baud rate determined by clocking module, 16x oversampling
 *   for Rx data, Rx filtering, and configurable FIFO buffers for receiving
 *   and transmitting.
 *
 *   Described as '_min' due to lack of overflow and other status signals
 *   as well as the use of only Tx and Rx signals.
 */
module uart_min(
   input clock,
   input reset,
   input write,
   input [7:0] data_in,   // tx going into uart, out of serial port
   input read,
   output [7:0] data_out, // rx coming in from serial port, out of uart
   output data_ready,
   output [8:0] rx_count,
   /*------------------------*/
   input RxD,
   output TxD
   );

   localparam DATA_WIDTH = 8; // Bit-width of FIFO data (should be 8)
   localparam ADDR_WIDTH = 8; // 2^ADDR_WIDTH words of FIFO space

   /* Clocking Signals */
   wire uart_tick, uart_tick_16x;
```

```
/* Receive Signals */
wire [7:0] rx_data;     // Raw bytes coming in from uart
wire rx_data_ready;     // Synchronous pulse indicating this (^)
wire rx_fifo_empty;

/* Send Signals */
reg tx_fifo_deQ = 0;
reg tx_start = 0;
wire tx_free;
wire tx_fifo_empty;
wire [7:0] tx_fifo_data_out;

assign data_ready = ~rx_fifo_empty;

always @(posedge clock) begin
   if (reset) begin
      tx_fifo_deQ <= 0;
      tx_start <= 0;
   end
   else begin
      if (~tx_fifo_empty & tx_free & uart_tick) begin
         tx_fifo_deQ <= 1;
         tx_start <= 1;
      end
      else begin
         tx_fifo_deQ <= 0;
         tx_start <= 0;
      end
   end
end

uart_clock clocks (
   .clock          (clock),
   .uart_tick      (uart_tick),
   .uart_tick_16x  (uart_tick_16x)
);

uart_tx tx (
   .clock          (clock),
   .reset          (reset),
   .uart_tick      (uart_tick),
   .TxD_data       (tx_fifo_data_out),
   .TxD_start      (tx_start),
   .ready          (tx_free),
   .TxD            (TxD)
);

uart_rx rx (
   .clock          (clock),
   .reset          (reset),
   .RxD            (RxD),
   .uart_tick_16x  (uart_tick_16x),
   .RxD_data       (rx_data),
   .data_ready     (rx_data_ready)
);

FIFO_NoFull_Count #(
   .DATA_WIDTH     (DATA_WIDTH),
   .ADDR_WIDTH     (ADDR_WIDTH))
tx_buffer (
   .clock          (clock),
   .reset          (reset),
   .enQ            (write),
   .deQ            (tx_fifo_deQ),
   .data_in        (data_in),
   .data_out       (tx_fifo_data_out),
   .empty          (tx_fifo_empty),
   .count          ()
```

```
   );

   FIFO_NoFull_Count #(
      .DATA_WIDTH     (DATA_WIDTH),
      .ADDR_WIDTH     (ADDR_WIDTH))
   rx_buffer (
      .clock          (clock),
      .reset          (reset),
      .enQ            (rx_data_ready),
      .deQ            (read),
      .data_in        (rx_data),
      .data_out       (data_out),
      .empty          (rx_fifo_empty),
      .count          (rx_count)
   );

endmodule
```

(3) uart_clock.v

```
`timescale 1ns / 1ps
/*
 * File         : uart_clock.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   24-May-2010  GEA       Initial design.
 *   2.0    1-Mar-2016  ASJ       Modified to support 50MHZ clock
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Takes a 100 MHz clock and generates synchronous pulses for 115200 baud
 *   and 16x 115200 baud (synchronized).
 *
 *   This timing can be adjusted to allow for other baud rates.
 */

module uart_clock(
   input clock,
   output uart_tick,
   output uart_tick_16x
   );

   // 100MHz / (2^13 / 151) == 16 * 115203.857 Hz
   // 100MHz / (2^17 / 151) == 115203.857 Hz
   //  66MHz / (2^14 / 453) == 16 * 115203.857 Hz
   //  66MHz / (2^18 / 453) == 115203.857 Hz


/* // 66 MHz version
   reg [14:0] accumulator = 15'h0000;
   always @(posedge clock) begin
      accumulator <= accumulator[13:0] + 906;
   end
   assign uart_tick_16x = accumulator[14];*/


   // 50 MHz version
   reg [13:0] accumulator = 14'h0000;
   always @(posedge clock) begin
      accumulator <= accumulator[12:0] + 302; // + "151*(100 / input clock in
                                              // MHz))"
   end
   assign uart_tick_16x = accumulator[13];
```

```
   //------------------------------
   reg [3:0] uart_16x_count = 4'h0;
   always @(posedge clock) begin
      uart_16x_count <= (uart_tick_16x) ? uart_16x_count + 1 : uart_16x_count;
   end
   assign uart_tick = (uart_tick_16x && (uart_16x_count == 4'b1111));

endmodule
```

(4) uart_tx.v

```
`timescale 1ns / 1ps
/*
 * File         : uart_tx.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   25-Mar-2010  GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Transmits bytes of data from the serial port. Capable of back-to-back
 *   transmission of data for maximum bandwidth utilization.
 *   'TxD_start' must only pulse with a 'uart_tick' pulse. 8N1.
 */
module uart_tx (
   input clock,
   input reset,
   input uart_tick,
   input [7:0] TxD_data,
   input TxD_start,    // Must happen with a uart_tick
   output ready,
   output reg TxD
   );

   localparam [3:0] IDLE=0, START=1, BIT_0=2, BIT_1=3, BIT_2=4, BIT_3=5,
                    BIT_4=6, BIT_5=7, BIT_6=8, BIT_7=9, STOP=10;

   reg [3:0] tx_state = IDLE;
   reg [7:0] TxD_data_r = 8'h00; // Registered input data so it doesn't need to
                                 // be held

   assign ready = (tx_state == IDLE) || (tx_state == STOP);

   always @(posedge clock) begin
      TxD_data_r <= (ready & TxD_start) ? TxD_data : TxD_data_r;
   end

   always @(posedge clock) begin
      if (reset) tx_state <= IDLE;
      else begin
         case (tx_state)
            IDLE:   if (TxD_start) tx_state <= START;
            START:  if (uart_tick) tx_state <= BIT_0;
            BIT_0:  if (uart_tick) tx_state <= BIT_1;
            BIT_1:  if (uart_tick) tx_state <= BIT_2;
            BIT_2:  if (uart_tick) tx_state <= BIT_3;
            BIT_3:  if (uart_tick) tx_state <= BIT_4;
            BIT_4:  if (uart_tick) tx_state <= BIT_5;
            BIT_5:  if (uart_tick) tx_state <= BIT_6;
            BIT_6:  if (uart_tick) tx_state <= BIT_7;
            BIT_7:  if (uart_tick) tx_state <= STOP;
            STOP:   if (uart_tick) tx_state <= (TxD_start) ? START : IDLE;
```

```
            default: tx_state <= 4'bxxxx;
         endcase
      end
   end

   always @(tx_state, TxD_data_r) begin
      case (tx_state)
         IDLE:   TxD <= 1;
         START:  TxD <= 0;
         BIT_0:  TxD <= TxD_data_r[0];
         BIT_1:  TxD <= TxD_data_r[1];
         BIT_2:  TxD <= TxD_data_r[2];
         BIT_3:  TxD <= TxD_data_r[3];
         BIT_4:  TxD <= TxD_data_r[4];
         BIT_5:  TxD <= TxD_data_r[5];
         BIT_6:  TxD <= TxD_data_r[6];
         BIT_7:  TxD <= TxD_data_r[7];
         STOP:   TxD <= 1;
         default: TxD <= 1'bx;
      endcase
   end

endmodule
```

(5) uart_rx.v

```
`timescale 1ns / 1ps
/*
 * File         : uart_rx.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   26-May-2010  GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   Recovers received data from the serial port with 16x clock over-sampling.
 *   'data_ready' is a synchronous pulse indicator. 8N1.
 */
module uart_rx(
   input clock,
   input reset,
   input RxD,
   input uart_tick_16x,
   output reg [7:0] RxD_data = 0,
   output data_ready
   );

   /* Synchronize incoming RxD */
   reg [1:0] RxD_sync = 2'b11; //0;
   always @(posedge clock) RxD_sync <= (uart_tick_16x) ? {RxD_sync[0], RxD} :
                                       RxD_sync;

   /* Filter Input */
   reg [1:0] RxD_cnt = 0;
   reg RxD_bit = 1; //0;
   always @(posedge clock) begin
      if (uart_tick_16x) begin
         case (RxD_sync[1])
            0:  RxD_cnt <= (RxD_cnt == 2'b11) ? RxD_cnt : RxD_cnt + 1;
            1:  RxD_cnt <= (RxD_cnt == 2'b00) ? RxD_cnt : RxD_cnt - 1;
         endcase
         RxD_bit <= (RxD_cnt == 2'b11) ? 0 : ((RxD_cnt == 2'b00) ? 1 : RxD_bit);
      end
```

```
      else begin
         RxD_cnt <= RxD_cnt;
         RxD_bit <= RxD_bit;
      end
   end

   /* State Definitions */
   localparam [3:0] IDLE=0, BIT_0=1, BIT_1=2, BIT_2=3, BIT_3=4, BIT_4=5, BIT_5=6,
                    BIT_6=7, BIT_7=8, STOP=9;
   reg [3:0] state = IDLE;

   /* Next-bit spacing and clock locking */
   reg clock_lock = 0;
   reg [3:0] bit_spacing = 4'b1110;   // Enable quick jumping from IDLE to BIT_0
                                      // when line was idle.
   always @(posedge clock) begin
      if (uart_tick_16x) begin
         if (~clock_lock) clock_lock <= ~RxD_bit; // We lock on when we detect a
                                                  // filtered 0 from idle
         else clock_lock <= ((state == IDLE) && (RxD_bit == 1'b1)) ? 0 :
                            clock_lock;
         bit_spacing <= (clock_lock) ? bit_spacing + 1 : 4'b1110;
      end
      else begin
         clock_lock <= clock_lock;
         bit_spacing <= bit_spacing;
      end
   end
   wire next_bit = (bit_spacing == 4'b1111);

   /* State Machine */
   always @(posedge clock) begin
      if (reset) state <= IDLE;
      else if (uart_tick_16x) begin
         case (state)
            IDLE: state <= (next_bit & (RxD_bit == 1'b0)) ? BIT_0 : IDLE; // Start
                                                                          // bit is 0
            BIT_0:  state <= (next_bit) ? BIT_1 : BIT_0;
            BIT_1:  state <= (next_bit) ? BIT_2 : BIT_1;
            BIT_2:  state <= (next_bit) ? BIT_3 : BIT_2;
            BIT_3:  state <= (next_bit) ? BIT_4 : BIT_3;
            BIT_4:  state <= (next_bit) ? BIT_5 : BIT_4;
            BIT_5:  state <= (next_bit) ? BIT_6 : BIT_5;
            BIT_6:  state <= (next_bit) ? BIT_7 : BIT_6;
            BIT_7:  state <= (next_bit) ? STOP  : BIT_7;
            STOP:   state <= (next_bit) ? IDLE  : STOP;
            default: state <= 4'bxxxx;
         endcase
      end
      else state <= state;
   end

   /* Shift Register to Collect Rx bits as they come */
   wire capture = (uart_tick_16x & next_bit & (state!=IDLE) & (state!=STOP));
   always @(posedge clock) RxD_data <= (capture) ? {RxD_bit, RxD_data[7:1]} :
                                       RxD_data[7:0];
   assign data_ready = (uart_tick_16x & next_bit & (state==STOP));

endmodule
```

(6) FIFO_NoFull_Count.v

```
`timescale 1ns / 1ps
/*
 * File         : FIFO_NoFull_Count.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
```

```
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   24-May-2010  GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A synchronous FIFO of variable data width and depth. ‘enQ’ is ignored when
 *   the FIFO is full and ‘deQ’ is ignored when the FIFO is empty. If ‘enQ’ and
 *   ‘deQ’ are asserted simultaneously, the FIFO is unchanged and the output data
 *   is the same as the input data.
 *
 *   This FIFO is “First word fall-through” meaning data can be read without
 *   asserting ‘deQ’ by merely supplying an address. However, when ‘deQ’ is
 *   asserted, the data is “removed” from the FIFO and one location is freed.
 *   If the FIFO is empty and ‘enQ’ and ‘deQ’ are not asserted simultaneously,
 *   the output data will be 0s.
 *
 * Variation:
 *   - There is no output to indicate the FIFO is full.
 *   - Output ‘count’ indicates how many elements are in the FIFO, from 0 to 256
 *     (for 8-bit ADDR_WIDTH).
 */

module FIFO_NoFull_Count(clock, reset, enQ, deQ, data_in, data_out, empty, count);
   parameter DATA_WIDTH = 8;
   parameter ADDR_WIDTH = 8;
   parameter RAM_DEPTH = 1 << ADDR_WIDTH;
   input clock;
   input reset;
   input enQ;
   input deQ;
   input [(DATA_WIDTH-1):0] data_in;
   output [(DATA_WIDTH-1):0] data_out;
   output empty;
   output reg [(ADDR_WIDTH):0] count; // How many elements are in the FIFO(0->256)

   reg [(ADDR_WIDTH-1):0] enQ_ptr, deQ_ptr; // Addresses for reading from and
                                            // writing to internal memory

   assign empty = (count == 0);
   wire full = (count == (1 << ADDR_WIDTH));

   wire [(DATA_WIDTH-1):0] w_data_out;
   assign data_out = (empty) ? ((enQ & deQ) ? data_in : 0) : w_data_out;

   wire w_enQ = (full) ? 0 : enQ;   // Mask ‘enQ’ when the FIFO is full
   wire w_deQ = (empty) ? 0 : deQ;  // Mask ‘deQ’ when the FIFO is empty

   always @(posedge clock) begin
      if (reset) begin
         enQ_ptr <= 0;
         deQ_ptr <= 0;
         count <= 0;
      end
      else begin
         enQ_ptr <= (w_enQ) ? enQ_ptr + 1 : enQ_ptr;
         deQ_ptr <= (w_deQ) ? deQ_ptr + 1 : deQ_ptr;
         count <= (w_enQ ~^ w_deQ) ? count : ((w_enQ) ? count + 1 : count - 1);
      end
   end

   SRAM #(
      .DATA_WIDTH (DATA_WIDTH),
      .ADDR_WIDTH (ADDR_WIDTH),
      .RAM_DEPTH  (RAM_DEPTH))
   ram(
      .clock   (clock),
```

```
      .wEn     (w_enQ),
      .rAddr   (deQ_ptr),
      .wAddr   (enQ_ptr),
      .dIn     (data_in),
      .dOut    (w_data_out)
   );

endmodule
```

(7) SRAM.v

```
`timescale 1ns / 1ps
/*
 * File         : SRAM.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   4-Apr-2010   GEA       Initial design.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   A simple memory of varying width and depth. Reads are asynchronous,
 *   writes are synchronous. Defaults to 8-bit width and 8-bit depth for
 *   a total of 8-bit * 256 entry or 256 bytes of storage.
 */

module SRAM(clock, wEn, rAddr, wAddr, dIn, dOut);
   parameter DATA_WIDTH = 8;
   parameter ADDR_WIDTH = 8;
   parameter RAM_DEPTH = 1 << ADDR_WIDTH;
   input clock;
   input wEn;
   input [(ADDR_WIDTH-1):0] rAddr;
   input [(ADDR_WIDTH-1):0] wAddr;
   input [(DATA_WIDTH-1):0] dIn;
   output [(DATA_WIDTH-1):0] dOut;

   reg [(DATA_WIDTH-1):0] mem [0:(RAM_DEPTH-1)];
   assign dOut = mem[rAddr];

   always @(posedge clock) begin
      if (wEn) mem[wAddr] <= dIn;
   end

endmodule
```

(8) NPSAT1_ARM_CFTP_Interface

```
`timescale 1ns / 1ps
`default_nettype none

/*
 * File         : NPSAT1_ARM_CFTP_Interface.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
```

```
 *
 * Description:
 *   Interface between the ARM and CFTP. The ARM uses bus timing for the LH*****
 *   with 16 wait cycles. The CFTP uses a 4-way handshake. Incoming and outgoing
 *   IP FIFOs are used with independent clocks, clocked by the respective
 *   source/destination for writes/reads respectively. This mitigates skewing
 *   effects and possible metastability to ensure synchronous operation on either
 *   side of the FIFOs.
 *
 *   XILINX's FIFO IP core was used to generate the FIFO's with the following
 *   settings:
 *
 *   Incoming : Independent Clocks, BRAM, Wirte Width 16, Write Depth 1024, Read
 *   Depth 32, Data Cout Outputs Selected, First-Word Fall-Through, Asynchronous,
 *   and 0 read latency.
 *
 *   Outgoing : Independent Clocks, BRAM, Wirte Width 32, Write Depth 512, Read
 *   Depth 16, Data Cout Outputs Selected, First-Word Fall-Through, Asynchronous,
 *   and 0 read latency.
 */

module NPSAT1_ARM_CFTP_Interface(
   // ARM Interface
   input wire [15:0] ARM_Data_in,
   input wire [10:0] ARM_Address_in,
   input wire ARM_Read_in, //active-low
   input wire ARM_Write_in, //active-low
   input wire ARM_CS_in, //active-low
   input wire ARM_BCLK_in, //51MHz
   output reg [15:0] ARM_Data_out,
   output wire ARM_IRQ_out, //active-high(???)
   output wire ARM_Data_Direction, //High_in, Low_out
   output wire PC104_Tristate,
   // CFTP Interface
   input wire CFTP_Address_in,
   input wire [31:0] CFTP_Data_in,
   output wire [31:0] CFTP_Data_out,
   input wire CFTP_Read_in, //active-high
   input wire CFTP_Write_in, //active-high
   output wire CFTP_Ready_out, //active-high
   input wire CFTP_BCLK_in, //200MHz
   input wire CFTP_Reset_in, //active-high
   output wire CFTP_IRQ_out //active-high
   );

   wire Incoming_Full, Incoming_Empty, Outgoing_Full, Outgoing_Empty;

   //ARM Interface Signals and Parameters
   parameter ARM_STATE_IDLE        = 3'b000;
   parameter ARM_STATE_READ        = 3'b001;
   parameter ARM_STATE_READ_WAIT   = 3'b010;
   parameter ARM_STATE_HIZ         = 3'b011;
   parameter ARM_STATE_WRITE       = 3'b100;
   parameter ARM_STATE_WRITE_WAIT  = 3'b101;
   reg [2:0] ARM_CurrentState, ARM_NextState;
   wire Decode;
   wire [15:0] ARM_Data_FIFO, ARM_Status;
   wire [10:0] Outgoing_FIFO_Count;


   //CFTP Interface Signals and Parameters
   parameter CFTP_STATE_IDLE   = 2'b00;
   parameter CFTP_STATE_WRITE  = 2'b01;
   parameter CFTP_STATE_READ   = 2'b10;
   parameter CFTP_STATE_ACK    = 2'b11;
   reg [1:0] CFTP_CurrentState, CFTP_NextState;
   reg [7:0] Top8;
   wire [9:0] Incoming_FIFO_Count;
   wire [31:0] CFTP_Data;
```

```
ARM_FIFO_in INCOMING ( //Uses first word fall through FIFO
   .rst            (CFTP_Reset_in),
   .wr_clk         (ARM_BCLK_in),
   .rd_clk         (CFTP_BCLK_in),
   .din            ({Top8,ARM_Data_in[7:0]}),
   .wr_en          ((ARM_CurrentState == ARM_STATE_WRITE) && (ARM_Address_in ==
                   11'h340)),
   .rd_en          ((CFTP_CurrentState == CFTP_STATE_READ) && (CFTP_Address_in
                   == 1'b0)),
   .dout           (CFTP_Data),
   .full           (Incoming_Full),
   .empty          (Incoming_Empty),
   .rd_data_count  (Incoming_FIFO_Count)
);

ARM_FIFO_out OUTGOING ( //Uses regular FIFO
   .rst            (CFTP_Reset_in),
   .wr_clk         (CFTP_BCLK_in),
   .rd_clk         (ARM_BCLK_in),
   .din            (CFTP_Data_in),
   .wr_en          ((CFTP_CurrentState == CFTP_STATE_WRITE)  &&
                   (CFTP_Address_in == 1'b0)),
   .rd_en          ((ARM_CurrentState == ARM_STATE_READ) && (ARM_Address_in ==
                   11'h341)),
   .dout           (ARM_Data_FIFO),
   .full           (Outgoing_Full),
   .empty          (Outgoing_Empty),
   .rd_data_count  (Outgoing_FIFO_Count)
);

//ARM Interface Assignments
assign ARM_Status = {3'b0, Outgoing_FIFO_Count, Incoming_Full,
                     ~Outgoing_Empty};
assign Decode = (((ARM_Address_in >= 11'h340) && (ARM_Address_in <= 11'h34F))
                && ~ARM_CS_in);
assign ARM_Data_Direction = ~(Decode && ~ARM_Read_in); //High is into FPGA, Low
                                                       // is out of FPGA...
assign PC104_Tristate = ~(Decode && ~ARM_Read_in && (ARM_CurrentState ==
                        ARM_STATE_READ||ARM_CurrentState ==
                        ARM_STATE_READ_WAIT));
assign ARM_IRQ_out = ~Outgoing_Empty;

//ARM Interface Synchronous Transitions
always@(posedge ARM_BCLK_in) begin
   ARM_Data_out = (ARM_CurrentState == ARM_STATE_READ) ? (ARM_Address_in ==
                  11'h340 || ARM_Address_in == 11'h341) ? {ARM_Data_FIFO[7:0],
                  ARM_Data_FIFO[15:8]} : (ARM_Address_in == 11'h342) ?
                  ARM_Status : ARM_Data_out : ARM_Data_out;
   //Temporarily Store the High Byte Since the ARM is doing 8-bit transactions
   Top8 = ((ARM_CurrentState == ARM_STATE_WRITE) && (ARM_Address_in ==
          11'h341)) ? ARM_Data_in[15:8] : Top8;
end

always@(negedge ARM_BCLK_in) ARM_CurrentState = ARM_NextState;

//ARM Interface State Transitions
always@(*) begin
   case (ARM_CurrentState)
      ARM_STATE_IDLE:
         ARM_NextState = (Decode) ? (~ARM_Write_in) ? ARM_STATE_WRITE :
                         (~ARM_Read_in) ? ARM_STATE_HIZ : ARM_STATE_IDLE :
                         ARM_STATE_IDLE;
      ARM_STATE_WRITE:
         ARM_NextState = ARM_STATE_WRITE_WAIT;
      ARM_STATE_WRITE_WAIT:
         ARM_NextState = (ARM_Write_in) ? ARM_STATE_IDLE :
                         ARM_STATE_WRITE_WAIT;
      ARM_STATE_HIZ:
```

```
            ARM_NextState = ARM_STATE_READ; //Allow one clock cycle to transition
                                            // the bus transcievers to prevent
                                            // driving bus from both the FPGA and
                                            // Bus Transcievers.
         ARM_STATE_READ:
            ARM_NextState = ARM_STATE_READ_WAIT;
         ARM_STATE_READ_WAIT:
            ARM_NextState = (ARM_Read_in) ? ARM_STATE_IDLE : ARM_STATE_READ_WAIT;
         default :
            ARM_NextState = ARM_STATE_IDLE;
      endcase
   end


   //CFTP Interface Assignments
   assign CFTP_IRQ_out = ~Incoming_Empty;
   assign CFTP_Ready_out = (CFTP_CurrentState == CFTP_STATE_ACK);
   assign CFTP_Data_out = (CFTP_Address_in) ? {22'd0 ,Incoming_FIFO_Count} :
                          CFTP_Data;

   //CFTP Interface Synchronous Transitions
   always@(negedge CFTP_BCLK_in) CFTP_CurrentState = CFTP_NextState;

   //CFTP Interface State Transitions
   always@(*) begin
      case (CFTP_CurrentState)
         CFTP_STATE_IDLE:
            CFTP_NextState = (CFTP_Write_in) ? CFTP_STATE_WRITE : (CFTP_Read_in) ?
                             CFTP_STATE_READ : CFTP_STATE_IDLE;
         CFTP_STATE_WRITE:
            CFTP_NextState = CFTP_STATE_ACK;
         CFTP_STATE_READ:
            CFTP_NextState = CFTP_STATE_ACK;
         CFTP_STATE_ACK:
            CFTP_NextState = ~(CFTP_Write_in || CFTP_Read_in) ? CFTP_STATE_IDLE :
                             CFTP_STATE_ACK;
         default :
            CFTP_NextState = CFTP_STATE_IDLE;
      endcase
   end

endmodule
`default_nettype wire
```

d. Memory Map / Interrupt controller (MMU.v)

```
`timescale 1ns / 1ps
/*
 * File         : MMU.v
 * Project      : University of Utah, XUM Project MIPS32 core
 * Creator(s)   : Grant Ayers (ayers@cs.utah.edu)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   23-Jul-2011  GEA       Initial design.
 *   2.0   26-Feb-2016  ASJ       Moved from Top Level. Unused IO devices removed.
 *
 * Standards/Formatting:
 *   Verilog 2001, 4 soft tab, wide column.
 *
 * Description:
 *   This module controls the mapping of address produced by the
 *   to either memory or the IO ports, and drives the control signals
 *   to each appropriately. It also collects the interrrupts to provide
 *   the MIPS32 processor. For the flight board, the UART address becomes the PC104
 *   address and we use the entire 32 bit data bus vice 17 bit.
 */
```

```
module Memory_Mapping_Unit(
   input MIPS32_InstMem_Read,
   input MIPS32_DataMem_Read,
   input [3:0] MIPS32_DataMem_WE,
   input [29:0] MIPS32_InstMem_Address,
   input [29:26] MIPS32_DataMem_Address,
   input UART_Interrupt,
   input UART_Ack,
   input UART_BootResetCPU,
   input UART_BootWriteMem_pre,
   input [16:0] UART_DOUT,
   input [29:0] UART_BootAddress,
   input [31:0] UART_BootData,
   input Data_Ready,
   input [31:0] Data_Data_out,
   output reg MIPS32_DataMem_Ready,
   output MIPS32_NMI,
   output [4:0] MIPS32_Interrupts,
   output reg [31:0] MIPS32_DataMem_In,
   output UART_WE,
   output UART_RE,
   output reg Inst_Read,
   output Data_Read,
   output reg [3:0] Inst_Write,
   output [3:0] Data_Write,
   output reg [29:0] Inst_Address,
   output reg [31:0] Inst_Data_in
);

wire MIPS32_IO_WE;
wire [3:0] UART_BootWriteMem = (UART_BootWriteMem_pre) ? 4'hF : 4'h0;
assign MIPS32_IO_WE = (MIPS32_DataMem_WE == 4'hF) ? 1'b1 : 1'b0;
assign MIPS32_Interrupts[4:1] = 4'h0;
assign MIPS32_Interrupts[0]   = UART_Interrupt;
assign MIPS32_NMI             = 1'b0;

// Allow writes to Instruction Memory Port when bootloading
always @(*) begin
   Inst_Read   <= (UART_BootResetCPU) ? 0 : MIPS32_InstMem_Read;
   Inst_Write   <= (UART_BootResetCPU) ? UART_BootWriteMem : 4'h0;
   Inst_Address <= (UART_BootResetCPU) ? UART_BootAddress : MIPS32_InstMem_Address;
   Inst_Data_in  <= (UART_BootResetCPU) ? UART_BootData : 32'h0000_0000;
end

always @(*) begin
   case (MIPS32_DataMem_Address[29])
      0 : begin
         MIPS32_DataMem_In    <= Data_Data_out;
         MIPS32_DataMem_Ready <= Data_Ready;
      end
      1 : begin
         // Memory-mapped I/O
         case (MIPS32_DataMem_Address[28:26])
            // UART
            3'b011 : begin
               MIPS32_DataMem_In    <= {15'h0000, UART_DOUT[16:0]};
               MIPS32_DataMem_Ready <= UART_Ack;
            end
            default : begin
               MIPS32_DataMem_In    <= 32'h0000_0000;
               MIPS32_DataMem_Ready <= 0;
            end
         endcase
      end
   endcase
end

// Memory
assign Data_Read = (MIPS32_DataMem_Address[29]) ? 0 : MIPS32_DataMem_Read;
```

```
assign Data_Write = (MIPS32_DataMem_Address[29]) ? 4'h0 : MIPS32_DataMem_WE;
// I/O
assign UART_WE = (MIPS32_DataMem_Address[29:26] == 4'b1011) ? MIPS32_IO_WE : 0;
assign UART_RE = (MIPS32_DataMem_Address[29:26] == 4'b1011) ? MIPS32_DataMem_Read : 0;

endmodule
```

e. Soft Error Mitigation Module

The SEM was implemented using XILINX's soft error mitigation module with the following settings set: Enable Error Correction, Repair, 100MHz.

4. Voters

Subsections one through three contain all the Verilog files to implement DTMR and BTMR of all registers in CFTP.

(1) Voter.v

```
`timescale 1ns / 1ps

/*
 * File         : Voter.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module performs a bit-wise majority vote between 3 input vectors.
 */

module Voter #(parameter WIDTH = 1)(
   input  [(WIDTH-1):0] A, B, C,
   output [(WIDTH-1):0] True/*, A_error, B_error, C_error, Bit_error*/
   );

   //wire [(WIDTH-1):0] AB_error, AC_error, BC_error;
   genvar i;

   generate
      for (i = 0; i < WIDTH; i = i +1) begin : Vote_Bit
//       assign AB_error[i] = A[i] ^ B[i];
//       assign AC_error[i] = A[i] ^ C[i];
//       assign BC_error[i] = B[i] ^ C[i];
//       assign A_error[i] = AB_error[i] && AC_error[i];
//       assign B_error[i] = AB_error[i] && BC_error[i];
//       assign C_error[i] = AC_error[i] && BC_error[i];
         assign True[i] =  (A[i] && B[i]) || (A[i] && C[i]) || (B[i] && C[i]);
//       assign Bit_error[i] = AB_error[i] || AC_error[i] || BC_error[i];
      end
   endgenerate

endmodule
```

(2) Processor_Voter.v

```
`timescale 1ns / 1ps

/*
 * File         : Processor_Voter.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module instantiates all DTMR voters.
 */

module Processor_Voter(
   input [1853:0] Vote_0_in,
   input [1853:0] Vote_1_in,
   input [1853:0] Vote_2_in,
   output [1853:0] Vote_out
   );

   Voter #(.WIDTH(2)) InstructionInterface_VOTER (
      .A     (Vote_0_in[1:0]),
      .B     (Vote_1_in[1:0]),
      .C     (Vote_2_in[1:0]),
      .True  (Vote_out[1:0])
   );

   Voter #(.WIDTH(193)) CP0_VOTER (
      .A     (Vote_0_in[194:2]),
      .B     (Vote_1_in[194:2]),
      .C     (Vote_2_in[194:2]),
      .True  (Vote_out[194:2])
   );

   Voter #(.WIDTH(32)) PC_VOTER (
      .A     (Vote_0_in[226:195]),
      .B     (Vote_1_in[226:195]),
      .C     (Vote_2_in[226:195]),
      .True  (Vote_out[226:195])
   );

   Voter #(.WIDTH(98)) IFID_VOTER (
      .A     (Vote_0_in[324:227]),
      .B     (Vote_1_in[324:227]),
      .C     (Vote_2_in[324:227]),
      .True  (Vote_out[324:227])
   );

   Voter #(.WIDTH(992)) RegisterFile_VOTER (
      .A     (Vote_0_in[1316:325]),
      .B     (Vote_1_in[1316:325]),
      .C     (Vote_2_in[1316:325]),
      .True  (Vote_out[1316:325])
   );

   Voter #(.WIDTH(154)) IDEX_VOTER (
      .A     (Vote_0_in[1470:1317]),
      .B     (Vote_1_in[1470:1317]),
      .C     (Vote_2_in[1470:1317]),
      .True  (Vote_out[1470:1317])
   );
```

```
   Voter #(.WIDTH(65)) ALU_VOTER (
      .A     (Vote_0_in[1535:1471]),
      .B     (Vote_1_in[1535:1471]),
      .C     (Vote_2_in[1535:1471]),
      .True  (Vote_out[1535:1471])
   );

   Voter #(.WIDTH(98)) Divider_VOTER (
      .A     (Vote_0_in[1633:1536]),
      .B     (Vote_1_in[1633:1536]),
      .C     (Vote_2_in[1633:1536]),
      .True  (Vote_out[1633:1536])
   );

   Voter #(.WIDTH(117)) EXMEM_VOTER (
      .A     (Vote_0_in[1750:1634]),
      .B     (Vote_1_in[1750:1634]),
      .C     (Vote_2_in[1750:1634]),
      .True  (Vote_out[1750:1634])
   );

   Voter #(.WIDTH(32)) DataInterface_VOTER (
      .A     (Vote_0_in[1782:1751]),
      .B     (Vote_1_in[1782:1751]),
      .C     (Vote_2_in[1782:1751]),
      .True  (Vote_out[1782:1751])
   );

   Voter #(.WIDTH(71)) MEMWB_VOTER (
      .A     (Vote_0_in[1853:1783]),
      .B     (Vote_1_in[1853:1783]),
      .C     (Vote_2_in[1853:1783]),
      .True  (Vote_out[1853:1783])
   );

endmodule
```

(3) Output_Voter.v

```
`timescale 1ns / 1ps
`default_nettype none

/*
 * File         : Processor_Voter.v
 * Project      : Naval Postgraduate School, CFTP Project
 * Creator(s)   : Andrew Jackson(The7thPres@gmail.com)
 *
 * Modification History:
 *   Rev   Date         Initials  Description of Change
 *   1.0   1-Mar-2016   ASJ       Initial design.
 *
 * Standards:
 *   Verilog 2001.
 *
 * Description:
 *   This module instantiates output voters from the TMR level to the top level.
 */

module Output_Voter(
   input wire [333:0] Vote_SOC_A,
   input wire [333:0] Vote_SOC_B,
   input wire [333:0] Vote_SOC_C,
   output wire [28:0] app_addr,
   output wire [2:0] app_cmd,
   output wire app_en,
   output wire [255:0] app_wdf_data,
   output wire app_wdf_end,
   output wire app_wdf_wren,
```

```
    output wire UART_Tx_out,
    output wire status_heartbeat,
    output wire icap_csib,
    output wire icap_rdwrb,
    output wire [31:0] icap_i,
    output wire [2:0] SD_MOSI,
    output wire [2:0] SD_CS,
    output wire SD_SClk
    );

    wire [333:0] True_vote;

    Voter #(.WIDTH(334)) Output_DataMem_Read (
       .A     (Vote_SOC_A),
       .B     (Vote_SOC_B),
       .C     (Vote_SOC_C),
       .True  (True_vote)
    );

    assign app_addr = True_vote[28:0];
    assign app_cmd = True_vote[31:29];
    assign app_en = True_vote[32];
    assign app_wdf_data = True_vote[288:33];
    assign app_wdf_end = True_vote[289];
    assign app_wdf_wren = True_vote[290];
    assign UART_Tx_out = True_vote[291];
    assign status_heartbeat = True_vote[292];
    assign icap_csib = True_vote[293];
    assign icap_rdwrb = True_vote[294];
    assign icap_i = True_vote[326:295];
    assign SD_MOSI = True_vote[329:327];
    assign SD_CS = True_vote[332:330];
    assign SD_SClk = True_vote[333];

endmodule
`default_nettype wire
```

5. Constraints

This subsection details the signal constraints used in CFTP. Subsection one constrains the IO lines on the Genesys Board, subsection two constrains the IO lines on the CFTP flight board, and subsection three constrains the clock lines in both designs.

(1) Genesys2_H.xdc

```
# Compress bitstream and configure for fast Flash programming
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
#set_property BITSTREAM.STARTUP.STARTUPCLK JTAGCLK [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]

#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
# Reset
#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-

#IO_0_14 Sch=cpu_reset_n
set_property PACKAGE_PIN R19 [get_ports reset_n]
set_property IOSTANDARD LVCMOS33 [get_ports reset_n]

#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
# LEDs
#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
```

```
#IO_L11N_T1_SRCC_14 Sch=led[0]
set_property PACKAGE_PIN T28 [get_ports Heartbeat_out]
set_property IOSTANDARD LVCMOS33 [get_ports Heartbeat_out]

#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
# SD
#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-

#set_property -dict { PACKAGE_PIN P28   IOSTANDARD LVCMOS33 } [get_ports {
sd_cd_tri_io[0] }]; #IO_L8N_T1_D12_14 Sch=sd_cd
set_property -dict {PACKAGE_PIN R29 IOSTANDARD LVCMOS33} [get_ports SD_MOSI]
set_property -dict {PACKAGE_PIN R26 IOSTANDARD LVCMOS33} [get_ports SD_MISO]
#set_property -dict { PACKAGE_PIN R30   IOSTANDARD LVCMOS33 } [get_ports { sd_dat1_i }];
#IO_L9P_T1_DQS_14 Sch=sd_dat[1]
#set_property -dict { PACKAGE_PIN P29   IOSTANDARD LVCMOS33 } [get_ports { sd_dat2_i }];
#IO_L7P_T1_D09_14 Sch=sd_dat[2]
set_property -dict {PACKAGE_PIN T30 IOSTANDARD LVCMOS33} [get_ports SD_CS]
set_property -dict {PACKAGE_PIN AE24 IOSTANDARD LVCMOS33} [get_ports SD_PWR]
set_property -dict {PACKAGE_PIN R28 IOSTANDARD LVCMOS33} [get_ports SD_SClk]

#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
# Sysclk, 200MHz
#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
set_property IOSTANDARD LVDS [get_ports clock_200MHz_n]
set_property PACKAGE_PIN AD12 [get_ports clock_200MHz_p]
set_property IOSTANDARD LVDS [get_ports clock_200MHz_p]

#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
# UART (PC-FPGA)
#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-

# TXD, RXD naming uses terminal-centric naming convention
# RXD is reception at the terminal (PC), output from the FPGA
set_property PACKAGE_PIN Y23 [get_ports UART_Tx]
set_property IOSTANDARD LVCMOS33 [get_ports UART_Tx]
# TXD is transmission at the terminal (PC), input to the FPGA
set_property PACKAGE_PIN Y20 [get_ports UART_Rx]
set_property IOSTANDARD LVCMOS33 [get_ports UART_Rx]
```

(2) mercury_kx1_top.xdc

```
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]

# ---------------------
# global clock inputs
# ---------------------

set_property IOSTANDARD LVDS [get_ports clock_200MHz_n]
set_property DIFF_TERM FALSE [get_ports clock_200MHz_n]

set_property PACKAGE_PIN AC18 [get_ports clock_200MHz_p]
set_property IOSTANDARD LVDS [get_ports clock_200MHz_p]
set_property DIFF_TERM FALSE [get_ports clock_200MHz_p]

#set_property PACKAGE_PIN AA3 [get_ports Clk100]
#set_property IOSTANDARD LVCMOS15 [get_ports Clk100]

# ---------------------
# ddr3 sdram device b
# ---------------------

set_property DCI_CASCADE {33} [get_iobanks 34]
# Add VCCAUX_IO HIGH (FFG676 package) or VCCAUX_IO = NORMAL (FBG676 package) to all ddr3*
nets.

set_property IOSTANDARD DIFF_SSTL15 [get_ports ddr3_ck_n]
set_property IOSTANDARD DIFF_SSTL15 [get_ports ddr3_ck_p]
```

```
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_n[0]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_n[1]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_n[2]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_n[3]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_p[0]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_p[1]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_p[2]}]
set_property IOSTANDARD DIFF_SSTL15_T_DCI [get_ports {ddr3_dqs_p[3]}]
set_property IOSTANDARD SSTL15 [get_ports ddr3_cas_n]
set_property IOSTANDARD SSTL15 [get_ports ddr3_cke]
set_property IOSTANDARD SSTL15 [get_ports ddr3_cs_n]
set_property IOSTANDARD SSTL15 [get_ports ddr3_odt]
set_property IOSTANDARD SSTL15 [get_ports ddr3_ras_n]
set_property IOSTANDARD SSTL15 [get_ports ddr3_reset_n]
set_property IOSTANDARD SSTL15 [get_ports ddr3_we_n]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[0]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[10]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[11]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[12]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[13]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[14]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[15]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[1]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[2]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[3]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[4]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[5]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[6]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[7]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[8]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_addr[9]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_ba[0]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_ba[1]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_ba[2]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_dm[0]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_dm[1]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_dm[2]}]
set_property IOSTANDARD SSTL15 [get_ports {ddr3_dm[3]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[0]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[10]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[11]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[12]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[13]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[14]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[15]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[16]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[17]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[18]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[19]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[1]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[20]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[21]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[22]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[23]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[24]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[25]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[26]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[27]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[28]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[29]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[2]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[30]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[31]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[3]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[4]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[5]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[6]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[7]}]
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[8]}]
```

```
set_property IOSTANDARD SSTL15_T_DCI [get_ports {ddr3_dq[9]}]
set_property PACKAGE_PIN AA10 [get_ports {ddr3_dq[18]}]
set_property PACKAGE_PIN AA12 [get_ports {ddr3_dq[21]}]
set_property PACKAGE_PIN AA13 [get_ports {ddr3_dq[20]}]
set_property PACKAGE_PIN AA17 [get_ports {ddr3_ba[2]}]
set_property PACKAGE_PIN AA18 [get_ports ddr3_ras_n]
set_property PACKAGE_PIN AA19 [get_ports {ddr3_addr[0]}]
set_property PACKAGE_PIN AA20 [get_ports {ddr3_addr[1]}]
set_property PACKAGE_PIN AA7 [get_ports {ddr3_dq[11]}]
set_property PACKAGE_PIN AA8 [get_ports {ddr3_dq[10]}]
set_property PACKAGE_PIN AA9 [get_ports {ddr3_dq[14]}]
set_property PACKAGE_PIN AB10 [get_ports {ddr3_dq[19]}]
set_property PACKAGE_PIN AB11 [get_ports {ddr3_dq[16]}]
set_property PACKAGE_PIN AB12 [get_ports {ddr3_dqs_p[2]}]
set_property PACKAGE_PIN AB14 [get_ports {ddr3_ba[0]}]
set_property PACKAGE_PIN AB15 [get_ports {ddr3_ba[1]}]
set_property PACKAGE_PIN AB16 [get_ports ddr3_cas_n]
set_property PACKAGE_PIN AB17 [get_ports ddr3_odt]
set_property PACKAGE_PIN AB19 [get_ports {ddr3_addr[4]}]
set_property PACKAGE_PIN AB20 [get_ports {ddr3_addr[5]}]
set_property PACKAGE_PIN AB7 [get_ports {ddr3_dq[12]}]
set_property PACKAGE_PIN AB9 [get_ports {ddr3_dq[15]}]
set_property PACKAGE_PIN AC11 [get_ports {ddr3_dq[17]}]
set_property PACKAGE_PIN AC13 [get_ports {ddr3_dq[22]}]
set_property PACKAGE_PIN AC16 [get_ports ddr3_we_n]
set_property PACKAGE_PIN AC17 [get_ports ddr3_cke]
set_property PACKAGE_PIN AC19 [get_ports {ddr3_addr[2]}]
set_property PACKAGE_PIN AC7 [get_ports {ddr3_dq[13]}]
set_property PACKAGE_PIN AC8 [get_ports {ddr3_dqs_p[1]}]
set_property PACKAGE_PIN AC9 [get_ports {ddr3_dm[1]}]
set_property PACKAGE_PIN AD10 [get_ports {ddr3_dq[24]}]
set_property PACKAGE_PIN AD11 [get_ports {ddr3_dm[3]}]
set_property PACKAGE_PIN AD13 [get_ports {ddr3_dq[23]}]
set_property PACKAGE_PIN AD19 [get_ports {ddr3_addr[3]}]
set_property PACKAGE_PIN AD20 [get_ports ddr3_ck_p]
set_property PACKAGE_PIN AE10 [get_ports {ddr3_dq[25]}]
set_property PACKAGE_PIN AE12 [get_ports {ddr3_dqs_p[3]}]
set_property PACKAGE_PIN AE13 [get_ports {ddr3_dq[28]}]
set_property PACKAGE_PIN AE7 [get_ports {ddr3_dq[8]}]
set_property PACKAGE_PIN AE8 [get_ports {ddr3_dq[26]}]
set_property PACKAGE_PIN AF10 [get_ports {ddr3_dq[30]}]
set_property PACKAGE_PIN AF13 [get_ports {ddr3_dq[29]}]
set_property PACKAGE_PIN AF7 [get_ports {ddr3_dq[9]}]
set_property PACKAGE_PIN AF8 [get_ports {ddr3_dq[27]}]
set_property PACKAGE_PIN AF9 [get_ports {ddr3_dq[31]}]
set_property PACKAGE_PIN V11 [get_ports {ddr3_dq[0]}]
set_property PACKAGE_PIN V14 [get_ports {ddr3_addr[15]}]
set_property PACKAGE_PIN V16 [get_ports {ddr3_addr[7]}]
set_property PACKAGE_PIN V17 [get_ports {ddr3_addr[8]}]
set_property PACKAGE_PIN V18 [get_ports {ddr3_addr[13]}]
set_property PACKAGE_PIN V19 [get_ports {ddr3_addr[14]}]
set_property PACKAGE_PIN V7 [get_ports {ddr3_dq[3]}]
set_property PACKAGE_PIN V8 [get_ports {ddr3_dq[2]}]
set_property PACKAGE_PIN V9 [get_ports {ddr3_dm[0]}]
set_property PACKAGE_PIN W10 [get_ports {ddr3_dqs_p[0]}]
set_property PACKAGE_PIN W11 [get_ports {ddr3_dq[1]}]
set_property PACKAGE_PIN W15 [get_ports {ddr3_addr[11]}]
set_property PACKAGE_PIN W16 [get_ports {ddr3_addr[12]}]
set_property PACKAGE_PIN W18 [get_ports {ddr3_addr[9]}]
set_property PACKAGE_PIN W19 [get_ports {ddr3_addr[10]}]
set_property PACKAGE_PIN Y10 [get_ports {ddr3_dq[7]}]
set_property PACKAGE_PIN Y11 [get_ports {ddr3_dq[6]}]
set_property PACKAGE_PIN Y13 [get_ports {ddr3_dm[2]}]
set_property PACKAGE_PIN Y15 [get_ports ddr3_reset_n]
set_property PACKAGE_PIN Y16 [get_ports ddr3_cs_n]
set_property PACKAGE_PIN Y17 [get_ports {ddr3_addr[6]}]
set_property PACKAGE_PIN Y7 [get_ports {ddr3_dq[5]}]
set_property PACKAGE_PIN Y8 [get_ports {ddr3_dq[4]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[18]}]
```

```
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[21]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[20]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_ba[2]}]
set_property VCCAUX_IO HIGH [get_ports ddr3_ras_n]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[0]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[1]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[11]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[10]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[14]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[19]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[16]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dqs_p[2]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_ba[0]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_ba[1]}]
set_property VCCAUX_IO HIGH [get_ports ddr3_cas_n]
set_property VCCAUX_IO HIGH [get_ports ddr3_odt]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[4]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[5]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[12]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[15]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[17]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[22]}]
set_property VCCAUX_IO HIGH [get_ports ddr3_we_n]
set_property VCCAUX_IO HIGH [get_ports ddr3_cke]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[2]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[13]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dqs_p[1]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dm[1]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[24]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dm[3]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[23]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[3]}]
set_property VCCAUX_IO HIGH [get_ports ddr3_ck_p]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[25]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dqs_p[3]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[28]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[8]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[26]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[30]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[29]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[9]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[27]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[31]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[0]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[15]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[7]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[8]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[13]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[14]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[3]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[2]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dm[0]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dqs_p[0]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[1]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[11]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[12]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[9]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[10]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[7]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[6]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dm[2]}]
set_property VCCAUX_IO HIGH [get_ports ddr3_reset_n]
set_property VCCAUX_IO HIGH [get_ports ddr3_cs_n]
set_property VCCAUX_IO HIGH [get_ports {ddr3_addr[6]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[5]}]
set_property VCCAUX_IO HIGH [get_ports {ddr3_dq[4]}]


## ---------------------
```

```
## spi flash
## ---------------------

#set_property IOSTANDARD LVCMOS33 [get_ports Flash_Clk]
#set_property IOSTANDARD LVCMOS33 [get_ports Flash_Cs_N]
#set_property IOSTANDARD LVCMOS33 [get_ports Flash_Di]
#set_property IOSTANDARD LVCMOS33 [get_ports Flash_Do]
#set_property IOSTANDARD LVCMOS33 [get_ports Flash_Hold_N]
#set_property IOSTANDARD LVCMOS33 [get_ports Flash_Wp_N]
#set_property PACKAGE_PIN A22 [get_ports Flash_Hold_N]
#set_property PACKAGE_PIN A25 [get_ports Flash_Do]
#set_property PACKAGE_PIN B22 [get_ports Flash_Wp_N]
#set_property PACKAGE_PIN B24 [get_ports Flash_Di]
#set_property PACKAGE_PIN C23 [get_ports Flash_Cs_N]
#set_property PACKAGE_PIN K21 [get_ports Flash_Clk]
#set_property SLEW FAST [get_ports Flash_Clk]
#set_property SLEW FAST [get_ports Flash_Cs_N]
#set_property SLEW FAST [get_ports Flash_Di]
#set_property SLEW FAST [get_ports Flash_Do]
#set_property SLEW FAST [get_ports Flash_Hold_N]
#set_property SLEW FAST [get_ports Flash_Wp_N]


## --------------------
## led
## --------------------

set_property IOSTANDARD LVCMOS33 [get_ports Heartbeat_out]
#set_property IOSTANDARD LVCMOS33 [get_ports {Led_N[1]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {Led_N[2]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {Led_N[3]}]
#set_property PACKAGE_PIN K18 [get_ports {Led_N[3]}]
#set_property PACKAGE_PIN L17 [get_ports {Led_N[2]}]
#set_property PACKAGE_PIN L18 [get_ports {Led_N[1]}]
set_property PACKAGE_PIN M17 [get_ports Heartbeat_out]
set_property SLEW SLOW [get_ports Heartbeat_out]
#set_property SLEW SLOW [get_ports {Led_N[1]}]
#set_property SLEW SLOW [get_ports {Led_N[2]}]
#set_property SLEW SLOW [get_ports {Led_N[3]}]

## --------------------
## bank 12 - module connector
## --------------------

set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[5]}]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L11N_SRCC_AB24]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L11P_SRCC_AA23]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L12N_MRCC_AA24]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L12P_MRCC_Y23]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L13N_MRCC_AA22]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L13P_MRCC_Y22]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L14N_SRCC_AC24]
set_property IOSTANDARD LVCMOS33 [get_ports ARM_BCLK_in]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[5]}]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L18N_AC21]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L18P_AB21]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L19N_VREF_AE21]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[12]}]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L1N_V22]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L1P_U22]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[2]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports ARM_Data_Direction]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports ARM_CS_in]
#set_property IOSTANDARD LVCMOS33 [get_ports Reset]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports ARM_Read_in]
set_property IOSTANDARD LVCMOS33 [get_ports ARM_Write_in]
#set_property IOSTANDARD LVCMOS33 [get_ports IO_B12_L6N_VREF_W21]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Address_in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports ARM_IRQ_out]
#set_property IOSTANDARD LVCMOS33 [get_ports Pwr_Req]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {ARM_Data_inout[10]}]
#set_property PACKAGE_PIN AA22 [get_ports IO_B12_L13N_MRCC_AA22]
#set_property PACKAGE_PIN AA23 [get_ports IO_B12_L11P_SRCC_AA23]
#set_property PACKAGE_PIN AA24 [get_ports IO_B12_L12N_MRCC_AA24]
set_property PACKAGE_PIN AA25 [get_ports {ARM_Data_inout[8]}]
#set_property PACKAGE_PIN AB21 [get_ports IO_B12_L18P_AB21]
set_property PACKAGE_PIN AB22 [get_ports {ARM_Data_inout[5]}]
#set_property PACKAGE_PIN AB24 [get_ports IO_B12_L11N_SRCC_AB24]
set_property PACKAGE_PIN AB25 [get_ports {ARM_Data_inout[9]}]
set_property PACKAGE_PIN AB26 [get_ports {ARM_Data_inout[10]}]
#set_property PACKAGE_PIN AC21 [get_ports IO_B12_L18N_AC21]
set_property PACKAGE_PIN AC22 [get_ports {ARM_Data_inout[4]}]
set_property PACKAGE_PIN AC23 [get_ports ARM_BCLK_in]
#set_property PACKAGE_PIN AC24 [get_ports IO_B12_L14N_SRCC_AC24]
set_property PACKAGE_PIN AC26 [get_ports {ARM_Data_inout[11]}]
set_property PACKAGE_PIN AD21 [get_ports {ARM_Data_inout[12]}]
set_property PACKAGE_PIN AD23 [get_ports {ARM_Address_in[7]}]
set_property PACKAGE_PIN AD24 [get_ports {ARM_Address_in[8]}]
set_property PACKAGE_PIN AD25 [get_ports {ARM_Data_inout[1]}]
set_property PACKAGE_PIN AD26 [get_ports {ARM_Data_inout[3]}]
#set_property PACKAGE_PIN AE21 [get_ports IO_B12_L19N_VREF_AE21]
set_property PACKAGE_PIN AE22 [get_ports {ARM_Data_inout[15]}]
set_property PACKAGE_PIN AE23 [get_ports {ARM_Data_inout[13]}]
set_property PACKAGE_PIN AE25 [get_ports {ARM_Data_inout[0]}]
set_property PACKAGE_PIN AE26 [get_ports {ARM_Data_inout[2]}]
set_property PACKAGE_PIN AF22 [get_ports ARM_Data_Direction]
set_property PACKAGE_PIN AF23 [get_ports {ARM_Data_inout[14]}]
set_property PACKAGE_PIN AF24 [get_ports {ARM_Address_in[9]}]
set_property PACKAGE_PIN AF25 [get_ports {ARM_Address_in[10]}]
#set_property PACKAGE_PIN U22 [get_ports IO_B12_L1P_U22]
#set_property PACKAGE_PIN U24 [get_ports Reset]
set_property PACKAGE_PIN U25 [get_ports ARM_CS_in]
set_property PACKAGE_PIN U26 [get_ports {ARM_Address_in[3]}]
set_property PACKAGE_PIN V21 [get_ports {ARM_Address_in[0]}]
#set_property PACKAGE_PIN V22 [get_ports IO_B12_L1N_V22]
set_property PACKAGE_PIN V23 [get_ports {ARM_Address_in[1]}]
set_property PACKAGE_PIN V24 [get_ports {ARM_Address_in[2]}]
set_property PACKAGE_PIN V26 [get_ports {ARM_Address_in[4]}]
set_property PACKAGE_PIN W20 [get_ports {ARM_Data_inout[7]}]
#set_property PACKAGE_PIN W21 [get_ports IO_B12_L6N_VREF_W21]
#set_property PACKAGE_PIN W23 [get_ports Pwr_Req]
set_property PACKAGE_PIN W24 [get_ports ARM_IRQ_out]
set_property PACKAGE_PIN W25 [get_ports ARM_Write_in]
set_property PACKAGE_PIN W26 [get_ports ARM_Read_in]
set_property PACKAGE_PIN Y21 [get_ports {ARM_Data_inout[6]}]
#set_property PACKAGE_PIN Y22 [get_ports IO_B12_L13P_MRCC_Y22]
#set_property PACKAGE_PIN Y23 [get_ports IO_B12_L12P_MRCC_Y23]
```

```
set_property PACKAGE_PIN Y25 [get_ports {ARM_Address_in[5]}]
set_property PACKAGE_PIN Y26 [get_ports {ARM_Address_in[6]}]

## --------------------
## bank 16 - module connector
## --------------------

set_property IOSTANDARD LVCMOS33 [get_ports SD_PWRn]
#set_property IOSTANDARD LVCMOS33 [get_ports SD3_Dat2]
#set_property IOSTANDARD LVCMOS33 [get_ports SD1_Dat2]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L11P_SRCC_G11]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L12N_MRCC_D10]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L12P_MRCC_E10]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L13N_MRCC_C11]
#set_property IOSTANDARD LVCMOS33 [get_ports SD3_CD]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L14N_SRCC_D11]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L14P_SRCC_E11]
#set_property IOSTANDARD LVCMOS33 [get_ports SD2_Dat1]
set_property IOSTANDARD LVCMOS33 [get_ports SD_MISO[1]]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L16N_F12]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L16P_G12]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L17N_D13]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L17P_D14]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L18N_E12]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L18P_E13]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L19N_VREF_C13]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L19P_C14]
set_property IOSTANDARD LVCMOS33 [get_ports SD_SClk[1]]
set_property IOSTANDARD LVCMOS33 [get_ports SD_MOSI[1]]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L20N_B11]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L20P_B12]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L21N_A14]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L21P_B14]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L22N_A10]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L22P_B10]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L23N_A15]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L23P_B15]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L24N_A12]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L24P_A13]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L2N_G9]
set_property IOSTANDARD LVCMOS33 [get_ports SD_CS[0]]
set_property IOSTANDARD LVCMOS33 [get_ports SD_SClk[0]]
set_property IOSTANDARD LVCMOS33 [get_ports SD_MISO[0]]
set_property IOSTANDARD LVCMOS33 [get_ports SD_MISO[2]]
#set_property IOSTANDARD LVCMOS33 [get_ports SD3_Dat1]
#set_property IOSTANDARD LVCMOS33 [get_ports SD1_Dat1]
set_property IOSTANDARD LVCMOS33 [get_ports SD_MOSI[0]]
##set_property IOSTANDARD LVCMOS33 [get_ports IO_B16_L6N_VREF_H11]
set_property IOSTANDARD LVCMOS33 [get_ports SD_SClk[2]]
#set_property IOSTANDARD LVCMOS33 [get_ports SD2_CD]
#set_property IOSTANDARD LVCMOS33 [get_ports SD1_CD]
set_property IOSTANDARD LVCMOS33 [get_ports SD_CS[2]]
set_property IOSTANDARD LVCMOS33 [get_ports SD_MOSI[2]]
set_property IOSTANDARD LVCMOS33 [get_ports SD_CS[1]]
#set_property IOSTANDARD LVCMOS33 [get_ports SD2_Dat2]
##set_property PACKAGE_PIN A10 [get_ports IO_B16_L22N_A10]
##set_property PACKAGE_PIN A12 [get_ports IO_B16_L24N_A12]
##set_property PACKAGE_PIN A13 [get_ports IO_B16_L24P_A13]
##set_property PACKAGE_PIN A14 [get_ports IO_B16_L21N_A14]
##set_property PACKAGE_PIN A15 [get_ports IO_B16_L23N_A15]
set_property PACKAGE_PIN A8 [get_ports SD_CS[1]]
#set_property PACKAGE_PIN A9 [get_ports SD2_Dat2]
##set_property PACKAGE_PIN B10 [get_ports IO_B16_L22P_B10]
##set_property PACKAGE_PIN B11 [get_ports IO_B16_L20N_B11]
##set_property PACKAGE_PIN B12 [get_ports IO_B16_L20P_B12]
##set_property PACKAGE_PIN B14 [get_ports IO_B16_L21P_B14]
##set_property PACKAGE_PIN B15 [get_ports IO_B16_L23P_B15]
set_property PACKAGE_PIN B9 [get_ports SD_PWRn]
##set_property PACKAGE_PIN C11 [get_ports IO_B16_L13N_MRCC_C11]
```

```
#set_property PACKAGE_PIN C12 [get_ports SD3_CD]
##set_property PACKAGE_PIN C13 [get_ports IO_B16_L19N_VREF_C13]
##set_property PACKAGE_PIN C14 [get_ports IO_B16_L19P_C14]
#set_property PACKAGE_PIN C9 [get_ports SD3_Dat2]
##set_property PACKAGE_PIN D10 [get_ports IO_B16_L12N_MRCC_D10]
##set_property PACKAGE_PIN D11 [get_ports IO_B16_L14N_SRCC_D11]
##set_property PACKAGE_PIN D13 [get_ports IO_B16_L17N_D13]
##set_property PACKAGE_PIN D14 [get_ports IO_B16_L17P_D14]
set_property PACKAGE_PIN D8 [get_ports SD_CS[2]]
set_property PACKAGE_PIN D9 [get_ports SD_MOSI[2]]
##set_property PACKAGE_PIN E10 [get_ports IO_B16_L12P_MRCC_E10]
##set_property PACKAGE_PIN E11 [get_ports IO_B16_L14P_SRCC_E11]
##set_property PACKAGE_PIN E12 [get_ports IO_B16_L18N_E12]
##set_property PACKAGE_PIN E13 [get_ports IO_B16_L18P_E13]
#set_property PACKAGE_PIN F10 [get_ports SD1_Dat2]
##set_property PACKAGE_PIN F12 [get_ports IO_B16_L16N_F12]
#set_property PACKAGE_PIN F13 [get_ports SD2_Dat1]
set_property PACKAGE_PIN F14 [get_ports SD_MISO[1]]
#set_property PACKAGE_PIN F8 [get_ports SD2_CD]
#set_property PACKAGE_PIN F9 [get_ports SD1_CD]
set_property PACKAGE_PIN G10 [get_ports SD_CS[0]]
##set_property PACKAGE_PIN G11 [get_ports IO_B16_L11P_SRCC_G11]
##set_property PACKAGE_PIN G12 [get_ports IO_B16_L16P_G12]
#set_property PACKAGE_PIN G14 [get_ports SD1_Dat1]
##set_property PACKAGE_PIN G9 [get_ports IO_B16_L2N_G9]
##set_property PACKAGE_PIN H11 [get_ports IO_B16_L6N_VREF_H11]
set_property PACKAGE_PIN H12 [get_ports SD_SClk[2]]
set_property PACKAGE_PIN H13 [get_ports SD_SClk[0]]
set_property PACKAGE_PIN H14 [get_ports SD_MISO[0]]
set_property PACKAGE_PIN H8 [get_ports SD_SClk[1]]
set_property PACKAGE_PIN H9 [get_ports SD_MOSI[1]]
set_property PACKAGE_PIN J10 [get_ports SD_MISO[2]]
#set_property PACKAGE_PIN J11 [get_ports SD3_Dat1]
set_property PACKAGE_PIN J13 [get_ports SD_MOSI[0]]

set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.STARTUP.STARTUPCLK JTAGCLK [current_design]
```

(3) timing.xdc

```
#Fast to Slow Clock
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *clk_out2_Master_Clock_Divider}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *clk_out2_Master_Clock_Divider}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *clk_out3_Master_Clock_Divider}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *clk_out3_Master_Clock_Divider}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *SD_Clock[0]}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *SD_Clock[0]}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *SD_Clock[1]}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *SD_Clock[1]}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *SD_Clock[2]}] 4
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_pll_i}] -to
   [get_clocks -filter {NAME =~ *SD_Clock[2]}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~
   *clk_out7_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] 4
```

```
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~
   *clk_out7_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] 3
set_multicycle_path -setup -start -from [get_clocks -filter {NAME =~ *clk_ref_mmcm_400}]
-
   to [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] 8
set_multicycle_path -hold -start -from [get_clocks -filter {NAME =~ *clk_ref_mmcm_400}] -
   to [get_clocks -filter {NAME =~ *clk_out1_Master_Clock_Divider}] 7


#Slow to Fast Clock
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~
   *clk_out2_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~
   *clk_out2_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~
   *clk_out3_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~
   *clk_out3_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_pll_i}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~ *SD_Clock[0]}] -to
   [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~ *SD_Clock[0]}] -to
   [get_clocks -filter {NAME =~ *clk_pll_i}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~ *SD_Clock[1]}] -to
   [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~ *SD_Clock[1]}] -to
   [get_clocks -filter {NAME =~ *clk_pll_i}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~ *SD_Clock[2]}] -to
   [get_clocks -filter {NAME =~ *clk_pll_i}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~ *SD_Clock[2]}] -to
   [get_clocks -filter {NAME =~ *clk_pll_i}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~
   *clk_out7_Master_Clock_Divider}] 4
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~
   *clk_out7_Master_Clock_Divider}] 3
set_multicycle_path -setup -end -from [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_ref_mmcm_400}]
   8
set_multicycle_path -hold -end -from [get_clocks -filter {NAME =~
   *clk_out1_Master_Clock_Divider}] -to [get_clocks -filter {NAME =~ *clk_ref_mmcm_400}]
   7
```

B. XUM SOFTWARE SOURCE CODE

The following source code is from demo program 4, the UART Demo, written by Grant Ayers for the XUM project. Modifications were made to cause the program to send an incrementing ascii character every half second and also to echo back to the terminal any character received. Unnecessary IO modules were removed as they would not be needed for the CFTP flight board / programs. For the flight programs, the uart.h and uart.c are modified to support the PC-104 interface.

(1) boot.asm

```
################################################################################
# TITLE: Boot Up Code
# AUTHOR: Grant Ayers (ayers@cs.utah.edu)
# DATE: 19 July 2011
# FILENAME: boot.asm
# PROJECT: University of Utah XUM Single Core
# DESCRIPTION:
#   Initializes the global pointer and stack pointer.
#   Zeros BSS memory region and jumps to main().
#
################################################################################


        .text
        .balign 4
        .global boot
        .ent    boot
        .set    noreorder
boot:
        la      $t0, _bss_start       # Defined in linker script
        la      $t1, _bss_end
        la      $sp, _sp
        la      $gp, _gp

$bss_clear:
        beq     $t0, $t1, $cp0_setup  # Loop until BSS is cleared
        nop
        sb      $0, 0($t0)
        j       $bss_clear
        addiu   $t0, $t0, 1

$cp0_setup:
        la      $26, $run             # Load the address of $run into
        mtc0    $26, $30, 0           #   the ErrorEPC
        mfc0    $26, $13, 0           # Load Cause register
        lui     $27, 0x0080           # Use “special” interrupt vector
        or      $26, $26, $27
        mtc0    $26, $13, 0           # Commit new Cause register
        mfc0    $26, $12, 0           # Load Status register
        lui     $27, 0x0fff           # Disable access to Coprocessors,
        ori     $27, $27, 0xffef      # Base operating mode is Kernel
        and     $26, $26, $27
        ori     $27, $0, 0xff01              # Enable all interrupts
        or      $26, $26, $27
        mtc0    $26, $12, 0           # Commit new Status register

        #lui    $26, 0x0000           # 1ms timer (50 MHz)
        #ori    $26, $26, 0xc350
        #lui    $26, 0x0007           # 10ms timer (50 MHz)
        #ori    $26, $26, 0xa120
        #lui    $26, 0x004c           # 100ms timer (50 MHz)
        #ori    $26, $26, 0x4b40
        #lui    $26, 0x00be           # 250ms timer (50 MHz)
        #ori    $26, 0xbc20
        lui     $26, 0x017d           # 500ms timer (50 MHz)
        ori     $26, 0x7840
        #lui    $26, 0x02fa           # 1 sec timer (50 MHz)
        #ori    $26, $26, 0xf080
        mtc0    $26, $11, 0           # Set Compare register to timer value

        eret                          # Return from Reset Exception

$run:
        jal     main
        nop

$done:
```

```
        j       $done
        nop

        .end boot
```

(2) vectors.asm

```
##############################################################################
# TITLE: Exception Vectors
# AUTHOR: Grant Ayers (ayers@cs.utah.edu)
# DATE: 23 May 2012
# FILENAME: exceptions.asm
# PROJECT: University of Utah XUM Single Core
# DESCRIPTION:
#   Provides the exception vectors which jump to
#   exception-handling routines.
#
##############################################################################


#  Current setup:
#    1. The exception vector begins at address 0x0.
#    2. The interrupt vector begins at address 0x8.
#    3. Each vector has room for 2 instructions (8 bytes) with which
#       it must jump to its demultiplexing routine. The demultiplexing
#       routine calls individual exception-specific handlers.
#    4. The linker script must ensure that this code is placed at the
#       correct address.


        .text
        .balign 4
        .ent    exception_vector
        .set    noreorder
exception_vector:
        j       mips32_general_exception
        nop
        .end    exception_vector


        .ent    interrupt_vector
interrupt_vector:
        j       mips32_interrupt_exception
        nop
        .end    interrupt_vector
```

(3) exceptions.asm

```
##############################################################################
# TITLE: Exception Vectors
# AUTHOR: Grant Ayers (ayers@cs.utah.edu)
# DATE: 23 May 2012
# FILENAME: exceptions.asm
# PROJECT: University of Utah XUM Single Core
# DESCRIPTION:
#   Provides the exception vectors which jump to
#   exception-handling routines.
#
##############################################################################

        .text
        .balign 4
        .set    noreorder
        .set    noat

exc_save:
```

```
        # Save all registers except k0 k1 sp ra
        addiu   $sp, $sp, -112
        sw      $1,    0($sp)
        sw      $2,    4($sp)
        sw      $3,    8($sp)
        sw      $4,   12($sp)
        sw      $5,   16($sp)
        sw      $6,   20($sp)
        sw      $7,   24($sp)
        sw      $8,   28($sp)
        sw      $9,   32($sp)
        sw      $10,  36($sp)
        sw      $11,  40($sp)
        sw      $12,  44($sp)
        sw      $13,  48($sp)
        sw      $14,  52($sp)
        sw      $15,  56($sp)
        sw      $16,  60($sp)
        sw      $17,  64($sp)
        sw      $18,  68($sp)
        sw      $19,  72($sp)
        sw      $20,  76($sp)
        sw      $21,  80($sp)
        sw      $22,  84($sp)
        sw      $23,  88($sp)
        sw      $24,  92($sp)
        sw      $25,  96($sp)
        sw      $28, 100($sp)
        jr      $ra
        sw      $30, 104($sp)

exc_restore:
        # Restore all registers except k0 k1 sp ra
        lw      $1,    0($sp)
        lw      $2,    4($sp)
        lw      $3,    8($sp)
        lw      $4,   12($sp)
        lw      $5,   16($sp)
        lw      $6,   20($sp)
        lw      $7,   24($sp)
        lw      $8,   28($sp)
        lw      $9,   32($sp)
        lw      $10,  36($sp)
        lw      $11,  40($sp)
        lw      $12,  44($sp)
        lw      $13,  48($sp)
        lw      $14,  52($sp)
        lw      $15,  56($sp)
        lw      $16,  60($sp)
        lw      $17,  64($sp)
        lw      $18,  68($sp)
        lw      $19,  72($sp)
        lw      $20,  76($sp)
        lw      $21,  80($sp)
        lw      $22,  84($sp)
        lw      $23,  88($sp)
        lw      $24,  92($sp)
        lw      $25,  96($sp)
        lw      $28, 100($sp)
        lw      $30, 104($sp)
        jr      $ra
        addiu   $sp, $sp, 112


        .global mips32_general_exception
        .ent    mips32_general_exception
mips32_general_exception:
        or      $26, $0, $ra
        jal     exc_save
```

```
        nop
        mfc0    $27, $13, 0           # Read Cause which has ExcCode bits
        srl     $27, $27, 2           # Extract exception code to $k1
        andi    $27, $27, 0x001f

        la      $ra, $end_exception   # Jump to the appropriate handler
        addiu   $t0, $0, 4
        addiu   $t1, $0, 5
        addiu   $t2, $0, 8
        addiu   $t3, $0, 9
        beq     $t0, $27, mips32_handler_AdEL
        addiu   $t0, $0, 10
        beq     $t1, $27, mips32_handler_AdES
        addiu   $t1, $0, 11
        beq     $t2, $27, mips32_handler_Sys
        addiu   $t2, $0, 12
        beq     $t3, $27, mips32_handler_Bp
        addiu   $t3, $0, 13
        beq     $t0, $27, mips32_handler_RI
        nop
        beq     $t1, $27, mips32_handler_CpU
        nop
        beq     $t2, $27, mips32_handler_Ov
        nop
        beq     $t3, $27, mips32_handler_Tr
        nop

$end_exception:
        jal     exc_restore
        nop
        xor     $27, $0, $0
        or      $ra, $0, $26
        xor     $26, $0, $0
        eret
        .end    mips32_general_exception



### “Special” Interrupt Vector: Cause_IV must be set.

        .ent    mips32_interrupt_exception
        .global mips32_interrupt_exception
mips32_interrupt_exception:
        mfc0    $26, $12, 0           # Status register for IM bits
        mfc0    $27, $13, 0           # Cause register for IP bits
        and     $26, $26, $27         # Extract pending, unmasked interrupts
        srl     $26, $26, 8
        andi    $26, $26, 0x00ff

        addu    $27, $0, $ra
        jal     exc_save
        nop
        clz     $26, $26
        la      $ra, $end_interrupt   # All C functions will return here
        addiu   $t0, $0, 24
        addiu   $t1, $0, 25
        addiu   $t2, $0, 26
        beq     $26, $t0, fixed_timer
        addiu   $t0, $0, 27
        beq     $26, $t1, mips32_handler_HwInt4
        addiu   $t1, $0, 28
        beq     $26, $t2, mips32_handler_HwInt3
        addiu   $t2, $0, 29
        beq     $26, $t0, mips32_handler_HwInt2
        addiu   $t0, $0, 30
        beq     $26, $t1, mips32_handler_HwInt1
        addiu   $t1, $0, 31
        beq     $26, $t2, mips32_handler_HwInt0
        nop
```

```
        beq     $26, $t0, mips32_handler_SwInt1
        nop
        beq     $26, $t1, mips32_handler_SwInt0
        nop


$end_interrupt:
        jal     exc_restore
        nop
        xor     $26, $0, $0
        or      $ra, $0, $27
        xor     $27, $0, $0
        eret
        .end    mips32_general_exception

fixed_timer:
    li      $t0, 0x17d7840      # Hardcoded value
    mfc0    $t1, $11, 0         # Current value of Compare
    addu    $t1, $t0, $t1       # Add ticks for the next interval
    mtc0    $t1, $11, 0         # Commit new Compare (clears interrupt)
    j mips32_handler_HwInt5     # Run the user handler
    nop
```

(4) excpetion_handler.h

```
#ifndef __exception_handler_h__
#define __exception_handler_h__

/* MIPS32 Exception Handlers */
void mips32_handler_AdEL(void);
void mips32_handler_AdES(void);
void mips32_handler_Bp(void);
void mips32_handler_CpU(void);
void mips32_handler_Ov(void);
void mips32_handler_RI(void);
void mips32_handler_Sys(void);
void mips32_handler_Tr(void);

/* MIPS32 Interrupt Handlers */
void mips32_handler_HwInt5(void);
void mips32_handler_HwInt4(void);
void mips32_handler_HwInt3(void);
void mips32_handler_HwInt2(void);
void mips32_handler_HwInt1(void);
void mips32_handler_HwInt0(void);
void mips32_handler_SwInt1(void);
void mips32_handler_SwInt0(void);

#endif
```

(5) exception_handler.c

```
#include “drivers/monitor.h”
#include “drivers/uart.h”

void dead_loop(void)
{
        for (;;) {}
}

void mips32_handler_AdEL(void)
{
        dead_loop();
}

void mips32_handler_AdES(void)
{
```

```
        dead_loop();
}

void mips32_handler_Bp(void)
{
        dead_loop();
}

void mips32_handler_CpU(void)
{
        dead_loop();
}

void mips32_handler_Ov(void)
{
        dead_loop();
}

void mips32_handler_RI(void)
{
        dead_loop();
}

void mips32_handler_Sys(void)
{
        dead_loop();
}

void mips32_handler_Tr(void)
{
        dead_loop();
}

/* Timer */
void mips32_handler_HwInt5(void)
{
        static volatile uint8_t cursor = 0;

        UART_writeByte(cursor);

        cursor++;
}

void mips32_handler_HwInt4(void)
{
}

void mips32_handler_HwInt3(void)
{
}

void mips32_handler_HwInt2(void)
{
}

void mips32_handler_HwInt1(void)
{
}


/* UART */
void mips32_handler_HwInt0(void)
{
        uint32_t recv_msg;
        uint32_t bytes_avail;
        uint8_t  read_byte;

        recv_msg = UART_readMessage();
        read_byte = (uint8_t)recv_msg;
```

```
        bytes_avail = (recv_msg >> 8);

        while (bytes_avail > 0) {
                if (read_byte == 0x7f) {  // delete
                        UART_writeByte(read_byte);
                }
                else {
                        UART_writeByte(read_byte);
                }
                bytes_avail--;
                read_byte = UART_readByte();
        }
}

void mips32_handler_SwInt1(void)
{
}

void mips32_handler_SwInt0(void)
{
}
```

(6) uart.h

```
#ifndef __UART_H__
#define __UART_H__

#include <stdint.h>

#define UART_ADDRESS 0xB0000000

void UART_disableBoot(void);
uint8_t UART_readByte(void);
uint32_t UART_readMessage(void);
void UART_writeByte(uint8_t byte);

#endif
```

(7) uart.c

```
#include “uart.h”

void UART_disableBoot(void)
{
        volatile uint32_t *uart = (volatile uint32_t *)UART_ADDRESS;
        uint32_t data;

        data = 0x00000100;

        *uart = data;
}

uint8_t UART_readByte(void)
{
        volatile uint32_t *uart = (volatile uint32_t *)UART_ADDRESS;
        uint32_t data;

        data = *uart;

        return (uint8_t)data;
}

uint32_t UART_readMessage(void)
{
```

```
	volatile uint32_t *uart = (volatile uint32_t *)UART_ADDRESS;

	return *uart;
}

void UART_writeByte(uint8_t byte)
{
	volatile uint32_t *uart = (volatile uint32_t *)UART_ADDRESS;
	uint32_t data;

	data = (uint32_t)byte;
	*uart = data;
}
```

(8) app.c

```
#include “drivers/uart.h”


int main(void)
{
	UART_disableBoot();

	while (1) {};

	return 0;
}
```

(9) xum.ls

```
/* Linker script for MIPS32 (Single Core) FPGA, intended for XUM */


/* Entry Point
 *
 * Set it to be the label “boot” (likely in boot.asm)
 *
 */
/*ENTRY(boot)*/


/* Memory Section
 *
 * The FPGA currently uses one region of Block RAM, which is 592 KB.
 *
 * Instruction Memory starts at address 0.
 *
 * Data Memory ends 592KB later, at address 0x00094000 (the last
 * usable word address is 0x00093ffc).
 *
 *   Instructions :    0x00000000 -> 0x0000fffc    ( 64KB)
 *   Data / BSS   :    0x00010000 -> 0x00017ffc    ( 32KB)
 *   Stack / Heap :    0x00018000 -> 0x00093ffc    (496KB)
 *
 *
 */

/* Sections
 *
 */

SECTIONS
{
	_sp = 0x00094000;
```

```
        . = 0 ;

        .text :
        {
                vectors.o(.text)
                . = 0x10 ;
                boot.o(.text)
                exceptions.o(.text)
                *(.*text*)
        }

        . = 0x00010000 ;

        .data :
        {
                *(.rodata*)
                *(.data*)
        }

        _gp = ALIGN(16) + 0x7ff0;

        .got :
        {
                *(.got)
        }

        .sdata :
        {
                *(.*sdata*)
        }

        _bss_start = . ;

        .sbss :
        {
                *(.*sbss)
        }

        .bss :
        {
                *(.*bss)
        }

        _bss_end = . ;

}
```

(10) Makefile

```
# Makefile for XUM
#
# Compiles code to run on the XUM platform, which
# is based on MIPS32 and a GCC cross-compiler toolchain.
#
# Author: Grant Ayers (ayers@cs.utah.edu)
# Date:   3 July 2012
#
# Modified by: Andrew Jackson (The7thPres@gmail.com
# Date:   1 March 2016
#

SHELL = /bin/sh

SRC          = ../src
MIPS_PREFIX  = /home/Thesis/Software/mips32/mips_tc
MIPS_BIN     = $(MIPS_PREFIX)/bin
MIPS_LIB     = $(MIPS_PREFIX)/mips-elf/lib
MIPS_CC      = $(MIPS_BIN)/mips-elf-gcc-5.2.0.exe
```

```
MIPS_AS      = $(MIPS_BIN)/mips-elf-as.exe
MIPS_LD      = $(MIPS_BIN)/mips-elf-ld.exe
MIPS_OBJDUMP = $(MIPS_BIN)/mips-elf-objdump.exe
MIPS_OBJCOPY = $(MIPS_BIN)/mips-elf-objcopy.exe
UTIL_PREFIX  = /home/Thesis/Programs/util
UTIL_CONVBIN = $(UTIL_PREFIX)/bintohex.exe
UTIL_CONVXUM = $(UTIL_PREFIX)/bintoxum.exe

AS_FLAGS      = -march=mips32 -EB -G0
LD_FLAGS      = -EB -static -Map app.map -T ../src/os/xum.ls
LD_LIBS       = -lm -lc -lgcc
LD_SEARCH     = -L$(MIPS_PREFIX)/mips-elf/lib \
                -L$(MIPS_PREFIX)/lib/gcc/mips-elf/5.2.0
LD_DRIVER     = $(MIPS_LD) $(LD_FLAGS) $(LD_SEARCH) $(LD_LIBS)
CC_FLAGS_ARCH = -march=mips32 -EB -msoft-float -mno-mips16 -mno-branch-likely \
                -mgpopt
CC_FLAGS_LANG = -Wall -O2
CC_FLAGS_INC  = -I../src/
CC_FLAGS_AS   = -Wa,-EB,-mips32,-msoft-float
CC_FLAGS_LD   = -nostdlib -nostartfiles -static -T ../src/os/xum.ls
CC_FLAGS_LIB  = -lm -lc -lgcc
CC_DRIVER     = $(MIPS_CC) $(CC_FLAGS_ARCH) $(CC_FLAGS_LANG) \
                $(CC_FLAGS_AS) $(CC_FLAGS_INC)


all : app

app : app.o uart.o boot.o vectors.o exceptions.o exception_handler.o
       $(LD_DRIVER) $^ -o app.exe
       @$(MIPS_OBJDUMP) -EB --disassemble app.exe > app.lst
       @$(MIPS_OBJCOPY) -O binary -j .text  app.exe app-code.bin
       @$(MIPS_OBJCOPY) -O binary -j .data  app.exe app-data1.bin
       @$(MIPS_OBJCOPY) -O binary -j .sdata app.exe app-data2.bin
       @$(MIPS_OBJCOPY) -O binary -j .sbss  app.exe app-data3.bin
       @$(MIPS_OBJCOPY) -O binary -j .bss   app.exe app-data4.bin
       @cat app-data1.bin app-data2.bin app-data3.bin app-data4.bin >> app-data.bin
       @$(UTIL_CONVXUM) -d 4096 app-code.bin app-data.bin app.xum
       #@$(UTIL_CONVBIN) -c -b app-code.bin app-code.coe
       $(UTIL_CONVBIN) -c -b app.xum app.coe



app.o : $(SRC)/app/app.c
       $(CC_DRIVER) -c $(SRC)/app/app.c -o app.o

uart.o : $(SRC)/drivers/uart.c $(SRC)/drivers/uart.h
       $(CC_DRIVER) -c $(SRC)/drivers/uart.c -o uart.o

i2c.o : $(SRC)/drivers/i2c.c $(SRC)/drivers/i2c.h
       $(CC_DRIVER) -c $(SRC)/drivers/i2c.c -o i2c.o

monitor.o : $(SRC)/drivers/monitor.c $(SRC)/drivers/monitor.h i2c.o
       $(CC_DRIVER) -c $(SRC)/drivers/monitor.c -o monitor.o

exception_handler.o : $(SRC)/os/exception_handler.c $(SRC)/os/exception_handler.h
monitor.o
       $(CC_DRIVER) -c $(SRC)/os/exception_handler.c -o exception_handler.o

boot.o : $(SRC)/os/boot.asm
       $(MIPS_AS) $(AS_FLAGS) -o boot.o $(SRC)/os/boot.asm

vectors.o : $(SRC)/os/vectors.asm
       $(MIPS_AS) $(AS_FLAGS) -o vectors.o $(SRC)/os/vectors.asm

exceptions.o : $(SRC)/os/exceptions.asm
       $(MIPS_AS) $(AS_FLAGS) -o exceptions.o $(SRC)/os/exceptions.asm

clean :
       rm -f *.o *.exe *.map *.coe *.bin *.map *.xum *.lst
```

APPENDIX B. ARM SOURCE CODE

The following source code provides the most basic interface to allow the CFTP flight board to communicate with the C&DH ARM processor. The source code for the Read and Write programs is the same as was developed for the first itereation of CFTP. The JTAG code had to be modified as the protocol and registers sizes between the FPGAs used on the first flight board and the FPGA used in this iteration are different.

(1) cftp_arm.h

```
#define PORTNUM_DATA    10701
#define PORTNUM_STAT    10702
#define PORTNUM_POLL    10703
#define IP_CFTP         “192.168.4.2”
#define IP_MIDSTAR      “192.168.4.1”

#define NUMSEMS         3
#define NONSTATUS_RDY   0
#define STATUS_RDY      1
#define FTP_BLOCK       2

#define POWERDOWN_ACTIVE    0xFF
#define STANDBY_ACTIVE      0xFF
#define POWERDOWN_INACTIVE  0x00
#define STANDBY_INACTIVE    0x00

#define DEST_GROUND_AND_MIDSTAR 0xFF
#define DEST_GROUND             0xF0
#define DEST_MIDSTAR            0x0F

#define CFTP_PATH            “/cftp/”
//#define MIDSTAR_PATH         “/home/mid-test/data/CFTP/”
//#define MID_TO_GROUND_PATH   “to_ground/”
//#define MID_TO_CFTP_PATH     “to_cftp/”
//#define MID_STARTUPFILE_PATH “startupfile/”
//#define MID_POWERDOWN_PATH   “powerdown/”
//#define MID_STARTUPFILE_PATH “startupfile/”
//#define MID_STORE_PATH       “store/”
//
#define STARTUPFILE_FN        “cftp_startfiles.tgz”

#define CFTP_STARTUPFILES   “/cftp/cftp10_startupfiles &” //script to run when
cftp_startfiles.tgz received from Midstar
#define CFTP_RECEIVEDFILE   “/cftp/cftp20_rcvdfile” //script to handle files received
from ground
#define CFTP_PRESTANDBY     “/cftp/cftp30_prestandby &” //script to run when standby
triggered
#define CFTP_STANDBY        “/cftp/cftp40_standby &” //script to run when standby
triggered
#define CFTP_STANDINGBY     “/cftp/cftp50_standingby &” //script to run when standby
triggered
#define CFTP_RESUME         “/cftp/cftp60_resume &” //script to run coming out of standby
#define CFTP_SHUTDOWNRQST   “/cftp/cftp70_shutdownrqst &” //script to run as soon as
shutdown triggered (60s before actual shutdown)
#define CFTP_PRESHUTDOWN    “/cftp/cftp80_preshutdown &” //script to run just before
system shutdown takes place (60s after trigger)

#define CFTP_FTPNONSTATUS   “/cftp/cftp90_ftpnonstat” //script to prepare and ftp
nonstatus files (called from ftp_nonstatus)
```

```
#define CFTP_FTPSTATUS      “/cftp/cftp95_ftpstatus” //script to prepare and ftp status
files (called from ftp_status)

#define CFTP_POWEROFF_CMD   “cftp_28v_off”
#define CFTP_POWERON_CMD    “cftp_28v_on”

#define ONE_SEC_IN_MICROSECS    (1000000)

#define MAX_SYSCMD_SZ           (256)
//#define MAX_SYSCMD_SZ       2 * (strlen(CFTP_PATH) + strlen(MIDSTAR_PATH) + \
//                            strlen(MID_TO_GROUND_PATH) + strlen(MID_TO_CFTP_PATH) + \
//                            strlen(MID_STARTUPFILE_PATH) + strlen(MID_STORE_PATH) + \
//                            strlen(STARTUPFILE_FN) + strlen(CFTP_STARTUPFILES) + \
//                            strlen(CFTP_RECEIVEDFILE) + strlen(CFTP_PRESTANDBY) + \
//                            strlen(CFTP_STANDBY) + strlen(CFTP_STANDINGBY) + \
//                            strlen(CFTP_RESUME) + strlen(CFTP_SHUTDOWNRQST) + \
//                            strlen(CFTP_PRESHUTDOWN) + strlen(CFTP_FTPNONSTATUS) + \
//                            strlen(CFTP_FTPSTATUS) + 50 )
```

(2) rd_arm_poll.c

```
#include <sys/io.h>
#include <sys/types.h> //open
#include <sys/stat.h> //open
#include <fcntl.h> //open
#include <sys/mman.h> //mmap
#include <asm/io.h>
#include <unistd.h>
#include <errno.h> //errno
#include <stdio.h>
#include <sys/socket.h>
#include “cftp_arm.h”
#include <netinet/in.h>

//Base physical address allocated to CFTP on the modified PC/104 bus on the ARM
#define BASE_ADDR      0x4C000000
// Actual ports used by CFTP (defined in PC/104 Interface VHDL module)
#define DATA_OFFSET    0x340
#define STATUS_OFFSET  0x342

#define DEFAULT_BUFFER_SIZE         (512)
#define DATA_WAIT_IN_MICROSECS      (ONE_SEC_IN_MICROSECS)

#define EXIT_OK     (1)
#define EXIT_ERR    (0)

// throttle the data rate:
// select an arbitrary # of bytes;
// if we collect that many bytes in less than an
// arbitrary amount of time, the data rate is excessive
#define TIME_INTERVAL_BYTES (10000)
#define TIME_INTERVAL_SECS  (60)

void write_socket(int sock_fd,
                  unsigned char *ip,
                  int port,
                  unsigned char *data,
                  int data_len);
void output_data(int bUpload, unsigned char *data_buf, int byte_count);
void output_status(int bUpload, unsigned char *data_buf, int byte_count);
//void saveStatus(unsigned char status, unsigned char data, int bUpload);

int _sock_fd;

void printUsage()
{
    fprintf(stderr, “\n”);
```

```
    fprintf(stderr, “Usage: rd_arm_poll [-u] [-c bytes] [-t time]\n”);
    fprintf(stderr, “       -u  upload data to host computer (write to socket)\n”);
    fprintf(stderr, “       -c  max bytes per time period (default 10,000)\n”);
    fprintf(stderr, “       -t  time period in seconds (default 60)\n\n”);
    exit(EXIT_ERR);
}

main(int argc, char *argv[])
{
    int i;
    int fd;
    void *ptr, *status_ptr, *data_ptr;
    unsigned char status;
    uint16_t idata;
    unsigned char *data_buf;
    int byte_count;
    size_t buffer_size;
    time_t timeIntervalStart;
    int nDataRateBytes;
    int bUpload = 0;
    int time_interval_secs = 0;
    int time_interval_bytes = 0;
    int k = 0;

    for (i = 1; i < argc; i++)
    {
        if (strcmp(argv[i], “-u”) == 0)
        {
            if (bUpload)
                printUsage();
            bUpload = 1;
        }
        else if (strcmp(argv[i], “-c”) == 0)
        {
            if (time_interval_bytes > 0 || ++i >= argc)
                printUsage();
            time_interval_bytes = atoi(argv[i]);
            if (time_interval_bytes <= 0)
                printUsage();
        }
        else if (strcmp(argv[i], “-t”) == 0)
        {
            if (time_interval_secs > 0 || ++i >= argc)
                printUsage();
            time_interval_secs = atoi(argv[i]);
            if (time_interval_secs <= 0)
                printUsage();
        }
        else
        {
            printUsage();
        }
    }
    // use default values for byte count & time interval if none given
    if (time_interval_secs == 0)
        time_interval_secs = TIME_INTERVAL_SECS;
    if (time_interval_bytes == 0)
        time_interval_bytes = TIME_INTERVAL_BYTES;

    if (bUpload)
    {
        if ((_sock_fd = socket (AF_INET, SOCK_DGRAM, 0)) < 0 )
        {
            perror(“socket”);
            return EXIT_ERR;
        }
    }

    buffer_size = DEFAULT_BUFFER_SIZE;
```

```
data_buf = malloc(buffer_size);
if (data_buf == NULL)
{
    fprintf(stderr, “Unable to allocate %d bytes for data buffer\n,” buffer_size);
    return EXIT_ERR;
}

//Opening the /dev/mem Linux kernel driver device gives
//this user program access to the system’s physical memory
//O_RDONLY gives us only read access
//  (we do not need to write to the PC/104 from the read program)
//O_SYNC: something with data integrity for writes... not sure it does
//  anything for reads...
if ( (fd = open(“/dev/mem,”O_RDONLY|O_SYNC)) < 0) {
    fprintf(stderr,”open(/dev/mem) failed”);
    return EXIT_ERR;
}

//Map
//0x1000 ports
//BASE_ADDR:  starting at physical address BASE_ADDR (0x4C000000)
//ptr:        into a virtual address in this programs user space
//              stored in ptr
//NULL:       with a requested virtual address of NULL
//              (meaning we don’t care where it maps into)
//fd:         from the file pointed to in file descriptor fd
//PROT_READ:  only read from this address
//MAP_SHARED: actually modify the mapped object if we write to it
//              (MAP_PRIVATE makes a copy of the page, and all writes
//              reference the copy. No good!)
if ((ptr = mmap(NULL,0x1000,PROT_READ,MAP_SHARED,fd,BASE_ADDR)) == (void *)-1)
{
    fprintf(stderr,
            “mmap failed, ptr: %08x, errno: %d, fd: %d\n,”
            ptr, errno, fd);
    perror(“mmap failed”);
    return EXIT_ERR;
}

//We must allocate a single block of ports including any that we need to
//access (STATUS_OFFSET=0x342, and DATA_OFFSET=0x340) using one call to
//mmap, and then access an offset from the pointer returned from mmap.
status_ptr = ptr + STATUS_OFFSET;
data_ptr = ptr + DATA_OFFSET;
byte_count = 0;

nDataRateBytes = 0;

// Continuously run -- we’re polling...
while (1)
{
    // We must dereference a volatile unsigned char *. if it is not
    // specified as volatile, the compiler will optimize it out thinking
    // the data never changes at that location, so it will not recognize
    // any changes to the data on that port.
    status = *((volatile unsigned char *)status_ptr);

    // If bit 0 of the status byte is ‘1’, then we have data available.
    //
    // When the board first starts up, all the output lines are briefly
    // set to ‘1’, so we are ignoring this case (when the status byte is
    // 0xFF, we are assuming we don’t actually have data. Never set the
    // status byte to 0xFF in the VHDL!!  All unused bits should be ‘0’,
    // and we should always keep at least one unused bit in the status
    // byte (to differentiate from the initial case))
    if ((1 & status) && (status != 0xFF))
    {
        idata = *((volatile uint16_t *)data_ptr);
```

```
/*k++;
if (k >= 50)
{
    fprintf(stderr, “Data: %04x  status=%02x\n,” idata, status);
    fflush(stderr);
    k = 0;
}*/
            // set interval start time when 1st byte received after
            // byte count is reset
            if (nDataRateBytes == 0)
                timeIntervalStart = time(NULL);

            // check whether data rate has gone out of bounds
            if (++nDataRateBytes >= time_interval_bytes)
            {
                // how long did it take to collect this much data?
                int numSecs = (int)(time(NULL) - timeIntervalStart);
                fprintf(stderr,
                        “%s: Interval byte count reached: bytes=%d secs=%d\n,”
                        argv[0], nDataRateBytes, numSecs);
                // if we’ve collected the byte count in less than
                // allotted time, the data rate has exceeded its limit
                if (numSecs < time_interval_secs)
                {
                    fprintf(stderr,
                            “%s: *** Data rate exceeded ***\n,” argv[0]);
                    // insert msg into data stream
                    strcpy(data_buf,
                            “*** DATA RATE EXCEEDED -- Reloading X1 ***”);
                    output_data(bUpload, data_buf, strlen(data_buf));
                    // reload write_flash.bin into X1 to stop output
                    system(“/cftp/cftp_load_write_flash >/dev/null 2>&1”);
                    //usleep(60 * ONE_SEC_IN_MICROSECS);
                    nDataRateBytes = 0;
                    continue;
                }
                nDataRateBytes = 0;
            }
            // if we’ve filled the data buffer, send the packet
            // and zero the byte count to start filling it again
            if (byte_count >= buffer_size)
            {
                fprintf(stderr, “Outputting data... status=%d\n,” status);
                fflush(stderr);
                //saveStatus(status, data[0], bUpload);
                // if we’re uploading to the C&DH (Midstar computer), write the
                // data to the socket, otherwise print the data to stdout
                // (you can redirect the output of rd_arm_poll to a file,
                // and then look at the contents of the file)
                output_data(bUpload, data_buf, byte_count);
                byte_count = 0;
            }
            memcpy(&data_buf[byte_count], &idata, 2);
            byte_count += 2;
        }
        else // no data ready
        {
            //saveStatus(status, data[0], bUpload);
            //fprintf(stderr, “%s: No data ready -- taking a nap\n,” argv[0]);
            // if there’s no data, sleep awhile,
            // but first flush any data already collected
            if (byte_count > 0)
            {
                output_data(bUpload, data_buf, byte_count);
                byte_count = 0;
            }
            // be sure to flush stdout
            if (!bUpload)
            {
```

```
                fflush(NULL);
            }
            usleep(DATA_WAIT_IN_MICROSECS);
        }
    }
    // close fd and unmap ptr!
    close(fd);
    munmap(ptr,0x1000);
    return EXIT_OK;
}
/*
void saveStatus(unsigned char status, unsigned char data, int bUpload)
{
    // status byte stuff:
    static time_t timeStatusStart = 0;
    static unsigned char statusBytes[30];
    static unsigned char dataBytes[30];
    static int numStatusBytes = 0;
    int i;
    int numSecs = 0;
    if (timeStatusStart == 0)
        timeStatusStart = time(NULL);
    else
        numSecs = (int)(time(NULL) - timeStatusStart);
    // collect another status byte
    statusBytes[numStatusBytes] = status;
    dataBytes[numStatusBytes] = data;
    numStatusBytes++;
    if (numSecs >= 60 || numStatusBytes >= 30)
    {
        char statMsg[256];
        int nchars = sprintf(statMsg, "\nstat/data:");
        char *cp = &statMsg[0] + nchars;
        for (i = 0; i < numStatusBytes; i++)
        {
            nchars = sprintf(cp, " %02x/%02x," statusBytes[i], dataBytes[i]);
            cp += nchars;
        }
        sprintf(cp, "\n");
        output_status(bUpload, statMsg, strlen(statMsg));
        timeStatusStart = time(NULL);
        numStatusBytes = 0;
    }
}
*/
void write_socket(int sock_fd,
                  unsigned char *ip,
                  int port,
                  unsigned char *data,
                  int data_len)
{
    struct sockaddr_in  comm_addr;
    socklen_t comm_len;

    comm_addr.sin_family = AF_INET;
    comm_addr.sin_addr.s_addr = inet_addr(ip);
    comm_addr.sin_port = htons(port);

    comm_len = sizeof(comm_addr);
    sendto( sock_fd,                        // file descriptor for socket
            (void *) data,                  // data to write to socket
            data_len,                       // length of message to be written
            0,                              // no flags
            (struct sockaddr *) &comm_addr, // structure containing IP, port
                                            // number, socket type
            comm_len );                     // length of comm_addr struct
}

void output_data(int bUpload, unsigned char *data_buf, int byte_count)
```

```
{
    if (byte_count <= 0)
        return;

    if (bUpload)
    {
        write_socket(_sock_fd, IP_MIDSTAR, PORTNUM_DATA,
                     data_buf, byte_count);
    }
    else
    {
        int i;
        unsigned char *cp = data_buf;
        for (i = 0; i < byte_count; i++)
            putchar(*(cp++));
    }
}

void output_status(int bUpload, unsigned char *data_buf, int byte_count)
{
    if (byte_count <= 0)
        return;

    if (bUpload)
    {
        write_socket(_sock_fd, IP_MIDSTAR, PORTNUM_STAT,
                     data_buf, byte_count);
    }
    else
    {
        int i;
        unsigned char *cp = data_buf;
        for (i = 0; i < byte_count; i++)
            putchar(*(cp++));
    }
}
```

(3) write_arm_poll.c

```
/*****************************************************************************
*** Write to PC/104, polling the status register before writes              ***
***   Usage: wr SRC_FILE [options]                                          ***
***       SRC_FILE:   SRC_FILE = stdin for reading data from stdin,         ***
***                   SRC_FILE = FILENAME to read from file FILENAME        ***
***       with options:                                                     ***
***       -p          print values written to PC/104 to stdout (default no) ***
***       -d DELAY    delay between writes to PC/104 (number of ARM clocks, ***
***                   default 0)                                            ***
***       -i INTERVAL write current byte to stdout every INTERVAL bytes     ***
***                   (0 for off, default 100, 0 when -p)                   ***
*****************************************************************************/
#include<sys/io.h>
#include<stdio.h>
#include<stdint.h>
#include<unistd.h>
#include<sys/types.h> //open
#include<sys/stat.h> //open
#include<fcntl.h> //open
#include<sys/mman.h> //mmap
#include<errno.h> //errno

//Base physical address allocated to CFTP on the modified PC/104 bus on the ARM
#define BASE_ADDR      0x4C000000
// Actual ports used by CFTP (defined in PC/104 Interface VHDL module)
#define DATA_OFFSET    0x340
#define STATUS_OFFSET  0x342
```

```
int print_usage();

main(int argc, char **argv)
{

    FILE *in;
    unsigned char status;
    union
    {
        unsigned char cdata[2];
        uint16_t idata;
    } data;
    int fd;
    void *io_ptr;
    unsigned char *io_status_ptr;
    unsigned char *io_data_ptr;
    int i,display=0,interval=100;
    unsigned long delay = 0;
    long cnt,filesize;
    struct stat buf,*bufptr;

    bufptr = &buf;

    // parse command line options (starting after SRC_FILE)
    // see usage above...
    for (i=2; i<argc; i++) {
        if (strcmp(argv[i], “-p”) == 0) {
            display = 1;
            interval = 0;
        }
        else if (strcmp(argv[i], “-d”) == 0) {
            if (i+1 <= argc-1) {
                i++;
                delay = (unsigned long) atoi(argv[i]);
            }
            else { print_usage(); }
        }
        else if (strcmp(argv[i], “-i”) == 0) {
            if (i+1 <= argc-1) {
                i++;
                interval = atoi(argv[i]);
            }
            else { return print_usage(); }
        }
        else { return print_usage(); }
    }
    // if they didn’t even include SRC_FILE, no go
    if (argc < 2) { return print_usage(); }

    // if they are reading from stdin, just set file descriptor in = stdin
    // filesize limit is arbitrarily set to 4000000
    if (strcmp(argv[1],”stdin”) == 0) {
        in = stdin;
        filesize = 4000000;
    }
    //otherwise, open the file, and get size
    else {

        //open file we will write to PC/104
        if ((in = fopen(argv[1],”rb”)) == NULL)
        {
            printf (“Unable to open file for read\n\n”);
            return 0;
        }
        //get size of file we will write to PC/104
        if (stat(argv[1],bufptr) != 0)
        {
            printf(“Error getting filesize\n\n”);
            return 0;
```

```
        }
        filesize = (long) bufptr->st_size;
    }
    // open file that handles memory mapped IO
    //Opening the /dev/mem Linux kernel driver device gives
    //      this user program access to the system's physical memory
    //O_RDWR gives us read and write access (we need to read the status address,
    //          and write to the data address)
    //O_SYNC: something with data integrity for writes
    if ( (fd = open("/dev/mem,"O_RDWR|O_SYNC)) < 0) {
        printf("open(/dev/mem) failed");
        return 0;
    }
    //map the physical address of CFTP's memory to memory space of this process
    //0x1000:     number of ports in range we have access to
    //BASE_ADDR:  starting at physical address BASE_ADDR (0x4C000000)
    //ptr:        into a virtual address in this programs user space
    //              stored in ptr
    //NULL:       with a requested virtual address of NULL
    //              (meaning we don't care where it maps into)
    //fd:         from the file pointed to in file descriptor fd
    //PROT_READ:  allow reads from this address
    //PROT_WRITE: allow writes to this address
    //MAP_SHARED: actually modify the mapped object if we write to it
    //              (MAP_PRIVATE makes a copy of the page, and all writes
    //              reference the copy. No good!)
    if ((io_ptr = mmap(NULL,0x1000,PROT_READ|PROT_WRITE,MAP_SHARED,fd,BASE_ADDR)) ==
(void *)-1) {
        printf("mmap failed, ptr: %08x, errno: %d, fd: %d\n,"io_ptr,errno, fd);
        perror("mmap failed");
        return 0;
    }
    // status byte is on port 0x342 (0x4C000342), data 0x340 (0x4C000340)
    //We must allocate a single block of ports including any that we need to
    //access (STATUS_OFFSET=0x342, and DATA_OFFSET=0x340) using one call to
    //mmap, and then access an offset from the pointer returned from mmap.
    io_status_ptr = (unsigned char *)(io_ptr + STATUS_OFFSET);
    io_data_ptr   = (unsigned char *)(io_ptr + DATA_OFFSET);
    fprintf(stderr,"ptr: %08x, status_ptr: %08x, data_ptr:
%08x\n,"io_ptr,io_status_ptr,io_data_ptr);

    cnt = 0;

    //if we're not echoing the bytes we're writing to the PC/104 to stdout
    if (!display)
    {
        printf ("\nWriting file %s to the PC/104. \n"
          "%d bytes, %d delay, status every %d bytes\n\n,"
          argv[1],filesize,delay,interval);
        fflush(stdin);
    }
    while (cnt < filesize)
    {
        // read 2 bytes from file/stdin;
        // if only 1 byte left, arbitrarily null-terminate
        data.cdata[0] = getc(in);
        cnt++;
        if (cnt < filesize)
        {
            data.cdata[1] = getc(in);
            cnt++;
        }
        else
        {
            data.cdata[1] = 0x0;
        }
        // print hex value to stdout if echoing file!
        if (display==1)
        {
```

```
            //printf("%02x%02x," data.cdata[0], data.cdata[1]);
            //printf("%04x," data.idata);
            printf("%04x," (((uint16_t)data.cdata[0]) << 8) | ((uint16_t)data.cdata[1]));
            fflush(NULL);
        }
        // if we're printing status, not echoing file, and we're at the correct
        // interval, print the byte number we're on to stdout
        if ( (interval) && (!display) && ((cnt % interval) == 0) )
        {
            //MLS
            printf("%d..,"cnt);
            //printf("byte: %02x, cnt: %d..\n,"byte,cnt);
            fflush(NULL);
        }
        // wait until status bit 1 is low, meaning
        // we are not currently writing from the processor to the FPGA
        // once the FPGA acknowledges receipt of a byte, status_bit(1)
        // goes low again
        status = *io_status_ptr;
        while ((2 & status) == 2)
        {
            // hand over the CPU while waiting for status change
            usleep(100);
            //get data from status address
            status = *io_status_ptr;
        }
        //write data to the data address
//        memcpy(io_data_ptr, data.cdata, 2);
//        *(io_data_ptr + 1) = data.cdata[0];
        *((uint16_t *)io_data_ptr) = (((uint16_t)data.cdata[0]) << 8) |
((uint16_t)data.cdata[1]);

        //busy loop for "delay" clocks
        for (i=0;i<delay;i++);
    }


    fclose(in);
    // usually when the file is echoed to stdout, it is so there can be some
    // sort of formatting/visualization of the inputs to the FPGA. Since all
    // numbers would be written to stdout (the hex representation of the
    // values), a Q can signify the end of the stream. The formatting code
    // can use this Q to terminate. Lame. I know.
    if (display)
    {
        printf("Q");
        fflush(NULL);
    }

    //unmap io_ptr!
    munmap(io_ptr,0x1000);

}

int print_usage() {
    printf("Usage: wr SRC_FILE [options]\n"
        "SRC_FILE:   SRC_FILE = stdin for reading data from stdin and writing\n"
        "                       to PC/104\n"
        "            SRC_FILE = FILENAME to read from file FILENAME and write\n"
        "                       to PC/104\n"
        "with options: \n"
        "-p          print values written to PC/104 to stdout (default no)\n"
        "-d DELAY    delay between writes to PC/104 (number of ARM clocks, \n"
        "            default 0)\n"
        "-i INTERVAL write current byte to stdout every INTERVAL bytes\n"
        "            (0 for off, default 100, 0 when -p)\n");
    return 0;
}
```

(4) jtag.h

```
#include<sys/io.h>
#include<sys/types.h> //open
#include<sys/stat.h> //open
#include<fcntl.h> //open
#include<sys/mman.h> //mmap
#include<asm/io.h>
#include<asm/types.h>
#include<unistd.h>
#include<errno.h> //errno
#include<stdio.h>


//set ACTIVE to 1 if you want to actually program the part
//set ACTIVE to 0 if you don't want to access the PC104
#define ACTIVE  1
//#define DEBUG
#ifdef DEBUG
        #define DPRINTF(x)    {x}
#else
        #define DPRINTF(x)    {}
#endif

#define NUM_BITS            (450996 * 8)
#define NUM_BITS_IN_COMMAND (32)

#define FDRI_WRITE          0x30004000

// If we see the Write to COR command,
// we want to verify that the startup clock
// bit in the COR value is set to JTAG.
// bit 21 (LSB is bit 0) of the COR must be set
// to '1' in order for JTAG startup clock to be used
// so the 11th bit we shift into JTAG must be a '1'
#define WRITE_TO_COR        0x30012001
#define STARTUP_CLOCK_BIT   10
#define CCLK_STARTUP_CLOCK  0
#define JTAG_STARTUP_CLOCK  1

#define TMS_HIGH        1
#define TMS_LOW         0
#define TDI_HIGH        1
#define TDI_LOW         0
#define TDI_DK          0


// JTAG Support from the C&DH ARM
// JTAG definitions
// JTAG for CFTP uses GPIOPE
// C&DH Bootloader ensures that GPIOPE set set up for:
//             TCK_ptr = jtag_ptr:D7  (output)
//             TMS_ptr = jtag_ptr:D6  (output)
//             TDI_ptr = jtag_ptr:D5  (output)
//             TDO_ptr = jtag_ptr:D1  (input)
// And ALL outputs are set to zero after processor initialization in the bootloader
#ifndef u8
        typedef unsigned char u8;
#endif
#define APB_BASE                            0xFFFC0000
#define GPIOPEDR_APB_OFFSET           (0xFFFDD000 - APB_BASE)
#define JTAG_PTR       (*(apb_base_ptr + GPIOPEDR_APB_OFFSET))
#define TCK_HIGH       (JTAG_PTR |= 0x80)
#define TCK_LOW               (JTAG_PTR &= 0x7F)
#define TMS_SET(x)     {\
        if (x)\
                JTAG_PTR |= 0x40;\
        else\
```

```
                JTAG_PTR &= 0xBF;\
        }
#define TDI_SET(x)     {\
        if (x)\
                JTAG_PTR |= 0x20;\
        else\
                JTAG_PTR &= 0xDF;\
        }
#define TDO_GET        ((JTAG_PTR&0x02)>>1)
u8 set_JTAG(u8 TMS_val, u8 TDI_val, int num_TCKs);
```

(5) set_JTAG.h

```
/*****************************************************************************
 *
 *      u8 set_JTAG(u8 TMS_val, u8 TDI_val, int num_TCKs)
 *
 *      First time called the mmap is setup.
 *      Sending a value of less than zero for num_TCKs closes the mmap interface.
 *
 *****************************************************************************/

u8 set_JTAG(u8 TMS_val, u8 TDI_val, int num_TCKs)
{
        static volatile u8 *apb_base_ptr = NULL;
        static int fd = -1;
        u8 ret_val;
        int i;


        if (ACTIVE && num_TCKs < 0)
        {
                munmap((void *)apb_base_ptr, GPIOPEDR_APB_OFFSET+4);
                close(fd);
                return 0;
        }

        if (ACTIVE && fd == -1 )
        {
                if ( (fd = open("/dev/mem,"O_RDWR|O_SYNC)) < 0)
                {
                        fprintf(stderr, "open(/dev/mem) failed");
                        exit(-1);
                }
                if ((apb_base_ptr = (u8 *)mmap(NULL, GPIOPEDR_APB_OFFSET+4,
                   PROT_READ|PROT_WRITE, MAP_SHARED, fd, APB_BASE)) == (void *)-1)
                {
                        fprintf(stderr, "mmap failed, ptr: %p, errno: %d, fd: %d\n,"
                                apb_base_ptr, errno, fd);
                        exit(-1);
                }
                DPRINTF(printf("apb_base_ptr = %p, jtag_ptr = %p\n," apb_base_ptr);)
        }

        DPRINTF(printf("TMS_SET(%c), TDI_SET(%c), TCKs(%d) ," (TMS_val)?'1':'0',
                        (TDI_val)?'1':'0', num_TCKs);)

    if (ACTIVE) {
                TCK_HIGH;
            TMS_SET(TMS_val);
            TDI_SET(TDI_val);
    }

        // C&DH ARM timing is such that there are at least 2 Tstates between
        //   instructions (20 nsecs * 2 = 40 nsecs); thus NO delays between
```

```
        //   TCK toggling is required.
        // TMS % TDI setups are min 10 nsecs, no delay required

    for (i = 0; i < num_TCKs; i++)
        {
               DPRINTF(printf("TCK=L ");)
               if (ACTIVE) TCK_LOW;
               if (ACTIVE) ret_val = TDO_GET;
        else ret_val = 0;
               DPRINTF(printf(" TDO=%u ," (unsigned int)(ret_val));)
               DPRINTF(printf("TCK=H ");)
               if (ACTIVE) TCK_HIGH;
    }
    DPRINTF(printf("\n");)
    return ret_val;

}  /* End of set_JTAG() */
```

(6) jtag-7.c

```
// Source: jtagK7-2.c
// Description: Performs a JTAG load of a BIN or BIT file to CFTP-7.
//     Based on sequence listed in UG470(v1.10 / June 24, 2015)
//     Series 7 Configuration Guide - Advanced JTAG USAGE - p.174
// Requirements: To use this file, the JTAG Clock must be selected
//     when building BIN / BIT Files. In Vivado : Open Implemented
//     Design, then Tools --> Edit Device Proerties --> Startup -->
//     Select Startup CLock = JTAGCLK. Keep everything else default.
// Author : Ron Aikins
// Modified by: Andrew S Jackson
// Date last Modified : Mar 1, 2016


// change DEBUG and ACTIVE settings in jtag.h
#include "jtag.h"

// set_JTAG.h is Jim's set_JTAG code
#include "set_JTAG.h"


unsigned int shiftIntoRegister( unsigned int *pValues_, int numBits_ )
{
    // MUST BE IN EITHER SHIFT-DR OR SHIFT-IR WHEN ENTERING FUNCTION
    int i, j, bit;
    unsigned int tdo_val = 0;
    unsigned int tdo_u8;
    unsigned int shift = 0;

    for (i = 0; numBits_ > 0; i++)
    {
        for (j = NUM_BITS_IN_COMMAND - 1; (j >= 0) && (numBits_ > 0); j--)
        {
            bit = (*pValues_ >> j) & 0x01;
            // If last bit, shift to EXIT-1 State while sending last bit
           if (--numBits_ == 0)
                tdo_u8 = (unsigned int)set_JTAG(TMS_HIGH, bit, 1);
            // If not last bit, Stay in Shift-IR or Shift-DR and send bit
           else
                tdo_u8 = (unsigned int)set_JTAG(TMS_LOW, bit, 1);
            tdo_val |= (tdo_u8 << shift++);
        }
        pValues_++;
    }
    return tdo_val;
}
```

```
// This function places you in the Test-Logic/Reset State from anywhere in the JTAG state
// machine
void stateReset()
{
    set_JTAG(TMS_HIGH,  TDI_DK,     5); // Test Logic Reset
}


//Main Function : Requires 1 input arguement specifiying the BIN/BIT file to load
main(int argc, char **argv)
{
    FILE *bitin;
    int mixedup_word = 0, word = 0, old_word = 0, bit;
    int i, j, data_start = 0;
    unsigned int configCommands[20];
    unsigned int CheckValue;
    long count=0,bits_in_file;
    long block_count = 0;
    struct stat buf,*bufptr;
    unsigned int value;

    //Open file and perform some intial checks to verify system is setup correctly to
    //JTAG
    bufptr = &buf;
    if (stat(argv[1],bufptr) != 0)
    {
        printf("Error getting filesize\n\n");
       return 0;
    }
    bits_in_file = ((long) bufptr->st_size) * 8;
    if (bits_in_file != NUM_BITS ) {
        fprintf (stderr,"\nWARNING: Not a full length bitstream! %ld bytes, not %ld.
                Proceeding anyway.\n,"bits_in_file/8,NUM_BITS/8);
        DPRINTF(printf ("\n\nWARNING: Not a full length bitstream! %ld bytes, not %ld.
               Proceeding anyway.\n\n,"bits_in_file/8,NUM_BITS/8);)
    }

    if (!ACTIVE) {
        fprintf (stderr,"\nWARNING: Not in ACTIVE mode! Not actually toggling JTAG
                pins!\n\n");
        DPRINTF(printf ("\n\nWARNING: Not in ACTIVE mode! Not actually toggling JTAG
               pins!\n");)
    }
    else {
        fprintf (stderr,"\nNOTICE: ACTIVE mode!\n\n");
        DPRINTF(printf ("\n\nNOTICE: ACTIVE mode!\n");)
    }
    if ( (bitin = fopen(argv[1],"rb")) == NULL )
    {
        printf("Unable to open bit file for read \n\n");
        return 0;
    }

    DPRINTF(printf("\nSetting up JTAG ... :\n\n");)
    fprintf(stderr,"Setting up JTAG ... :\n");

    // Move to a known state in the JTAG State Machine
    stateReset();

    // Send Device ID Command to FPGA
    printf("Sending Device ID Command to FPGA\n");
    set_JTAG(TMS_LOW,  TDI_DK,  1); // Run-Test/Idle
    set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-IR-SCAN
    set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
    configCommands[0] = 0x90000000; // Device ID
    CheckValue = shiftIntoRegister( configCommands, 6 );
    printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

    set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-DR-SCAN
```

```
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-DR
configCommands[0] = 0x00000000; // Dummy Word
CheckValue = shiftIntoRegister( configCommands, 32 );
printf("Shifted %08x into DR and recieved %08x\n," configCommands[0],CheckValue);

// Send Bypass Command to FPGA
printf("Sending Bypass Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xfc000000; // Bypass
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Send JProgram Command to FPGA
printf("Sending JProgram Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xd0000000; // JProgram
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Send ISC_NOOP Command to FPGA
printf("Sending ISC_NOOP Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0x28000000; // IR Capture
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Run-Test for 100000 Clock Cycles
printf("Running Test...\n");
set_JTAG(TMS_HIGH, TDI_DK,  1); // Update-IR
set_JTAG(TMS_LOW,  TDI_DK,  100001); // Run-Test/Idle
printf("Test Complete.\n");

// Send IR Capture Command to FPGA
printf("Sending IR_Capture Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0x28000000; // IR Capture
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Send Cfg-In Command to FPGA
printf("Sending CFG-IN Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xa0000000; // Config In
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Shift Bitstream into DR
set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-DR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-DR

while (count < bits_in_file) { // shift in all bits in bitstream

    old_word = word;

    // Reads LSByte first, so we have to rearrange the bytes.
    // 44332211 -> 11223344
    fread(&mixedup_word,1,4,bitin);
    word = ( (0xFF & mixedup_word) << 24) +
           ( (0xFF00 & mixedup_word) << 8) +
           ( (0xFF0000 & mixedup_word) >> 8) +
           ( (0xFF000000 & mixedup_word) >> 24) ;

    if (count < 1024)
    {
```

```
        fflush(stderr);
        fprintf(stderr, "\nword %08x %08x :," mixedup_word, word);
    }

    for (i=0;i<32;i++) {
        // Grab the MSB first to shift into JTAG TDI
        bit = (word >> (31-i)) & 0x01 ;

        //Display info to track progress of load.
       if (count < 256)
        {
            fprintf(stderr, "%d," bit);
        }

        //TMS high when last bit shifted in
        if (++count == bits_in_file)
            set_JTAG(TMS_HIGH,bit,1); // last bit; go to EXIT1-DR
        else
            set_JTAG(TMS_LOW,bit,1);
    }
   // More Display Information to Track progress of load.
   if (count < 64)
        continue;
    if (count%0x20000 == 0) {
        fprintf(stderr,."");
        if (count%0x100000 == 0)
            fprintf(stderr,"\n%d,"count/8);
        fflush(stderr);
    }
}

// Run-Test for 100000 Clock Cycles
printf("Running Test...\n");
set_JTAG(TMS_HIGH, TDI_DK,  1); // Update-IR
set_JTAG(TMS_LOW,  TDI_DK,  100001); // Run-Test/Idle
printf("Test Complete.\n");

// Send Cfg-In Command to FPGA
printf("Sending CFG-IN Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xa0000000; // Config In
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-DR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-DR
configCommands[0] = 0xffffffff; // Dummy Word
configCommands[1] = 0xaa995566; // Sync Word
configCommands[2] = 0x20000000; // NOP
configCommands[3] = 0x2802c001; // Read Boot History Status Register
configCommands[4] = 0x20000000; // NOP
configCommands[5] = 0x20000000; // NOP
configCommands[6] = 0x30008001; // Command Register
configCommands[7] = 0x0000000d; // Desync Word
configCommands[8] = 0x20000000; // NOP
configCommands[9] = 0x20000000; // NOP
CheckValue = shiftIntoRegister( configCommands, 320 );
printf("Shifted %08x into DR and recieved %08x\n," configCommands[0], CheckValue);

// Send Cfg-Out Command to FPGA
printf("Sending CFG-Out Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0x20000000;
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-DR-SCAN
```

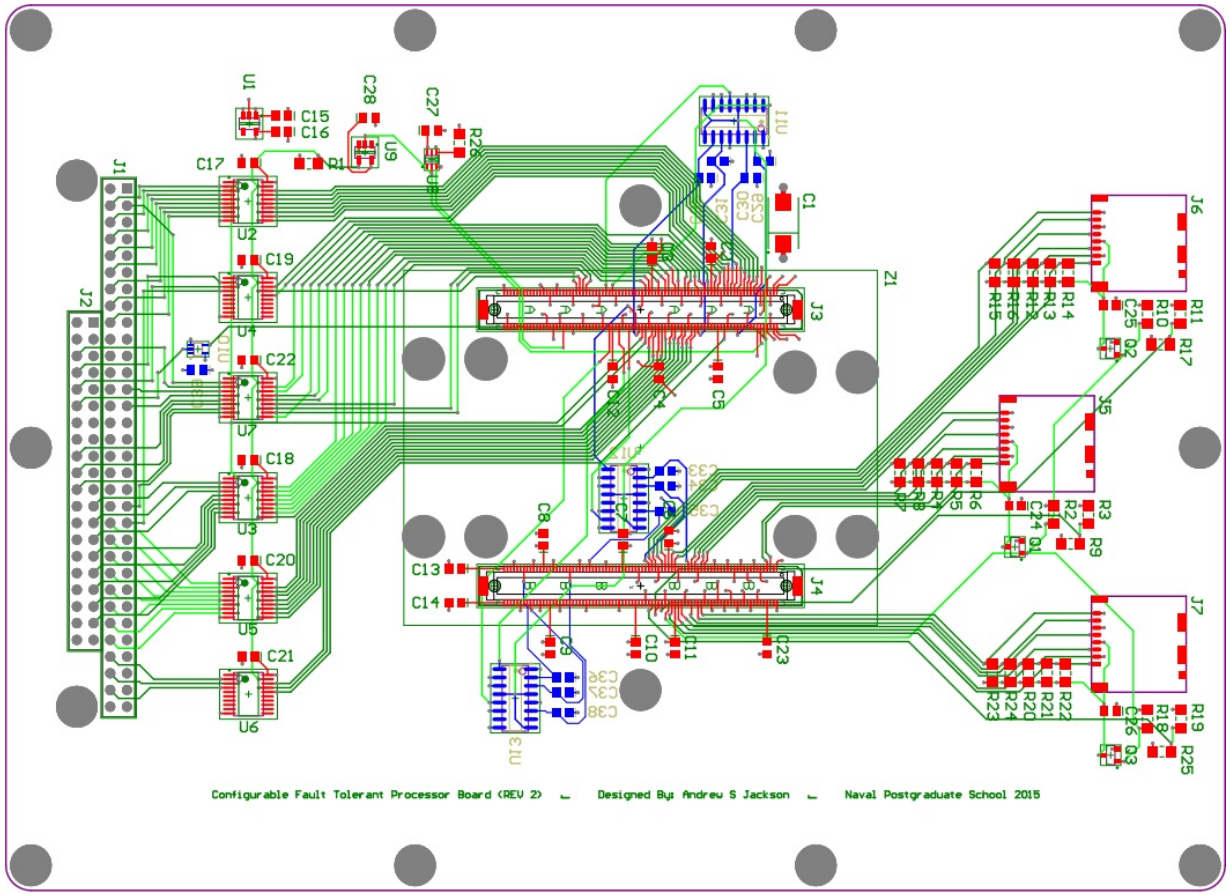
```
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-DR
configCommands[0] = 0x00000000; // Dummy Word
CheckValue = shiftIntoRegister( configCommands, 32 );
printf("Shifted %08x into DR and recieved %08x\n," configCommands[0], CheckValue);

// Send Bypass Command to FPGA
printf("Sending Bypass Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xfc000000; // Bypass
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Send Cfg-In Command to FPGA
printf("Sending CFG-IN Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xa0000000; // Config In
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-DR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-DR
configCommands[0] = 0xffffffff; // Dummy Word
configCommands[1] = 0xaa995566; // Sync Word
configCommands[2] = 0x20000000; // NOP
configCommands[3] = 0x2800e001; // Read Status Register
configCommands[4] = 0x20000000; // NOP
configCommands[5] = 0x20000000; // NOP
configCommands[6] = 0x30008001; // Command Register
configCommands[7] = 0x0000000d; // Desync Word
configCommands[8] = 0x20000000; // NOP
configCommands[9] = 0x20000000; // NOP
CheckValue = shiftIntoRegister( configCommands, 320 );
printf("Shifted %08x into DR and recieved %08x\n," configCommands[0], CheckValue);

// Send Cfg-Out Command to FPGA
printf("Sending CFG-Out Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0x20000000;
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-DR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-DR
configCommands[0] = 0x00000000; // Dummy Word
CheckValue = shiftIntoRegister( configCommands, 32 );
printf("Shifted %08x into DR and recieved %08x\n," configCommands[0], CheckValue);

// Send JStart Command to FPGA
printf("Sending JStart Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0x30000000; // JStart
CheckValue = shiftIntoRegister( configCommands, 6 );
printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

// Run-Test for 100000 Clock Cycles
printf("Running Test...\n");
set_JTAG(TMS_HIGH, TDI_DK,  1); // Update-IR
set_JTAG(TMS_LOW,  TDI_DK,  101); // Run-Test/Idle
printf("Test Complete.\n");

// Send Bypass Command to FPGA
printf("Sending Bypass Command to FPGA\n");
set_JTAG(TMS_HIGH, TDI_DK,  2); // SELECT-IR-SCAN
set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
configCommands[0] = 0xfc000000; // Bypass
```

```
    CheckValue = shiftIntoRegister( configCommands, 6 );
    printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

    // Send Bypass Command to FPGA
    printf("Sending Bypass Command to FPGA\n");
    set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
    set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
    configCommands[0] = 0xfc000000; // Bypass
    CheckValue = shiftIntoRegister( configCommands, 6 );
    printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

    // Send Bypass Command to FPGA
    printf("Sending Bypass Command to FPGA\n");
    set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
    set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
    configCommands[0] = 0xfc000000; // Bypass
    CheckValue = shiftIntoRegister( configCommands, 6 );
    printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

    // Send JStart Command to FPGA
    printf("Sending JStart Command to FPGA\n");
    set_JTAG(TMS_HIGH, TDI_DK,  3); // SELECT-IR-SCAN
    set_JTAG(TMS_LOW,  TDI_DK,  2); // SHIFT-IR
    configCommands[0] = 0x30000000; // JStart
    CheckValue = shiftIntoRegister( configCommands, 6 );
    printf("Shifted %08x into IR and recieved %08x\n," configCommands[0],CheckValue);

    // Run-Test for 100000 Clock Cycles
    printf("Running Test...\n");
    set_JTAG(TMS_HIGH, TDI_DK,  1); // Update-IR
    set_JTAG(TMS_LOW,  TDI_DK,  100001); // Run-Test/Idle
    printf("Test Complete. Program loaded.\n");

    // close up //
    set_JTAG(0, 0, -1);
}
```

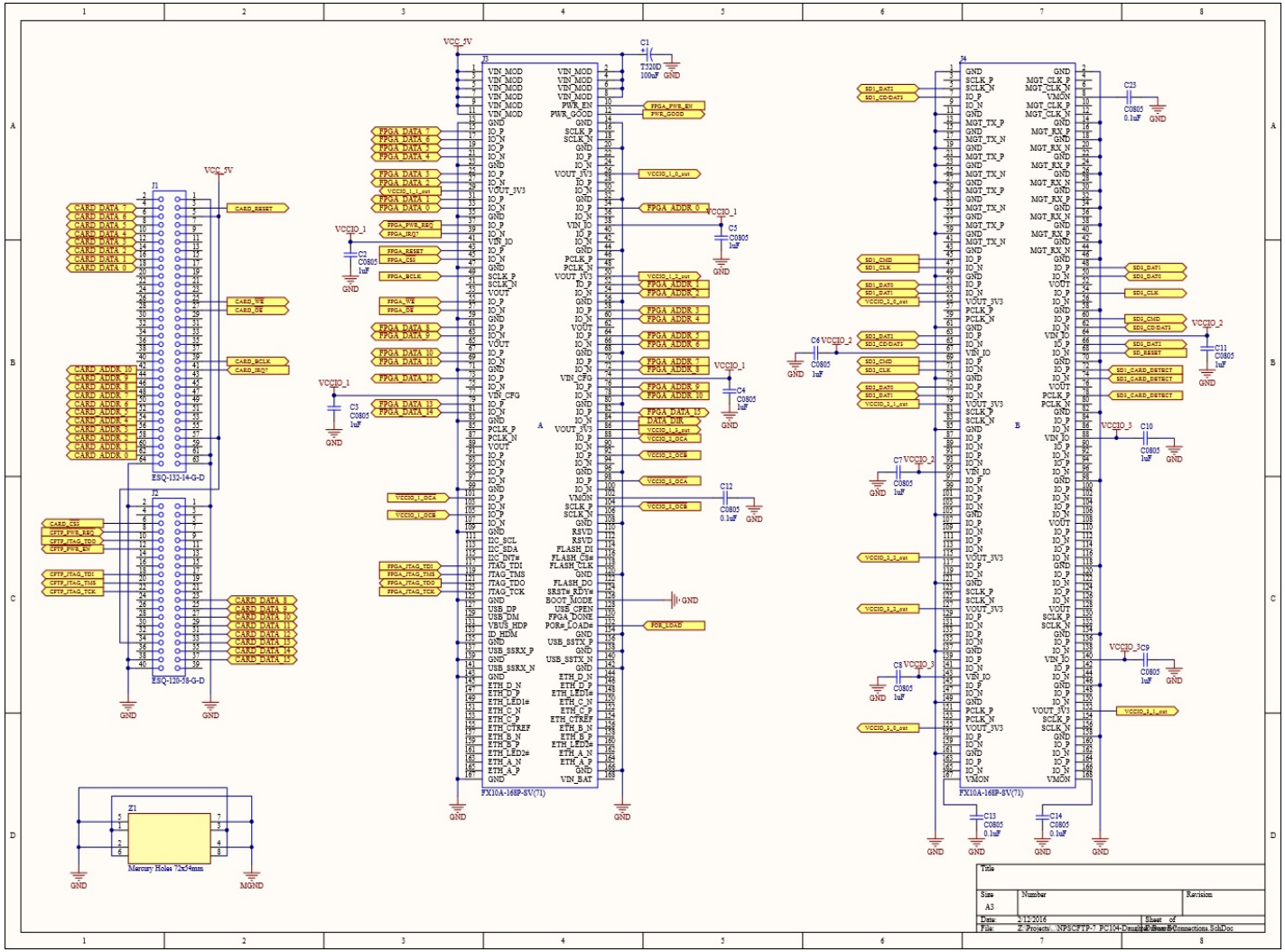
APPENDIX C. CFTP PRINTED CIRCUIT BOARD

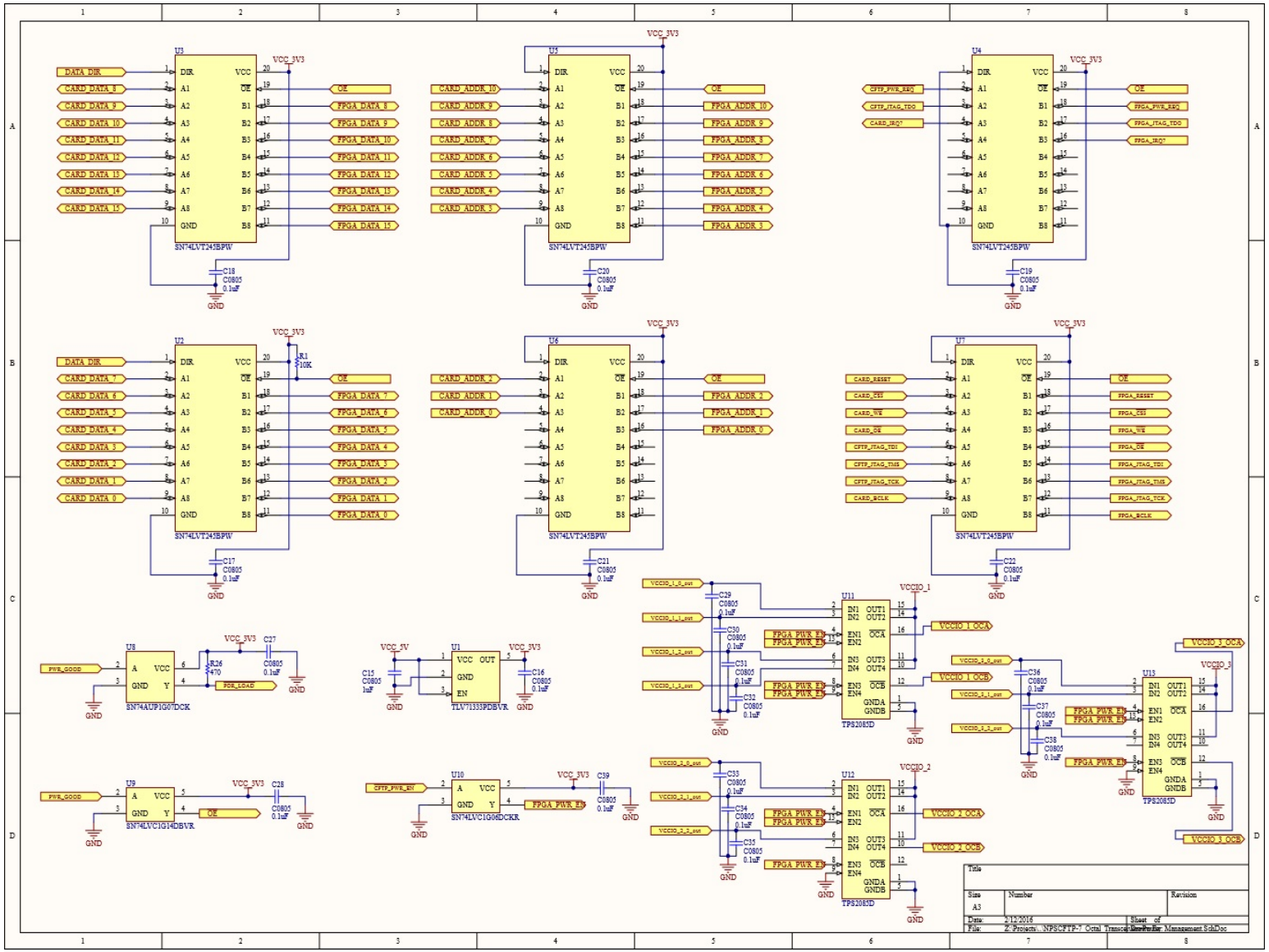
CFTP was designed using Altium Designer 15. A composite drawing, complete schematics, and all plots required for fabrication is contained in this appendix. Schematics or drawings for the Mercury KX-1 module are not included.

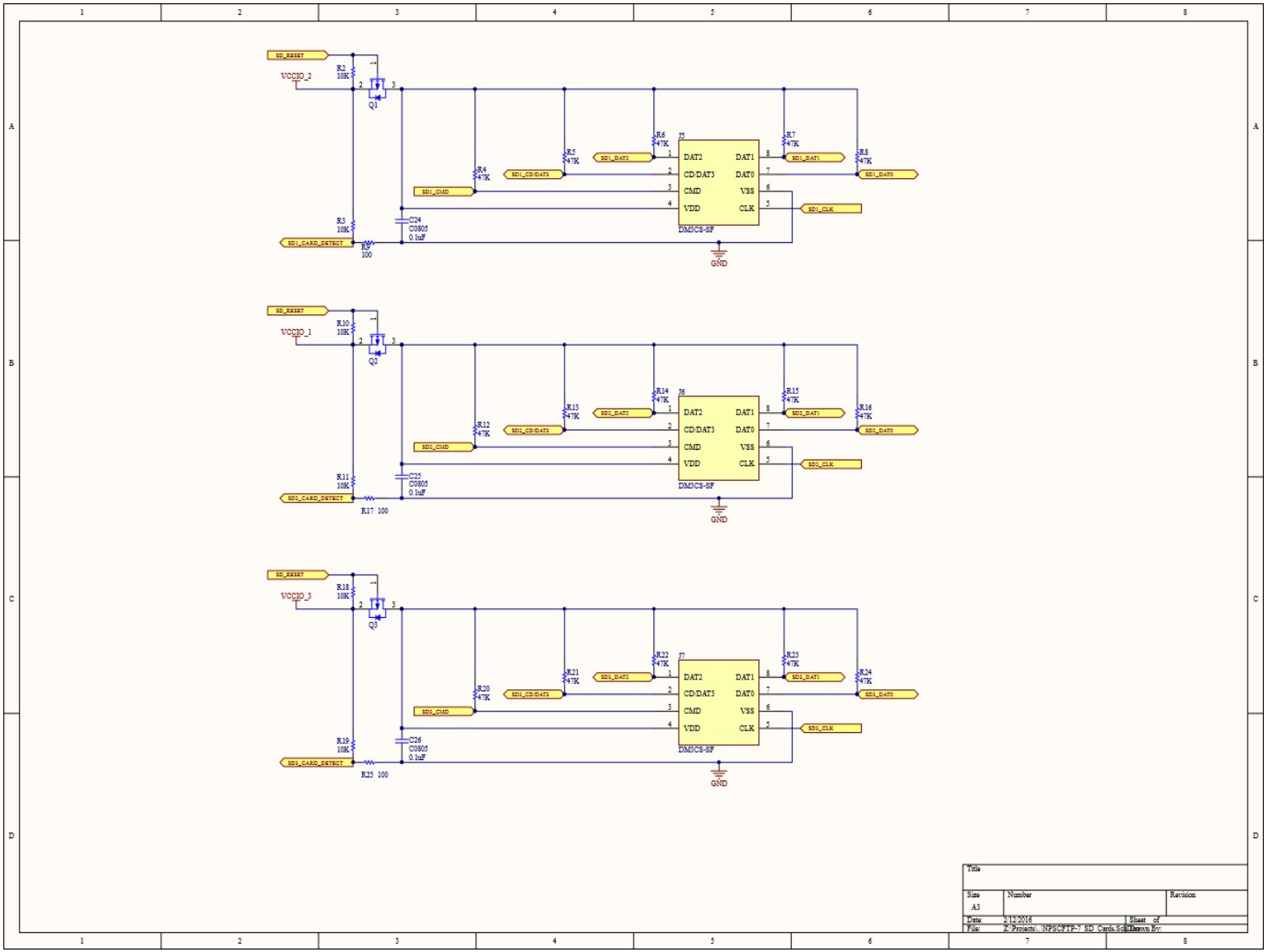
A. CFTP COMPOSITE DRAWING



B. CFTP SCHEMATIC DRAWINGS

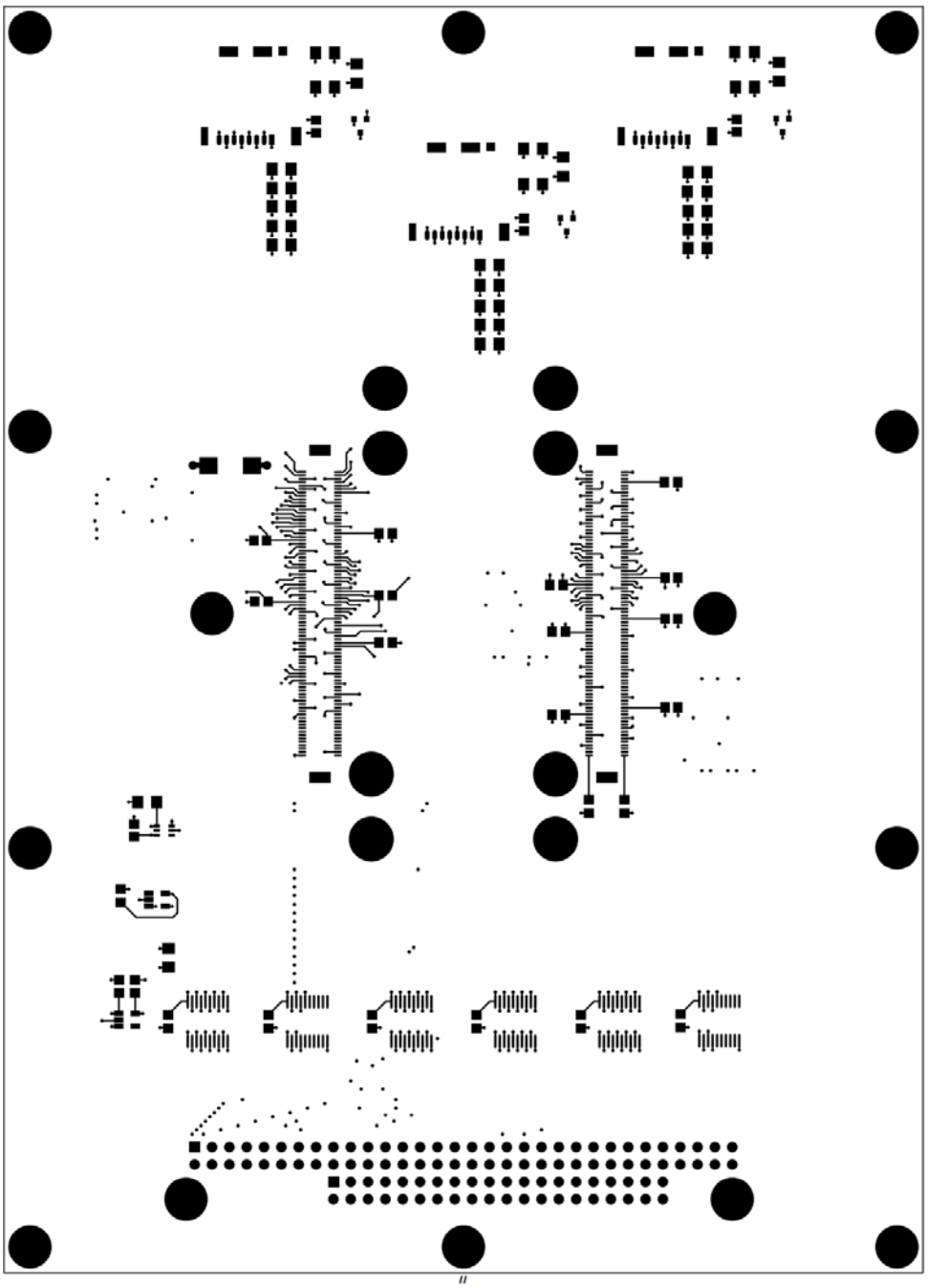



U3
SN74LVT245BPW
DATA_DIR
CARD_DATA_8
CARD_DATA_9
CARD_DATA_10
CARD_DATA_11
CARD_DATA_12
CARD_DATA_13
CARD_DATA_14
CARD_DATA_15
FPGA_DATA_8
FPGA_DATA_9
FPGA_DATA_10
FPGA_DATA_11
FPGA_DATA_12
FPGA_DATA_13
FPGA_DATA_14
FPGA_DATA_15
U5
CARD_ADDR_10
CARD_ADDR_9
CARD_ADDR_8
CARD_ADDR_7
CARD_ADDR_6
CARD_ADDR_5
CARD_ADDR_4
CARD_ADDR_3
FPGA_ADDR_10
FPGA_ADDR_9
FPGA_ADDR_8
FPGA_ADDR_7
FPGA_ADDR_6
FPGA_ADDR_5
FPGA_ADDR_4
FPGA_ADDR_3
U4
CFTP_PWR_REQ
CFTP_JTAG_TDO
CARD_IRQ?
FPGA_PWR_REQ
FPGA_JTAG_TDO
FPGA_IRQ?
U2
DATA_DIR
CARD_DATA_7
CARD_DATA_6
CARD_DATA_5
CARD_DATA_4
CARD_DATA_3
CARD_DATA_2
CARD_DATA_1
CARD_DATA_0
FPGA_DATA_7
FPGA_DATA_6
FPGA_DATA_5
FPGA_DATA_4
FPGA_DATA_3
FPGA_DATA_2
FPGA_DATA_1
FPGA_DATA_0
R1
10K
U6
CARD_ADDR_2
CARD_ADDR_1
CARD_ADDR_0
FPGA_ADDR_2
FPGA_ADDR_1
FPGA_ADDR_0
U7
CARD_RESET
CARD_CES
CARD_WE
CARD_OE
CFTP_JTAG_TDI
CFTP_JTAG_TMS
CFTP_JTAG_TCK
CARD_RCLK
FPGA_RESET
FPGA_CES
FPGA_WE
FPGA_OE
FPGA_JTAG_TDI
FPGA_JTAG_TMS
FPGA_JTAG_TCK
FPGA_RCLK
C18
C20
C19
C17
C21
C22
C0805
0.1uF
VCC_3V3
GND
U8
SN74AUP1G07DCK
PWR_GOOD
R26
470
PGR_LOAD
C27
U1
TLV71333PDBVR
VCC_5V
C15
1uF
C16
U9
SN74LVC1G14DBVR
C28
U10
SN74LVC1G06DCKR
CFTP_PWR_EN
FPGA_PWR_EN
C39
U11
TPS2085D
VCCIO_1_0_out
VCCIO_1_1_out
VCCIO_1_2_out
VCCIO_1_3_out
C29
C30
C31
C32
VCCIO_1
VCCIO_1_OCA
VCCIO_1_OCB
U12
VCCIO_2_0_out
VCCIO_2_1_out
VCCIO_2_2_out
C33
C34
C35
VCCIO_2
VCCIO_2_OCA
VCCIO_2_OCB
U13
C36
C37
C38
VCCIO_3
VCCIO_3_OCA
VCCIO_3_OCB
Title
Size
A3
Number
Revision
Date:
2/12/2016
Sheet of
File:
Z:\Projects\...\PBSCFTP-7 Octal Transceiver Power Management.SchDoc

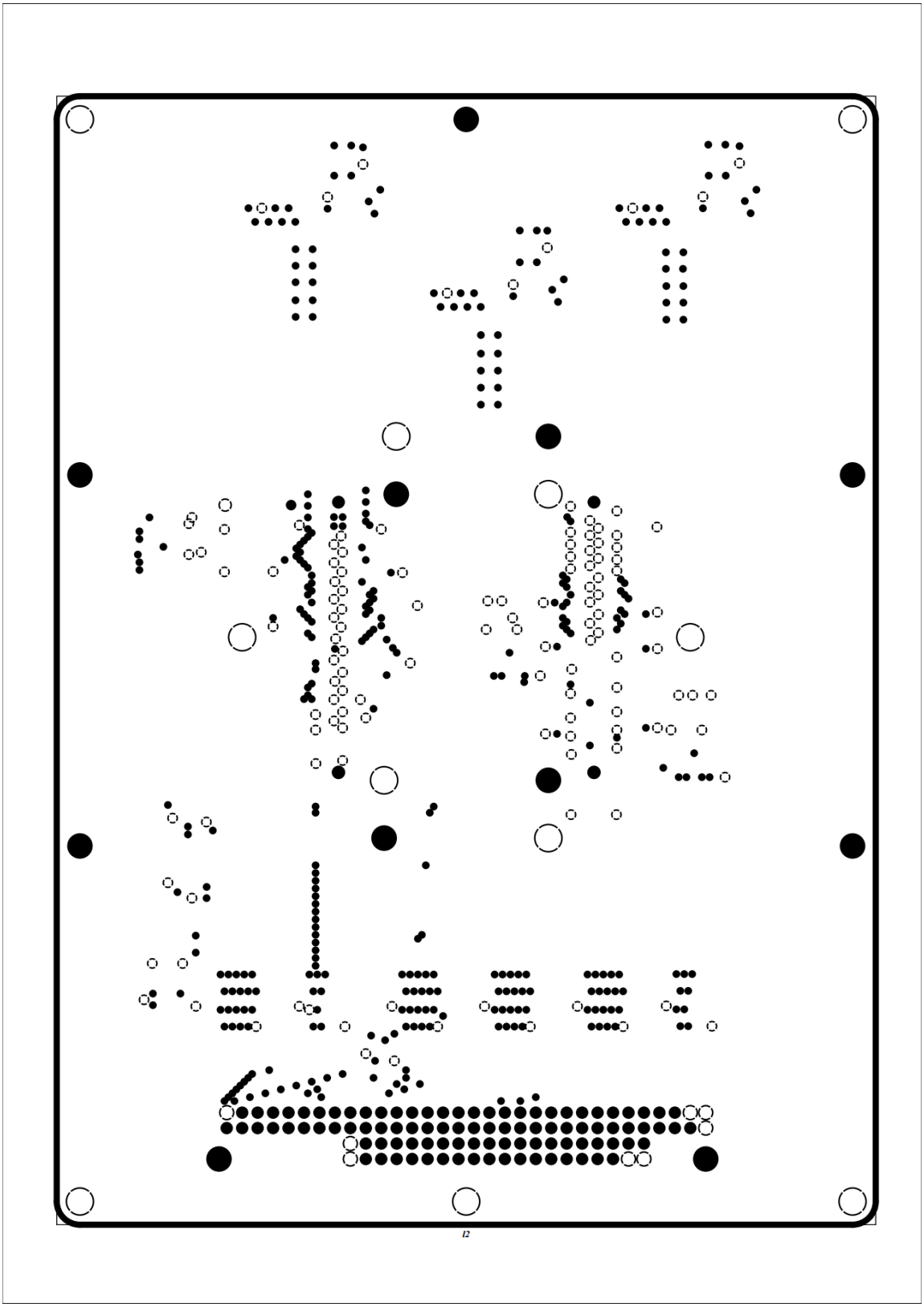

SD_RESET
VCCIO_2
R2 10K
Q1
R3 10K
SD1_CARD_DETECT
R9 100
C24 C0805 0.1uF
R4 47K
R5 47K
R6 47K
R7 47K
R8 47K
SD1_CD/DAT3
SD1_DAT2
SD1_DAT1
SD1_DAT0
SD1_CMD
SD1_CLK
J5
DAT2
CD/DAT3
CMD
VDD
DAT1
DAT0
VSS
CLK
DMSCS-SF
GND
VCCIO_1
R10 10K
Q2
R11 10K
SD2_CARD_DETECT
R17 100
C25 C0805 0.1uF
R12 47K
R13 47K
R14 47K
R15 47K
R16 47K
SD2_CD/DAT3
SD2_DAT2
SD2_DAT1
SD2_DAT0
SD2_CMD
SD2_CLK
J6
VCCIO_3
R18 10K
Q3
R19 10K
SD3_CARD_DETECT
R25 100
C26 C0805 0.1uF
R20 47K
R21 47K
R22 47K
R23 47K
R24 47K
SD3_CD/DAT3
SD3_DAT2
SD3_DAT1
SD3_DAT0
SD3_CMD
SD3_CLK
J7
Title
Size
A3
Number
Revision
Date:
2/12/2016
Sheet of
File:
Drawn By:

C. CFTP PCB GERBER PLOTS

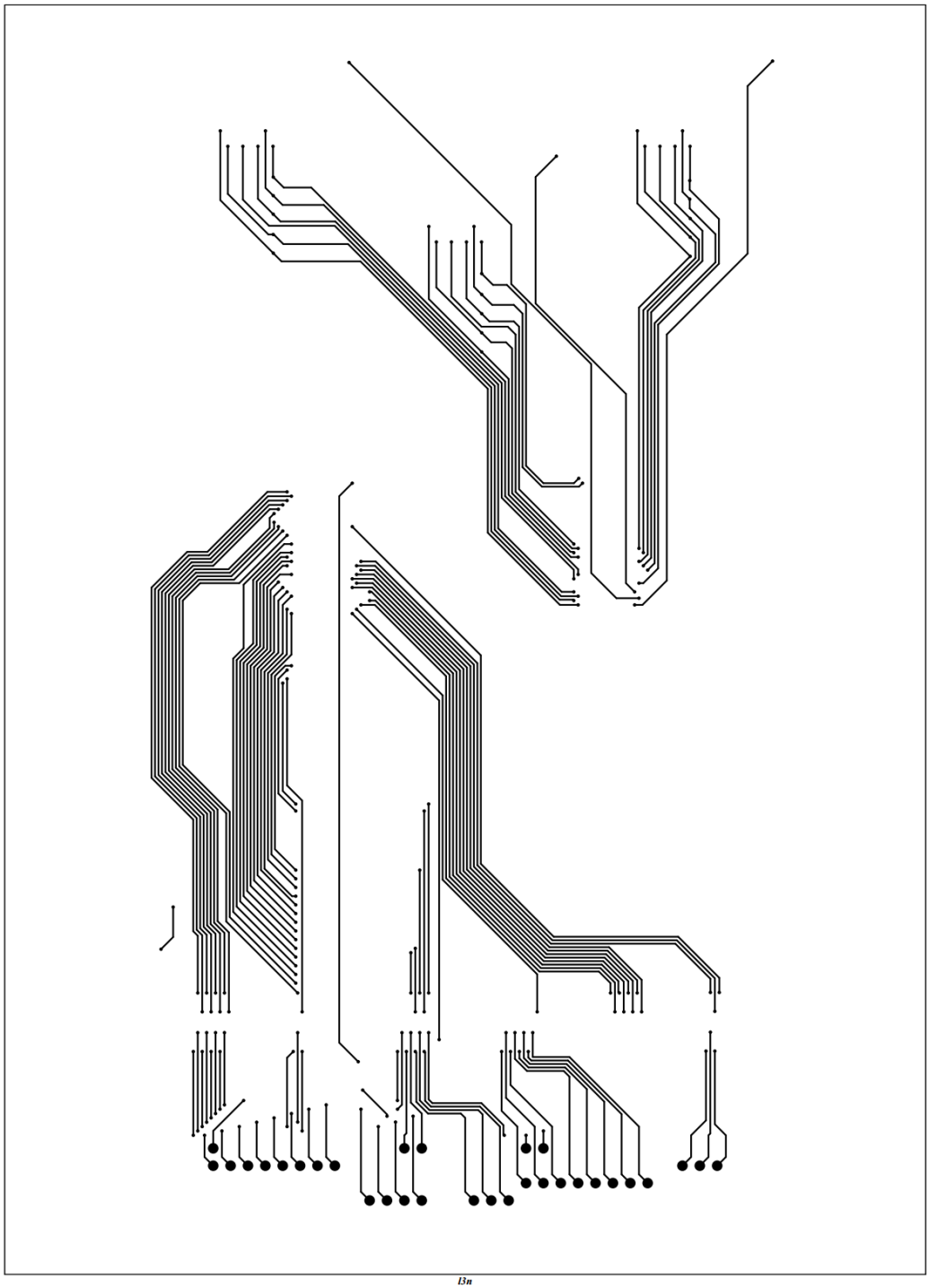
(1) Top Copper Layer



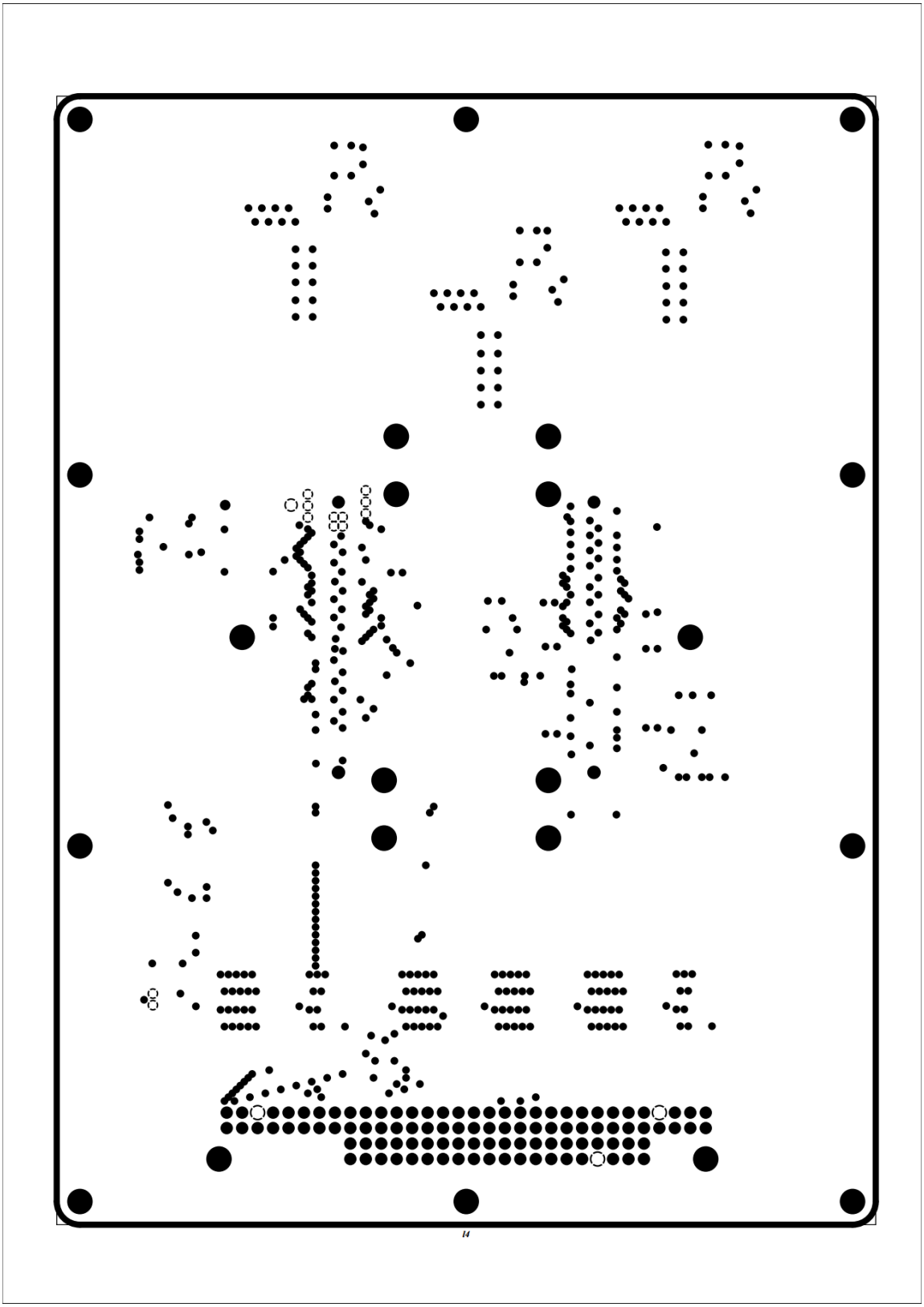
(2) Ground Plane A, Inner Layer 1



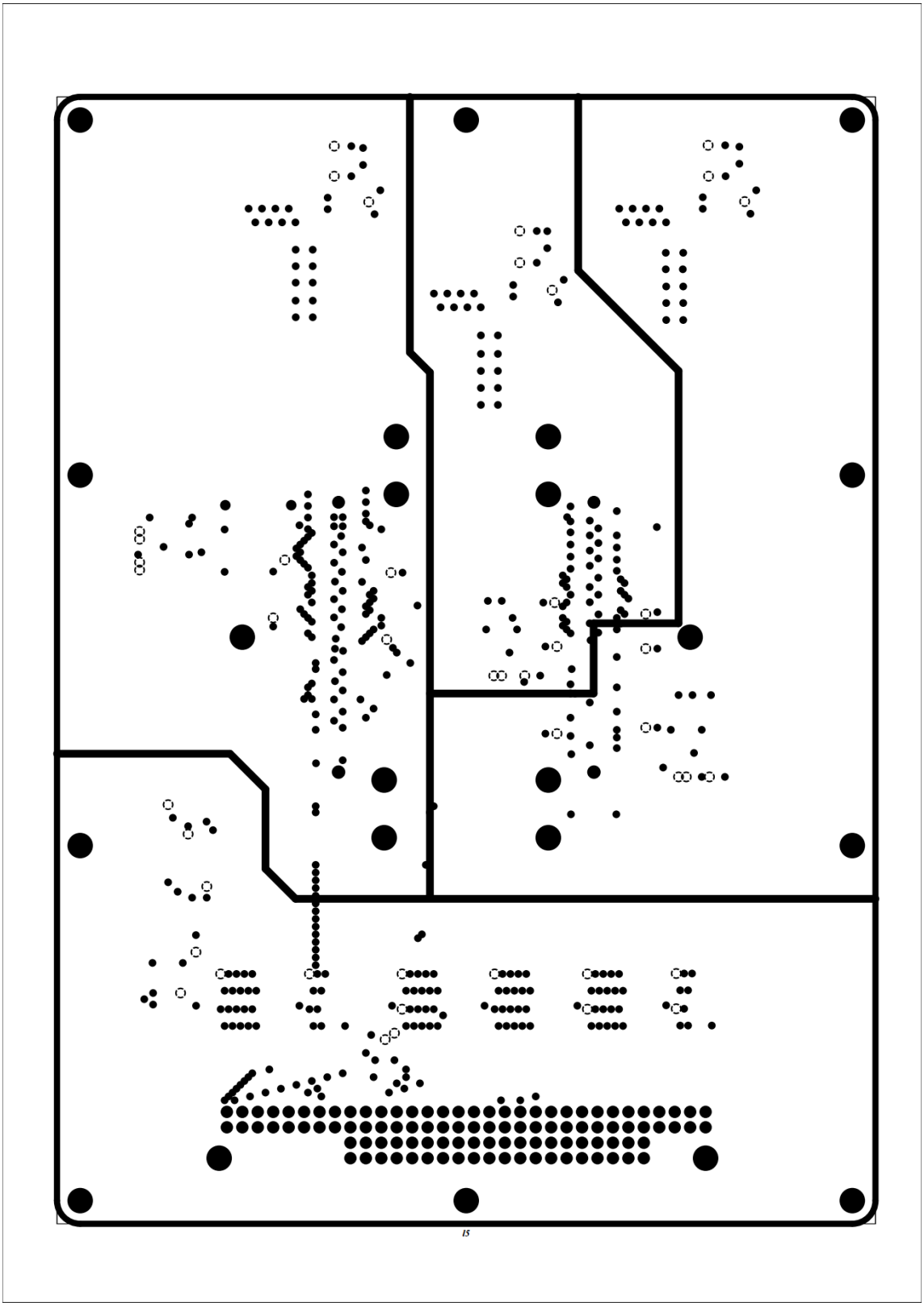
(3) Signal Plane A, Inner Layer 2



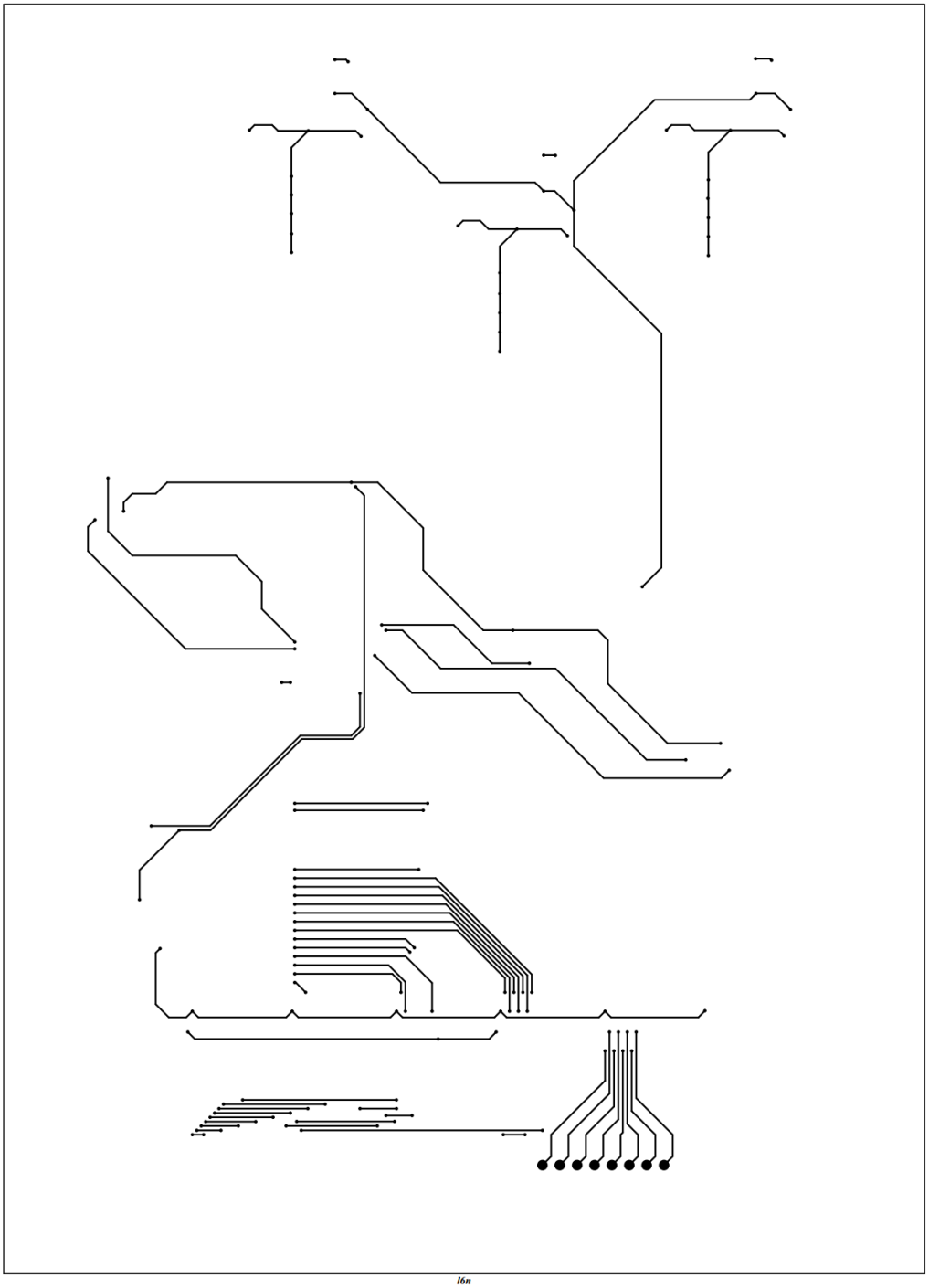
(4) Power Plane, 5V, Inner Layer 3



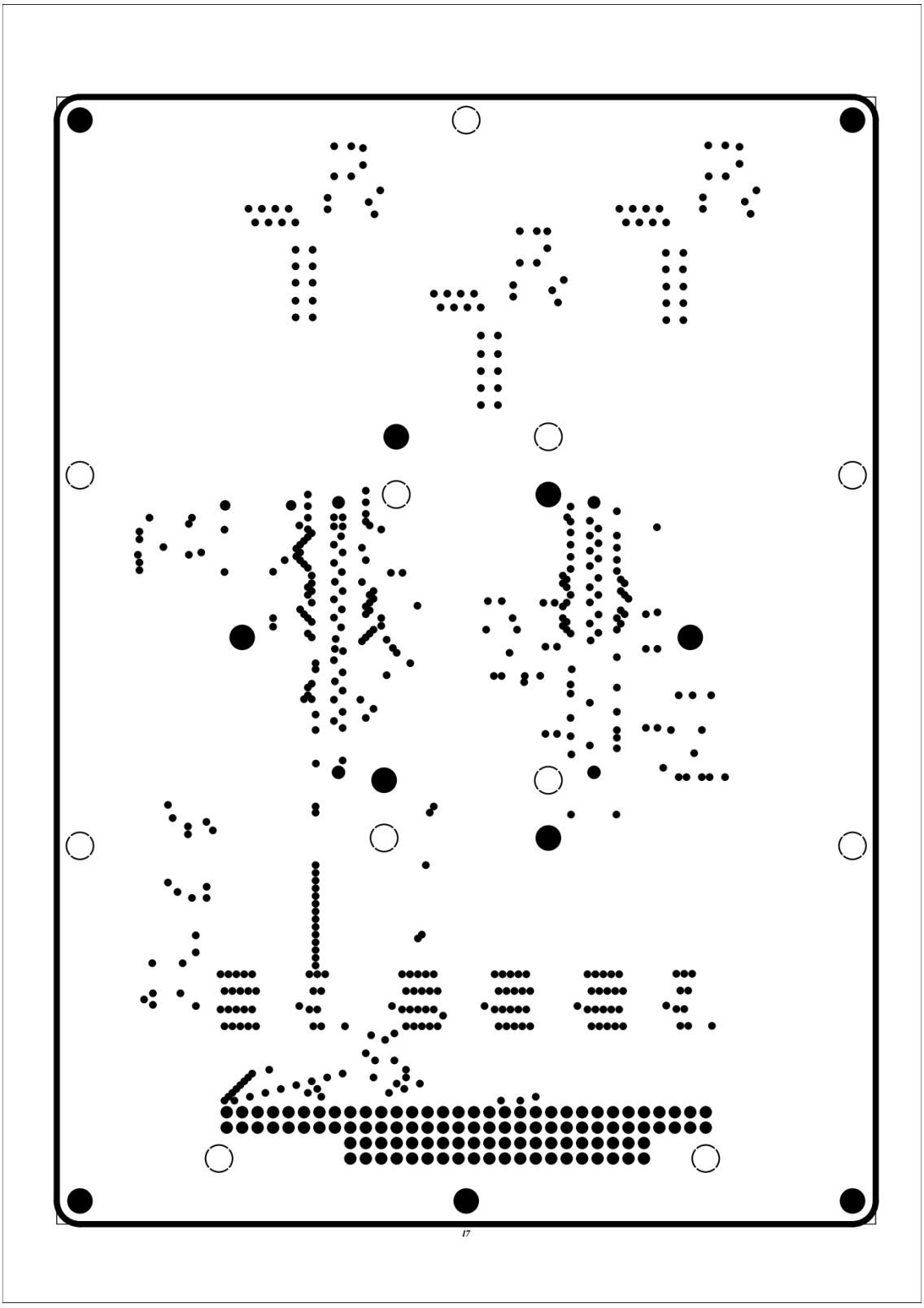
(5) Power Plane, 3V3, Inner Layer 4



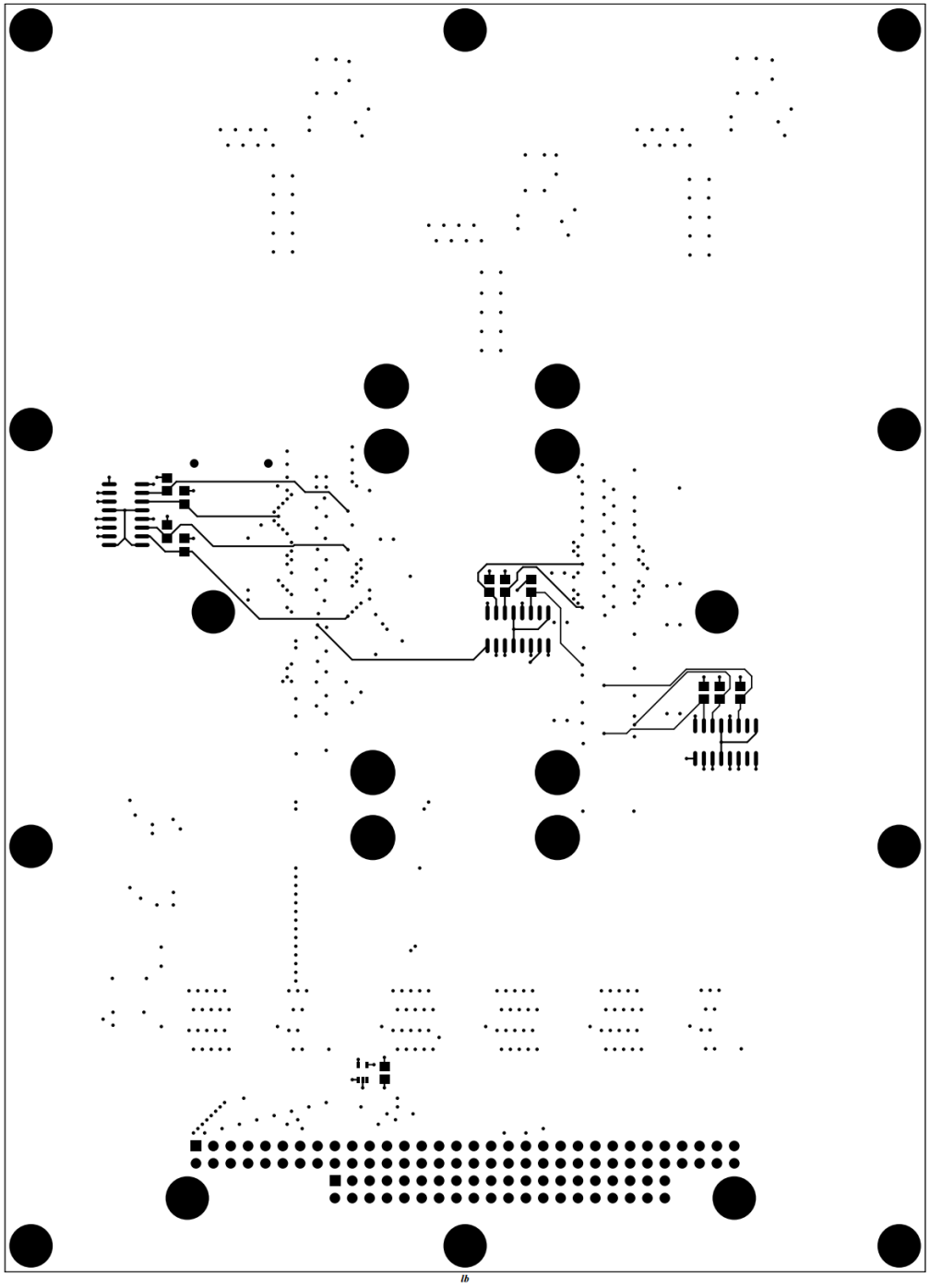
(6) Signal Plane B, Inner Layer 5



(7) Ground Plane B, Inner Layer 6



(8) Bottom Copper Layer



THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] D. C. Wilkinson, S.C. Daughtridge, J.L. Stone, H.H. Sauer and P. Darling, "TDRS-1 single event upsets and the effect of the space environment," *IEEE Trans. on Nuc. Sci.*, vol. 38, no. 6, pp. 1708–1712, Dec. 1991.

[2] D. Sinclair and J. Dyer, "Radiation effects and COTS parts in SmallSats," in *27th Annual AIAA/USU Conference on Small Satellites*, Logan, UT, 2013. [Online]. Available: http://digitalcommons.usu.edu/smallsat/2013/all2013/69

[3] D. A. Ebert, "Design and development of a configurable fault-tolerant processor (CFTP) for space applications," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2003.

[4] C. H. McLean, W. D. Deininger, J. Joniatis, P. K. Aggarwal, R. A. Spores, M. Deans, J. T. Yim, K. Bury, J. Martinez, E.H. Cardiff and C. E. Bacha, "Green propellant infusion mission program development and technology maturation," in *50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, Cleveland, OH, 2014. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2014-3481

[5] *Trapped particle radiation models.* (2013, Sep. 30). SPENVIS. [Online]. Available: https://www.spenvis.oma.be/help/background/traprad/traprad.html#APAE

[6] J. W. Howard and D. M. Hardage, "Spacecraft environments interactions: space radiation and its effects on electronic systems," NASA, Hampton, VA, Tech. Pap. TP-1999-209373, Oct. 1999.

[7] J. C. Payne, "Fault tolerant computing testbed: A tool for the analysis of hardware and software fault handling techniques," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 1998.

[8] D. C. Summers [8], "Implementation of a fault tolerant computing testbed: a tool for the analysis of hardware and software fault handling techniques," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2000.

[9] D. O. Hofheinz, "Completion and testing of a TMR computing testbed and recommendations for a flight-ready follow-on design," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2000.

[10] P. A. Lashomb, "Triple modular redundant (TMR) microprocessor system for field programmable gate array (FPGA) implementation," M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2002.

[11] S. A. Johnson, “Implementation of a configurable fault tolerant processor (CFTP),” M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2003.

[12] M.D. Berg, “Single event effects in FPGA devices 2014–1015,” presented at NASA Electronic Parts and Packaging Program Electronics Technology Workshop, Greenbelt, MD, 2015.

[13] R. Yuan, “Triple modular redundancy (TMR) in a configurable fault-tolerant processor (CFTP) for space applications,” M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2003.

[14] J. C. Coudeyras, “Radiation testing of the configurable fault tolerant processor (CFTP) for space-based applications,” M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2005.

[15] P. J. Majewicz, “Implementation of a configurable fault tolerant processor (CFTP) using internal triple modular redundancy (TMR),” M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2005.

[16] D. E. Dwiggins, “Fault tolerant microcontroller for the configurable fault tolerant processor,” M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2008.

[17] J. K. Goff, “Adaption of a fault-tolerant FPGA-based launch sequencer as a CubeSat payload processor,” M.S. thesis, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2014.

[18] G. Ayers, Salt Lake City, UT. (2014). [Online]. *The eXtensible Utah Multicore*. Available: https://github.com/grantae/mips32r1_xum. Accessed May. 5, 2015.

[19] D. A. Patterson and J. L. Hennessy, “Large and fast: Exploiting memory hierarchy,” in *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Burlington, MA: Morgan Kaufmann, 2014, pp. 372–398

[20] Xilinx, Inc., *Soft Error Mitigation Controller v4.1*, Sep. 2015.

[21] H. Takai, M.Wirthlin, and A. Harding, “Soft error rate estimations of the Kintex-7 FPGA within the ATLAS liquid argon (Lar) calorimeter,” presented at Topical Workshop on Electronics for Particle Physics, Perugia, Italy, 2013.

[22] D. S. Lee, M.Wirthlin, G. Swift, and A. C. Le, “Single-event characterization of the 28 nm Xilinx Kintex-7 field-programmable gate array under heavy ion irradiation,” in *Radiation Effects Data Workshop*, Paris, France, 2014, pp. 1–5.

[23] R. Kingsbury, F. Schmidt, K. Cahoy, and D. Sklair, “TID tolerance of popular CubeSat components,” in *Radiation Effects Data Workshop*, San Francisco, CA, 2013, pp. 1–5.

[24] Digilent, Inc. (2015). [Online]. *Genesys 2 Schematic*. Available: https://reference.digilentinc.com/_media/genesys2:genesys2_public_sch.pdf. Accessed Dec. 13, 2015.

[25] Xilinx, Inc., *7 Series FPGAs Configuration User Guide v1.10*, Jun. 2015.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California