



**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**

DISTRIBUTION STATEMENT A  
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Disclaimer: The views expressed in this paper are those of the authors, and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U. S. Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. This document has been approved for public release; distribution unlimited, case number 88ABW-2019-5336

## **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



## 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 processed 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**

<b>Name</b>	<b>Type</b>	<b>Width</b>	<b>Description</b>
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*

<b>Name</b>	<b>Type</b>	<b>Width</b>	<b>Description</b>
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*

<b>Name</b>	<b>Type</b>	<b>Width</b>	<b>Description</b>
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**

<b>Name</b>	<b>Description</b>
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 signal 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 $\neq$ k_TSR_END and f_rec_flag = 1	Wait for MIPS0 read signal, MIPS0 to read an instruction from the TSR instruction set, and the 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 occurred 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*

<b>Current State</b>	<b>Outputs</b>
	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.

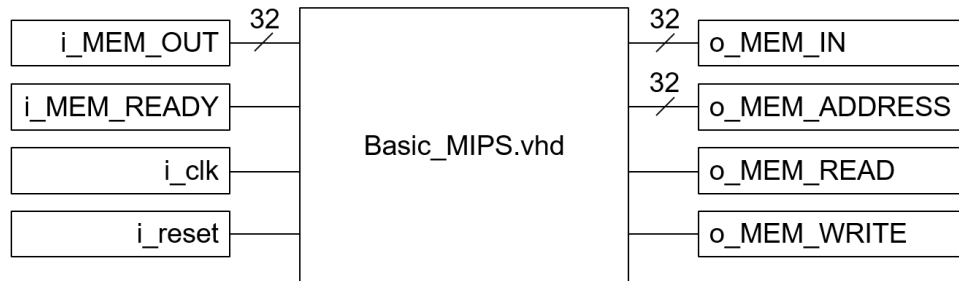
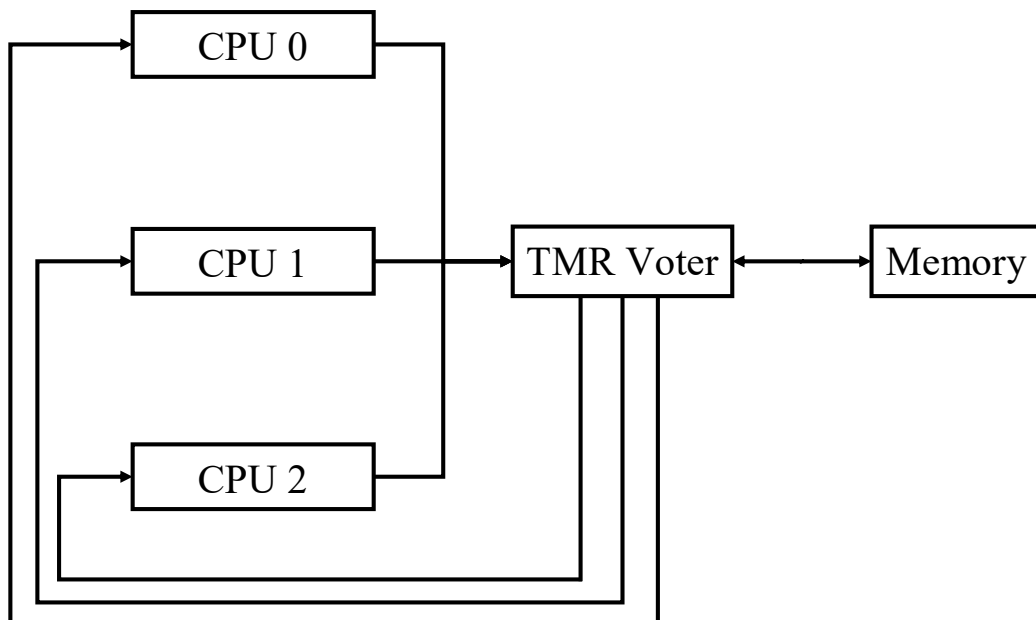


Figure 1. Basic MIPS Inputs and Outputs

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`).

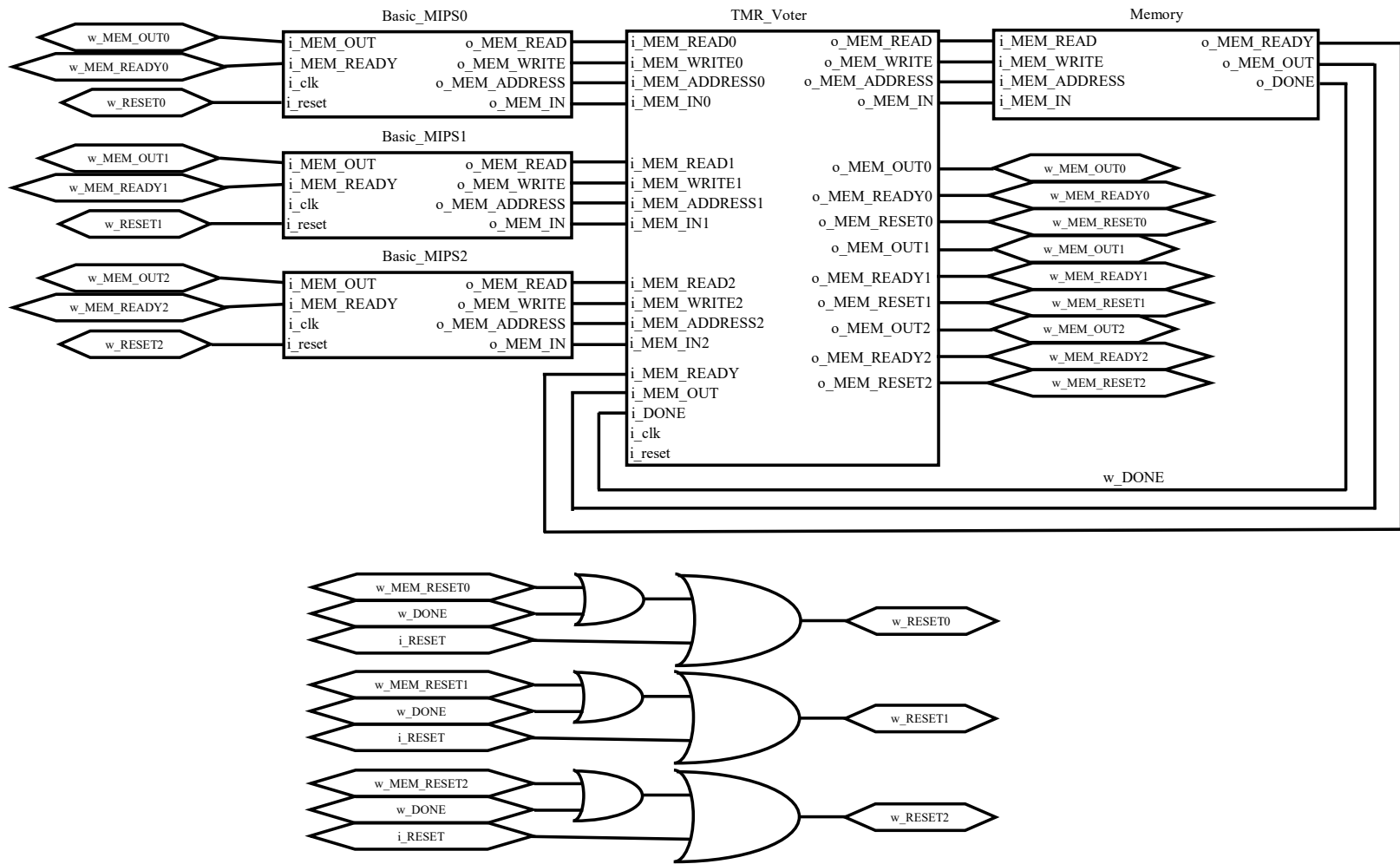


Figure 3. TMR MIPS Detailed Block Diagram

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).



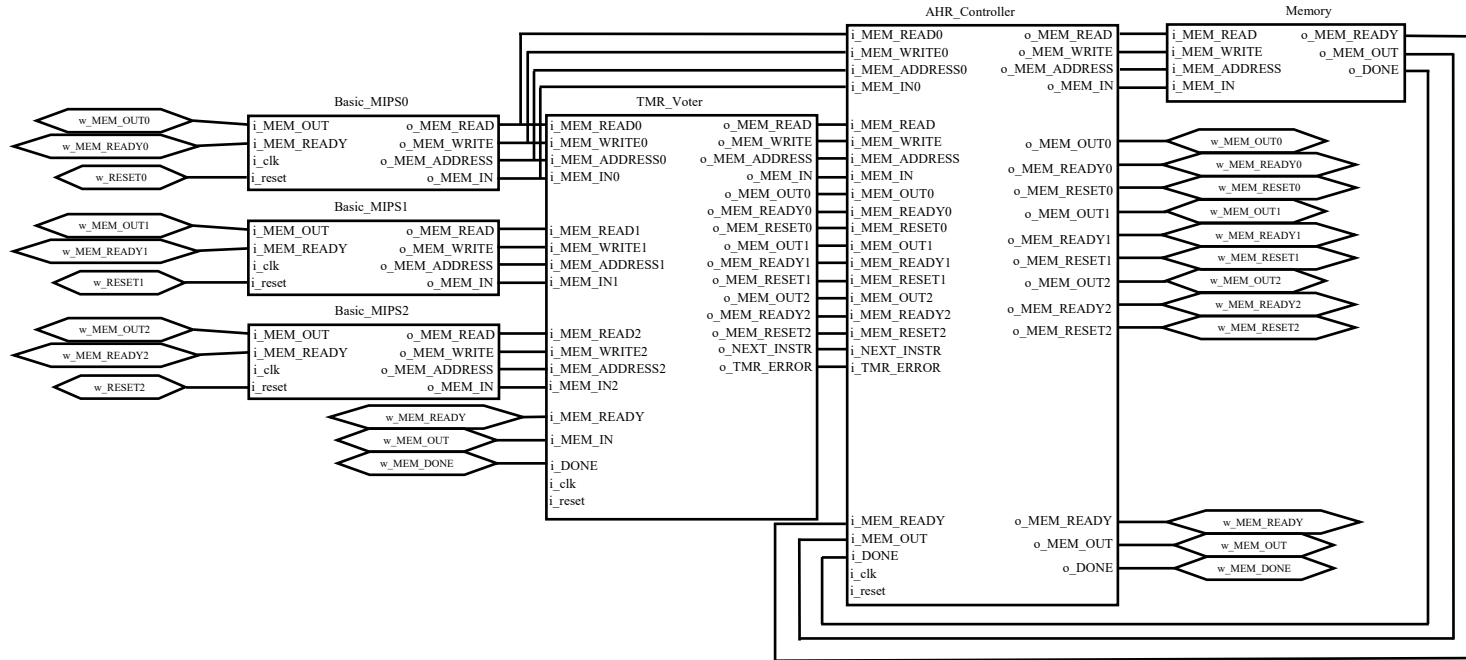


Figure 4. AHR MIPS Detailed Block Diagram

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.

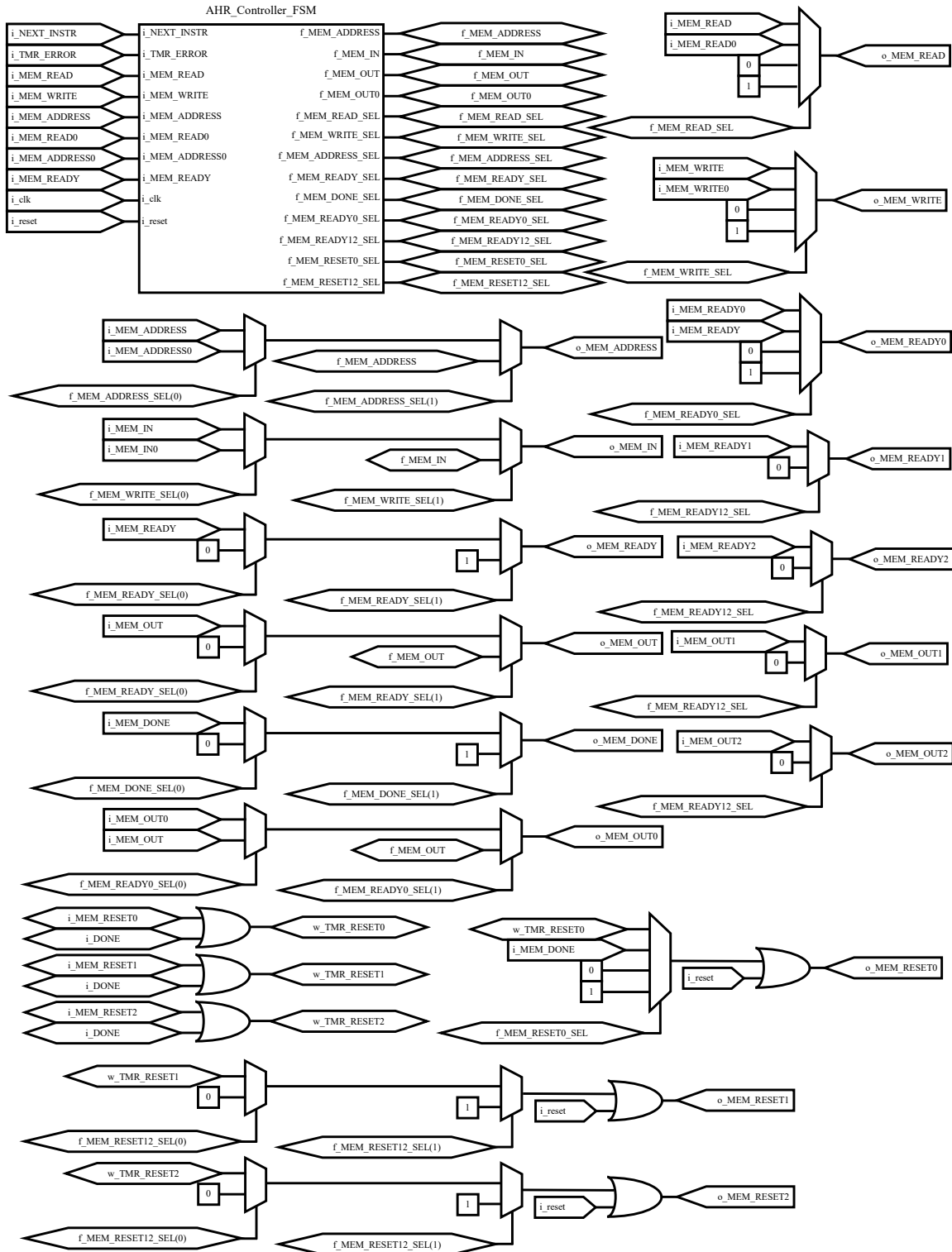


Figure 5. AHR Controller Detailed Block Diagram

## 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.

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

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

```

17         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
19         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
21         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
23         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
25         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
27         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
29         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

```

```

31         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
33         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
35         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
37         i_RESET2         : in  std_logic;
      -- From Voter
controller can modify this signal
        o_MEM_READ         : out  std_logic;
      -- To Memory
39         o_MEM_WRITE      : out  std_logic;
      -- To Memory
        o_MEM_ADDRESS     : out  std_logic_vector(31 downto 0);
      -- To Memory
41         o_MEM_IN         : out  std_logic_vector(31 downto 0);
      -- To Memory
        o_MEM_READY       : out  std_logic;
      -- To TMR Voter
43         o_MEM_OUT        : out  std_logic_vector(31 downto 0);
      -- To TMR Voter
        o_MEM_DONE        : out  std_logic;
      -- To TMR Voter
45         o_MEM_READY0     : out  std_logic;
      -- To MIPS0
        o_MEM_OUT0        : out  std_logic_vector(31 downto 0);
      -- To MIPS0
47         o_RESET0         : out  std_logic;
      -- To MIPS0
        o_MEM_READY1     : out  std_logic;
      -- To MIPS1

```

```

49     o_MEMOUT1      : out std_logic_vector(31 downto 0);
   -- To MIPS1
     o_RESET1       : out std_logic;
   -- To MIPS1
51     o_MEMREADY2   : out std_logic;
   -- To MIPS2
     o_MEMOUT2      : out std_logic_vector(31 downto 0);
   -- To MIPS2
53     o_RESET2      : out std_logic);
   -- To MIPS2
end AHR_Controller_v2_Test1001;

55
architecture a_AHR_Controller_v2_Test1001 of
  AHR_Controller_v2_Test1001 is
57 --| Declare components
   -- 2-input 1-bit mux
59 component myMUX2_1 is
     port (i_0 : in  std_logic;
61         i_1 : in  std_logic;
         i_S : in  std_logic;
63         o_Z : out std_logic
         );
65 end component;

67 -- 2-input 32-bit mux
component myMUX2N is
69   generic (m_width : integer := 32);
     port (i_0 : in  std_logic_vector(m_width-1 downto 0);
71         i_1 : in  std_logic_vector(m_width-1 downto 0);
         i_S : in  std_logic;
73         o_Z : out std_logic_vector(m_width-1 downto 0)
         );
75 end component;

   -- 4-input 1-bit mux
77 component myMUX4_1 is
     port (i_0 : in  std_logic;
79         i_1 : in  std_logic;
         i_2 : in  std_logic;
81         i_3 : in  std_logic;
         i_S : in  std_logic_vector(1 downto 0);
83         o_Z : out std_logic
         );
85 end component;

```

```

87 —| Create state machine types
— Create states for the controller finite state machine
89 type sm_c fsm is (s_c fsm_0 , s_c fsm_1 , s_c fsm_2 , s_c fsm_3 ,
s_c fsm_4 ,
s_c fsm_5 , s_c fsm_6 , s_c fsm_7 , s_c fsm_8 ,
s_c fsm_9 ,
91 s_c fsm_10 , s_c fsm_11 , s_c fsm_12 , s_c fsm_13 ,
s_c fsm_14 ,
s_c fsm_15 , s_c fsm_16 , s_c fsm_17 , s_c fsm_18 ,
s_c fsm_19 ,
93 s_c fsm_20 , s_c fsm_21 , s_c fsm_22 , s_c fsm_23 ,
s_c fsm_24 ,
s_c fsm_25 , s_c fsm_26 , s_c fsm_27 , s_c fsm_28 ,
s_c fsm_29 ,
95 s_c fsm_30 , s_c fsm_31 , s_c fsm_32 , s_c fsm_33 ,
s_c fsm_34 ,
s_c fsm_35 , s_c fsm_36 , s_c fsm_37 , s_c fsm_38 ,
s_c fsm_39 ,
97 s_c fsm_40 , s_c fsm_41 , s_c fsm_42 , s_c fsm_43 ,
s_c fsm_44 ,
s_c fsm_45 , s_c fsm_46 , s_c fsm_47 , s_c fsm_48 ,
s_c fsm_49 ,
99 s_c fsm_50 , s_c fsm_51 , s_c fsm_52 , s_c fsm_25a)
;

101 — Initialize the controller finite state machine register
signal f_c fsm_state : sm_c fsm := s_c fsm_0;
103
—| Define Signals
105 — Counts instructions to determine when to transition
from TMR to TSR
signal f_instr_count : unsigned(31 downto 0) := (others =>
'0');
107
— Signals to determine status of TSR error recovery
progress
109 signal f_err_flag : std_logic := '0';
signal f_rec_flag : std_logic := '0';
111
— Used to store the current iteration of the program loop
when switching from TMR to TSR or vice versa – modified
as needed

```



```

113 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
115 signal f_loop1 : unsigned(31 downto 0) := (others => '0');

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

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

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

— Wires for routing intermediate output values
131
133 signal w_MEM_ADDRESS : std_logic_vector(31 downto 0);
signal w_MEM_IN : std_logic_vector(31 downto 0);
135 signal w_MEM_READY : std_logic;
signal w_MEM_OUT : std_logic_vector(31 downto 0);
137 signal w_MEM_DONE : std_logic;
signal w_MEM_OUT0 : std_logic_vector(31 downto 0);
139 signal w_TMR_RESET0 : std_logic;
signal w_TMR_RESET1 : std_logic;
141 signal w_TMR_RESET2 : std_logic;
signal w_TMR_RESET0a : std_logic;
143 signal w_TMR_RESET1a : std_logic;
signal w_TMR_RESET2a : std_logic;
145 signal w_MEM_RESET1 : std_logic;
signal w_MEM_RESET2 : std_logic;
147

```

```

149  — Registers for controlling flow of outputs
— f_MEM_READ_SEL selects what read signal should be
output to memory
— 00 – Voter Pass through
151 — 01 – MIPS0 Pass Through
— 10 – 0
153 — 11 – 1
signal f_MEM_READ_SEL    : std_logic_vector(1 downto 0) :=
(others => '0');

155  — f_MEM_WRITE_SEL selects what write and data signals
should be output to memory
— 00 – Voter Pass through
157 — 01 – MIPS0 Pass Through
— 10 – 0
159 — 11 – 1
signal f_MEM_WRITE_SEL   : std_logic_vector(1 downto 0) :=
(others => '0');

163  — f_MEM_ADDRESS_SEL selects what address signal should be
output to memory
— 00 – Voter Pass through
165 — 01 – MIPS0 Pass Through
— 10 – Controller Data Override
167 — 11 – Controller Data Override
signal f_MEM_ADDRESS_SEL : std_logic_vector(1 downto 0)
:= (others => '0');

169  — f_MEM_READY_SEL selects what memory ready and data
signals should be output to the voter
171 — 00 – Memory Pass Through
— 01 – 0
173 — 10 – 1
— 11 – 1
175 signal f_MEM_READY_SEL : std_logic_vector(1 downto 0) := (
others => '0');

177  — f_MEM_DONE_SEL selects what done signal should be
output to the voter
— 00 – Memory Pass Through
179 — 01 – 0
— 10 – 1
181 — 11 – 1

```

```

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

185  — f_MEM_READY0_SEL selects what ready and data signals
should be output to MIPS0
— 00 – Voter Pass Through
187  — 01 – Memory Pass Through
— 10 – 0
— 11 – 1
189  signal f_MEM_READY0_SEL : std_logic_vector(1 downto 0) := (
others => '0');

191  — f_MEM_READY12_SEL selects what ready and data signals
should be output to MIPS1 and MIPS2
— 0 – Voter Pass Through
193  — 1 – 0
signal f_MEM_READY12_SEL : std_logic := '0';

195  — f_MEM_RESET0_SEL selects what reset signal should be
output to MIPS0
197  — 00 – Voter Pass Through
— 01 – Memory Pass Through
199  — 10 – 0
— 11 – 1
201  signal f_MEM_RESET0_SEL : std_logic_vector(1 downto 0) := (
others => '0');

203  — f_MEM_RESET12_SEL selects what reset signal should be
output to MIPS1 and MIPS2
— 00 – Voter Pass Through
205  — 01 – 0
— 10 – 1
207  — 11 – 1
signal f_MEM_RESET12_SEL : std_logic_vector(1 downto 0) :=
(others => '0');

209

211  — Define Constants
— Constant used to compare instruction counter against
213  constant k_switch_point : unsigned(31 downto 0) := ”
0000000000000000000000001000111”; —71
--”0000000000000000000011101010011000”; --15,000

```

```

215  — Location at which backup memory starts
      constant k_mem_location : std_logic_vector(31 downto 0) :=
        "00000000000000000000000010111001000";
217  — Important backup memory locations
      constant k_14_16 : std_logic_vector(15 downto 0) := "
        0000000000111000";
219  constant k_02_32 : std_logic_vector(31 downto 0) := "
        000000000000000000000000000000001000";
      constant k_14_32 : std_logic_vector(31 downto 0) := "
        00000000000000000000000000000000111000";
221  constant k_29_32 : std_logic_vector(31 downto 0) := "
        000000000000000000000000000000001110100";
      constant k_30_32 : std_logic_vector(31 downto 0) := "
        000000000000000000000000000000001111000";
223  constant k_31_32 : std_logic_vector(31 downto 0) := "
        000000000000000000000000000000001111100";
      constant k_62_32 : std_logic_vector(31 downto 0) := "
        0000000000000000000000000000000011111000";
225  constant k_63_32 : std_logic_vector(31 downto 0) := "
        0000000000000000000000000000000011111100";
      constant k_64_32 : std_logic_vector(31 downto 0) := "
        00000000000000000000000000000000100000000";
227  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));
229  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));
231  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));
233  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));
235 constant k_mem_location64_32 : std_logic_vector(31 downto
0) := std_logic_vector(unsigned(k_mem_location) + unsigned
(k_64_32));

— Additional constants
237 constant k_20_32 : std_logic_vector (31 downto 0) := ”
0000000000000000000000000000000010100”;

239

— Location at which TMR loop starts
241 constant k_TMR_LOOP_START : std_logic_vector(31 downto 0)
:= ”000000000000000000000000000000001000”;
— Location at which TSR starts
243 constant k_TSR_START_BRANCH : std_logic_vector(15 downto
0) := ”0000000000111010”;
— Location at which TSR loop starts
245 constant k_TSR_LOOP_START_BRANCH : std_logic_vector(15
downto 0) := ”0000000000111100”;
— Location at which TSR Error Recovery Code Starts
247 constant k_TSR_RECOVERY : std_logic_vector(31 downto 0) :=
”00000000000000000000000010010011000”;
249 — Location at which TSR Save/Restore Creation Point Code
Starts
constant k_TSR_SAVE : std_logic_vector(31 downto 0) := ”
000000000000000000000000001011101100”;
251 — Location at which TSR Code Ends
constant k_TSR_END : std_logic_vector(31 downto 0) := ”
000000000000000000000000001011001000”;

253

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

257

— One constant
259 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)
261 constant k_256_32 : std_logic_vector(31 downto 0) := ”

```



```

295         i_3 => k_one_1 ,
          i_S => f_MEM_WRITE_SEL ,
          o_Z => o_MEM_WRITE);
297
— MUX to determine the memory address signal to be sent
to memory – selects between Voter or MIPS 0 – intermediate
selector
299 u_myMUX_MEM_ADDRESS_Intermediate: myMUX2_N
generic map (m_width => 32)
301 port map (i_0 => i_MEM_ADDRESS ,
          i_1 => i_MEM_ADDRESS0 ,
          i_S => f_MEM_ADDRESS_SEL(0) ,
          o_Z => w_MEM_ADDRESS);
303
305
— 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
307 u_myMUX_MEM_ADDRESS: myMUX2_N
generic map (m_width => 32)
309 port map (i_0 => w_MEM_ADDRESS ,
          i_1 => f_MEM_ADDRESS ,
          i_S => f_MEM_ADDRESS_SEL(1) ,
          o_Z => o_MEM_ADDRESS);
311
313
— MUX to determine the memory data input signal to be
sent to memory – selects between Voter or MIPS 0 –
intermediate selector
315 u_myMUX_MEM_IN_Intermediate: myMUX2_N
generic map (m_width => 32)
317 port map (i_0 => i_MEM_IN ,
          i_1 => i_MEM_IN0 ,
          i_S => f_MEM_WRITE_SEL(0) ,
          o_Z => w_MEM_IN);
319
321
— 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
323 u_myMUX_MEM_IN: myMUX2_N
generic map (m_width => 32)
325 port map (i_0 => w_MEM_IN ,
          i_1 => f_MEM_IN ,
          i_S => f_MEM_WRITE_SEL(1) ,
          o_Z => o_MEM_IN);
327

```

```

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

337 — 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
339 port map (i_0 => w_MEM_READY,
         i_1 => k_one_1 ,
341         i_S => f_MEM_READY_SEL(1) ,
         o_Z => o_MEM_READY);
343

— MUX to determine the memory data signal to be sent to
the voter — selects between Memory or 0 — intermediate
selector
345 u_myMUX_MEM_OUT_Intermediate: myMUX2_N
generic map (m_width => 32)
347 port map (i_0 => i_MEM_OUT,
         i_1 => k_zero_32 ,
349         i_S => f_MEM_READY_SEL(0) ,
         o_Z => w_MEM_OUT);
351

— 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
353 u_myMUX_MEM_OUT: myMUX2_N
generic map (m_width => 32)
355 port map (i_0 => w_MEM_OUT,
         i_1 => f_MEM_OUT,
357         i_S => f_MEM_READY_SEL(1) ,
         o_Z => o_MEM_OUT);
359

— MUX to determine the memory done signal to be sent to
the voter — selects between Memory or 0 — intermediate
selector
361 u_myMUX_MEM_DONE_Intermediate: myMUX2_1

```



```

363     port map ( i_0 => i_MEMDONE,
365               i_1 => k_zero_1 ,
               i_S => f_MEMDONE_SEL(0) ,
               o_Z => w_MEMDONE);

367     — 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_MEMDONE: myMUX2_1
369     port map ( i_0 => w_MEMDONE,
               i_1 => k_one_1 ,
371               i_S => f_MEMDONE_SEL(1) ,
               o_Z => o_MEMDONE);

373     — MUX to determine the ready signal to be sent to MIPS0
375     u_myMUX_MEMREADY0: myMUX4_1
     port map ( i_0 => i_MEM_READY0,
377               i_1 => i_MEM_READY ,
               i_2 => k_zero_1 ,
379               i_3 => k_one_1 ,
               i_S => f_MEM_READY0_SEL ,
381               o_Z => o_MEM_READY0);

383     — MUX to determine the ready signal to be sent to MIPS1
u_myMUX_MEMREADY1: myMUX2_1
385     port map ( i_0 => i_MEM_READY1,
               i_1 => k_zero_1 ,
387               i_S => f_MEM_READY12_SEL ,
               o_Z => o_MEM_READY1);

389     — MUX to determine the ready signal to be sent to MIPS1
391     u_myMUX_MEMREADY2: myMUX2_1
     port map ( i_0 => i_MEM_READY2,
393               i_1 => k_zero_1 ,
               i_S => f_MEM_READY12_SEL ,
395               o_Z => o_MEM_READY2);

397     — 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
399     generic map (m_width => 32)
     port map ( i_0 => i_MEM_OUT0,

```

```

401         i_1 => i_MEM_OUT,
         i_S => f_MEM_READY0_SEL(0),
403         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
407     generic map (m_width => 32)
     port map (i_0 => w_MEM_OUT0,
409             i_1 => f_MEM_OUT0,
             i_S => f_MEM_READY0_SEL(1),
411             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
415     generic map (m_width => 32)
     port map (i_0 => i_MEM_OUT1,
417             i_1 => k_zero_32,
             i_S => f_MEM_READY12_SEL,
419             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
423     generic map (m_width => 32)
     port map (i_0 => i_MEM_OUT2,
425             i_1 => k_zero_32,
             i_S => f_MEM_READY12_SEL,
427             o_Z => o_MEM_OUT2);

— MUX to determine the reset signal to be sent to MIPS0
u_myMUX_MEM_RESET0: myMUX4_1
431     port map (i_0 => w_TMR_RESET0,
             i_1 => i_MEM_DONE,
433             i_2 => k_zero_1,
             i_3 => k_one_1,
435             i_S => f_MEM_RESET0_SEL,
             o_Z => w_TMR_RESET0a);
437

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

```

```

439 u_myMUX_MEM_RESET1_Intermediate: myMUX2_1
port map ( i_0 => w_TMR_RESET1,
441         i_1 => k_zero_1 ,
         i_S => f_MEM_RESET12_SEL(0) ,
443         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
447 port map ( i_0 => w_MEM_RESET1,
         i_1 => k_one_1 ,
449         i_S => f_MEM_RESET12_SEL(1) ,
         o_Z => w_TMR_RESET1a);

451 -- 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
453 port map ( i_0 => w_TMR_RESET2,
         i_1 => k_zero_1 ,
455         i_S => f_MEM_RESET12_SEL(0) ,
457         o_Z => w_MEM_RESET2);

459 -- 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
461 port map ( i_0 => w_MEM_RESET2,
         i_1 => k_one_1 ,
463         i_S => f_MEM_RESET12_SEL(1) ,
         o_Z => w_TMR_RESET2a);

465

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

```

```

479         f_NEXT_INSTR <= i_NEXT_INSTR;
         case f_c fsm_state is
481             — TMR MIPS is running
             — Determine when TMR MIPS completes processing
             an instruction or encounters an error
             when s_c fsm_0 =>
483                 if (i_TMR_ERROR = '1') then
                     f_c fsm_state <= s_c fsm_2;
485                 elsif ((f_instr_count >= k_switch_point) and (
i_MEM_READ = '1') and (i_MEM_ADDRESS = k_TMR_LOOP_START))
then
                     f_c fsm_state <= s_c fsm_3;
487                 elsif ((i_NEXT_INSTR = '1') and (f_NEXT_INSTR
= '0')) then
                     f_c fsm_state <= s_c fsm_1;
489                 else
                     f_c fsm_state <= s_c fsm_0;
491                 end if;
                     f_instr_count <= f_instr_count;
493                     f_loop <= f_loop;
                     f_loop1 <= f_loop1;
495                     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                     f_err_flag <= '0';
497                     f_rec_flag <= '0';

499             — Return to state 0
             when s_c fsm_1 =>
501                 f_c fsm_state <= s_c fsm_0;
                     f_instr_count <= f_instr_count + 1;
503                     f_loop <= f_loop;
                     f_loop1 <= f_loop1;
505                     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
                     f_err_flag <= f_err_flag;
507                     f_rec_flag <= f_rec_flag;

509             — TMR MIPS has encountered an error
             when s_c fsm_2 =>
511                 if (i_TMR_ERROR = '0') then
                     f_c fsm_state <= s_c fsm_0;
513                     f_instr_count <= (others => '0');
                 else
515                     f_c fsm_state <= s_c fsm_2;
                     f_instr_count <= f_instr_count + 1;

```

```

517     end if;
519     f_loop <= f_loop;
521     f_loop1 <= f_loop1;
523     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
525     f_err_flag <= f_err_flag;
527     f_rec_flag <= f_rec_flag;
529
531 — Start transition from TMR MIPS to TSR MIPS
533 — Wait for Memory Ready signal
535 when s_c fsm_3 =>
537     if (i_MEM_READY = '1') then
539         f_c fsm_state <= s_c fsm_4;
541     else
543         f_c fsm_state <= s_c fsm_3;
545     end if;
547     f_instr_count <= f_instr_count;
549     f_loop <= f_loop;
551     f_loop1 <= f_loop1;
553     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
555     f_err_flag <= f_err_flag;
557     f_rec_flag <= f_rec_flag;
559
561 — Wait for Memory Ready signal to return to 0
563 when s_c fsm_4 =>
565     if (i_MEM_READY = '0') then
567         f_c fsm_state <= s_c fsm_5;
569     else
571         f_c fsm_state <= s_c fsm_4;
573     end if;
575     f_instr_count <= f_instr_count;
577     f_loop <= f_loop;
579     f_loop1 <= f_loop1;
581     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
583     f_err_flag <= f_err_flag;
585     f_rec_flag <= f_rec_flag;
587
589 — Wait for TMR Voter Read signal to return to 0
591 when s_c fsm_5 =>
593     if (i_MEM_READ = '0') then
595         f_c fsm_state <= s_c fsm_6;
597     else
599         f_c fsm_state <= s_c fsm_5;
601     end if;

```

```

561     f_instr_count <= f_instr_count;
562     f_loop <= f_loop;
563     f_loop1 <= f_loop1;
564     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
565     f_err_flag <= f_err_flag;
566     f_rec_flag <= f_rec_flag;
567
568     — Wait for TMR Voter Write signal and watch out
569     for an Error
570     when s_c fsm_6 =>
571         if (i_TMR_ERROR = '1') then
572             f_c fsm_state <= s_c fsm_2;
573         elsif (i_MEM_WRITE = '1') then
574             f_c fsm_state <= s_c fsm_7;
575         else
576             f_c fsm_state <= s_c fsm_6;
577         end if;
578     f_instr_count <= f_instr_count;
579     f_loop <= f_loop;
580     f_loop1 <= f_loop1;
581     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
582     f_err_flag <= f_err_flag;
583     f_rec_flag <= f_rec_flag;
584
585     — Wait for TMR Voter Write signal to return to 0
586     when s_c fsm_7 =>
587         if (i_MEM_WRITE = '0') then
588             f_c fsm_state <= s_c fsm_8;
589             f_loop <= f_loop;
590             f_loop1 <= f_loop1;
591         else
592             f_c fsm_state <= s_c fsm_7;
593             f_loop <= unsigned(i_MEM_IN);
594             f_loop1 <= unsigned(i_MEM_IN);
595         end if;
596     f_instr_count <= f_instr_count;
597     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
598     f_err_flag <= f_err_flag;
599     f_rec_flag <= f_rec_flag;
600
601     — Add 1 to the loop count
602     when s_c fsm_8 =>
603         f_c fsm_state <= s_c fsm_9;

```

```

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

609     — Wait for Memory Ready Signal
        when s_c fsm_9 =>
611         if (i_MEM_READY = '1') then
            f_c fsm_state <= s_c fsm_10;
613         else
            f_c fsm_state <= s_c fsm_9;
615         end if;
        f_instr_count <= f_instr_count;
617         f_loop <= f_loop;
        f_loop1 <= f_loop1;
619         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
621         f_rec_flag <= f_rec_flag;

623     — Wait for Memory Ready signal to return to 0
        when s_c fsm_10 =>
625         if (i_MEM_READY = '0') then
            f_c fsm_state <= s_c fsm_11;
627         else
            f_c fsm_state <= s_c fsm_10;
629         end if;
        f_instr_count <= f_instr_count;
631         f_loop <= f_loop;
        f_loop1 <= f_loop1;
633         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
635         f_rec_flag <= f_rec_flag;

637     — Wait for Memory Ready Signal
        when s_c fsm_11 =>
639         if (i_MEM_READY = '1') then
            f_c fsm_state <= s_c fsm_12;
641         else
            f_c fsm_state <= s_c fsm_11;
643         end if;
        f_instr_count <= f_instr_count;

```

```

645         f_loop <= f_loop;
        f_loop1 <= f_loop1;
647     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
649     f_rec_flag <= f_rec_flag;

651     — Wait for Memory Ready signal to return to 0
when s_c fsm_12 =>
653     if (iMEMREADY = '0') then
        f_c fsm_state <= s_c fsm_13;
655     else
        f_c fsm_state <= s_c fsm_12;
657     end if;
        f_instr_count <= f_instr_count;
659     f_loop <= f_loop;
        f_loop1 <= f_loop1;
661     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
663     f_rec_flag <= f_rec_flag;

665     — Wait for Memory Ready Signal
when s_c fsm_13 =>
667     if (iMEMREADY = '1') then
        f_c fsm_state <= s_c fsm_14;
669     else
        f_c fsm_state <= s_c fsm_13;
671     end if;
        f_instr_count <= f_instr_count;
673     f_loop <= f_loop;
        f_loop1 <= f_loop1;
675     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
677     f_rec_flag <= f_rec_flag;

679     — Wait for Memory Ready signal to return to 0
when s_c fsm_14 =>
681     if (iMEMREADY = '0') then
        f_c fsm_state <= s_c fsm_15;
683     else
        f_c fsm_state <= s_c fsm_14;
685     end if;
        f_instr_count <= f_instr_count;
687     f_loop <= f_loop;

```



```

689     f_loop1 <= f_loop1;
        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
691     f_err_flag <= f_err_flag;
        f_rec_flag <= f_rec_flag;

693     — Reset MIPS Processors
    when s_c fsm_15 =>
695         f_c fsm_state <= s_c fsm_16;
        f_instr_count <= f_instr_count;
697         f_loop <= f_loop;
        f_loop1 <= f_loop1;
699         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
701         f_rec_flag <= f_rec_flag;

703     — Wait for MIPS0 Read Signal
    when s_c fsm_16 =>
705         if (i_MEM_READ0 = '1') then
            f_c fsm_state <= s_c fsm_17;
707         else
            f_c fsm_state <= s_c fsm_16;
709         end if;
        f_instr_count <= f_instr_count;
711         f_loop <= f_loop;
        f_loop1 <= f_loop1;
713         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
715         f_rec_flag <= f_rec_flag;

717     — Wait for MIPS0 Read Signal to return to 0
    when s_c fsm_17 =>
719         if (i_MEM_READ0 = '0') then
            f_c fsm_state <= s_c fsm_18;
721         else
            f_c fsm_state <= s_c fsm_17;
723         end if;
        f_instr_count <= f_instr_count;
725         f_loop <= f_loop;
        f_loop1 <= f_loop1;
727         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
729         f_rec_flag <= f_rec_flag;

```

```

731 — Wait for MIPS0 Read Signal
when s_c fsm_18 =>
733   if (i_MEM_READ0 = '1') then
       f_c fsm_state <= s_c fsm_19;
735   else
       f_c fsm_state <= s_c fsm_18;
737   end if;
       f_instr_count <= f_instr_count;
739       f_loop <= f_loop;
       f_loop1 <= f_loop1;
741       f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
743       f_rec_flag <= f_rec_flag;

745 — Wait for MIPS0 Read Signal to return to 0
when s_c fsm_19 =>
747   if (i_MEM_READ0 = '0') then
       f_c fsm_state <= s_c fsm_20;
749   else
       f_c fsm_state <= s_c fsm_19;
751   end if;
       f_instr_count <= f_instr_count;
753       f_loop <= f_loop;
       f_loop1 <= f_loop1;
755       f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
757       f_rec_flag <= f_rec_flag;

759 — Wait for MIPS0 Read Signal
when s_c fsm_20 =>
761   if (i_MEM_READ0 = '1') then
       f_c fsm_state <= s_c fsm_21;
763   else
       f_c fsm_state <= s_c fsm_20;
765   end if;
       f_instr_count <= f_instr_count;
767       f_loop <= f_loop;
       f_loop1 <= f_loop1;
769       f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
771       f_rec_flag <= f_rec_flag;

773 — Wait for MIPS0 Read Signal to return to 0

```

```

775     when s_c fsm_21 =>
       if (i_MEM_READ0 = '0') then
           f_c fsm_state <= s_c fsm_22;
777     else
           f_c fsm_state <= s_c fsm_21;
779     end if;
       f_instr_count <= f_instr_count;
781     f_loop <= f_loop;
       f_loop1 <= f_loop1;
783     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
       f_err_flag <= f_err_flag;
785     f_rec_flag <= f_rec_flag;

787     — Wait for MIPS0 Read Signal
       when s_c fsm_22 =>
789         if (i_MEM_READ0 = '1') then
           f_c fsm_state <= s_c fsm_23;
791         else
           f_c fsm_state <= s_c fsm_22;
793         end if;
           f_instr_count <= f_instr_count;
795         f_loop <= f_loop;
           f_loop1 <= f_loop1;
797         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
           f_err_flag <= f_err_flag;
799         f_rec_flag <= f_rec_flag;

801     — Wait for MIPS0 Read signal to return to 0
       when s_c fsm_23 =>
803         if (i_MEM_READ0 = '0') then
           f_c fsm_state <= s_c fsm_24;
805         else
           f_c fsm_state <= s_c fsm_23;
807         end if;
           f_instr_count <= f_instr_count;
809         f_loop <= f_loop;
           f_loop1 <= f_loop1;
811         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
           f_err_flag <= f_err_flag;
813         f_rec_flag <= f_rec_flag;

815     — Wait for MIPS0 Read signal
       when s_c fsm_24 =>

```

```

817         if (i_MEM_READ0 = '1') then
            f_c fsm_state <= s_c fsm_25;
819         else
            f_c fsm_state <= s_c fsm_24;
821         end if;
            f_instr_count <= f_instr_count;
823         f_loop <= f_loop;
            f_loop1 <= f_loop1;
825         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
            f_err_flag <= f_err_flag;
827         f_rec_flag <= f_rec_flag;

            -- TSR MIPS is Running
            -- Wait for an error to occur or the program to
            finish
829         when s_c fsm_25 =>
            if ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0 =
831 k_TSR_RECOVERY) and (f_err_flag = '1')) then
            f_c fsm_state <= s_c fsm_29;
833         elsif ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0
            = k_TSR_RECOVERY) and (f_err_flag = '0')) then
835         f_c fsm_state <= s_c fsm_25a;
            f_err_flag <= '1';
837         elsif ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0
            = k_TSR_SAVE)) then
            f_c fsm_state <= s_c fsm_25;
839         f_rec_flag <= '1';
            elsif ((i_MEM_READ0 = '1') and (i_MEM_ADDRESS0
            <= k_TSR_END) and (f_rec_flag = '1')) then
841         f_c fsm_state <= s_c fsm_25;
            f_rec_flag <= '0';
843         f_err_flag <= '0';
            elsif (i_MEM_DONE = '1') then
845         f_c fsm_state <= s_c fsm_26;
            else
847         f_c fsm_state <= s_c fsm_25;
            end if;
849         f_instr_count <= f_instr_count;
            f_loop <= f_loop;
851         f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
853
            -- An error has occurred in TSR MIPS

```

```

855         when s_c fsm_25a =>
            if ((iMEM_READ0 = '1') and (iMEM_ADDRESS0 >
std_logic_vector(unsigned(k.TSR_RECOVERY) + unsigned(
k_20_32)))) then
857             f_c fsm_state <= s_c fsm_25;
            else
859             f_c fsm_state <= s_c fsm_25a;
            end if;
861             f_instr_count <= f_instr_count;
            f_loop <= f_loop;
863             f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
865             f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;
867
Memory. — Interrupt communications between MIPS0 and
Wait for MIPS0 Read Signal
869         when s_c fsm_26 =>
            if (iMEM_READ0 = '1') then
871             f_c fsm_state <= s_c fsm_27;
            else
873             f_c fsm_state <= s_c fsm_26;
            end if;
875             f_instr_count <= f_instr_count;
            f_loop <= f_loop;
877             f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
879             f_err_flag <= f_err_flag;
            f_rec_flag <= f_rec_flag;
881
Start. — Transmit branch instruction to branch to TSR
Wait for MIPS0 Read Signal to return to 0
883         when s_c fsm_27 =>
            if (iMEM_READ0 = '0') then
885             f_c fsm_state <= s_c fsm_28;
            else
887             f_c fsm_state <= s_c fsm_27;
            end if;
889             f_instr_count <= f_instr_count;
            f_loop <= f_loop;
891             f_loop1 <= f_loop1;
            f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
893             f_err_flag <= f_err_flag;

```

```

895         f_rec_flag <= f_rec_flag;
897     — Wait for MIPS0 Read Signal
    when s_c fsm_28 =>
899         if (iMEM_READ0 = '1') then
                f_c fsm_state <= s_c fsm_25;
901         else
                f_c fsm_state <= s_c fsm_28;
        end if;
903         f_instr_count <= f_instr_count;
        f_loop <= f_loop;
905         f_loop1 <= f_loop1;
        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
907         f_err_flag <= f_err_flag;
        f_rec_flag <= f_rec_flag;
909
    — Start transition from TSR MIPS to TMR MIPS
    — Wait for Memory Ready Signal
    when s_c fsm_29 =>
913         if (iMEM_READY = '1') then
                f_c fsm_state <= s_c fsm_30;
915         else
                f_c fsm_state <= s_c fsm_29;
917         end if;
        f_instr_count <= f_instr_count;
919         f_loop <= f_loop;
        f_loop1 <= f_loop1;
921         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= '0';
923         f_rec_flag <= '0';
925
    — Wait for Memory Ready signal to return to 0
    when s_c fsm_30 =>
927         if (iMEM_READY = '0') then
                f_c fsm_state <= s_c fsm_31;
929         else
                f_c fsm_state <= s_c fsm_30;
931         end if;
        f_instr_count <= f_instr_count;
933         f_loop <= f_loop;
        f_loop1 <= f_loop1;
935         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;

```

```

937         f_rec_flag <= f_rec_flag;

939     — Wait for Memory Ready Signal
when s_c fsm_31 =>
941         if (iMEMREADY = '1') then
            f_c fsm_state <= s_c fsm_32;
943         else
            f_c fsm_state <= s_c fsm_31;
945         end if;
        f_instr_count <= f_instr_count;
947        f_loop <= f_loop;
        f_loop1 <= f_loop1;
949        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
951        f_rec_flag <= f_rec_flag;

953     — Wait for Memory Ready signal to return to 0
when s_c fsm_32 =>
955         if (iMEMREADY = '0') then
            f_c fsm_state <= s_c fsm_33;
957         else
            f_c fsm_state <= s_c fsm_32;
959         end if;
        f_instr_count <= f_instr_count;
961        f_loop <= f_loop;
        f_loop1 <= f_loop1;
963        f_TEMP_ADDRESS <= iMEMOUT;
        f_err_flag <= f_err_flag;
965        f_rec_flag <= f_rec_flag;

967     — Wait for Memory Ready Signal
when s_c fsm_33 =>
969         if (iMEMREADY = '1') then
            f_c fsm_state <= s_c fsm_34;
971         else
            f_c fsm_state <= s_c fsm_33;
973         end if;
        f_instr_count <= f_instr_count;
975        f_loop <= f_loop;
        f_loop1 <= f_loop1;
977        f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
        f_err_flag <= f_err_flag;
979        f_rec_flag <= f_rec_flag;

```

```

981  — Wait for Memory Ready signal to return to 0
when s_c fsm_34 =>
983      if (iMEM_READY = '0') then
          f_c fsm_state <= s_c fsm_35;
985          f_loop <= f_loop;
          f_loop1 <= f_loop1;
987      else
          f_c fsm_state <= s_c fsm_34;
989          f_loop <= unsigned(iMEM_OUT);
          f_loop1 <= unsigned(iMEM_OUT);
991      end if;
          f_loop <= unsigned(iMEM_OUT);
          f_loop1 <= unsigned(iMEM_OUT);
993      f_instr_count <= f_instr_count;
          f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
995      f_err_flag <= f_err_flag;
          f_rec_flag <= f_rec_flag;
997

999  — Subtract 1 from the loop count
when s_c fsm_35 =>
1001      f_c fsm_state <= s_c fsm_36;
          f_instr_count <= f_instr_count;
1003      f_loop <= f_loop - 1;
          f_loop1 <= f_loop1;
1005      f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= f_err_flag;
1007      f_rec_flag <= f_rec_flag;

1009  — Wait for Memory Ready Signal
when s_c fsm_36 =>
1011      if (iMEM_READY = '1') then
          f_c fsm_state <= s_c fsm_37;
1013      else
          f_c fsm_state <= s_c fsm_36;
1015      end if;
          f_instr_count <= f_instr_count;
1017      f_loop <= f_loop;
          f_loop1 <= f_loop1;
1019      f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
          f_err_flag <= f_err_flag;
1021      f_rec_flag <= f_rec_flag;

```



```

1023 — Wait for Memory Ready signal to return to 0
      when s_c fsm_37 =>
1025         if (iMEM_READY = '0') then
              f_c fsm_state <= s_c fsm_38;
1027         else
              f_c fsm_state <= s_c fsm_37;
1029         end if;
              f_instr_count <= f_instr_count;
1031         f_loop <= f_loop;
              f_loop1 <= f_loop1;
1033         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
              f_err_flag <= f_err_flag;
1035         f_rec_flag <= f_rec_flag;

1037 — Wait for Memory Ready Signal
      when s_c fsm_38 =>
1039         if (iMEM_READY = '1') then
              f_c fsm_state <= s_c fsm_39;
1041         else
              f_c fsm_state <= s_c fsm_38;
1043         end if;
              f_instr_count <= f_instr_count;
1045         f_loop <= f_loop;
              f_loop1 <= f_loop1;
1047         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
              f_err_flag <= f_err_flag;
1049         f_rec_flag <= f_rec_flag;

1051 — Wait for Memory Ready signal to return to 0
      when s_c fsm_39 =>
1053         if (iMEM_READY = '0') then
              f_c fsm_state <= s_c fsm_40;
1055         else
              f_c fsm_state <= s_c fsm_39;
1057         end if;
              f_instr_count <= f_instr_count;
1059         f_loop <= f_loop;
              f_loop1 <= f_loop1;
1061         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
              f_err_flag <= f_err_flag;
1063         f_rec_flag <= f_rec_flag;

1065 — Wait for Memory Ready Signal

```

```

1067     when s_c fsm_40 =>
1068         if (iMEM_READY = '1') then
1069             f_c fsm_state <= s_c fsm_41;
1070         else
1071             f_c fsm_state <= s_c fsm_40;
1072         end if;
1073         f_instr_count <= f_instr_count;
1074         f_loop <= f_loop;
1075         f_loop1 <= f_loop1;
1076         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1077         f_err_flag <= f_err_flag;
1078         f_rec_flag <= f_rec_flag;
1079
1080     — Wait for Memory Ready signal to return to 0
1081     when s_c fsm_41 =>
1082         if (iMEM_READY = '0') then
1083             f_c fsm_state <= s_c fsm_42;
1084         else
1085             f_c fsm_state <= s_c fsm_41;
1086         end if;
1087         f_instr_count <= f_instr_count;
1088         f_loop <= f_loop;
1089         f_loop1 <= f_loop1;
1090         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1091         f_err_flag <= f_err_flag;
1092         f_rec_flag <= f_rec_flag;
1093
1094     — Wait for Memory Ready Signal
1095     when s_c fsm_42 =>
1096         if (iMEM_READY = '1') then
1097             f_c fsm_state <= s_c fsm_43;
1098         else
1099             f_c fsm_state <= s_c fsm_42;
1100         end if;
1101         f_instr_count <= f_instr_count;
1102         f_loop <= f_loop;
1103         f_loop1 <= f_loop1;
1104         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1105         f_err_flag <= f_err_flag;
1106         f_rec_flag <= f_rec_flag;
1107
1108     — Wait for Memory Ready signal to return to 0
1109     when s_c fsm_43 =>

```

```

1109         if (iMEM_READY = '0') then
1110             f_c fsm_state <= s_c fsm_44;
1111         else
1112             f_c fsm_state <= s_c fsm_43;
1113         end if;
1114         f_instr_count <= f_instr_count;
1115         f_loop <= f_loop;
1116         f_loop1 <= f_loop1;
1117         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1118         f_err_flag <= f_err_flag;
1119         f_rec_flag <= f_rec_flag;
1120
1121     — Wait for Memory Ready Signal
1122     when s_c fsm_44 =>
1123         if (iMEM_READY = '1') then
1124             f_c fsm_state <= s_c fsm_45;
1125         else
1126             f_c fsm_state <= s_c fsm_44;
1127         end if;
1128         f_instr_count <= f_instr_count;
1129         f_loop <= f_loop;
1130         f_loop1 <= f_loop1;
1131         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1132         f_err_flag <= f_err_flag;
1133         f_rec_flag <= f_rec_flag;
1134
1135     — Wait for Memory Ready signal to return to 0
1136     when s_c fsm_45 =>
1137         if (iMEM_READY = '0') then
1138             f_c fsm_state <= s_c fsm_46;
1139         else
1140             f_c fsm_state <= s_c fsm_45;
1141         end if;
1142         f_instr_count <= f_instr_count;
1143         f_loop <= f_loop;
1144         f_loop1 <= f_loop1;
1145         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1146         f_err_flag <= f_err_flag;
1147         f_rec_flag <= f_rec_flag;
1148
1149     — Wait for Voter Read Signal
1150     when s_c fsm_46 =>
1151         if (iMEM_READ = '1') then

```

```

1153         f_c fsm_state <= s_c fsm_47;
1154     else
1155         f_c fsm_state <= s_c fsm_46;
1156     end if;
1157     f_instr_count <= f_instr_count;
1158     f_loop <= f_loop;
1159     f_loop1 <= f_loop1;
1160     f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1161     f_err_flag <= f_err_flag;
1162     f_rec_flag <= f_rec_flag;
1163
1164     — Wait for Voter Read Signal to return to 0
1165     when s_c fsm_47 =>
1166         if (iMEMREAD = '0') then
1167             f_c fsm_state <= s_c fsm_48;
1168         else
1169             f_c fsm_state <= s_c fsm_47;
1170         end if;
1171         f_instr_count <= f_instr_count;
1172         f_loop <= f_loop;
1173         f_loop1 <= f_loop1;
1174         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1175         f_err_flag <= f_err_flag;
1176         f_rec_flag <= f_rec_flag;
1177
1178     — Wait for Voter Read Signal
1179     when s_c fsm_48 =>
1180         if (iMEMREAD = '1') then
1181             f_c fsm_state <= s_c fsm_49;
1182         else
1183             f_c fsm_state <= s_c fsm_48;
1184         end if;
1185         f_instr_count <= f_instr_count;
1186         f_loop <= f_loop;
1187         f_loop1 <= f_loop1;
1188         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1189         f_err_flag <= f_err_flag;
1190         f_rec_flag <= f_rec_flag;
1191
1192     — Wait for Voter Read Signal to return to 0
1193     when s_c fsm_49 =>
1194         if (iMEMREAD = '0') then
1195             f_c fsm_state <= s_c fsm_50;

```

```

1195         else
1196             f_c fsm_state <= s_c fsm_49;
1197         end if;
1198         f_instr_count <= f_instr_count;
1199         f_loop <= f_loop;
1200         f_loop1 <= f_loop1;
1201         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1202         f_err_flag <= f_err_flag;
1203         f_rec_flag <= f_rec_flag;
1204
1205     — Wait for Voter Read Signal
1206     when s_c fsm_50 =>
1207         if (i_MEM_READ = '1') then
1208             f_c fsm_state <= s_c fsm_51;
1209         else
1210             f_c fsm_state <= s_c fsm_50;
1211         end if;
1212         f_instr_count <= f_instr_count;
1213         f_loop <= f_loop;
1214         f_loop1 <= f_loop1;
1215         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1216         f_err_flag <= f_err_flag;
1217         f_rec_flag <= f_rec_flag;
1218
1219     — Wait for Voter Read Signal to return to 0
1220     when s_c fsm_51 =>
1221         if (i_MEM_READ = '0') then
1222             f_c fsm_state <= s_c fsm_52;
1223         else
1224             f_c fsm_state <= s_c fsm_51;
1225         end if;
1226         f_instr_count <= f_instr_count;
1227         f_loop <= f_loop;
1228         f_loop1 <= f_loop1;
1229         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1230         f_err_flag <= f_err_flag;
1231         f_rec_flag <= f_rec_flag;
1232
1233     — Return to normal TMR operation
1234     when s_c fsm_52 =>
1235         f_c fsm_state <= s_c fsm_0;
1236         f_instr_count <= f_instr_count;
1237         f_loop <= f_loop;

```

```

1239         f_loop1 <= f_loop1;
1240         f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1241         f_err_flag <= f_err_flag;
1242         f_rec_flag <= f_rec_flag;
1243
1244         — This should never happen
1245         when others =>
1246             f_c fsm_state <= s_c fsm_0;
1247             f_instr_count <= (others => '0');
1248             f_loop <= (others => '0');
1249             f_loop1 <= (others => '0');
1250             f_TEMP_ADDRESS <= f_TEMP_ADDRESS;
1251             f_err_flag <= f_err_flag;
1252             f_rec_flag <= f_rec_flag;
1253         end case;
1254     end if;
1255 end process c fsm;
1256
1257 controller_output_fsm: process(i_clk , i_reset ,
1258 f_c fsm_state)
1259 begin
1260     if (i_reset = '1') then
1261         f_MEM_ADDRESS <= (others => '0');
1262         f_MEM_IN <= (others => '0');
1263         f_MEM_OUT <= (others => '0');
1264         f_MEM_OUT0 <= (others => '0');
1265         f_MEM_READ_SEL <= (others => '0');
1266         f_MEM_WRITE_SEL <= (others => '0');
1267         f_MEM_ADDRESS_SEL <= (others => '0');
1268         f_MEM_READY_SEL <= (others => '0');
1269         f_MEM_DONE_SEL <= (others => '0');
1270         f_MEM_READY0_SEL <= (others => '0');
1271         f_MEM_READY12_SEL <= '0';
1272         f_MEM_RESET0_SEL <= (others => '0');
1273         f_MEM_RESET12_SEL <= (others => '0');
1274     elsif rising_edge(i_clk) then
1275         case f_c fsm_state is
1276             — TMR MIPS is running
1277             — Determine when TMR MIPS completes processing
1278             an instruction or encounters an error
1279             when s_c fsm_0 =>

```

```

1279     f_MEM_ADDRESS <= (others => '0');
1281     f_MEM_IN <= (others => '0');
1283     f_MEM_OUT <= (others => '0');
1285     f_MEM_OUT0 <= (others => '0');
1287     f_MEM_READ_SEL <= "00";
1289     f_MEM_WRITE_SEL <= "00";
1291     f_MEM_ADDRESS_SEL <= "00";
1293     f_MEM_READY_SEL <= "00";
1295     f_MEM_DONE_SEL <= "00";
1297     f_MEM_READY0_SEL <= "00";
1299     f_MEM_READY12_SEL <= '0';
1301     f_MEM_RESET0_SEL <= "00";
1303     f_MEM_RESET12_SEL <= "00";

```

— Return to state 0

```

when s_cfsm_1 =>
1295     f_MEM_ADDRESS <= (others => '0');
1297     f_MEM_IN <= (others => '0');
1299     f_MEM_OUT <= (others => '0');
1301     f_MEM_OUT0 <= (others => '0');
1303     f_MEM_READ_SEL <= "00";
1305     f_MEM_WRITE_SEL <= "00";
1307     f_MEM_ADDRESS_SEL <= "00";
1309     f_MEM_READY_SEL <= "00";
1311     f_MEM_DONE_SEL <= "00";
1313     f_MEM_READY0_SEL <= "00";
1315     f_MEM_READY12_SEL <= '0';
1317     f_MEM_RESET0_SEL <= "00";
1319     f_MEM_RESET12_SEL <= "00";

```

— TMR MIPS has encountered an error

```

when s_cfsm_2 =>
1311     f_MEM_ADDRESS <= (others => '0');
1313     f_MEM_IN <= (others => '0');
1315     f_MEM_OUT <= (others => '0');
1317     f_MEM_OUT0 <= (others => '0');
1319     f_MEM_READ_SEL <= "00";
1321     f_MEM_WRITE_SEL <= "00";
1323     f_MEM_ADDRESS_SEL <= "00";
1325     f_MEM_READY_SEL <= "00";
1327     f_MEM_DONE_SEL <= "00";
1329     f_MEM_READY0_SEL <= "00";
1331     f_MEM_READY12_SEL <= '0';

```

```

1323         f_MEM_RESET0_SEL <= "00";
1324         f_MEM_RESET12_SEL <= "00";

1325     — Start transition from TMR MIPS to TSR MIPS
1326     — Wait for Memory Ready signal
1327     when s_c fsm_3 =>
1328         f_MEM_ADDRESS <= (others => '0');
1329         f_MEM_IN <= (others => '0');
1330         f_MEM_OUT <= (others => '0');
1331         f_MEM_OUT0 <= (others => '0');
1332         f_MEM_READ_SEL <= "00";
1333         f_MEM_WRITE_SEL <= "00";
1334         f_MEM_ADDRESS_SEL <= "00";
1335         f_MEM_READY_SEL <= "01";
1336         f_MEM_DONE_SEL <= "01";
1337         f_MEM_READY0_SEL <= "00";
1338         f_MEM_READY12_SEL <= '0';
1339         f_MEM_RESET0_SEL <= "00";
1340         f_MEM_RESET12_SEL <= "00";

1341

1342     — Wait for Memory Ready signal to return to 0
1343     when s_c fsm_4 =>
1344         f_MEM_ADDRESS <= (others => '0');
1345         f_MEM_IN <= (others => '0');
1346         f_MEM_OUT <= "1010110000011111" &
k_mem_location14_16;
1347         f_MEM_OUT0 <= (others => '0');
1348         f_MEM_READ_SEL <= "10";
1349         f_MEM_WRITE_SEL <= "10";
1350         f_MEM_ADDRESS_SEL <= "10";
1351         f_MEM_READY_SEL <= "11";
1352         f_MEM_DONE_SEL <= "01";
1353         f_MEM_READY0_SEL <= "00";
1354         f_MEM_READY12_SEL <= '0';
1355         f_MEM_RESET0_SEL <= "00";
1356         f_MEM_RESET12_SEL <= "00";

1357

1358     — Wait for TMR Voter Read signal to return to 0
1359     when s_c fsm_5 =>
1360         f_MEM_ADDRESS <= (others => '0');
1361         f_MEM_IN <= (others => '0');
1362         f_MEM_OUT <= (others => '0');
1363         f_MEM_OUT0 <= (others => '0');

```



```

1365         f_MEM_OUT <= "1010110000011111" &
k_mem_location14_16;
1367         f_MEM_OUT0 <= (others => '0');
1369         f_MEM_READ_SEL <= "10";
1371         f_MEM_WRITE_SEL <= "10";
1373         f_MEM_ADDRESS_SEL <= "10";
1375         f_MEM_READY_SEL <= "01";
1377         f_MEM_DONE_SEL <= "01";
1379         f_MEM_READY0_SEL <= "00";
1381         f_MEM_READY12_SEL <= '0';
1383         f_MEM_RESET0_SEL <= "00";
1385         f_MEM_RESET12_SEL <= "00";

1387         — Wait for TMR Voter Write signal and watch out
for an Error
1389         when s_c fsm_6 =>
1391         f_MEM_ADDRESS <= (others => '0');
1393         f_MEM_IN <= (others => '0');
1395         f_MEM_OUT <= "1010110000011111" &
k_mem_location14_16;
1397         f_MEM_OUT0 <= (others => '0');
1399         f_MEM_READ_SEL <= "10";
1401         f_MEM_WRITE_SEL <= "10";
1403         f_MEM_ADDRESS_SEL <= "10";
1405         f_MEM_READY_SEL <= "01";
1407         f_MEM_DONE_SEL <= "01";
1409         f_MEM_READY0_SEL <= "00";
1411         f_MEM_READY12_SEL <= '0';
1413         f_MEM_RESET0_SEL <= "00";
1415         f_MEM_RESET12_SEL <= "00";

1417         — Wait for TMR Voter Write signal to return to 0
1419         when s_c fsm_7 =>
1421         f_MEM_ADDRESS <= (others => '0');
1423         f_MEM_IN <= (others => '0');
1425         f_MEM_OUT <= (others => '0');
1427         f_MEM_OUT0 <= (others => '0');
1429         f_MEM_READ_SEL <= "10";
1431         f_MEM_WRITE_SEL <= "10";
1433         f_MEM_ADDRESS_SEL <= "10";
1435         f_MEM_READY_SEL <= "10";

```

```

1405     f_MEM_DONE_SEL <= "01";
1406     f_MEM_READY0_SEL <= "00";
1407     f_MEM_READY12_SEL <= '0';
1408     f_MEM_RESET0_SEL <= "00";
1409     f_MEM_RESET12_SEL <= "00";

1411 — Add 1 to the loop count
1412 when s_c fsm_8 =>
1413     f_MEM_ADDRESS <= (others => '0');
1414     f_MEM_IN <= (others => '0');
1415     f_MEM_OUT <= (others => '0');
1416     f_MEM_OUT0 <= (others => '0');
1417     f_MEM_READ_SEL <= "10";
1418     f_MEM_WRITE_SEL <= "10";
1419     f_MEM_ADDRESS_SEL <= "10";
1420     f_MEM_READY_SEL <= "01";
1421     f_MEM_DONE_SEL <= "01";
1422     f_MEM_READY0_SEL <= "00";
1423     f_MEM_READY12_SEL <= '0';
1424     f_MEM_RESET0_SEL <= "00";
1425     f_MEM_RESET12_SEL <= "00";

1427 — Wait for Memory Ready Signal
1428 when s_c fsm_9 =>
1429     f_MEM_ADDRESS <= k_mem_location14_32;
1430     f_MEM_IN <= std_logic_vector(f_loop);
1431     f_MEM_OUT <= (others => '0');
1432     f_MEM_OUT0 <= (others => '0');
1433     f_MEM_READ_SEL <= "10";
1434     f_MEM_WRITE_SEL <= "11";
1435     f_MEM_ADDRESS_SEL <= "11";
1436     f_MEM_READY_SEL <= "01";
1437     f_MEM_DONE_SEL <= "01";
1438     f_MEM_READY0_SEL <= "00";
1439     f_MEM_READY12_SEL <= '0';
1440     f_MEM_RESET0_SEL <= "00";
1441     f_MEM_RESET12_SEL <= "00";

1443
1445 — Wait for Memory Ready signal to return to 0
1446 when s_c fsm_10 =>

```

```

1447     f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
1449     f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= (others => '0');
1451     f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
1453     f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
1455     f_MEM_DONE_SEL <= "01";
        f_MEM_READY0_SEL <= "00";
1457     f_MEM_READY12_SEL <= '0';
        f_MEM_RESET0_SEL <= "00";
1459     f_MEM_RESET12_SEL <= "00";

```

— Wait for Memory Ready Signal

```

1463     when s_c fsm_11 =>
        f_MEM_ADDRESS <= k_mem_location29_32;
1465     f_MEM_IN <= std_logic_vector(f_loop);
        f_MEM_OUT <= (others => '0');
1467     f_MEM_OUT0 <= (others => '0');
        f_MEM_READ_SEL <= "10";
1469     f_MEM_WRITE_SEL <= "11";
        f_MEM_ADDRESS_SEL <= "11";
1471     f_MEM_READY_SEL <= "01";
        f_MEM_DONE_SEL <= "01";
1473     f_MEM_READY0_SEL <= "00";
        f_MEM_READY12_SEL <= '0';
1475     f_MEM_RESET0_SEL <= "00";
        f_MEM_RESET12_SEL <= "00";
1477

```

— Wait for Memory Ready signal to return to 0

```

1479     when s_c fsm_12 =>
        f_MEM_ADDRESS <= (others => '0');
1481     f_MEM_IN <= (others => '0');
        f_MEM_OUT <= (others => '0');
1483     f_MEM_OUT0 <= (others => '0');
        f_MEM_READ_SEL <= "10";
1485     f_MEM_WRITE_SEL <= "10";
        f_MEM_ADDRESS_SEL <= "10";
1487     f_MEM_READY_SEL <= "01";
        f_MEM_DONE_SEL <= "01";
1489

```

```

1491     f_MEM_READY0_SEL <= "00";
1492     f_MEM_READY12_SEL <= '0';
1493     f_MEM_RESET0_SEL <= "00";
1494     f_MEM_RESET12_SEL <= "00";
1495
1496     — Wait for Memory Ready Signal
1497     when s_c fsm_13 =>
1498         f_MEM_ADDRESS <= k_mem_location30_32;
1499         f_MEM_IN <= k_zero_32;
1500         f_MEM_OUT <= (others => '0');
1501         f_MEM_OUT0 <= (others => '0');
1502         f_MEM_READ_SEL <= "10";
1503         f_MEM_WRITE_SEL <= "11";
1504         f_MEM_ADDRESS_SEL <= "11";
1505         f_MEM_READY_SEL <= "01";
1506         f_MEM_DONE_SEL <= "01";
1507         f_MEM_READY0_SEL <= "00";
1508         f_MEM_READY12_SEL <= '0';
1509         f_MEM_RESET0_SEL <= "00";
1510         f_MEM_RESET12_SEL <= "00";
1511
1512     — Wait for Memory Ready signal to return to 0
1513     when s_c fsm_14 =>
1514         f_MEM_ADDRESS <= (others => '0');
1515         f_MEM_IN <= (others => '0');
1516         f_MEM_OUT <= (others => '0');
1517         f_MEM_OUT0 <= (others => '0');
1518         f_MEM_READ_SEL <= "10";
1519         f_MEM_WRITE_SEL <= "10";
1520         f_MEM_ADDRESS_SEL <= "10";
1521         f_MEM_READY_SEL <= "01";
1522         f_MEM_DONE_SEL <= "01";
1523         f_MEM_READY0_SEL <= "00";
1524         f_MEM_READY12_SEL <= '0';
1525         f_MEM_RESET0_SEL <= "00";
1526         f_MEM_RESET12_SEL <= "00";
1527
1528     — Reset all processors
1529     when s_c fsm_15 =>
1530         f_MEM_ADDRESS <= (others => '0');
1531         f_MEM_IN <= (others => '0');

```

```

1533     f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= (others => '0');
1535     f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
1537     f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
1539     f_MEM_DONE_SEL <= "11";
        f_MEM_READY0_SEL <= "10";
1541     f_MEM_READY12_SEL <= '1';
        f_MEM_RESET0_SEL <= "11";
1543     f_MEM_RESET12_SEL <= "11";

```

— Wait for MIPS0 Read Signal

```

1545     when s_c fsm_16 =>
1547         f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
1549         f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= (others => '0');
1551         f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
1553         f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
1555         f_MEM_DONE_SEL <= "11";
        f_MEM_READY0_SEL <= "10";
1557         f_MEM_READY12_SEL <= '1';
        f_MEM_RESET0_SEL <= "10";
1559         f_MEM_RESET12_SEL <= "11";

```

— Wait for MIPS0 Read Signal to return to 0

```

1561     when s_c fsm_17 =>
1563         f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
1565         f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= "
10001100000111110000000000000000";
1567         f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
1569         f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
1571         f_MEM_DONE_SEL <= "11";
        f_MEM_READY0_SEL <= "11";
1573         f_MEM_READY12_SEL <= '1';
        f_MEM_RESET0_SEL <= "10";

```

```

1575         f_MEM_RESET12_SEL <= "11";

1577     — Wait for MIPS0 Read Signal
    when s_c fsm_18 =>
1579         f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
1581         f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= "
10001100000111110000000000000000";
1583         f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
1585         f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
1587         f_MEM_DONE_SEL <= "11";
        f_MEM_READY0_SEL <= "10";
1589         f_MEM_READY12_SEL <= '1';
        f_MEM_RESET0_SEL <= "10";
1591         f_MEM_RESET12_SEL <= "11";

1593     — Wait for MIPS0 Read Signal to return to 0
    when s_c fsm_19 =>
1595         f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
1597         f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= std_logic_vector(f_loop1);
1599         f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";
1601         f_MEM_ADDRESS_SEL <= "10";
        f_MEM_READY_SEL <= "01";
1603         f_MEM_DONE_SEL <= "11";
        f_MEM_READY0_SEL <= "11";
1605         f_MEM_READY12_SEL <= '1';
        f_MEM_RESET0_SEL <= "10";
1607         f_MEM_RESET12_SEL <= "11";

1609     — Wait for MIPS0 Read Signal
    when s_c fsm_20 =>
1611         f_MEM_ADDRESS <= (others => '0');
        f_MEM_IN <= (others => '0');
1613         f_MEM_OUT <= (others => '0');
        f_MEM_OUT0 <= std_logic_vector(f_loop1);
1615         f_MEM_READ_SEL <= "10";
        f_MEM_WRITE_SEL <= "10";

```

```

1617         f_MEM_ADDRESS_SEL <= "10";
1618         f_MEM_READY_SEL <= "01";
1619         f_MEM_DONE_SEL <= "11";
1620         f_MEM_READY0_SEL <= "10";
1621         f_MEM_READY12_SEL <= '1';
1622         f_MEM_RESET0_SEL <= "10";
1623         f_MEM_RESET12_SEL <= "11";

1624     — Wait for MIPS0 Read Signal to return to 0
1625     when s_cfsm_21 =>
1626         f_MEM_ADDRESS <= (others => '0');
1627         f_MEM_IN <= (others => '0');
1628         f_MEM_OUT <= (others => '0');
1629         f_MEM_OUT0 <= "
00100011111111100000000000000000";
1630         f_MEM_READ_SEL <= "10";
1631         f_MEM_WRITE_SEL <= "10";
1632         f_MEM_ADDRESS_SEL <= "10";
1633         f_MEM_READY_SEL <= "01";
1634         f_MEM_DONE_SEL <= "11";
1635         f_MEM_READY0_SEL <= "11";
1636         f_MEM_READY12_SEL <= '1';
1637         f_MEM_RESET0_SEL <= "10";
1638         f_MEM_RESET12_SEL <= "11";

1639

1640     — Wait for MIPS0 Read Signal
1641     when s_cfsm_22 =>
1642         f_MEM_ADDRESS <= (others => '0');
1643         f_MEM_IN <= (others => '0');
1644         f_MEM_OUT <= (others => '0');
1645         f_MEM_OUT0 <= "
00100011111111100000000000000000";
1646         f_MEM_READ_SEL <= "10";
1647         f_MEM_WRITE_SEL <= "10";
1648         f_MEM_ADDRESS_SEL <= "10";
1649         f_MEM_READY_SEL <= "01";
1650         f_MEM_DONE_SEL <= "11";
1651         f_MEM_READY0_SEL <= "10";
1652         f_MEM_READY12_SEL <= '1';
1653         f_MEM_RESET0_SEL <= "10";
1654         f_MEM_RESET12_SEL <= "11";
1655
1656
1657

```

```

1659      — Wait for MIPS0 Read signal to return to 0
1660      when s_c fsm_23 =>
1661          f_MEM_ADDRESS <= (others => '0');
1662          f_MEM_IN <= (others => '0');
1663          f_MEM_OUT <= (others => '0');
1664          f_MEM_OUT0 <= "0001000000000000" &
k_TSR_LOOP_START_BRANCH;
1665          f_MEM_READ_SEL <= "10";
1666          f_MEM_WRITE_SEL <= "10";
1667          f_MEM_ADDRESS_SEL <= "10";
1668          f_MEM_READY_SEL <= "01";
1669          f_MEM_DONE_SEL <= "11";
1670          f_MEM_READY0_SEL <= "11";
1671          f_MEM_READY12_SEL <= '1';
1672          f_MEM_RESET0_SEL <= "10";
1673          f_MEM_RESET12_SEL <= "11";
1674
1675      — Wait for MIPS0 Read signal
1676      when s_c fsm_24 =>
1677          f_MEM_ADDRESS <= (others => '0');
1678          f_MEM_IN <= (others => '0');
1679          f_MEM_OUT <= (others => '0');
1680          f_MEM_OUT0 <= "0001000000000000" &
k_TSR_LOOP_START_BRANCH;
1681          f_MEM_READ_SEL <= "10";
1682          f_MEM_WRITE_SEL <= "10";
1683          f_MEM_ADDRESS_SEL <= "10";
1684          f_MEM_READY_SEL <= "01";
1685          f_MEM_DONE_SEL <= "11";
1686          f_MEM_READY0_SEL <= "10";
1687          f_MEM_READY12_SEL <= '1';
1688          f_MEM_RESET0_SEL <= "10";
1689          f_MEM_RESET12_SEL <= "11";
1690
1691      — TSR MIPS is Running
1692      — Wait for an error to occur or the program to
finish
1693
1694      when s_c fsm_25 =>
1695          f_MEM_ADDRESS <= (others => '0');
1696          f_MEM_IN <= (others => '0');
1697

```



```

1699     f_MEM_OUT <= (others => '0');
1701     f_MEM_OUT0 <= (others => '0');
1703     f_MEM_READ_SEL <= "01";
1705     f_MEM_WRITE_SEL <= "01";
1707     f_MEM_ADDRESS_SEL <= "01";
1709     f_MEM_READY_SEL <= "01";
1711     f_MEM_DONE_SEL <= "11";
1713     f_MEM_READY0_SEL <= "01";
1715     f_MEM_READY12_SEL <= '1';
1717     f_MEM_RESET0_SEL <= "01";
1719     f_MEM_RESET12_SEL <= "11";

```

— An error has occurred in TSR MIPS

```

1721 when s_c fsm_25a =>
1723     f_MEM_ADDRESS <= (others => '0');
1725     f_MEM_IN <= (others => '0');
1727     f_MEM_OUT <= (others => '0');
1729     f_MEM_OUT0 <= (others => '0');
1731     f_MEM_READ_SEL <= "01";
1733     f_MEM_WRITE_SEL <= "01";
1735     f_MEM_ADDRESS_SEL <= "01";
1737     f_MEM_READY_SEL <= "01";
1739     f_MEM_DONE_SEL <= "11";
1741     f_MEM_READY0_SEL <= "01";
1743     f_MEM_READY12_SEL <= '1';
1745     f_MEM_RESET0_SEL <= "01";
1747     f_MEM_RESET12_SEL <= "11";

```

Memory. — Interrupt communications between MIPS0 and  
Wait for MIPS0 Read Signal

```

1727 when s_c fsm_26 =>
1729     f_MEM_ADDRESS <= (others => '0');
1731     f_MEM_IN <= (others => '0');
1733     f_MEM_OUT <= (others => '0');
1735     f_MEM_OUT0 <= (others => '0');
1737     f_MEM_READ_SEL <= "10";
1739     f_MEM_WRITE_SEL <= "10";
1741     f_MEM_ADDRESS_SEL <= "10";
1743     f_MEM_READY_SEL <= "01";
1745     f_MEM_DONE_SEL <= "11";
1747     f_MEM_READY0_SEL <= "10";
1749     f_MEM_READY12_SEL <= '1';
1751     f_MEM_RESET0_SEL <= "10";

```

```

1741         f_MEM_RESET12_SEL <= "11";
1743         — Transmit branch instruction to branch to TSR
Start. Wait for MIPS0 Read Signal to return to 0
1743         when s_c fsm_27 =>
1745             f_MEM_ADDRESS <= (others => '0');
1745             f_MEM_IN <= (others => '0');
1745             f_MEM_OUT <= (others => '0');
1747             f_MEM_OUT0 <= "0001000000000000" &
k_TSR_START_BRANCH;
1749             f_MEM_READ_SEL <= "10";
1749             f_MEM_WRITE_SEL <= "10";
1749             f_MEM_ADDRESS_SEL <= "10";
1751             f_MEM_READY_SEL <= "01";
1751             f_MEM_DONE_SEL <= "11";
1753             f_MEM_READY0_SEL <= "11";
1753             f_MEM_READY12_SEL <= '1';
1755             f_MEM_RESET0_SEL <= "10";
1755             f_MEM_RESET12_SEL <= "11";
1757
1757         — Wait for MIPS0 Read Signal
1759         when s_c fsm_28 =>
1759             f_MEM_ADDRESS <= (others => '0');
1761             f_MEM_IN <= (others => '0');
1761             f_MEM_OUT <= (others => '0');
1763             f_MEM_OUT0 <= "0001000000000000" &
k_TSR_START_BRANCH;
1765             f_MEM_READ_SEL <= "10";
1765             f_MEM_WRITE_SEL <= "10";
1765             f_MEM_ADDRESS_SEL <= "10";
1767             f_MEM_READY_SEL <= "01";
1767             f_MEM_DONE_SEL <= "11";
1769             f_MEM_READY0_SEL <= "10";
1769             f_MEM_READY12_SEL <= '1';
1771             f_MEM_RESET0_SEL <= "10";
1771             f_MEM_RESET12_SEL <= "11";
1773
1775         — Start transition from TSR MIPS to TMR MIPS
1775         — Wait for Memory Ready Signal
1777         when s_c fsm_29 =>
1777             f_MEM_ADDRESS <= (others => '0');
1779             f_MEM_IN <= (others => '0');

```

```

1781     f_MEM_OUT <= (others => '0');
1782     f_MEM_OUT0 <= (others => '0');
1783     f_MEM_READ_SEL <= "10";
1784     f_MEM_WRITE_SEL <= "10";
1785     f_MEM_ADDRESS_SEL <= "10";
1786     f_MEM_READY_SEL <= "01";
1787     f_MEM_DONE_SEL <= "11";
1788     f_MEM_READY0_SEL <= "10";
1789     f_MEM_READY12_SEL <= '1';
1790     f_MEM_RESET0_SEL <= "11";
1791     f_MEM_RESET12_SEL <= "11";

```

— Wait for Memory Ready signal to return to 0

```

1793     when s_c fsm_30 =>
1794         f_MEM_ADDRESS <= (others => '0');
1795         f_MEM_IN <= (others => '0');
1796         f_MEM_OUT <= (others => '0');
1797         f_MEM_OUT0 <= (others => '0');
1798         f_MEM_READ_SEL <= "10";
1799         f_MEM_WRITE_SEL <= "10";
1800         f_MEM_ADDRESS_SEL <= "10";
1801         f_MEM_READY_SEL <= "01";
1802         f_MEM_DONE_SEL <= "11";
1803         f_MEM_READY0_SEL <= "10";
1804         f_MEM_READY12_SEL <= '1';
1805         f_MEM_RESET0_SEL <= "11";
1806         f_MEM_RESET12_SEL <= "11";

```

— Wait for Memory Ready Signal

```

1809     when s_c fsm_31 =>
1810         f_MEM_ADDRESS <= k_mem_location30_32;
1811         f_MEM_IN <= (others => '0');
1812         f_MEM_OUT <= (others => '0');
1813         f_MEM_OUT0 <= (others => '0');
1814         f_MEM_READ_SEL <= "11";
1815         f_MEM_WRITE_SEL <= "10";
1816         f_MEM_ADDRESS_SEL <= "11";
1817         f_MEM_READY_SEL <= "01";
1818         f_MEM_DONE_SEL <= "11";
1819         f_MEM_READY0_SEL <= "10";
1820         f_MEM_READY12_SEL <= '1';
1821         f_MEM_RESET0_SEL <= "11";

```

```

1823         f_MEM_RESET12_SEL <= "11";

1825     — Wait for Memory Ready signal to return to 0
when s_c fsm_32 =>
1827         f_MEM_ADDRESS <= (others => '0');
         f_MEM_IN <= (others => '0');
1829         f_MEM_OUT <= (others => '0');
         f_MEM_OUT0 <= (others => '0');
1831         f_MEM_READ_SEL <= "10";
         f_MEM_WRITE_SEL <= "10";
1833         f_MEM_ADDRESS_SEL <= "10";
         f_MEM_READY_SEL <= "01";
1835         f_MEM_DONE_SEL <= "11";
         f_MEM_READY0_SEL <= "10";
1837         f_MEM_READY12_SEL <= '1';
         f_MEM_RESET0_SEL <= "11";
1839         f_MEM_RESET12_SEL <= "11";

1841     — Wait for Memory Ready Signal
when s_c fsm_33 =>
1843         f_MEM_ADDRESS <= std_logic_vector(unsigned(
k_mem_location14_32)+unsigned(f_TEMP_ADDRESS));
         f_MEM_IN <= (others => '0');
1845         f_MEM_OUT <= (others => '0');
         f_MEM_OUT0 <= (others => '0');
1847         f_MEM_READ_SEL <= "11";
         f_MEM_WRITE_SEL <= "10";
1849         f_MEM_ADDRESS_SEL <= "11";
         f_MEM_READY_SEL <= "01";
1851         f_MEM_DONE_SEL <= "11";
         f_MEM_READY0_SEL <= "10";
1853         f_MEM_READY12_SEL <= '1';
         f_MEM_RESET0_SEL <= "11";
1855         f_MEM_RESET12_SEL <= "11";

1857     — Wait for Memory Ready signal to return to 0
when s_c fsm_34 =>
1859         f_MEM_ADDRESS <= (others => '0');
         f_MEM_IN <= (others => '0');
1861         f_MEM_OUT <= (others => '0');
         f_MEM_OUT0 <= (others => '0');
1863         f_MEM_READ_SEL <= "10";
         f_MEM_WRITE_SEL <= "10";

```

```

1865         f_MEM_ADDRESS_SEL <= "10";
1866         f_MEM_READY_SEL <= "01";
1867         f_MEM_DONE_SEL <= "11";
1868         f_MEM_READY0_SEL <= "10";
1869         f_MEM_READY12_SEL <= '1';
1870         f_MEM_RESET0_SEL <= "11";
1871         f_MEM_RESET12_SEL <= "11";

1873
1874     — Subtract 1 from the loop count
1875     when s_c fsm_35 =>
1876         f_MEM_ADDRESS <= (others => '0');
1877         f_MEM_IN <= (others => '0');
1878         f_MEM_OUT <= (others => '0');
1879         f_MEM_OUT0 <= (others => '0');
1880         f_MEM_READ_SEL <= "10";
1881         f_MEM_WRITE_SEL <= "10";
1882         f_MEM_ADDRESS_SEL <= "10";
1883         f_MEM_READY_SEL <= "01";
1884         f_MEM_DONE_SEL <= "11";
1885         f_MEM_READY0_SEL <= "10";
1886         f_MEM_READY12_SEL <= '1';
1887         f_MEM_RESET0_SEL <= "11";
1888         f_MEM_RESET12_SEL <= "11";

1889
1890     — Wait for Memory Ready Signal
1891     when s_c fsm_36 =>
1892         f_MEM_ADDRESS <= k_mem_location30_32;
1893         f_MEM_IN <= std_logic_vector(f_loop);
1894         f_MEM_OUT <= (others => '0');
1895         f_MEM_OUT0 <= (others => '0');
1896         f_MEM_READ_SEL <= "10";
1897         f_MEM_WRITE_SEL <= "11";
1898         f_MEM_ADDRESS_SEL <= "11";
1899         f_MEM_READY_SEL <= "01";
1900         f_MEM_DONE_SEL <= "11";
1901         f_MEM_READY0_SEL <= "10";
1902         f_MEM_READY12_SEL <= '1';
1903         f_MEM_RESET0_SEL <= "11";
1904         f_MEM_RESET12_SEL <= "11";

1905
1906     — Wait for Memory Ready signal to return to 0

```

```

1909     when s_c fsm_37 =>
1910         f_MEM_ADDRESS <= (others => '0');
1911         f_MEM_IN <= (others => '0');
1912         f_MEM_OUT <= (others => '0');
1913         f_MEM_OUT0 <= (others => '0');
1914         f_MEM_READ_SEL <= "10";
1915         f_MEM_WRITE_SEL <= "10";
1916         f_MEM_ADDRESS_SEL <= "10";
1917         f_MEM_READY_SEL <= "01";
1918         f_MEM_DONE_SEL <= "11";
1919         f_MEM_READY0_SEL <= "10";
1920         f_MEM_READY12_SEL <= '1';
1921         f_MEM_RESET0_SEL <= "11";
1922         f_MEM_RESET12_SEL <= "11";

```

— Wait for Memory Ready Signal

```

1923     when s_c fsm_38 =>
1924         f_MEM_ADDRESS <= k_mem_location62_32;
1925         f_MEM_IN <= std_logic_vector(f_loop);
1926         f_MEM_OUT <= (others => '0');
1927         f_MEM_OUT0 <= (others => '0');
1928         f_MEM_READ_SEL <= "10";
1929         f_MEM_WRITE_SEL <= "11";
1930         f_MEM_ADDRESS_SEL <= "11";
1931         f_MEM_READY_SEL <= "01";
1932         f_MEM_DONE_SEL <= "11";
1933         f_MEM_READY0_SEL <= "10";
1934         f_MEM_READY12_SEL <= '1';
1935         f_MEM_RESET0_SEL <= "11";
1936         f_MEM_RESET12_SEL <= "11";

```

— Wait for Memory Ready signal to return to 0

```

1939     when s_c fsm_39 =>
1940         f_MEM_ADDRESS <= (others => '0');
1941         f_MEM_IN <= (others => '0');
1942         f_MEM_OUT <= (others => '0');
1943         f_MEM_OUT0 <= (others => '0');
1944         f_MEM_READ_SEL <= "10";
1945         f_MEM_WRITE_SEL <= "10";
1946         f_MEM_ADDRESS_SEL <= "10";
1947         f_MEM_READY_SEL <= "01";
1948         f_MEM_DONE_SEL <= "11";
1949         f_MEM_READY0_SEL <= "10";

```

```

1951         f_MEM_READY12_SEL <= '1';
1952         f_MEM_RESET0_SEL <= "11";
1953         f_MEM_RESET12_SEL <= "11";

1955     — Wait for Memory Ready Signal
1956     when s_c fsm_40 =>
1957         f_MEM_ADDRESS <= k_mem_location31_32;
1958         f_MEM_IN <= k_mem_location02_32;
1959         f_MEM_OUT <= (others => '0');
1960         f_MEM_OUT0 <= (others => '0');
1961         f_MEM_READ_SEL <= "10";
1962         f_MEM_WRITE_SEL <= "11";
1963         f_MEM_ADDRESS_SEL <= "11";
1964         f_MEM_READY_SEL <= "01";
1965         f_MEM_DONE_SEL <= "11";
1966         f_MEM_READY0_SEL <= "10";
1967         f_MEM_READY12_SEL <= '1';
1968         f_MEM_RESET0_SEL <= "11";
1969         f_MEM_RESET12_SEL <= "11";

1971     — Wait for Memory Ready signal to return to 0
1972     when s_c fsm_41 =>
1973         f_MEM_ADDRESS <= (others => '0');
1974         f_MEM_IN <= (others => '0');
1975         f_MEM_OUT <= (others => '0');
1976         f_MEM_OUT0 <= (others => '0');
1977         f_MEM_READ_SEL <= "10";
1978         f_MEM_WRITE_SEL <= "10";
1979         f_MEM_ADDRESS_SEL <= "10";
1980         f_MEM_READY_SEL <= "01";
1981         f_MEM_DONE_SEL <= "11";
1982         f_MEM_READY0_SEL <= "10";
1983         f_MEM_READY12_SEL <= '1';
1984         f_MEM_RESET0_SEL <= "11";
1985         f_MEM_RESET12_SEL <= "11";

1987     — Wait for Memory Ready Signal
1988     when s_c fsm_42 =>
1989         f_MEM_ADDRESS <= k_mem_location63_32;
1990         f_MEM_IN <= k_mem_location02_32;
1991         f_MEM_OUT <= (others => '0');
1992         f_MEM_OUT0 <= (others => '0');
1993         f_MEM_READ_SEL <= "10";

```

```

1995     f_MEM_WRITE_SEL <= "11";
1996     f_MEM_ADDRESS_SEL <= "11";
1997     f_MEM_READY_SEL <= "01";
1998     f_MEM_DONE_SEL <= "11";
1999     f_MEM_READY0_SEL <= "10";
2000     f_MEM_READY12_SEL <= '1';
2001     f_MEM_RESET0_SEL <= "11";
2002     f_MEM_RESET12_SEL <= "11";

2003     — Wait for Memory Ready signal to return to 0
2004     when s_c fsm_43 =>
2005         f_MEM_ADDRESS <= (others => '0');
2006         f_MEM_IN <= (others => '0');
2007         f_MEM_OUT <= (others => '0');
2008         f_MEM_OUT0 <= (others => '0');
2009         f_MEM_READ_SEL <= "10";
2010         f_MEM_WRITE_SEL <= "10";
2011         f_MEM_ADDRESS_SEL <= "10";
2012         f_MEM_READY_SEL <= "01";
2013         f_MEM_DONE_SEL <= "11";
2014         f_MEM_READY0_SEL <= "10";
2015         f_MEM_READY12_SEL <= '1';
2016         f_MEM_RESET0_SEL <= "11";
2017         f_MEM_RESET12_SEL <= "11";

2019     — Wait for Memory Ready Signal
2020     when s_c fsm_44 =>
2021         f_MEM_ADDRESS <= k_mem_location64_32;
2022         f_MEM_IN <= k_zero_32;
2023         f_MEM_OUT <= (others => '0');
2024         f_MEM_OUT0 <= (others => '0');
2025         f_MEM_READ_SEL <= "10";
2026         f_MEM_WRITE_SEL <= "11";
2027         f_MEM_ADDRESS_SEL <= "11";
2028         f_MEM_READY_SEL <= "01";
2029         f_MEM_DONE_SEL <= "11";
2030         f_MEM_READY0_SEL <= "10";
2031         f_MEM_READY12_SEL <= '1';
2032         f_MEM_RESET0_SEL <= "11";
2033         f_MEM_RESET12_SEL <= "11";

2035     — Wait for Memory Ready signal to return to 0
2036     when s_c fsm_45 =>

```



```

2037         f_MEM_ADDRESS <= (others => '0');
2039         f_MEM_IN <= (others => '0');
2041         f_MEM_OUT <= (others => '0');
2043         f_MEM_OUT0 <= (others => '0');
2045         f_MEM_READ_SEL <= "10";
2047         f_MEM_WRITE_SEL <= "10";
2049         f_MEM_ADDRESS_SEL <= "10";
2051         f_MEM_READY_SEL <= "01";
2053         f_MEM_DONE_SEL <= "11";
2055         f_MEM_READY0_SEL <= "10";
2057         f_MEM_READY12_SEL <= '1';
2059         f_MEM_RESET0_SEL <= "11";
2061         f_MEM_RESET12_SEL <= "11";

2063     — Wait for Voter Read Signal
2065     when s_cfsm_46 =>
2067         f_MEM_ADDRESS <= (others => '0');
2069         f_MEM_IN <= (others => '0');
2071         f_MEM_OUT <= (others => '0');
2073         f_MEM_OUT0 <= (others => '0');
2075         f_MEM_READ_SEL <= "10";
2077         f_MEM_WRITE_SEL <= "10";
2079         f_MEM_ADDRESS_SEL <= "10";
2081         f_MEM_READY_SEL <= "01";
2083         f_MEM_DONE_SEL <= "01";
2085         f_MEM_READY0_SEL <= "00";
2087         f_MEM_READY12_SEL <= '0';
2089         f_MEM_RESET0_SEL <= "10";
2091         f_MEM_RESET12_SEL <= "01";

2093     — Wait for Voter Read Signal to return to 0
2095     when s_cfsm_47 =>
2097         f_MEM_ADDRESS <= (others => '0');
2099         f_MEM_IN <= (others => '0');
2101         f_MEM_OUT <= "10001100000111110000000000000000
";
2103         f_MEM_OUT0 <= (others => '0');
2105         f_MEM_READ_SEL <= "10";
2107         f_MEM_WRITE_SEL <= "10";
2109         f_MEM_ADDRESS_SEL <= "10";
2111         f_MEM_READY_SEL <= "11";
2113         f_MEM_DONE_SEL <= "01";
2115         f_MEM_READY0_SEL <= "00";

```

```

2079         f_MEM_READY12_SEL <= '0';
2080         f_MEM_RESET0_SEL <= "10";
2081         f_MEM_RESET12_SEL <= "01";

2082
2083     — Wait for Voter Read Signal
2084     when s_c fsm_48 =>
2085         f_MEM_ADDRESS <= (others => '0');
2086         f_MEM_IN <= (others => '0');
2087         f_MEM_OUT <= (others => '0');
2088         f_MEM_OUT0 <= (others => '0');
2089         f_MEM_READ_SEL <= "10";
2090         f_MEM_WRITE_SEL <= "10";
2091         f_MEM_ADDRESS_SEL <= "10";
2092         f_MEM_READY_SEL <= "01";
2093         f_MEM_DONE_SEL <= "01";
2094         f_MEM_READY0_SEL <= "00";
2095         f_MEM_READY12_SEL <= '0';
2096         f_MEM_RESET0_SEL <= "10";
2097         f_MEM_RESET12_SEL <= "01";

2098
2099     — Wait for Voter Read Signal to return to 0
2100     when s_c fsm_49 =>
2101         f_MEM_ADDRESS <= (others => '0');
2102         f_MEM_IN <= (others => '0');
2103         f_MEM_OUT <= std_logic_vector(f_loop);
2104         f_MEM_OUT0 <= (others => '0');
2105         f_MEM_READ_SEL <= "10";
2106         f_MEM_WRITE_SEL <= "10";
2107         f_MEM_ADDRESS_SEL <= "10";
2108         f_MEM_READY_SEL <= "11";
2109         f_MEM_DONE_SEL <= "01";
2110         f_MEM_READY0_SEL <= "00";
2111         f_MEM_READY12_SEL <= '0';
2112         f_MEM_RESET0_SEL <= "10";
2113         f_MEM_RESET12_SEL <= "01";

2114
2115     — Wait for Voter Read Signal
2116     when s_c fsm_50 =>
2117         f_MEM_ADDRESS <= (others => '0');
2118         f_MEM_IN <= (others => '0');
2119         f_MEM_OUT <= (others => '0');
2120         f_MEM_OUT0 <= (others => '0');
2121

```

```

2123     f_MEM_READ_SEL <= "10";
2124     f_MEM_WRITE_SEL <= "10";
2125     f_MEM_ADDRESS_SEL <= "10";
2126     f_MEM_READY_SEL <= "01";
2127     f_MEM_DONE_SEL <= "01";
2128     f_MEM_READY0_SEL <= "00";
2129     f_MEM_READY12_SEL <= '0';
2130     f_MEM_RESET0_SEL <= "10";
2131     f_MEM_RESET12_SEL <= "01";
2132
2133     — Wait for Voter Read Signal to return to 0
2134     when s_c fsm_51 =>
2135         f_MEM_ADDRESS <= (others => '0');
2136         f_MEM_IN <= (others => '0');
2137         f_MEM_OUT <= k_zero_32;
2138         f_MEM_OUT0 <= (others => '0');
2139         f_MEM_READ_SEL <= "10";
2140         f_MEM_WRITE_SEL <= "10";
2141         f_MEM_ADDRESS_SEL <= "10";
2142         f_MEM_READY_SEL <= "11";
2143         f_MEM_DONE_SEL <= "01";
2144         f_MEM_READY0_SEL <= "00";
2145         f_MEM_READY12_SEL <= '0';
2146         f_MEM_RESET0_SEL <= "10";
2147         f_MEM_RESET12_SEL <= "01";
2148
2149     — Return to normal TMR operation
2150     when s_c fsm_52 =>
2151         f_MEM_ADDRESS <= (others => '0');
2152         f_MEM_IN <= (others => '0');
2153         f_MEM_OUT <= (others => '0');
2154         f_MEM_OUT0 <= (others => '0');
2155         f_MEM_READ_SEL <= "10";
2156         f_MEM_WRITE_SEL <= "10";
2157         f_MEM_ADDRESS_SEL <= "10";
2158         f_MEM_READY_SEL <= "01";
2159         f_MEM_DONE_SEL <= "01";
2160         f_MEM_READY0_SEL <= "00";
2161         f_MEM_READY12_SEL <= '0';
2162         f_MEM_RESET0_SEL <= "10";
2163         f_MEM_RESET12_SEL <= "01";
2164
2165     — This should never happen

```

```

2165     when others =>
2166         f_MEM_ADDRESS <= (others => '0');
2167         f_MEM_IN <= (others => '0');
2168         f_MEM_OUT <= (others => '0');
2169         f_MEM_OUT0 <= (others => '0');
2170         f_MEM_READ_SEL <= "00";
2171         f_MEM_WRITE_SEL <= "00";
2172         f_MEM_ADDRESS_SEL <= "00";
2173         f_MEM_READY_SEL <= "00";
2174         f_MEM_DONE_SEL <= "00";
2175         f_MEM_READY0_SEL <= "00";
2176         f_MEM_READY12_SEL <= '0';
2177         f_MEM_RESET0_SEL <= "00";
2178         f_MEM_RESET12_SEL <= "00";
2179     end case;
2180 end if;
2181 end process controller_output_fsm;
2182
2183 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

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

<b>1. REPORT DATE (DD-MM-YYYY)</b> 12-09-2019		<b>2. REPORT TYPE</b> Technical Report		<b>3. DATES COVERED (From — To)</b> Sept 2016 — Sept 2019	
<b>4. TITLE AND SUBTITLE</b>  Adaptive-Hybrid Redundancy MIPS Architecture Version 2.2				<b>5a. CONTRACT NUMBER</b>	
				<b>5b. GRANT NUMBER</b>	
				<b>5c. PROGRAM ELEMENT NUMBER</b>	
				<b>5d. PROJECT NUMBER</b> 18G169C	
				<b>5e. TASK NUMBER</b>	
<b>6. AUTHOR(S)</b>  Hamilton, Nicolas S, Maj, USAF				<b>5f. WORK UNIT NUMBER</b>	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Air Force Institute of Technology Graduate School of Engineering an Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>  AFIT/EN/TR-19-04	
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b>  Undisclosed				<b>10. SPONSOR/MONITOR'S ACRONYM(S)</b>	
				<b>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</b>	
<b>12. DISTRIBUTION / AVAILABILITY STATEMENT</b> DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.					
<b>13. SUPPLEMENTARY NOTES</b>					
<b>14. ABSTRACT</b>  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.					
<b>15. SUBJECT TERMS</b>  MIPS, Processor, Adaptive-Hybrid Redundancy					
<b>16. SECURITY CLASSIFICATION OF:</b>			<b>17. LIMITATION OF ABSTRACT</b>	<b>18. NUMBER OF PAGES</b>	<b>19a. NAME OF RESPONSIBLE PERSON</b>
<b>a. REPORT</b>	<b>b. ABSTRACT</b>	<b>c. THIS PAGE</b>			Maj Nicolas Hamilton, AFIT/ENG
U	U	U	UU	119	<b>19b. TELEPHONE NUMBER (include area code)</b> (937) 255-6565 x4220; nicolas.hamilton@afit.edu