ADAPTIVE-HYBRID REDUNDANCY MIPS ARCHITECTURE VERSION 2.2

TECHNICAL REPORT

Nicolas Hamilton, Major, USAF

AFIT/EN/TR-19-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/EN/TR-19-04

Abstract

This report describes in detail the architecture of an Adaptive-Hybrid Redundancy (AHR) MIPS processor based upon the Basic MIPS processor [1] and Triple Modular Redundancy (TMR) processor [2]. The AHR MIPS processor is the result of Adaptive-Hybrid Redundancy for Radiation-Hardening research and combines TMR and Temporal Software Redundancy. The AHR MIPS processor is hybrid in that it utilizes both hardware and software redundancy. It is adaptive because it has the capability to switch between TMR and TSR modes. There may be many other applications for the AHR MIPS processor beyond this specific research area.

Table of Contents

Page

Abstract ii

List of Figures iv

List of Tables v

I. Introduction 1

II. Inputs and Outputs 3

III. Internal Control Signals 6

IV. Finite State Machine (FSM) 10

V. Figures Illustrating AHR Controller Operation 46

Appendix A. AHR Controller VHDL Code 53

Appendix B. Version History 110

Bibliography 111

List of Figures

Figure Page

1 Basic MIPS Inputs and Outputs 46

2 TMR MIPS High-Level Block Diagram 47

3 TMR MIPS Detailed Block Diagram 48

4 AHR MIPS Detailed Block Diagram 50

5 AHR Controller Detailed Block Diagram 52

List of Tables

Table Page

1 AHR Controller Inputs and Outputs 3

2 AHR Controller Internal Control Signals Descriptions 7

3 AHR Controller FSM States 11

4 AHR Controller Outputs 18

ADAPTIVE-HYBRID REDUNDANCY MIPS ARCHITECTURE VERSION 2.2

I. Introduction

The AHR MIPS Architecture integrates both Triple Modular Redundancy (TMR) MIPS and Temporal Software Redundancy (TSR) MIPS. The key feature of this architecture is an overarching controller that selects to run TMR MIPS or TSR MIPS depending on the occurrence of errors. The controller begins executing in TMR MIPS mode. If no errors occur in TMR MIPS during a predefined time interval, the controller transitions operations from TMR MIPS to TSR MIPS. Operation continues in TSR MIPS until an error occurs. If no error has previously occurred during TSR MIPS operation or no error has occurred since the last time TSR MIPS created a save/restore point, TSR MIPS will be allowed to recover from the error; however, an error flag is set indicating that this error has occurred. After recovering from the error, if TSR MIPS successfully creates a save/restore point, the error flag is cleared. If a second error occurs before the error flag is cleared, the controller recovers from the error and returns to TMR MIPS operation.

The controller is able to detect when errors occur by monitoring the TMR Voter outputs when operating in TMR mode and monitoring the outputs of a single Basic MIPS processor when operating in TSR mode. The controller does not add clock cycle delays to the memory accesses performed by the TMR Voter or single Basic MIPS processor. The only added delay is due to combinational logic; this logic is utilized by the controller to reroute signals when the controller needs to change between TMR and TSR modes.

When switching between modes, the controller overrides the signals between the

TMR Voter and memory, the Basic MIPS processors and TMR Voter, and between the Basic MIPS processor being used in TSR mode and memory. The controller does this to send appropriate commands to the TMR Voter, Basic MIPS processors, and memory to effectively switch between modes. During the transition period, the controller will add clock cycle delays when relaying information to memory from the TMR Voter or Basic MIPS processor used for TSR mode and from memory to the TMR Voter or Basic MIPS processor used for TSR mode.

Special consideration must be made for the TSR mode when the program is complete. Program completion causes memory to issue a DONE signal back to the Basic MIPS processor running the TSR instructions. This DONE signal resets the Basic MIPS processor and causes its program counter to return to 0. However, program location 0 is the start of the TMR instructions. When this occurs, the controller intervenes and issues a branch command to the Basic MIPS processor so that its program counter will jump to the start of the TSR instructions.

II. Inputs and Outputs

The AHR Controller inputs and outputs are shown in Table 1. In order to make the AHR Controller work with the TMR Voter, two outputs were added to the TMR Voter that are inputs to the AHR Controller. These signals are the NEXT_INSTR and TMR_ERROR signals. The NEXT_INSTR signal is '1' when the TMR Voter is in state FSM_0 and '0' otherwise. The AHR Controller detects the change in this signal from '0' when the TMR Voter completes processing one instruction to '1' when the TMR Voter begins processing the next instruction. When this occurs, the AHR Controller increments an internal counter that tracks how many instructions have been processes since the last error occurred or since TMR MIPS operation began. The TMR_ERROR signal is '1' when the TMR Voter encounters a type 0 (one processor) or type 1 (multiple processor or timeout) error. When the TMR_ERROR signal is '1', the AHR Controller resets its internal instruction counter to zero.

Table 1. AHR Controller Inputs and Outputs

Name	**Type**	**Width**	**Description**
i_NEXT_INSTR	In	1	Signal indicating when the TMR Voter is ready to access the next instruction from memory
i_TMR_ERROR	In	1	Signal indicating when the TMR Voter is performing error recovery operations
i_MEM_READ	In	1	Voter's memory read signal
i_MEM_WRITE	In	1	Voter's memory write signal
i_MEM_ADDRESS	In	32	Voter's memory address signal
i_MEM_IN	In	32	Data the voter is attempting to write to memory
i_MEM_READY	In	1	Memory ready signal

Table 1 – *Continued on next page*

Table 1 – *Continued from previous page*

Name	**Type**	**Width**	**Description**
i_MEM_OUT	In	32	Data memory provides in response to a read request
i_MEM_DONE	In	1	Signal from memory at program completion
i_MEM_READ0	In	1	Basic MIPS Processor 0 read signal
i_MEM_WRITE0	In	1	Basic MIPS Processors 0, 1, and 2 write signals
i_MEM_ADDRESS0	In	32	Basic MIPS Processors 0, 1, and 2 address signals
i_MEM_IN0	In	32	Data Basic MIPS Processor 0 is attempting to write to the voter or memory
i_MEM_READY0 i_MEM_READY1 i_MEM_READY2	Out	1	Ready signals sent from the voter to Basic MIPS processors 0, 1, and 2
i_MEM_OUT0 i_MEM_OUT1 i_MEM_OUT2	Out	32	Data or instruction the voter is providing to Basic MPS processors 0, 1, and 2
i_RESET0 i_RESET1 i_RESET2	Out	32	Reset signals from the voter to Basic MIPS processors 0, 1, and 2
o_MEM_READ	In	1	Read signal sent to memory
o_MEM_WRITE	In	1	Write signal sent to memory
o_MEM_ADDRESS	In	32	Memory address signal sent to memory
o_MEM_IN	In	32	Data to write to memory
o_MEM_READY	In	1	Ready signal sent to the voter
o_MEM_OUT	In	32	Data sent to the voter in response to a read request
o_MEM_DONE	In	1	Done signal sent by the controller to the voter
o_MEM_READY0 o_MEM_READY1 o_MEM_READY2	Out	1	Ready signals sent from the controller to Basic MIPS processors 0, 1, and 2
o_MEM_OUT0 o_MEM_OUT1 o_MEM_OUT2	Out	32	Data or instruction the controller is providing to Basic MPS processors 0, 1, and 2
o_RESET0 o_RESET1	Out	32	Reset signals from the controller to Basic MIPS processors 0, 1, and 2

Table 1 – *Continued on next page*

Table 1 – *Continued from previous page*

Name	**Type**	**Width**	**Description**
o_RESET2			

III. Internal Control Signals

Table 2 describes the AHR Controller's internal control signals. These signals are used to determine the outputs of the AHR Controller to the Basic MIPS processors, TMR Voter, and Memory.

Table 2. AHR Controller Internal Control Signals Descriptions

Name	**Description**
f_MEM_ADDRESS	Address to possibly send to memory for read and write operations. Address sent to memory depends on f_MEM_ADDRESS_SEL
f_MEM_IN	Data to possibly send to memory for write operations. Data sent to memory depends on i_MEM_WRITE_SEL
f_MEM_OUT	Data to possibly send to the voter. Data sent to voter depends on i_MEM_READY_SEL
f_MEM_OUT0	Data to possibly send to Basic MIPS processor 0. Data sent to MIPS0 depends on i_MEM_READY0_SEL
f_MEM_READ_SEL	Determines which read signal to send to memory 00 - Pass Voter read signal to memory 01 - Pass MIPS0 read signal to memory 10 - Send a ‘0’ read signal to memory 11 - Send a ‘1’ read signal to memory
f_MEM_WRITE_SEL	Determines which write and data signals to send to memory 00 - Pass Voter write signal and i_MEM_IN signal to memory 01 - Pass MIPS0 write signal and i_MEM_IN signal to memory 10 - Send a ‘0’ write signal and f_MEM_IN signal to memory 11 - Send a ‘1’ write signal and f_MEM_IN signal to memory
f_MEM_ADDRESS_SEL	Determines which address signal to send to memory 00 - Pass Voter address signal to memory 01 - Pass MIPS0 address signal to memory 10 - Send f_MEM_ADDRESS signal to memory 11 - Send f_MEM_ADDRESS signal to memory
f_MEM_READY_SEL	Determines which ready and data signals to send to the voter 00 - Pass memory ready signal and i_MEM_OUT0 signal to the voter 01 - Send a ‘0‘ready signal and a zero o_MEM_OUT signal to the voter

Table 2 – *Continued on next page*

Table 2 – *Continued from previous page*

Name	**Description**
	10 - Send a '1' ready signal and f_MEM_OUT signal to the voter 11 - Send a '1' ready signal and f_MEM_OUT signal to the voter
f_MEM_DONE_SEL	Determine the done signal to send to the voter 00 - Pass memory done signal to the voter 01 - Send a '0'done signal to the voter 10 - Send a '1' done signal to the voter 11 - Send a '1' done signal to the voter
f_MEM_READY0_SEL	Determine which ready and data signals to send to MIPS0 00 - Pass memory ready signal and i_MEM_OUT0 signal from the voter to MIPS0 01 - Pass memory ready signal and i_MEM_OUT0 signal from memory to MIPS0 10 - Send a '0' ready signal and f_MEM_OUT0 signal to the voter 11 - Send a '1' ready signal and f_MEM_OUT0 signal to the voter
f_MEM_READY12_SEL	Determine which ready and data signals to send to MIPS1 and MIPS2 0 - Pass memory ready signal and i_MEM_OUT1(2) signal from the voter to MIPS1(2) 1 - Send a '0' ready signal and zero to MIPS1(2)
f_MEM_RESET0_SEL	Determine which reset signal to send to MIPS0 00 - Pass reset signal from the voter to MIPS0 01 - Send a '0' reset signal to MIPS 10 - Send a '1' reset signal to MIPS 11 - Send a '1' reset signal to MIPS
f_MEM_RESET12_SEL	Determine which reset signals to send to MIPS1 and MIPS2 00 - Pass reset signals from the voter to MIPS1(2) 01 - Send a '0' reset signal to MIPS1(2) 10 - Send a '1' reset signal to MIPS1(2) 11 - Send a '1' reset signal to MIPS1(2)
f_loop	Temporary storage used to hold the value of the current loop iteration during TMR to TSR and TSR to TMR transitions.

Table 2 – *Continued on next page*

Table 2 – *Continued from previous page*

Name	**Description**
	This value may be updated to accomodate these transitions
f_loop1	Temporary storage used to hold the value of the current loop iteration during TMR to TSR and TSR to TMR transitions. This value is not updated to accomodate these transitions
f_TEMP_ADDRESS	Stores the offset from the begininig of TSR save/recovery memory to the active save/recovery point in memory, then added to the save/recovery memory start point and the constant loop counter location offset
f_instr_count	Count the number of instructions since the last system reset or TMR Error. Returns to 0 upon reset or TMR Error
f_NEXT_INSTR	Stores the value of i_NEXT_INSTR
f_err_flag	Indicates that TSR MIPS has encountered an error when 1
f_rec_flag	Indicates that TSR MIPS is attempting to create a save/restore when 1

IV. Finite State Machine (FSM)

Table 3 shows the states in the AHR Controller Finite State Machine (FSM). The table shows the current state, next state, description, and condition upon which the current state transitions to the next state. States 0 and 1 represent normal TMR MIPS operation. State 2 is entered when TMR MIPS encounters an error. States 3 through 24 are the states that transition operations from TMR to TSR. State 25 represents normal TSR MIPS operation. Special handling is required when the program is completed while operating in TSR and states 26 to 28 facilitate this. States 29 through 52 are the states that transition from TSR to TMR.

The transition from TMR to TSR is only permitted to occur at the beginning of loops because there is no one-to-one mapping for internal registers, temporary memory, and save/restore point data between TMR and TSR, except for the loop counter. The registers and save/recovery memory containing the register values at the active save/recovery point will also not match because TMR allows 29 user defined registers while TSR only allows 14 (these counts exclude the loop counter and the zero register). While permanent memory locations are the same for TMR and TSR, any temporary memory usage will not. Variables stored to temporary memory by TMR and TSR will not match. When loops begin, the values of the registers are yet-to-be determined in the sense that normal program operation will overwrite the values in the registers to be the correct values for TMR or TSR operation, so copying the loop counter value from TMR to TSR or vice versa and starting at the beginning of that loop will result in correct transition between operating modes.

Table 3. AHR Controller FSM States

Current State	**Next State**	**Transition Condition**	**Description**
FSM_0	FSM_1	(i_NEXT_INSTR = 1 and f_NEXT_INSTR = 0)	Idle state. Waiting for TMR Voter next instruction signal
FSM_0	FSM_2	i_TMR_ERROR = 1	Idle state. Waiting for TMR error to occur
FSM_0	FSM_3	(f_instr_count $\geq$ k_switch_point and i_MEM_READ = 1 and i_MEM_ADDRESS = k_TMR_LOOP_START)	Idle state. Waiting to reach point at which TMR to TSR transition occurs
FSM_1	FSM_0	None	Increment f_instruction_count
FSM_2	FSM_0	i_TMR_ERROR = 0	Wait for TMR error to be resolved
FSM_3	FSM_4	i_MEM_READY = 1	Start transition from TMR MIPS to TSR MIPS. Wait for memory ready signal
FSM_4	FSM_5	i_MEM_READY = 0	Provide TMR Voter command to write loop count to TSR save/recovery memory location 14. Wait for memory read signal to return to 0.
FSM_5	FSM_6	i_MEM_READ = 0	Wait for TMR Voter ready signal signal to return to 0.
FSM_6	FSM_2	i_TMR_ERROR = 1	Wait for TMR error to occur
FSM_6	FSM_7	i_MEM_WRITE = 1	Wait for TMR Voter write signal signal
FSM_7	FSM_8	i_MEM_WRITE = 0	Wait for TMR Voter write signal signal to return to 0. Store the loop value presented as i_MEM_IN to f_loop and f_loop1
FSM_8	FSM_9	None	Add 1 to f_loop, but leave f_loop1

Table 3 – *Continued on next page*

Table 3 – *Continued from previous page*

Current State	**Next State**	**Transition Condition**	**Description**
			unchanged
FSM_9	FSM_10	i_MEM_READY = 1	Write f_loop to TSR save/recovery memory location 14. Wait for memory ready signal
FSM_10	FSM_11	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_11	FSM_12	i_MEM_READY = 1	Write f_loop to TSR save/recovery memory location 29. Wait for memory ready signal
FSM_12	FSM_13	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_13	FSM_14	i_MEM_READY = 1	Write active save/recovery point to TSR save/recovery memory location 30. Wait for memory ready signal
FSM_14	FSM_15	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_15	FSM_16	None	Reset all MIPS Processors
FSM_16	FSM_17	i_MEM_READ0 = 1	Wait for MIPS0 read signal
FSM_17	FSM_18	i_MEM_READ0 = 0	Send LW R31 0 command to MIPS 0. Wait for MIPS0 read signal to return to 0
FSM_18	FSM_19	i_MEM_READ0 = 1	Wait for MIPS0 read signal
FSM_19	FSM_20	i_MEM_READ0 = 0	Send f_loop1 to MIPS 0. Wait for MIPS0 read signal to return to 0

Table 3 – *Continued on next page*

Table 3 – *Continued from previous page*

Current State	**Next State**	**Transition Condition**	**Description**
FSM_20	FSM_21	i_MEM_READ0 = 1	Wait for MIPS0 read signal
FSM_21	FSM_22	i_MEM_READ0 = 0	Send ADDI R31 R30 0 to MIPS 0. Wait for MIPS0 read signal to return to 0
FSM_22	FSM_23	i_MEM_READ0 = 1	Wait for MIPS0 read signal
FSM_23	FSM_24	i_MEM_READ0 = 0	Send BEQ R0 R0 TSR Branch Distance to MIPS0. Wait for MIPS0 read signal to return to 0
FSM_24	FSM_25	i_MEM_READ0 = 1	Wait for MIPS0 read signal. Once in FSM_25, normal TSR operations begin
FSM_25	FSM_25a	i_MEM_READ0 = 1 and i_MEM_ADDRESS0 = k_TSR_RECOVERY and f_err_flag = 0	Wait for MIPS0 read signal, MIPS0 to read an instruction from the recovery code, and the error flag to be 0. This is the first error to occur since the start of the TSR program or the creation of the last save/restore point
FSM_25a	FSM_25	i_MEM_READ0 = 0	Wait for MIPS0 read singal to to return to 0. This ensures that the read operation is completed and the error will not trigger TMR recovery
FSM_25	FSM_25	i_MEM_READ0 = 1 and i_MEM_ADDRESS0 = k_TSR_SAVE and	Wait for MIPS0 read signal and MIPS0 to read an instruction

Table 3 – *Continued on next page*

Table 3 – *Continued from previous page*

Current State	**Next State**	**Transition Condition**	**Description**
			from the save/restore creation code. Set f_rec_flag = 1
FSM_25	FSM_25	i_MEM_READ0 = 1 and i_MEM_ADDRESS0 ¡= k_TSR_END and f_rec_flag = 1	Wait for MIPS0 read signal, MIPS0 to read an instruction from the TSR instruction set, and thre recovery flag to be 1. TSR has successfully created a new save/restore point after an error. Set the f_err_flag = 0 and the f_rec_flag = 0
FSM_25	FSM_26	i_MEM_DONE = 1	Wait for memory done signal
FSM_26	FSM_27	i_MEM_READ0 = 1	The program is complete. Without appropriate intervention, MIPS0 will reset and try to read address zero (TMR instructions) rather than the TSR instructions. Interrupt communications between MIPS0 and Memory. Wait for MIPS0 Read Signal after reset. Set the f_err_flag = 0 and the f_rec_flag = 0
FSM_27	FSM_28	i_MEM_READ0 = 0	Send branch instruction to branch to start of TSR instructions to MIPS0. Wait for MIPS0 read signal to return to 0
FSM_28	FSM_25	i_MEM_READ0 = 1	Wait for MIPS0 read signal

Table 3 – *Continued on next page*

Table 3 – *Continued from previous page*

Current State	**Next State**	**Transition Condition**	**Description**
			to resume normal TSR operation
FSM_25	FSM_29	i_MEM_READ0 = 1 and i_MEM_ADDRESS0 = k_TSR_RECOVERY and f_err_flag = 1	Wait for MIPS0 read signal, MIPS0 to read an instruction from the recovery code, and the error flag to be 1. A second error has occured before save/restore point creation could be completed
FSM_29	FSM_30	i_MEM_READY = 0	Intercept all communications from memory to MIPS0. Wait for memory ready signal to return to 0. Reset MIPS0
FSM_30	FSM_31	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_31	FSM_32	i_MEM_READY = 1	Read active save/recovery point from memory and wait for memory ready signal
FSM_32	FSM_33	i_MEM_READY = 0	Wait for memory ready signal to return to 0. Store i_MEM_IN to f_TEMP_ADDRESS to keep track of the active save/recovery memory location
FSM_33	FSM_34	i_MEM_READY = 1	Read active save/recovery loop counter from memory and wait for memory ready signal
FSM_34	FSM_35	i_MEM_READY = 0	Wait for memory ready signal to return to 0. Store the loop value

Table 3 – *Continued on next page*

Table 3 – *Continued from previous page*

Current State	**Next State**	**Transition Condition**	**Description**
			presented as i_MEM_IN to f_loop and f_loop1
FSM_35	FSM_36	None	Subtract 1 from f_loop, but leave f_loop1 unchanged
FSM_36	FSM_37	i_MEM_READY = 1	Write f_loop to save/recovery memory location 30. Wait for memory ready signal
FSM_37	FSM_38	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_38	FSM_38	i_MEM_READY = 1	Write f_loop to save/recovery memory location 62. Wait for memory ready signal
FSM_39	FSM_40	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_40	FSM_41	i_MEM_READY = 1	Write the memory address for the start of the TMR instruction loop (2) to recovery memory location 31. Wait for memory ready signal
FSM_41	FSM_42	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_42	FSM_43	i_MEM_READY = 1	Write the memory address for the start of the TMR instruction loop (2) to recovery memory location 63. Wait for memory ready signal
FSM_43	FSM_44	i_MEM_READY = 0	Wait for memory ready signal

Table 3 – *Continued on next page*

Table 3 – *Continued from previous page*

Current State	**Next State**	**Transition Condition**	**Description**
			to return to 0
FSM_44	FSM_45	i_MEM_READY = 1	Write active save/recovery point to recovery memory location 64. Wait for memory ready signal
FSM_45	FSM_46	i_MEM_READY = 0	Wait for memory ready signal to return to 0
FSM_46	FSM_47	i_MEM_READ = 1	Allow TMR MIPS operations to begin. Wait for voter read signal
FSM_47	FSM_48	i_MEM_READ = 0	Send LW R31 R0 0 instruction to the TMR Voter. Wait for voter read signal to return to 0
FSM_48	FSM_49	i_MEM_READ = 1	Wait for voter read signal
FSM_49	FSM_50	i_MEM_READ = 0	Send f_loop to the TMR Voter. to load the loop counter into R31 Wait for voter read signal to return to 0
FSM_50	FSM_51	i_MEM_READ = 1	Wait for voter read signal
FSM_51	FSM_52	i_MEM_READ = 0	Send NOP command to Voter so TMR MIPS can proceed to instruction 2, the beginning of TMR MIPS instruction loop
FSM_52	FSM_0	None	Begin normal TMR operations

Table 4. AHR Controller Outputs

Current State	Outputs
FSM_0	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 00
	f_MEM_WRITE_SEL = 00
	f_MEM_ADDRESS_SEL = 00
	f_MEM_READY_SEL = 00
	f_MEM_DONE_SEL = 00
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00
FSM_1	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 00
	f_MEM_WRITE_SEL = 00
	f_MEM_ADDRESS_SEL = 00
	f_MEM_READY_SEL = 00
	f_MEM_DONE_SEL = 00
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00
FSM_2	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 00
	f_MEM_WRITE_SEL = 00
	f_MEM_ADDRESS_SEL = 00
	f_MEM_READY_SEL = 00
	f_MEM_DONE_SEL = 00
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00
FSM_3	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 00
	f_MEM_WRITE_SEL = 00
	f_MEM_ADDRESS_SEL = 00
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00

FSM_4	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = SW R31 R0 save/recovery memory location 14
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 11
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00

FSM_5	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = SW R31 R0 save/recovery memory location 14
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00
FSM_6	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = SW R31 R0 save/recovery memory location 14
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00
FSM_7	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 10
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00

FSM_8	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00

FSM_9	f_MEM_ADDRESS = Save/recovery memory location 14
	f_MEM_IN = f_loop
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00

FSM_10	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00

FSM_11	f_MEM_ADDRESS = Save/recovery memory location 29
	f_MEM_IN = f_loop
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00
FSM_12	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00
FSM_13	f_MEM_ADDRESS = Save/recovery memory location 30
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 00

FSM_14	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 00
	f_MEM_RESET12_SEL = 00

FSM_15	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_16	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 11

FSM_17	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = LW R31 R0 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 11
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_18	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = LW R31 R0 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 11

FSM_19	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = f_loop1
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 11
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_20	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = f_loop1
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 11

FSM_21	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = ADDI R30 R31 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 11
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_22	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = ADDI R30 R31 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 11

FSM_23	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = BEQ R0 R0 TSR Branch Distance
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 11
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_24	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = BEQ R0 R0 TSR Branch Distance
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 11

FSM_25	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 01
	f_MEM_WRITE_SEL = 01
	f_MEM_ADDRESS_SEL = 01
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 01
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 01

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11
FSM_25a	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 01
	f_MEM_WRITE_SEL = 01
	f_MEM_ADDRESS_SEL = 01
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 01
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 01
	f_MEM_RESET12_SEL = 11
FSM_26	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_27	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = BEQ R0 R0 TSR Branch Distance
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 11
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 11

FSM_28	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = BEQ R0 R0 TSR Branch Distance
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11
FSM_29	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11
FSM_30	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_31	f_MEM_ADDRESS = Save/recovery memory location 30
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 11
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11

FSM_32	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_33	f_MEM_ADDRESS = Save/recovery memory location 14 + f_TEMP_ADDRESS
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 11
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11

FSM_34	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_35	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11

FSM_36	f_MEM_ADDRESS = Save/recovery memory location 30
	f_MEM_IN = f_loop
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	Outputs
	f_MEM_RESET12_SEL = 11

FSM_37	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11

FSM_38	f_MEM_ADDRESS = Save/recovery memory location 62
	f_MEM_IN = f_loop
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11
FSM_39	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11
FSM_40	f_MEM_ADDRESS = Save/recovery memory location 31
	f_MEM_IN = TMR loop start address
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_41	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11

FSM_42	f_MEM_ADDRESS = Save/recovery memory location 63
	f_MEM_IN = TMR loop start address
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11

FSM_43	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11

FSM_44	f_MEM_ADDRESS = Save/recovery memory location 64
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 11
	f_MEM_ADDRESS_SEL = 11
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 11
FSM_45	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 11
	f_MEM_READY0_SEL = 10
	f_MEM_READY12_SEL = 1
	f_MEM_RESET0_SEL = 11
	f_MEM_RESET12_SEL = 11
FSM_46	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 01
FSM_47	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = LW R31 R0 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 11
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 01
FSM_48	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 01

FSM_49	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = f_loop
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 11
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 01

FSM_50	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 01

FSM_51	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = NOP
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 11
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10
	f_MEM_RESET12_SEL = 01

FSM_52	f_MEM_ADDRESS = 0
	f_MEM_IN = 0
	f_MEM_OUT = 0
	f_MEM_OUT0 = 0
	f_MEM_READ_SEL = 10
	f_MEM_WRITE_SEL = 10
	f_MEM_ADDRESS_SEL = 10
	f_MEM_READY_SEL = 01
	f_MEM_DONE_SEL = 01
	f_MEM_READY0_SEL = 00
	f_MEM_READY12_SEL = 0
	f_MEM_RESET0_SEL = 10

Table 4 – *Continued on next page*

Table 4 – *Continued from previous page*

Current State	**Outputs**
	f_MEM_RESET12_SEL = 01

V. Figures Illustrating AHR Controller Operation

Figure 1 shows the external connections to the Basic MIPS processor, excluding the clock and reset inputs. The inputs are on the left and the outputs are on the right. Signals with an "i_" preceding them are input signals and those with an "o_" preceding them are output signals.

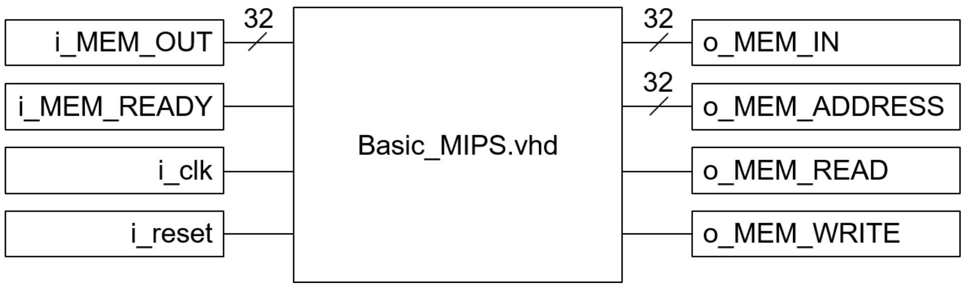

Figure 1. Basic MIPS Inputs and Outputs

Figure 2 shows how Basic MIPS processors are connected to the TMR Voter and Memory at a high-level where the individual memory access and memory response signals are replaced by single lines for simplicity.

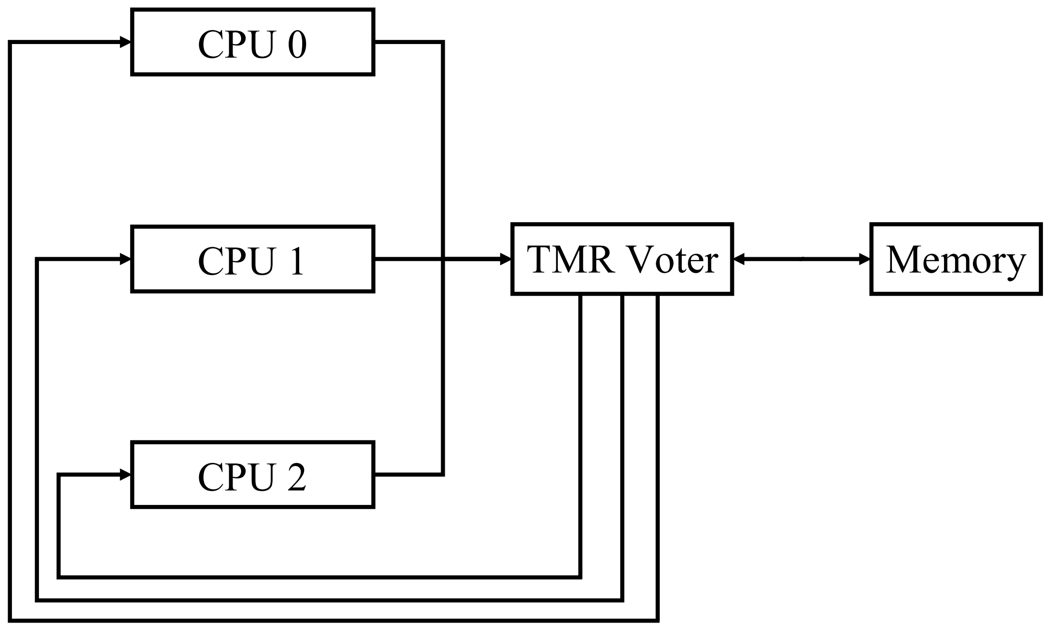

Figure 2. TMR MIPS High-Level Block Diagram

Figure 3 shows a detailed view of the interconnections between the Basic MIPS processors, TMR Voter, and Memory. This figure illustrates each individual connecting wire. The elongated hexagon blocks are used to label signals. If the routing of these signals were shown in the figure, it would be very difficult to discern where all of the signals were connecting. The gates used to determine the reset signals to be sent to the Basic MIPS processors are two and three input OR gates. The i_clk inputs on the Basic MIPS processors and TMR Voter are connected to the master clock and the i_reset signal on the TMR Voter is connected to the master reset (i_RESET).

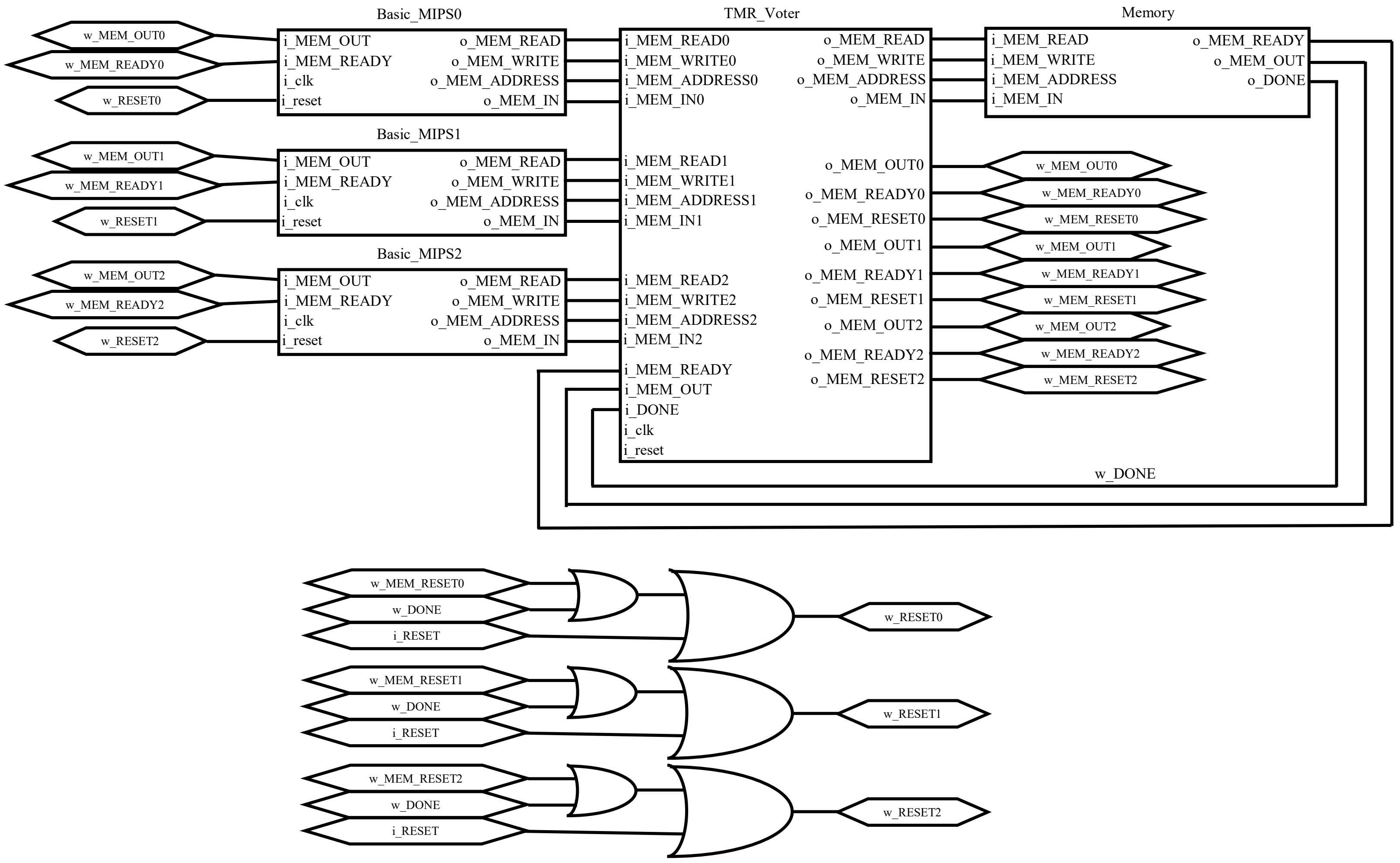

Figure 3. TMR MIPS Detailed Block Diagram

Figure 4 illustrates how the AHR Controller is integrated into the Basic MIPS, TMR Voter, and Memory structure. The logic determining the reset signals to be sent to the individual Basic MIPS processors has been integrated into the AHR Controller. Signal wires that cross over one another do not connect to one another. Signal wires that intersect at a "T" are connected and represent the same signal. This is true for the MEM_READ0, MEM_WRITE0, MEM_ADDRESS0, and MEM_IN0 signals. The i_clk inputs on the Basic MIPS processors, TMR Voter, and AHR Controller are connected to the master clock and the i_reset signals on the TMR Voter and AHR Controller are connected to the master reset (i_RESET).

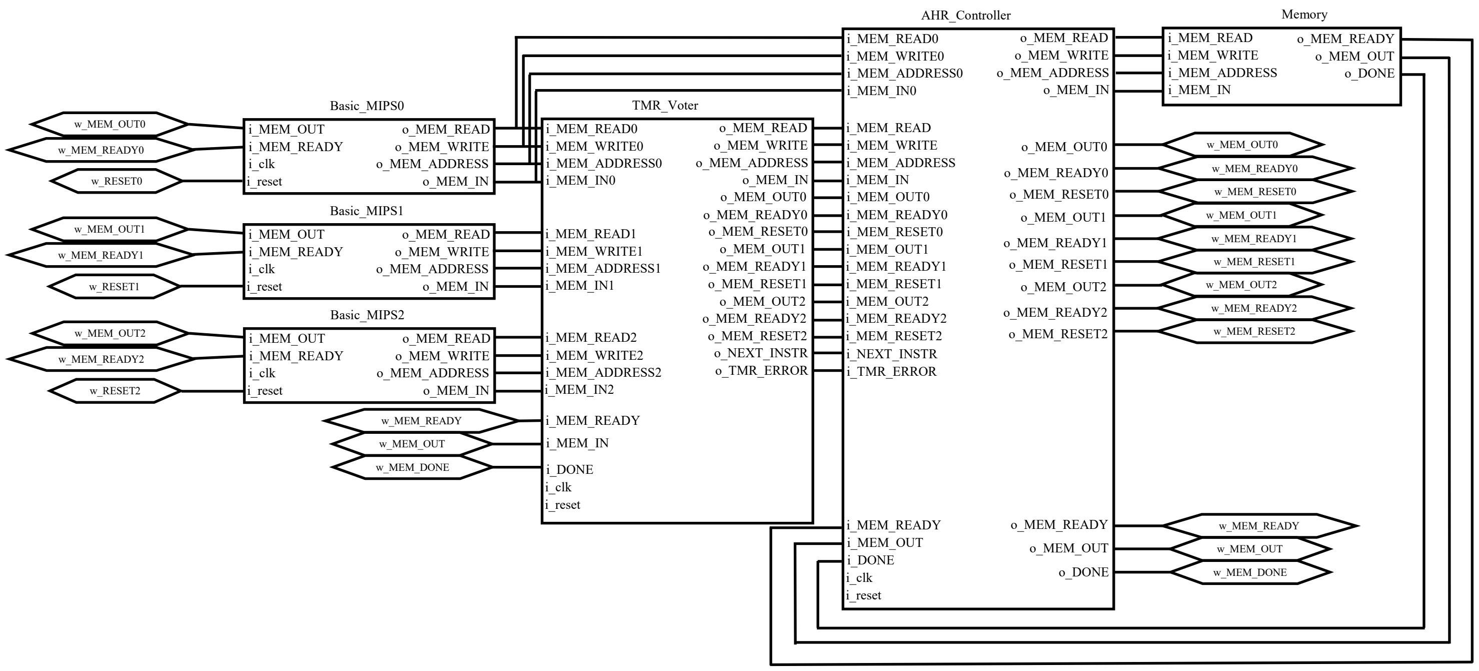

Figure 4. AHR MIPS Detailed Block Diagram

Figure 5 shows the internals of the AHR Controller. The state machine determines the control signals which are used to determine the outputs to the Basic MIPS processors, the TMR Voter, and Memory. The state machine control signals were previously described in Table 4. The elongated pentagons with tips pointing to the right are used to denote input signals. The elongated pentagons with tips pointing to the left denote output signals. The trapezoids in this figure represent multiplexers. The top input on the left side of the two input multiplexers is the 0 input and the bottom input is the 1 input. Similarly, the inputs are the 00, 01, 10, and 11 inputs from top to bottom on the left side of the four input multiplexers. The select signals shown as connecting to the bottom of the multiplexers determine which input is passed through the multiplexer such that a 0 signal passes the 0 input, a 1 signal passes the 1 input, a 00 signal passes the 00 input, a 01 signal passes the 01 input, and so on.

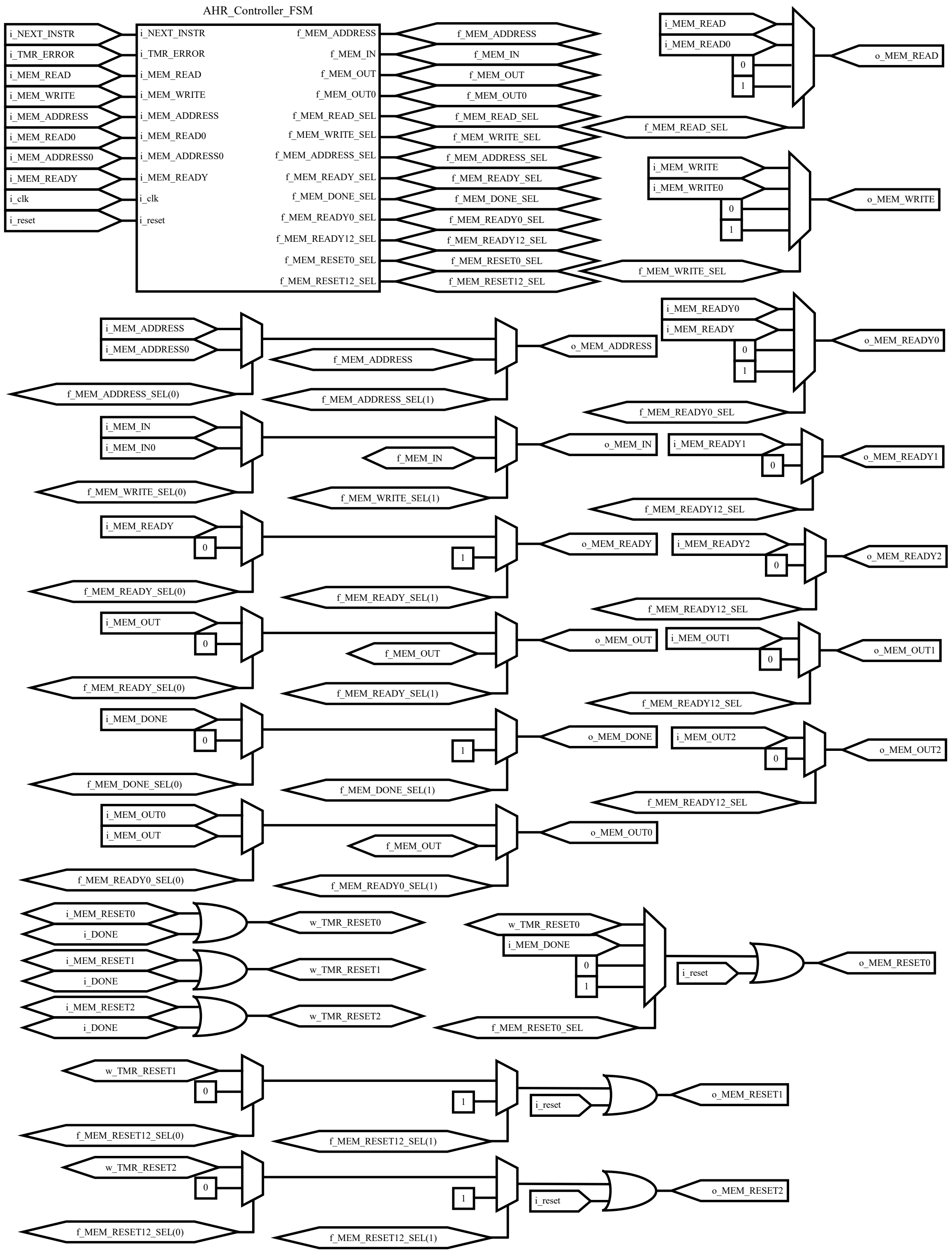

Figure 5. AHR Controller Detailed Block Diagram

Appendix A. AHR Controller VHDL Code

This appendix provides an example of the Adaptive-Hybrid Redundancy (AHR) Controller. This voter has been customized so so that it knows the start of TMR MIPS and TSR MIPS instructions in memory and switches from TMR to TSR mode after 15,000 instructions are processed without error. The start of the TMR MIPS program is always memory location 0 and the start of the TMR MIPS program loop is indicated by *k_TMR_LOOP_START*. The start of the TSR MIPS program is indicated by *k_TSR_START_BRANCH* while the start of the TSR MIPS program loop is indicated by *k_TSR_LOOP_START_BRANCH*. The transition point is set by the *k_switch_point* variable. The AHR Controller also has knowledge of where the TSR MIPS error recovery instructions(*k_TSR_RECOVERY*) and save/restore point creation instructions (*k_TSR_SAVE*) reside in memory. It has also been customized to point to a specific memory location for the save/restore point. This value is set by the *k_mem_location* variable.

```
--| AHR_Controller_v2_Test1001.vhd
--| Author: Nicolas Hamilton using write_AHR_Controller_v2.m
--| Created:  23 July 2019 at 20:19:02
--| Switch between TMR MIPS and TSR MIPS operation depending
   on the radiation
--| environment.  Starts operating in TMR MIPS.  If no errors
    occur over a
--| predifined time period, automatically switches to
   operating in TSR MIPS.
--| If an error occurs while operating in TSR MIPS,
   automatically switches to
--| TMR MIPS.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity AHR_Controller_v2_Test1001 is
   port (i_clk            : in   std_logic;
         i_reset          : in   std_logic;
```

```
        i_NEXT_INSTR    : in  std_logic;
    -- From TMR Voter   - used to determine next state
        i_TMR_ERROR     : in  std_logic;
    -- From TMR Voter   - used to determine next state
        i_MEM_READ      : in  std_logic;
    -- From TMR Voter   - used to determine next state   -
controller can modify this signal
        i_MEM_WRITE     : in  std_logic;
    -- From TMR Voter   - used to determine next state   -
controller can modify this signal
        i_MEM_ADDRESS   : in  std_logic_vector(31 downto 0);
  -- From TMR Voter   - used to determine next state   -
controller can modify this signal
        i_MEM_IN        : in  std_logic_vector(31 downto 0);
  -- From TMR Voter                                    -
controller can modify this signal
        i_MEM_READY     : in  std_logic;
    -- From Memory      - used to determine next state   -
controller can modify this signal
        i_MEM_OUT       : in  std_logic_vector(31 downto 0);
  -- From Memory                                       -
controller can modify this signal
        i_MEM_DONE      : in  std_logic;
    -- From Memory                                       -
controller can modify this signal
        i_MEM_READ0     : in  std_logic;
    -- From MIPS0      - used to determine next state   -
controller can modify this signal
        i_MEM_WRITE0    : in  std_logic;
    -- From MIPS0                                      -
controller can modify this signal
        i_MEM_ADDRESS0 : in  std_logic_vector(31 downto 0);
  -- From MIPS0      - used to determine next state   -
controller can modify this signal
        i_MEM_IN0       : in  std_logic_vector(31 downto 0);
  -- From MIPS0                                        -
controller can modify this signal
        i_MEM_READY0    : in  std_logic;
    -- From Voter                                      -
controller can modify this signal
        i_MEM_OUT0      : in  std_logic_vector(31 downto 0);
  -- From Voter                                        -
controller can modify this signal
```

```
        i_RESET0        : in   std_logic;
   -- From Voter                                          -
controller can modify this signal
        i_MEM_READY1    : in   std_logic;
   -- From Voter                                          -
controller can modify this signal
        i_MEM_OUT1      : in   std_logic_vector(31 downto 0);
  -- From Voter                                          -
controller can modify this signal
        i_RESET1        : in   std_logic;
   -- From Voter                                          -
controller can modify this signal
        i_MEM_READY2    : in   std_logic;
   -- From Voter                                          -
controller can modify this signal
        i_MEM_OUT2      : in   std_logic_vector(31 downto 0);
  -- From Voter                                          -
controller can modify this signal
        i_RESET2        : in   std_logic;
   -- From Voter                                          -
controller can modify this signal
        o_MEM_READ      : out  std_logic;
   -- To Memory
        o_MEM_WRITE     : out  std_logic;
   -- To Memory
        o_MEM_ADDRESS   : out  std_logic_vector(31 downto 0);
  -- To Memory
        o_MEM_IN        : out  std_logic_vector(31 downto 0);
  -- To Memory
        o_MEM_READY     : out  std_logic;
   -- To TMR Voter
        o_MEM_OUT       : out  std_logic_vector(31 downto 0);
  -- To TMR Voter
        o_MEM_DONE      : out  std_logic;
   -- To TMR Voter
        o_MEM_READY0    : out  std_logic;
   -- To MIPS0
        o_MEM_OUT0      : out  std_logic_vector(31 downto 0);
  -- To MIPS0
        o_RESET0        : out  std_logic;
   -- To MIPS0
        o_MEM_READY1    : out  std_logic;
   -- To MIPS1
```

```
        o_MEM_OUT1      : out std_logic_vector(31 downto 0);
    -- To MIPS1
        o_RESET1        : out std_logic;
     -- To MIPS1
        o_MEM_READY2    : out std_logic;
     -- To MIPS2
        o_MEM_OUT2      : out std_logic_vector(31 downto 0);
    -- To MIPS2
        o_RESET2        : out std_logic);
      -- To MIPS2
end AHR_Controller_v2_Test1001;

architecture a_AHR_Controller_v2_Test1001 of
   AHR_Controller_v2_Test1001 is
--| Declare components
   -- 2-input 1-bit mux
   component myMUX2_1 is
      port (i_0 : in  std_logic;
            i_1 : in  std_logic;
            i_S : in  std_logic;
            o_Z : out std_logic
            );
   end component;

   -- 2-input 32-bit mux
   component myMUX2_N is
      generic (m_width : integer := 32);
      port (i_0 : in  std_logic_vector(m_width-1 downto 0);
            i_1 : in  std_logic_vector(m_width-1 downto 0);
            i_S : in  std_logic;
            o_Z : out std_logic_vector(m_width-1 downto 0)
            );
   end component;
   -- 4-input 1-bit mux
   component myMUX4_1 is
      port (i_0 : in  std_logic;
            i_1 : in  std_logic;
            i_2 : in  std_logic;
            i_3 : in  std_logic;
            i_S : in  std_logic_vector(1 downto 0);
            o_Z : out std_logic
            );
   end component;
```

```
--| Create state machine types
    -- Create states for the controller finite state machine
    type sm_cfsm is (s_cfsm_0, s_cfsm_1, s_cfsm_2, s_cfsm_3,
    s_cfsm_4,
                     s_cfsm_5, s_cfsm_6, s_cfsm_7, s_cfsm_8,
    s_cfsm_9,
                     s_cfsm_10,s_cfsm_11,s_cfsm_12,s_cfsm_13,
    s_cfsm_14,
                     s_cfsm_15,s_cfsm_16,s_cfsm_17,s_cfsm_18,
    s_cfsm_19,
                     s_cfsm_20,s_cfsm_21,s_cfsm_22,s_cfsm_23,
    s_cfsm_24,
                     s_cfsm_25,s_cfsm_26,s_cfsm_27,s_cfsm_28,
    s_cfsm_29,
                     s_cfsm_30,s_cfsm_31,s_cfsm_32,s_cfsm_33,
    s_cfsm_34,
                     s_cfsm_35,s_cfsm_36,s_cfsm_37,s_cfsm_38,
    s_cfsm_39,
                     s_cfsm_40,s_cfsm_41,s_cfsm_42,s_cfsm_43,
    s_cfsm_44,
                     s_cfsm_45,s_cfsm_46,s_cfsm_47,s_cfsm_48,
    s_cfsm_49,
                     s_cfsm_50,s_cfsm_51,s_cfsm_52,s_cfsm_25a)
    ;

    -- Initialize the controller finite state machine register
    signal f_cfsm_state : sm_cfsm := s_cfsm_0;

--| Define Signals
    -- Counts instructions to determine when to transition
    from TMR to TSR
    signal f_instr_count : unsigned(31 downto 0) := (others =>
     '0');

    -- Signals to determine status of TSR error recovery
    progress
    signal f_err_flag : std_logic := '0';
    signal f_rec_flag : std_logic := '0';

    -- Used to store the current iteration of the program loop
     when switching from TMR to TSR or vice versa - modified
    as needed
    signal f_loop : unsigned(31 downto 0) := (others => '0');
    -- Used to store an unaltered copy of the current
    iteration of the program loop when switching from TMR to
    TSR or vice versa
    signal f_loop1 : unsigned(31 downto 0) := (others => '0');

    -- Signal used to determine last value of i_NEXT_INSTR
    signal
    signal f_NEXT_INSTR       : std_logic := '0';

    -- Registers for holding output values when temporary
    values are needed
    --- Outputs to Memory
    signal f_MEM_ADDRESS       : std_logic_vector(31 downto 0)
    := (others => '0');
    signal f_MEM_IN           : std_logic_vector(31 downto 0) :=
     (others => '0');
    -- Outputs to Voter
    signal f_MEM_OUT           : std_logic_vector(31 downto 0)
    := (others => '0');
    -- Outputs to MIPS Processors
    signal f_MEM_OUT0          : std_logic_vector(31 downto 0)
    := (others => '0');

    -- Intermediate address used for accessing recovery memory
    signal f_TEMP_ADDRESS    : std_logic_vector(31 downto 0);

    -- Wires for routing intermediate output values
    signal w_MEM_ADDRESS     : std_logic_vector(31 downto 0);
    signal w_MEM_IN          : std_logic_vector(31 downto 0);
    signal w_MEM_READY       : std_logic;
    signal w_MEM_OUT         : std_logic_vector(31 downto 0);
    signal w_MEM_DONE        : std_logic;
    signal w_MEM_OUT0        : std_logic_vector(31 downto 0);
    signal w_TMR_RESET0       : std_logic;
    signal w_TMR_RESET1       : std_logic;
    signal w_TMR_RESET2       : std_logic;
    signal w_TMR_RESET0a       : std_logic;
    signal w_TMR_RESET1a       : std_logic;
    signal w_TMR_RESET2a       : std_logic;
    signal w_MEM_RESET1      : std_logic;
    signal w_MEM_RESET2      : std_logic;

```

```
-- Registers for controlling flow of outputs
-- f_MEM_READ_SEL selects what read signal should be
output to memory
--- 00 - Voter Pass through
--- 01 - MIPS0 Pass Through
--- 10 - 0
--- 11 - 1
signal f_MEM_READ_SEL    : std_logic_vector(1 downto 0) :=
(others => '0');

-- f_MEM_WRITE_SEL selects what write and data signals
should be output to memory
--- 00 - Voter Pass through
--- 01 - MIPS0 Pass Through
--- 10 - 0
--- 11 - 1
signal f_MEM_WRITE_SEL    : std_logic_vector(1 downto 0) :=
 (others => '0');

-- f_MEM_ADDRESS_SEL selects what address signal should be
 output to memory
--- 00 - Voter Pass through
--- 01 - MIPS0 Pass Through
--- 10 - Controller Data Override
--- 11 - Controller Data Override
signal f_MEM_ADDRESS_SEL    : std_logic_vector(1 downto 0)
:= (others => '0');

-- f_MEM_READY_SEL selects what memory ready and data
signals should be output to the voter
--- 00 - Memory Pass Through
--- 01 - 0
--- 10 - 1
--- 11 - 1
signal f_MEM_READY_SEL :std_logic_vector(1 downto 0) := (
others => '0');

-- f_MEM_DONE_SEL selects what done signal should be
output to the voter
--- 00 - Memory Pass Through
--- 01 - 0
--- 10 - 1
--- 11 - 1
```

```
signal f_MEM_DONE_SEL :std_logic_vector(1 downto 0) := (
others => '0');

-- f_MEM_READY0_SEL selects what ready and data signals
should be output to MIPS0
--- 00 - Voter Pass Through
--- 01 - Memory Pass Through
--- 10 - 0
--- 11 - 1
signal f_MEM_READY0_SEL :std_logic_vector(1 downto 0) := (
others => '0');

-- f_MEM_READY12_SEL selects what ready and data signals
should be output to MIPS1 and MIPS2
--- 0 - Voter Pass Through
--- 1 - 0
signal f_MEM_READY12_SEL :std_logic := '0';

-- f_MEM_RESET0_SEL selects what resetignal should be
output to MIPS0
--- 00 - Voter Pass Through
--- 01 - Memory Pass Through
--- 10 - 0
--- 11 - 1
signal f_MEM_RESET0_SEL :std_logic_vector(1 downto 0) := (
others => '0');

-- f_MEM_RESET12_SEL selects what reset signal should be
output to MIPS1 and MIPS2
--- 00 - Voter Pass Through
--- 01 - 0
--- 10 - 1
--- 11 - 1
signal f_MEM_RESET12_SEL :std_logic_vector(1 downto 0) :=
(others => '0');


-- Define Constants
-- Constant used to compare instruction counter against
constant k_switch_point : unsigned(31 downto 0) := "
00000000000000000000000001000111"; --71
--"00000000000000000011101010011000"; --15,000
```

```
  -- Location at which backup memory starts
  constant k_mem_location : std_logic_vector(31 downto 0) := "00000000000000000000010111001000";
  -- Important backup memory locations
  constant k_14_16 : std_logic_vector(15 downto 0) := "0000000000111000";
  constant k_02_32 : std_logic_vector(31 downto 0) := "00000000000000000000000000001000";
  constant k_14_32 : std_logic_vector(31 downto 0) := "00000000000000000000000000111000";
  constant k_29_32 : std_logic_vector(31 downto 0) := "00000000000000000000000001110100";
  constant k_30_32 : std_logic_vector(31 downto 0) := "00000000000000000000000001111000";
  constant k_31_32 : std_logic_vector(31 downto 0) := "00000000000000000000000001111100";
  constant k_62_32 : std_logic_vector(31 downto 0) := "00000000000000000000000011111000";
  constant k_63_32 : std_logic_vector(31 downto 0) := "00000000000000000000000011111100";
  constant k_64_32 : std_logic_vector(31 downto 0) := "00000000000000000000000100000000";
  constant k_mem_location14_16 : std_logic_vector(15 downto 0) := std_logic_vector(unsigned(k_mem_location(15 downto 0)) + unsigned(k_14_16));
  constant k_mem_location02_32 : std_logic_vector(31 downto 0) := std_logic_vector(unsigned(k_mem_location) + unsigned(k_02_32));
  constant k_mem_location14_32 : std_logic_vector(31 downto 0) := std_logic_vector(unsigned(k_mem_location) + unsigned(k_14_32));
  constant k_mem_location29_32 : std_logic_vector(31 downto 0) := std_logic_vector(unsigned(k_mem_location) + unsigned(k_29_32));
  constant k_mem_location30_32 : std_logic_vector(31 downto 0) := std_logic_vector(unsigned(k_mem_location) + unsigned(k_30_32));
  constant k_mem_location31_32 : std_logic_vector(31 downto 0) := std_logic_vector(unsigned(k_mem_location) + unsigned(k_31_32));
  constant k_mem_location62_32 : std_logic_vector(31 downto 0) := std_logic_vector(unsigned(k_mem_location) + unsigned(k_62_32));
```

```
constant k_mem_location63_32 : std_logic_vector(31 downto
0) := std_logic_vector(unsigned(k_mem_location) + unsigned
(k_63_32));
constant k_mem_location64_32 : std_logic_vector(31 downto
0) := std_logic_vector(unsigned(k_mem_location) + unsigned
(k_64_32));

-- Additional constants
constant k_20_32 : std_logic_vector (31 downto 0) := "
00000000000000000000000000010100";


-- Location at which TMR loop starts
constant k_TMR_LOOP_START : std_logic_vector(31 downto 0)
:= "00000000000000000000000000001000";
-- Location at which TSR starts
constant k_TSR_START_BRANCH : std_logic_vector(15 downto
0) := "0000000000111010";
-- Location at which TSR loop starts
constant k_TSR_LOOP_START_BRANCH : std_logic_vector(15
downto 0) := "0000000000111100";
-- Location at which TSR Error Recovery Code Starts
constant k_TSR_RECOVERY : std_logic_vector(31 downto 0) :=
 "00000000000000000000010010011000";
-- Location at which TSR Save/Restore Creation Point Code
Starts
constant k_TSR_SAVE : std_logic_vector(31 downto 0) := "
00000000000000000000001011101100";
-- Location at which TSR Code Ends
constant k_TSR_END : std_logic_vector(31 downto 0) := "
00000000000000000000001011001000";

-- Zero constants
constant k_zero_1 : std_logic := '0';
constant k_zero_32 : std_logic_vector(31 downto 0) := (
others => '0');

-- One constant
constant k_one_1 : std_logic := '1';

-- Constant used to access the 64th memory location (0-30=
R1-R31, 31=PC, 32-62=R1-R31, 63=PC, 64=sav_point)
constant k_256_32 : std_logic_vector(31 downto 0) := "
```

```
    00000000000000000000000100000000";
    -- Constant used to access the 32nd memory location (0-30=
    R1-R31, 31=PC, 32-62=R1-R31, 63=PC, 64=sav_point)
    constant k_128_16 : std_logic_vector(15 downto 0) := "
    0000000010000000";
    -- Constant used to access the 31st memory location (0-30=
    R1-R31, 31=PC, 32-62=R1-R31, 63=PC, 64=sav_point)
    constant k_124_32 : std_logic_vector(31 downto 0) := "
    00000000000000000000000001111100";
    -- Constant used to access the 63rd memory location (0-30=
    R1-R31, 31=PC, 32-62=R1-R31, 63=PC, 64=sav_point)
    constant k_252_32 : std_logic_vector(31 downto 0) := "
    00000000000000000000000011111100";

begin
    w_TMR_RESET0 <= i_RESET0 or i_MEM_DONE; -- Reset MIPS0 on
    a Voter Reset0 signal or memory DONE signal
    w_TMR_RESET1 <= i_RESET1 or i_MEM_DONE; -- Reset MIPS1 on
    a Voter Reset1 signal or memory DONE signal
    w_TMR_RESET2 <= i_RESET2 or i_MEM_DONE; -- Reset MIPS2 on
    a Voter Reset2 signal or memory DONE signal

    -- Assign output reset signals
    o_RESET0 <= w_TMR_RESET0a or i_reset;
    o_RESET1 <= w_TMR_RESET1a or i_reset;
    o_RESET2 <= w_TMR_RESET2a or i_reset;

    -- MUX to determine the memory read signal to be sent to
    memory
    u_myMUX_MEM_READ: myMUX4_1
    port map (i_0 => i_MEM_READ,
              i_1 => i_MEM_READ0,
              i_2 => k_zero_1,
              i_3 => k_one_1,
              i_S => f_MEM_READ_SEL,
              o_Z => o_MEM_READ);

    -- MUX to determine the memory write signal to be sent to
    memory
    u_myMUX_MEM_WRITE: myMUX4_1
    port map (i_0 => i_MEM_WRITE,
              i_1 => i_MEM_WRITE0,
              i_2 => k_zero_1,
                i_3  => k_one_1,
                i_S  => f_MEM_WRITE_SEL,
                o_Z  => o_MEM_WRITE);

    -- MUX to determine the memory address signal to be sent
    to memory - selects between Voter or MIPS 0 - intermediate
     selector
    u_myMUX_MEM_ADDRESS_Intermediate: myMUX2_N
    generic map (m_width => 32)
    port map (i_0  => i_MEM_ADDRESS,
                i_1  => i_MEM_ADDRESS0,
                i_S  => f_MEM_ADDRESS_SEL(0),
                o_Z  => w_MEM_ADDRESS);

    -- MUX to determine the memory address signal to be sent
    to memory - selects between output of Voter or MIPS 0
    selector MUX and the controller override
    u_myMUX_MEM_ADDRESS: myMUX2_N
    generic map (m_width => 32)
    port map (i_0  => w_MEM_ADDRESS,
                i_1  => f_MEM_ADDRESS,
                i_S  => f_MEM_ADDRESS_SEL(1),
                o_Z  => o_MEM_ADDRESS);

    -- MUX to determine the memory data input signal to be
    sent to memory - selects between Voter or MIPS 0 -
    intermediate selector
    u_myMUX_MEM_IN_Intermediate: myMUX2_N
    generic map (m_width => 32)
    port map (i_0  => i_MEM_IN,
                i_1  => i_MEM_IN0,
                i_S  => f_MEM_WRITE_SEL(0),
                o_Z  => w_MEM_IN);

    -- MUX to determine the memory data input signal to be
    sent to memory - selects between output of Voter or MIPS 0
     selector MUX and the controller override
    u_myMUX_MEM_IN: myMUX2_N
    generic map (m_width => 32)
    port map (i_0  => w_MEM_IN,
                i_1  => f_MEM_IN,
                i_S  => f_MEM_WRITE_SEL(1),
                o_Z  => o_MEM_IN);

    -- MUX to determine the memory ready signal to be sent to the voter - selects between Memory or 0 - intermediate selector
    u_myMUX_MEM_READY_Intermediate: myMUX2_1
    port map (i_0 => i_MEM_READY,
              i_1 => k_zero_1,
              i_S => f_MEM_READY_SEL(0),
              o_Z => w_MEM_READY);

    -- MUX to determine the memory ready signal to be sent to the voter - selects between output of Memory or 0 selector MUX and 1
    u_myMUX_MEM_READY: myMUX2_1
    port map (i_0 => w_MEM_READY,
              i_1 => k_one_1,
              i_S => f_MEM_READY_SEL(1),
              o_Z => o_MEM_READY);

    -- MUX to determine the memory data signal to be sent to the voter - selects between Memory or 0 - intermediate selector
    u_myMUX_MEM_OUT_Intermediate: myMUX2_N
    generic map (m_width => 32)
    port map (i_0 => i_MEM_OUT,
              i_1 => k_zero_32,
              i_S => f_MEM_READY_SEL(0),
              o_Z => w_MEM_OUT);

    -- MUX to determine the memory data signal to be sent to the voter - selects between output of Memory or 0 selector MUX and the controller override
    u_myMUX_MEM_OUT: myMUX2_N
    generic map (m_width => 32)
    port map (i_0 => w_MEM_OUT,
              i_1 => f_MEM_OUT,
              i_S => f_MEM_READY_SEL(1),
              o_Z => o_MEM_OUT);

    -- MUX to determine the memory done signal to be sent to the voter - selects between Memory or 0 - intermediate selector
    u_myMUX_MEM_DONE_Intermediate: myMUX2_1
    port map (i_0 => i_MEM_DONE,
              i_1 => k_zero_1,
              i_S => f_MEM_DONE_SEL(0),
              o_Z => w_MEM_DONE);

    -- MUX to determine the memory done signal to be sent to
    the voter - selects between output of Memory or 0 selector
     MUX and 1
    u_myMUX_MEM_DONE: myMUX2_1
    port map (i_0 => w_MEM_DONE,
              i_1 => k_one_1,
              i_S => f_MEM_DONE_SEL(1),
              o_Z => o_MEM_DONE);

    -- MUX to determine the ready signal to be sent to MIPS0
    u_myMUX_MEM_READY0: myMUX4_1
    port map (i_0 => i_MEM_READY0,
              i_1 => i_MEM_READY,
              i_2 => k_zero_1,
              i_3 => k_one_1,
              i_S => f_MEM_READY0_SEL,
              o_Z => o_MEM_READY0);

    -- MUX to determine the ready signal to be sent to MIPS1
    u_myMUX_MEM_READY1: myMUX2_1
    port map (i_0 => i_MEM_READY1,
              i_1 => k_zero_1,
              i_S => f_MEM_READY12_SEL,
              o_Z => o_MEM_READY1);

    -- MUX to determine the ready signal to be sent to MIPS1
    u_myMUX_MEM_READY2: myMUX2_1
    port map (i_0 => i_MEM_READY2,
              i_1 => k_zero_1,
              i_S => f_MEM_READY12_SEL,
              o_Z => o_MEM_READY2);

    -- MUX to determine the memory data signal to be sent to
    MIPS0 - selects between Voter or Memory - intermediate
    selector
    u_myMUX_MEM_OUT0_Intermediate: myMUX2_N
    generic map (m_width => 32)
    port map (i_0 => i_MEM_OUT0,
                i_1 => i_MEM_OUT,
                i_S => f_MEM_READY0_SEL(0),
                o_Z => w_MEM_OUT0);

    -- MUX to determine the memory data signal to be sent to
    MIPS0 - selects between output of Voter or Memory selector
     MUX and the controller override
    u_myMUX_MEM_OUT0: myMUX2_N
    generic map (m_width => 32)
    port map (i_0 => w_MEM_OUT0,
                i_1 => f_MEM_OUT0,
                i_S => f_MEM_READY0_SEL(1),
                o_Z => o_MEM_OUT0);

    -- MUX to determine the memory data signal to be sent to
    MIPS1 - selects between output of Voter or 0
    u_myMUX_MEM_OUT1: myMUX2_N
    generic map (m_width => 32)
    port map (i_0 => i_MEM_OUT1,
                i_1 => k_zero_32,
                i_S => f_MEM_READY12_SEL,
                o_Z => o_MEM_OUT1);

    -- MUX to determine the memory data signal to be sent to
    MIPS2 - selects between output of Voter or 0
    u_myMUX_MEM_OUT2: myMUX2_N
    generic map (m_width => 32)
    port map (i_0 => i_MEM_OUT2,
                i_1 => k_zero_32,
                i_S => f_MEM_READY12_SEL,
                o_Z => o_MEM_OUT2);

    -- MUX to determine the reset signal to be sent to MIPS0
    u_myMUX_MEM_RESET0: myMUX4_1
    port map (i_0 => w_TMR_RESET0,
                i_1 => i_MEM_DONE,
                i_2 => k_zero_1,
                i_3 => k_one_1,
                i_S => f_MEM_RESET0_SEL,
                o_Z => w_TMR_RESET0a);

    -- MUX to determine the reset signal to be sent to MIPS1 -
     selects between output of Voter or 0
```

```
  u_myMUX_MEM_RESET1_Intermediate: myMUX2_1
  port map (i_0 => w_TMR_RESET1,
            i_1 => k_zero_1 ,
            i_S => f_MEM_RESET12_SEL(0) ,
            o_Z => w_MEM_RESET1);

  -- MUX to determine the reset signal to be sent to MIPS1 -
   selects between output of Voter or 0 selector MUX and 1
  u_myMUX_MEM_RESET1: myMUX2_1
  port map (i_0 => w_MEM_RESET1,
            i_1 => k_one_1 ,
            i_S => f_MEM_RESET12_SEL(1) ,
            o_Z => w_TMR_RESET1a);

  -- MUX to determine the reset signal to be sent to MIPS2 -
   selects between output of Voter or 0
  u_myMUX_MEM_RESET2_Intermediate: myMUX2_1
  port map (i_0 => w_TMR_RESET2,
            i_1 => k_zero_1 ,
            i_S => f_MEM_RESET12_SEL(0) ,
            o_Z => w_MEM_RESET2);

  -- MUX to determine the reset signal to be sent to MIPS2 -
   selects between output of Voter or 0 selector MUX and 1
  u_myMUX_MEM_RESET2: myMUX2_1
  port map (i_0 => w_MEM_RESET2,
            i_1 => k_one_1 ,
            i_S => f_MEM_RESET12_SEL(1) ,
            o_Z => w_TMR_RESET2a);


  cfsm: process(i_clk , i_reset , f_cfsm_state ,i_NEXT_INSTR,
 i_TMR_ERROR)
  begin
     if (i_reset = '1') then
        f_cfsm_state    <= s_cfsm_0;
        f_instr_count   <= (others => '0');
        f_loop          <= (others => '0');
        f_loop1         <= (others => '0');
        f_TEMP_ADDRESS  <= (others => '0');
        f_err_flag      <= '0';
        f_rec_flag      <= '0';
     elsif rising_edge(i_clk) then
```

```
      f_NEXT_INSTR <= i_NEXT_INSTR;
      case f_cfsm_state is
        -- TMR MIPS is running
        -- Determine when TMR MIPS completes processing an instruction or encounters an error
        when s_cfsm_0 =>
          if (i_TMR_ERROR = '1') then
            f_cfsm_state <= s_cfsm_2;
          elsif ((f_instr_count >= k_switch_point) and (i_MEM_READ = '1') and (i_MEM_ADDRESS = k_TMR_LOOP_START)) then
            f_cfsm_state <= s_cfsm_3;
          elsif ((i_NEXT_INSTR = '1') and (f_NEXT_INSTR = '0')) then
            f_cfsm_state <= s_cfsm_1;
          else
            f_cfsm_state <= s_cfsm_0;
          end if;
          f_instr_count <= f_instr_count;
          f_loop <= f_loop;
          f_loop1 <= f_loop1;
          f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= '0';
          f_rec_flag <= '0';

        -- Return to state 0
        when s_cfsm_1 =>
          f_cfsm_state <= s_cfsm_0;
          f_instr_count <= f_instr_count + 1;
          f_loop <= f_loop;
          f_loop1 <= f_loop1;
          f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= f_err_flag;
          f_rec_flag <= f_rec_flag;

        -- TMR MIPS has encountered an error
        when s_cfsm_2 =>
          if (i_TMR_ERROR = '0') then
            f_cfsm_state <= s_cfsm_0;
            f_instr_count <= (others => '0');
          else
            f_cfsm_state <= s_cfsm_2;
            f_instr_count <= f_instr_count + 1;
```

```
                    end if;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Start transition from TMR MIPS to TSR MIPS
                -- Wait for Memory Ready signal
                when s_cfsm_3 =>
                    if (i_MEM_READY = '1') then
                        f_cfsm_state <= s_cfsm_4;
                    else
                        f_cfsm_state <= s_cfsm_3;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_4 =>
                    if (i_MEM_READY = '0') then
                        f_cfsm_state <= s_cfsm_5;
                    else
                        f_cfsm_state <= s_cfsm_4;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for TMR Voter Read signal to return to 0
                when s_cfsm_5 =>
                    if (i_MEM_READ = '0') then
                        f_cfsm_state <= s_cfsm_6;
                    else
                        f_cfsm_state <= s_cfsm_5;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for TMR Voter Write signal and watch out
    for an Error
                when s_cfsm_6 =>
                    if (i_TMR_ERROR = '1') then
                        f_cfsm_state <= s_cfsm_2;
                    elsif (i_MEM_WRITE = '1') then
                        f_cfsm_state <= s_cfsm_7;
                    else
                        f_cfsm_state <= s_cfsm_6;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for TMR Voter Write signal to return to 0
                when s_cfsm_7 =>
                    if (i_MEM_WRITE = '0') then
                        f_cfsm_state <= s_cfsm_8;
                        f_loop <= f_loop;
                        f_loop1 <= f_loop1;
                    else
                        f_cfsm_state <= s_cfsm_7;
                        f_loop <= unsigned(i_MEM_IN);
                        f_loop1 <= unsigned(i_MEM_IN);
                    end if;
                    f_instr_count <= f_instr_count;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Add 1 to the loop count
                when s_cfsm_8 =>
                    f_cfsm_state <= s_cfsm_9;
```

```
            f_instr_count <= (others => '0');
            f_loop <= f_loop + 1;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready Signal
        when s_cfsm_9 =>
            if (i_MEM_READY = '1') then
               f_cfsm_state <= s_cfsm_10;
            else
               f_cfsm_state <= s_cfsm_9;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready signal to return to 0
        when s_cfsm_10 =>
            if (i_MEM_READY = '0') then
               f_cfsm_state <= s_cfsm_11;
            else
               f_cfsm_state <= s_cfsm_10;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready Signal
        when s_cfsm_11 =>
            if (i_MEM_READY = '1') then
               f_cfsm_state <= s_cfsm_12;
            else
               f_cfsm_state <= s_cfsm_11;
            end if;
            f_instr_count <= f_instr_count;
```

```
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_12 =>
                    if (i_MEM_READY = '0') then
                        f_cfsm_state <= s_cfsm_13;
                    else
                        f_cfsm_state <= s_cfsm_12;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for Memory Ready Signal
                when s_cfsm_13 =>
                    if (i_MEM_READY = '1') then
                        f_cfsm_state <= s_cfsm_14;
                    else
                        f_cfsm_state <= s_cfsm_13;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_14 =>
                    if (i_MEM_READY = '0') then
                        f_cfsm_state <= s_cfsm_15;
                    else
                        f_cfsm_state <= s_cfsm_14;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
```

```
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Reset MIPS Processors
        when s_cfsm_15 =>
            f_cfsm_state <= s_cfsm_16;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for MIPS0 Read Signal
        when s_cfsm_16 =>
            if (i_MEM_READ0 = '1') then
                f_cfsm_state <= s_cfsm_17;
            else
                f_cfsm_state <= s_cfsm_16;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for MIPS0 Read Signal to return to 0
        when s_cfsm_17 =>
            if (i_MEM_READ0 = '0') then
                f_cfsm_state <= s_cfsm_18;
            else
                f_cfsm_state <= s_cfsm_17;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;
        -- Wait for MIPS0 Read Signal
        when s_cfsm_18 =>
            if (i_MEM_READ0 = '1') then
                f_cfsm_state <= s_cfsm_19;
            else
                f_cfsm_state <= s_cfsm_18;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for MIPS0 Read Signal to return to 0
        when s_cfsm_19 =>
            if (i_MEM_READ0 = '0') then
                f_cfsm_state <= s_cfsm_20;
            else
                f_cfsm_state <= s_cfsm_19;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for MIPS0 Read Signal
        when s_cfsm_20 =>
            if (i_MEM_READ0 = '1') then
                f_cfsm_state <= s_cfsm_21;
            else
                f_cfsm_state <= s_cfsm_20;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for MIPS0 Read Signal to return to 0
```

```
            when s_cfsm_21 =>
               if (i_MEM_READ0 = '0') then
                  f_cfsm_state <= s_cfsm_22;
               else
                  f_cfsm_state <= s_cfsm_21;
               end if;
               f_instr_count <= f_instr_count;
               f_loop <= f_loop;
               f_loop1 <= f_loop1;
               f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
               f_err_flag <= f_err_flag;
               f_rec_flag <= f_rec_flag;

            -- Wait for MIPS0 Read Signal
            when s_cfsm_22 =>
               if (i_MEM_READ0 = '1') then
                  f_cfsm_state <= s_cfsm_23;
               else
                  f_cfsm_state <= s_cfsm_22;
               end if;
               f_instr_count <= f_instr_count;
               f_loop <= f_loop;
               f_loop1 <= f_loop1;
               f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
               f_err_flag <= f_err_flag;
               f_rec_flag <= f_rec_flag;

            -- Wait for MIPS0 Read signal to return to 0
            when s_cfsm_23 =>
               if (i_MEM_READ0 = '0') then
                  f_cfsm_state <= s_cfsm_24;
               else
                  f_cfsm_state <= s_cfsm_23;
               end if;
               f_instr_count <= f_instr_count;
               f_loop <= f_loop;
               f_loop1 <= f_loop1;
               f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
               f_err_flag <= f_err_flag;
               f_rec_flag <= f_rec_flag;

            -- Wait for MIPS0 Read signal
            when s_cfsm_24 =>
```

```
                    if (i_MEM_READ0 = '1') then
                        f_cfsm_state <= s_cfsm_25;
                    else
                        f_cfsm_state <= s_cfsm_24;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- TSR MIPS is Running
                -- Wait for an error to occur or the program to
    finish
                when s_cfsm_25 =>
                    if ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0 =
    k_TSR_RECOVERY) and (f_err_flag = '1')) then
                        f_cfsm_state <= s_cfsm_29;
                    elsif ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0
     = k_TSR_RECOVERY) and (f_err_flag = '0')) then
                        f_cfsm_state <= s_cfsm_25a;
                        f_err_flag <= '1';
                    elsif ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0
     = k_TSR_SAVE)) then
                        f_cfsm_state <= s_cfsm_25;
                        f_rec_flag <= '1';
                    elsif ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0
     <= k_TSR_END) and (f_rec_flag = '1')) then
                        f_cfsm_state <= s_cfsm_25;
                        f_rec_flag <= '0';
                        f_err_flag <= '0';
                    elsif (i_MEM_DONE = '1') then
                        f_cfsm_state <= s_cfsm_26;
                    else
                        f_cfsm_state <= s_cfsm_25;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;

            -- An error has occured in TSR MIPS
                when s_cfsm_25a =>
                    if ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0 >
    std_logic_vector(unsigned(k_TSR_RECOVERY) + unsigned(
    k_20_32)))) then
                        f_cfsm_state <= s_cfsm_25;
                    else
                        f_cfsm_state <= s_cfsm_25a;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Interrupt communications between MIPS0 and
    Memory.  Wait for MIPS0 Read Signal
                when s_cfsm_26 =>
                    if (i_MEM_READ0 = '1') then
                        f_cfsm_state <= s_cfsm_27;
                    else
                        f_cfsm_state <= s_cfsm_26;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Transmit branch instruction to branch to TSR
    Start.  Wait for MIPS0 Read Signal to return to 0
                when s_cfsm_27 =>
                    if (i_MEM_READ0 = '0') then
                        f_cfsm_state <= s_cfsm_28;
                    else
                        f_cfsm_state <= s_cfsm_27;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
```

```
          f_rec_flag <= f_rec_flag;

      -- Wait for MIPS0 Read Signal
      when s_cfsm_28 =>
          if (i_MEM_READ0 = '1') then
              f_cfsm_state <= s_cfsm_25;
          else
              f_cfsm_state <= s_cfsm_28;
          end if;
          f_instr_count <= f_instr_count;
          f_loop <= f_loop;
          f_loop1 <= f_loop1;
          f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= f_err_flag;
          f_rec_flag <= f_rec_flag;

      -- Start transition from TSR MIPS to TMR MIPS
      -- Wait for Memory Ready Signal
      when s_cfsm_29 =>
          if (i_MEM_READY = '1') then
              f_cfsm_state <= s_cfsm_30;
          else
              f_cfsm_state <= s_cfsm_29;
          end if;
          f_instr_count <= f_instr_count;
          f_loop <= f_loop;
          f_loop1 <= f_loop1;
          f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= '0';
          f_rec_flag <= '0';

      -- Wait for Memory Ready signal to return to 0
      when s_cfsm_30 =>
          if (i_MEM_READY = '0') then
              f_cfsm_state <= s_cfsm_31;
          else
              f_cfsm_state <= s_cfsm_30;
          end if;
          f_instr_count <= f_instr_count;
          f_loop <= f_loop;
          f_loop1 <= f_loop1;
          f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= f_err_flag;
```

```
               f_rec_flag <= f_rec_flag;

            -- Wait for Memory Ready Signal
            when s_cfsm_31 =>
               if (i_MEM_READY = '1') then
                  f_cfsm_state <= s_cfsm_32;
               else
                  f_cfsm_state <= s_cfsm_31;
               end if;
               f_instr_count <= f_instr_count;
               f_loop <= f_loop;
               f_loop1 <= f_loop1;
               f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
               f_err_flag <= f_err_flag;
               f_rec_flag <= f_rec_flag;

            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_32 =>
               if (i_MEM_READY = '0') then
                  f_cfsm_state <= s_cfsm_33;
               else
                  f_cfsm_state <= s_cfsm_32;
               end if;
               f_instr_count <= f_instr_count;
               f_loop <= f_loop;
               f_loop1 <= f_loop1;
               f_TEMP_ADDRESS <= i_MEM_OUT;
               f_err_flag <= f_err_flag;
               f_rec_flag <= f_rec_flag;

            -- Wait for Memory Ready Signal
            when s_cfsm_33 =>
               if (i_MEM_READY = '1') then
                  f_cfsm_state <= s_cfsm_34;
               else
                  f_cfsm_state <= s_cfsm_33;
               end if;
               f_instr_count <= f_instr_count;
               f_loop <= f_loop;
               f_loop1 <= f_loop1;
               f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
               f_err_flag <= f_err_flag;
               f_rec_flag <= f_rec_flag;
```

```vhdl
        -- Wait for Memory Ready signal to return to 0
        when s_cfsm_34 =>
            if (i_MEM_READY = '0') then
                f_cfsm_state <= s_cfsm_35;
--                f_loop <= f_loop;
--                f_loop1 <= f_loop1;
            else
                f_cfsm_state <= s_cfsm_34;
--                f_loop <= unsigned(i_MEM_OUT);
--                f_loop1 <= unsigned(i_MEM_OUT);
            end if;
                f_loop <= unsigned(i_MEM_OUT);
                f_loop1 <= unsigned(i_MEM_OUT);
            f_instr_count <= f_instr_count;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Subtract 1 from the loop count
        when s_cfsm_35 =>
            f_cfsm_state <= s_cfsm_36;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop -1;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready Signal
        when s_cfsm_36 =>
            if (i_MEM_READY = '1') then
                f_cfsm_state <= s_cfsm_37;
            else
                f_cfsm_state <= s_cfsm_36;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;
        -- Wait for Memory Ready signal to return to 0
        when s_cfsm_37 =>
            if (i_MEM_READY = '0') then
                f_cfsm_state <= s_cfsm_38;
            else
                f_cfsm_state <= s_cfsm_37;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready Signal
        when s_cfsm_38 =>
            if (i_MEM_READY = '1') then
                f_cfsm_state <= s_cfsm_39;
            else
                f_cfsm_state <= s_cfsm_38;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready signal to return to 0
        when s_cfsm_39 =>
            if (i_MEM_READY = '0') then
                f_cfsm_state <= s_cfsm_40;
            else
                f_cfsm_state <= s_cfsm_39;
            end if;
            f_instr_count <= f_instr_count;
            f_loop <= f_loop;
            f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;

        -- Wait for Memory Ready Signal
```

```
            when s_cfsm_40 =>
                if (i_MEM_READY = '1') then
                    f_cfsm_state <= s_cfsm_41;
                else
                    f_cfsm_state <= s_cfsm_40;
                end if;
                f_instr_count <= f_instr_count;
                f_loop <= f_loop;
                f_loop1 <= f_loop1;
                f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                f_err_flag <= f_err_flag;
                f_rec_flag <= f_rec_flag;

            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_41 =>
                if (i_MEM_READY = '0') then
                    f_cfsm_state <= s_cfsm_42;
                else
                    f_cfsm_state <= s_cfsm_41;
                end if;
                f_instr_count <= f_instr_count;
                f_loop <= f_loop;
                f_loop1 <= f_loop1;
                f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                f_err_flag <= f_err_flag;
                f_rec_flag <= f_rec_flag;

            -- Wait for Memory Ready Signal
            when s_cfsm_42 =>
                if (i_MEM_READY = '1') then
                    f_cfsm_state <= s_cfsm_43;
                else
                    f_cfsm_state <= s_cfsm_42;
                end if;
                f_instr_count <= f_instr_count;
                f_loop <= f_loop;
                f_loop1 <= f_loop1;
                f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                f_err_flag <= f_err_flag;
                f_rec_flag <= f_rec_flag;

            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_43 =>
```

```
        if (i_MEM_READY = '0') then
          f_cfsm_state <= s_cfsm_44;
        else
          f_cfsm_state <= s_cfsm_43;
        end if;
        f_instr_count <= f_instr_count;
        f_loop <= f_loop;
        f_loop1 <= f_loop1;
        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
        f_rec_flag <= f_rec_flag;

      -- Wait for Memory Ready Signal
      when s_cfsm_44 =>
        if (i_MEM_READY = '1') then
          f_cfsm_state <= s_cfsm_45;
        else
          f_cfsm_state <= s_cfsm_44;
        end if;
        f_instr_count <= f_instr_count;
        f_loop <= f_loop;
        f_loop1 <= f_loop1;
        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
        f_rec_flag <= f_rec_flag;

      -- Wait for Memory Ready signal to return to 0
      when s_cfsm_45 =>
        if (i_MEM_READY = '0') then
          f_cfsm_state <= s_cfsm_46;
        else
          f_cfsm_state <= s_cfsm_45;
        end if;
        f_instr_count <= f_instr_count;
        f_loop <= f_loop;
        f_loop1 <= f_loop1;
        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
        f_rec_flag <= f_rec_flag;

      -- Wait for Voter Read Signal
      when s_cfsm_46 =>
        if (i_MEM_READ = '1') then
```

```
          f_cfsm_state <= s_cfsm_47;
       else
          f_cfsm_state <= s_cfsm_46;
       end if;
       f_instr_count <= f_instr_count;
       f_loop <= f_loop;
       f_loop1 <= f_loop1;
       f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
       f_rec_flag <= f_rec_flag;

    -- Wait for Voter Read Signal to return to 0
    when s_cfsm_47 =>
       if (i_MEM_READ = '0') then
          f_cfsm_state <= s_cfsm_48;
       else
          f_cfsm_state <= s_cfsm_47;
       end if;
       f_instr_count <= f_instr_count;
       f_loop <= f_loop;
       f_loop1 <= f_loop1;
       f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
       f_rec_flag <= f_rec_flag;

    -- Wait for Voter Read Signal
    when s_cfsm_48 =>
       if (i_MEM_READ = '1') then
          f_cfsm_state <= s_cfsm_49;
       else
          f_cfsm_state <= s_cfsm_48;
       end if;
       f_instr_count <= f_instr_count;
       f_loop <= f_loop;
       f_loop1 <= f_loop1;
       f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
       f_rec_flag <= f_rec_flag;

    -- Wait for Voter Read Signal to return to 0
    when s_cfsm_49 =>
       if (i_MEM_READ = '0') then
          f_cfsm_state <= s_cfsm_50;
```

```
                    else
                        f_cfsm_state <= s_cfsm_49;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for Voter Read Signal
                when s_cfsm_50 =>
                    if (i_MEM_READ = '1') then
                        f_cfsm_state <= s_cfsm_51;
                    else
                        f_cfsm_state <= s_cfsm_50;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Wait for Voter Read Signal to return to 0
                when s_cfsm_51 =>
                    if (i_MEM_READ = '0') then
                        f_cfsm_state <= s_cfsm_52;
                    else
                        f_cfsm_state <= s_cfsm_51;
                    end if;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- Return to normal TMR operation
                when s_cfsm_52 =>
                    f_cfsm_state <= s_cfsm_0;
                    f_instr_count <= f_instr_count;
                    f_loop <= f_loop;
                    f_loop1 <= f_loop1;
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;

                -- This should never happen
                when others =>
                    f_cfsm_state <= s_cfsm_0;
                    f_instr_count <= (others => '0');
                    f_loop <= (others => '0');
                    f_loop1 <= (others => '0');
                    f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                    f_err_flag <= f_err_flag;
                    f_rec_flag <= f_rec_flag;
            end case;
        end if;
    end process cfsm;



    controller_output_fsm: process(i_clk, i_reset,
    f_cfsm_state)
    begin
        if (i_reset = '1') then
            f_MEM_ADDRESS <= (others => '0');
            f_MEM_IN <= (others => '0');
            f_MEM_OUT <= (others => '0');
            f_MEM_OUT0 <= (others => '0');
            f_MEM_READ_SEL <= (others => '0');
            f_MEM_WRITE_SEL <= (others => '0');
            f_MEM_ADDRESS_SEL <= (others => '0');
            f_MEM_READY_SEL <= (others => '0');
            f_MEM_DONE_SEL <= (others => '0');
            f_MEM_READY0_SEL <= (others => '0');
            f_MEM_READY12_SEL <= '0';
            f_MEM_RESET0_SEL <= (others => '0');
            f_MEM_RESET12_SEL <= (others => '0');
        elsif rising_edge(i_clk) then
            case f_cfsm_state is
                -- TMR MIPS is running
                -- Determine when TMR MIPS completes processing
    an instruction or encounters an error
                when s_cfsm_0 =>
```

```
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "00";
                f_MEM_WRITE_SEL <= "00";
                f_MEM_ADDRESS_SEL <= "00";
                f_MEM_READY_SEL <= "00";
                f_MEM_DONE_SEL <= "00";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "00";
                f_MEM_RESET12_SEL <= "00";

            -- Return to state 0
            when s_cfsm_1 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "00";
                f_MEM_WRITE_SEL <= "00";
                f_MEM_ADDRESS_SEL <= "00";
                f_MEM_READY_SEL <= "00";
                f_MEM_DONE_SEL <= "00";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "00";
                f_MEM_RESET12_SEL <= "00";

            -- TMR MIPS has encountered an error
            when s_cfsm_2 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "00";
                f_MEM_WRITE_SEL <= "00";
                f_MEM_ADDRESS_SEL <= "00";
                f_MEM_READY_SEL <= "00";
                f_MEM_DONE_SEL <= "00";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
```

```
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";

                -- Start transition from TMR MIPS to TSR MIPS
                -- Wait for Memory Ready signal
                when s_cfsm_3 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "00";
                    f_MEM_WRITE_SEL <= "00";
                    f_MEM_ADDRESS_SEL <= "00";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "01";
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";


                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_4 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= "1010110000011111" &
    k_mem_location14_16;
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "11";
                    f_MEM_DONE_SEL <= "01";
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";


                -- Wait for TMR Voter Read signal to return to 0
                when s_cfsm_5 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= "1010110000011111" &
    k_mem_location14_16;
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "01";
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";


                -- Wait for TMR Voter Write signal and watch out
    for an Error
                when s_cfsm_6 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= "1010110000011111" &
    k_mem_location14_16;
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "01";
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";


                -- Wait for TMR Voter Write signal to return to 0
                when s_cfsm_7 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "10";
```

```
        f_MEM_DONE_SEL <= "01";
        f_MEM_READY0_SEL <= "00";
        f_MEM_READY12_SEL <= '0';
        f_MEM_RESET0_SEL <= "00";
        f_MEM_RESET12_SEL <= "00";


    -- Add 1 to the loop count
    when s_cfsm_8 =>
        f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
        f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= (others => '0');
        f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
        f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
        f_MEM_DONE_SEL <= "01";
        f_MEM_READY0_SEL <= "00";
        f_MEM_READY12_SEL <= '0';
        f_MEM_RESET0_SEL <= "00";
        f_MEM_RESET12_SEL <= "00";


    -- Wait for Memory Ready Signal
    when s_cfsm_9 =>
        f_MEM_ADDRESS <= k_mem_location14_32;
        f_MEM_IN <= std_logic_vector(f_loop);
        f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= (others => '0');
        f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "11";
        f_MEM_ADDRESS_SEL <= "11";
        f_MEM_READY_SEL <= "01";
        f_MEM_DONE_SEL <= "01";
        f_MEM_READY0_SEL <= "00";
        f_MEM_READY12_SEL <= '0';
        f_MEM_RESET0_SEL <= "00";
        f_MEM_RESET12_SEL <= "00";


    -- Wait for Memory Ready signal to return to 0
    when s_cfsm_10 =>
```

```
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "00";
                f_MEM_RESET12_SEL <= "00";


            -- Wait for Memory Ready Signal
            when s_cfsm_11 =>
                f_MEM_ADDRESS <= k_mem_location29_32;
                f_MEM_IN <= std_logic_vector(f_loop);
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "11";
                f_MEM_ADDRESS_SEL <= "11";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "00";
                f_MEM_RESET12_SEL <= "00";


            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_12 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "01";
```

```
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";


                -- Wait for Memory Ready Signal
                when s_cfsm_13 =>
                    f_MEM_ADDRESS <= k_mem_location30_32;
                    f_MEM_IN <= k_zero_32;
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "11";
                    f_MEM_ADDRESS_SEL <= "11";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "01";
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";


                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_14 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "01";
                    f_MEM_READY0_SEL <= "00";
                    f_MEM_READY12_SEL <= '0';
                    f_MEM_RESET0_SEL <= "00";
                    f_MEM_RESET12_SEL <= "00";

                -- Reset all processors
                when s_cfsm_15 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "11";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal
                when s_cfsm_16 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal to return to 0
                when s_cfsm_17 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= "
        10001100000111110000000000000000";
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "11";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
```

```
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal
                when s_cfsm_18 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= "
    10001100000111110000000000000000";
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal to return to 0
                when s_cfsm_19 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= std_logic_vector(f_loop1);
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "11";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal
                when s_cfsm_20 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= std_logic_vector(f_loop1);
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal to return to 0
                when s_cfsm_21 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= "
    00100011111111100000000000000000";
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "11";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";


                -- Wait for MIPS0 Read Signal
                when s_cfsm_22 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= "
    00100011111111100000000000000000";
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

```

```
            -- Wait for MIPS0 Read signal to return to 0
            when s_cfsm_23 =>
               f_MEM_ADDRESS <= (others => '0');
               f_MEM_IN <= (others => '0');
               f_MEM_OUT <= (others => '0');
               f_MEM_OUT0 <= "0001000000000000" &
   k_TSR_LOOP_START_BRANCH;
               f_MEM_READ_SEL <= "10";
               f_MEM_WRITE_SEL <= "10";
               f_MEM_ADDRESS_SEL <= "10";
               f_MEM_READY_SEL <= "01";
               f_MEM_DONE_SEL <= "11";
               f_MEM_READY0_SEL <= "11";
               f_MEM_READY12_SEL <= '1';
               f_MEM_RESET0_SEL <= "10";
               f_MEM_RESET12_SEL <= "11";


            -- Wait for MIPS0 Read signal
            when s_cfsm_24 =>
               f_MEM_ADDRESS <= (others => '0');
               f_MEM_IN <= (others => '0');
               f_MEM_OUT <= (others => '0');
               f_MEM_OUT0 <= "0001000000000000" &
   k_TSR_LOOP_START_BRANCH;
               f_MEM_READ_SEL <= "10";
               f_MEM_WRITE_SEL <= "10";
               f_MEM_ADDRESS_SEL <= "10";
               f_MEM_READY_SEL <= "01";
               f_MEM_DONE_SEL <= "11";
               f_MEM_READY0_SEL <= "10";
               f_MEM_READY12_SEL <= '1';
               f_MEM_RESET0_SEL <= "10";
               f_MEM_RESET12_SEL <= "11";


            -- TSR MIPS is Running
            -- Wait for an error to occur or the program to
   finish
            when s_cfsm_25 =>
               f_MEM_ADDRESS <= (others => '0');
               f_MEM_IN <= (others => '0');
```

```
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "01";
                    f_MEM_WRITE_SEL <= "01";
                    f_MEM_ADDRESS_SEL <= "01";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "01";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "01";
                    f_MEM_RESET12_SEL <= "11";

          -- An error has occured in TSR MIPS
          when s_cfsm_25a =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "01";
                    f_MEM_WRITE_SEL <= "01";
                    f_MEM_ADDRESS_SEL <= "01";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "01";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "01";
                    f_MEM_RESET12_SEL <= "11";

                -- Interrupt communications between MIPS0 and
    Memory.     Wait for MIPS0 Read Signal
                when s_cfsm_26 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

                -- Transmit branch instruction to branch to TSR
      Start.  Wait for MIPS0 Read Signal to return to 0
                when s_cfsm_27 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= "0001000000000000" &
    k_TSR_START_BRANCH;
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "11";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for MIPS0 Read Signal
                when s_cfsm_28 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= "0001000000000000" &
    k_TSR_START_BRANCH;
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "10";
                    f_MEM_RESET12_SEL <= "11";


                -- Start transition from TSR MIPS to TMR MIPS
                -- Wait for Memory Ready Signal
                when s_cfsm_29 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
```

```
          f_MEM_OUT <= (others => '0');
          f_MEM_OUT0 <= (others => '0');
          f_MEM_READ_SEL <= "10";
          f_MEM_WRITE_SEL <= "10";
          f_MEM_ADDRESS_SEL <= "10";
          f_MEM_READY_SEL <= "01";
          f_MEM_DONE_SEL <= "11";
          f_MEM_READY0_SEL <= "10";
          f_MEM_READY12_SEL <= '1';
          f_MEM_RESET0_SEL <= "11";
          f_MEM_RESET12_SEL <= "11";


      -- Wait for Memory Ready signal to return to 0
      when s_cfsm_30 =>
          f_MEM_ADDRESS <= (others => '0');
          f_MEM_IN <= (others => '0');
          f_MEM_OUT <= (others => '0');
          f_MEM_OUT0 <= (others => '0');
          f_MEM_READ_SEL <= "10";
          f_MEM_WRITE_SEL <= "10";
          f_MEM_ADDRESS_SEL <= "10";
          f_MEM_READY_SEL <= "01";
          f_MEM_DONE_SEL <= "11";
          f_MEM_READY0_SEL <= "10";
          f_MEM_READY12_SEL <= '1';
          f_MEM_RESET0_SEL <= "11";
          f_MEM_RESET12_SEL <= "11";

      -- Wait for Memory Ready Signal
      when s_cfsm_31 =>
          f_MEM_ADDRESS <= k_mem_location30_32;
          f_MEM_IN <= (others => '0');
          f_MEM_OUT <= (others => '0');
          f_MEM_OUT0 <= (others => '0');
          f_MEM_READ_SEL <= "11";
          f_MEM_WRITE_SEL <= "10";
          f_MEM_ADDRESS_SEL <= "11";
          f_MEM_READY_SEL <= "01";
          f_MEM_DONE_SEL <= "11";
          f_MEM_READY0_SEL <= "10";
          f_MEM_READY12_SEL <= '1';
          f_MEM_RESET0_SEL <= "11";
```

```
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_32 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "11";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for Memory Ready Signal
                when s_cfsm_33 =>
                    f_MEM_ADDRESS <= std_logic_vector(unsigned(
    k_mem_location14_32)+unsigned(f_TEMP_ADDRESS));
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "11";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "11";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "11";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for Memory Ready signal to return to 0
                when s_cfsm_34 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "11";
                    f_MEM_RESET12_SEL <= "11";


                -- Subtract 1 from the loop count
                when s_cfsm_35 =>
                    f_MEM_ADDRESS <= (others => '0');
                    f_MEM_IN <= (others => '0');
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "10";
                    f_MEM_ADDRESS_SEL <= "10";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "11";
                    f_MEM_RESET12_SEL <= "11";


                -- Wait for Memory Ready Signal
                when s_cfsm_36 =>
                    f_MEM_ADDRESS <= k_mem_location30_32;
                    f_MEM_IN <= std_logic_vector(f_loop);
                    f_MEM_OUT <= (others => '0');
                    f_MEM_OUT0 <= (others => '0');
                    f_MEM_READ_SEL <= "10";
                    f_MEM_WRITE_SEL <= "11";
                    f_MEM_ADDRESS_SEL <= "11";
                    f_MEM_READY_SEL <= "01";
                    f_MEM_DONE_SEL <= "11";
                    f_MEM_READY0_SEL <= "10";
                    f_MEM_READY12_SEL <= '1';
                    f_MEM_RESET0_SEL <= "11";
                    f_MEM_RESET12_SEL <= "11";

                -- Wait for Memory Ready signal to return to 0
```

```
        when s_cfsm_37 =>
            f_MEM_ADDRESS <= (others => '0');
            f_MEM_IN <= (others => '0');
            f_MEM_OUT <= (others => '0');
            f_MEM_OUT0 <= (others => '0');
            f_MEM_READ_SEL <= "10";
            f_MEM_WRITE_SEL <= "10";
            f_MEM_ADDRESS_SEL <= "10";
            f_MEM_READY_SEL <= "01";
            f_MEM_DONE_SEL <= "11";
            f_MEM_READY0_SEL <= "10";
            f_MEM_READY12_SEL <= '1';
            f_MEM_RESET0_SEL <= "11";
            f_MEM_RESET12_SEL <= "11";

        -- Wait for Memory Ready Signal
        when s_cfsm_38 =>
            f_MEM_ADDRESS <= k_mem_location62_32;
            f_MEM_IN <= std_logic_vector(f_loop);
            f_MEM_OUT <= (others => '0');
            f_MEM_OUT0 <= (others => '0');
            f_MEM_READ_SEL <= "10";
            f_MEM_WRITE_SEL <= "11";
            f_MEM_ADDRESS_SEL <= "11";
            f_MEM_READY_SEL <= "01";
            f_MEM_DONE_SEL <= "11";
            f_MEM_READY0_SEL <= "10";
            f_MEM_READY12_SEL <= '1';
            f_MEM_RESET0_SEL <= "11";
            f_MEM_RESET12_SEL <= "11";

        -- Wait for Memory Ready signal to return to 0
        when s_cfsm_39 =>
            f_MEM_ADDRESS <= (others => '0');
            f_MEM_IN <= (others => '0');
            f_MEM_OUT <= (others => '0');
            f_MEM_OUT0 <= (others => '0');
            f_MEM_READ_SEL <= "10";
            f_MEM_WRITE_SEL <= "10";
            f_MEM_ADDRESS_SEL <= "10";
            f_MEM_READY_SEL <= "01";
            f_MEM_DONE_SEL <= "11";
            f_MEM_READY0_SEL <= "10";
```

```
                f_MEM_READY12_SEL <= '1';
                f_MEM_RESET0_SEL <= "11";
                f_MEM_RESET12_SEL <= "11";

            -- Wait for Memory Ready Signal
            when s_cfsm_40 =>
                f_MEM_ADDRESS <= k_mem_location31_32;
                f_MEM_IN <= k_mem_location02_32;
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "11";
                f_MEM_ADDRESS_SEL <= "11";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "11";
                f_MEM_READY0_SEL <= "10";
                f_MEM_READY12_SEL <= '1';
                f_MEM_RESET0_SEL <= "11";
                f_MEM_RESET12_SEL <= "11";

            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_41 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "11";
                f_MEM_READY0_SEL <= "10";
                f_MEM_READY12_SEL <= '1';
                f_MEM_RESET0_SEL <= "11";
                f_MEM_RESET12_SEL <= "11";

            -- Wait for Memory Ready Signal
            when s_cfsm_42 =>
                f_MEM_ADDRESS <= k_mem_location63_32;
                f_MEM_IN <= k_mem_location02_32;
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "11";
                f_MEM_ADDRESS_SEL <= "11";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "11";
                f_MEM_READY0_SEL <= "10";
                f_MEM_READY12_SEL <= '1';
                f_MEM_RESET0_SEL <= "11";
                f_MEM_RESET12_SEL <= "11";

            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_43 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "11";
                f_MEM_READY0_SEL <= "10";
                f_MEM_READY12_SEL <= '1';
                f_MEM_RESET0_SEL <= "11";
                f_MEM_RESET12_SEL <= "11";

            -- Wait for Memory Ready Signal
            when s_cfsm_44 =>
                f_MEM_ADDRESS <= k_mem_location64_32;
                f_MEM_IN <= k_zero_32;
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "11";
                f_MEM_ADDRESS_SEL <= "11";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "11";
                f_MEM_READY0_SEL <= "10";
                f_MEM_READY12_SEL <= '1';
                f_MEM_RESET0_SEL <= "11";
                f_MEM_RESET12_SEL <= "11";

            -- Wait for Memory Ready signal to return to 0
            when s_cfsm_45 =>
```

```
               f_MEM_ADDRESS <= (others => '0');
               f_MEM_IN <= (others => '0');
               f_MEM_OUT <= (others => '0');
               f_MEM_OUT0 <= (others => '0');
               f_MEM_READ_SEL <= "10";
               f_MEM_WRITE_SEL <= "10";
               f_MEM_ADDRESS_SEL <= "10";
               f_MEM_READY_SEL <= "01";
               f_MEM_DONE_SEL <= "11";
               f_MEM_READY0_SEL <= "10";
               f_MEM_READY12_SEL <= '1';
               f_MEM_RESET0_SEL <= "11";
               f_MEM_RESET12_SEL <= "11";

            -- Wait for Voter Read Signal
            when s_cfsm_46 =>
               f_MEM_ADDRESS <= (others => '0');
               f_MEM_IN <= (others => '0');
               f_MEM_OUT <= (others => '0');
               f_MEM_OUT0 <= (others => '0');
               f_MEM_READ_SEL <= "10";
               f_MEM_WRITE_SEL <= "10";
               f_MEM_ADDRESS_SEL <= "10";
               f_MEM_READY_SEL <= "01";
               f_MEM_DONE_SEL <= "01";
               f_MEM_READY0_SEL <= "00";
               f_MEM_READY12_SEL <= '0';
               f_MEM_RESET0_SEL <= "10";
               f_MEM_RESET12_SEL <= "01";

            -- Wait for Voter Read Signal to return to 0
            when s_cfsm_47 =>
               f_MEM_ADDRESS <= (others => '0');
               f_MEM_IN <= (others => '0');
               f_MEM_OUT <= "10001100000111110000000000000000
   ";
               f_MEM_OUT0 <= (others => '0');
               f_MEM_READ_SEL <= "10";
               f_MEM_WRITE_SEL <= "10";
               f_MEM_ADDRESS_SEL <= "10";
               f_MEM_READY_SEL <= "11";
               f_MEM_DONE_SEL <= "01";
               f_MEM_READY0_SEL <= "00";
```

```
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "10";
                f_MEM_RESET12_SEL <= "01";


            -- Wait for Voter Read Signal
            when s_cfsm_48 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "10";
                f_MEM_RESET12_SEL <= "01";

            -- Wait for Voter Read Signal to return to 0
            when s_cfsm_49 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= std_logic_vector(f_loop);
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "11";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "10";
                f_MEM_RESET12_SEL <= "01";

            -- Wait for Voter Read Signal
            when s_cfsm_50 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "10";
                f_MEM_RESET12_SEL <= "01";

            -- Wait for Voter Read Signal to return to 0
            when s_cfsm_51 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= k_zero_32;
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "11";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "10";
                f_MEM_RESET12_SEL <= "01";

            -- Return to normal TMR operation
            when s_cfsm_52 =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "10";
                f_MEM_WRITE_SEL <= "10";
                f_MEM_ADDRESS_SEL <= "10";
                f_MEM_READY_SEL <= "01";
                f_MEM_DONE_SEL <= "01";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "10";
                f_MEM_RESET12_SEL <= "01";

            -- This should never happen
            when others =>
                f_MEM_ADDRESS <= (others => '0');
                f_MEM_IN <= (others => '0');
                f_MEM_OUT <= (others => '0');
                f_MEM_OUT0 <= (others => '0');
                f_MEM_READ_SEL <= "00";
                f_MEM_WRITE_SEL <= "00";
                f_MEM_ADDRESS_SEL <= "00";
                f_MEM_READY_SEL <= "00";
                f_MEM_DONE_SEL <= "00";
                f_MEM_READY0_SEL <= "00";
                f_MEM_READY12_SEL <= '0';
                f_MEM_RESET0_SEL <= "00";
                f_MEM_RESET12_SEL <= "00";
        end case;
    end if;
  end process controller_output_fsm;

end a_AHR_Controller_v2_Test1001;
```

Listing A.1. AHR_Controller_v2.vhd Code

Appendix B. Version History

- Version 2.2
  - Converted document to AFIT Report Format
  - Renamed “Combined MIPS” to “Adaptive-Hybrid Redundancy (AHR) MIPS”
- Version 2.1
  - Added hyperlinks to Table of Contents and figure and table references to simplify document navigation
- Version 2.0
  - Previous version transitioned from TSR MIPS to TMR MIPS after a single error. This version allows TSR MIPS an opportunity to recover from the error. If a second error occurs before TSR MIPS creates a new save/restore point, TSR MIPS transitions to TMR MIPS. If TSR MIPS successfully creates a new save/restore point, TSR MIPS continues processing as if no error had occurred
- Version 1.0
  - Original Document.

Bibliography

1. N. S. Hamilton, “Basic MIPS Architecture Version 1.4,” Jul 2019.

2. ——, “Triple Modular Redundancy MIPS Architecture Version 1.4,” Jul 2019.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 12–09–2019	**2. REPORT TYPE** Technical Report	**3. DATES COVERED** *(From — To)* Sept 2016 — Sept 2019

4. TITLE AND SUBTITLE Adaptive-Hybrid Redundancy MIPS Architecture Version 2.2	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Hamilton, Nicolas S, Maj, USAF	**5d. PROJECT NUMBER** 18G169C
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering an Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/EN/TR-19-04
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Undisclosed	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes in detail the architecture of an Adaptive-Hybrid Redundancy (AHR) MIPS processor based upon the Basic MIPS processor [1] and Triple Modular Redundancy (TMR) processor [2]. The AHR MIPS processor is the result of Adaptive-Hybrid Redundancy for Radiation-Hardening research and combines TMR and Temporal Software Redundancy. The AHR MIPS processor is hybrid in that it utilizes both hardware and software redundancy. It is adaptive because it has the capability to switch between TMR and TSR modes. There may be many other applications for the AHR MIPS processor beyond this specific research area.

15. SUBJECT TERMS

MIPS, Processor, Adaptive-Hybrid Redundancy

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Maj Nicolas Hamilton, AFIT/ENG
U	U	U	UU	119	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-6565 x4220; nicolas.hamilton@afit.edu