MicroBlaze Triple Modular Redundancy (TMR) Subsystem v1.0

Product Guide

Vivado Design Suite

PG268 November 22, 2019

XILINX

Table of Contents

Appendix B: Upgrading

Appendix C: Debugging

Appendix D: Application Software Development

Appendix E: Benchmarks

Appendix F: Additional Resources and Legal Notices

Introduction

The complete high-reliability MicroBlaze™ Triple Modular Redundancy (TMR) solution for Vivado® provides soft error detection, correction and recovery for Xilinx devices.

The guide describes the IP cores that are part of the solution, explains typical use cases, and covers Vivado IP integrator automation.

Features

- Complete Triple Modular Redundancy solution for MicroBlaze providing:
  - TMR Manager to control the overall redundancy state and supervise soft error mitigation
  - TMR Voter to implement a self-checking voter that generates outputs from the triplicated sub-blocks
  - TMR Comparator to implement a self-checking comparison of outputs from the triplicated sub-blocks, and generate errors in the case of mismatch
  - TMR Inject to implement functional level fault injection for test purposes
  - TMR Soft Error Mitigation (SEM) interface, to encapsulate the Xilinx Soft Error Mitigation IP core.
- Vivado IP integrator automation to greatly simplify the creation of a triplicated MicroBlaze subsystem.
- TMR Manager example design.

IP Facts Table	
Subsystem Specifics	
Supported Device Family(1)	UltraScale+™ UltraScale™ Zynq®-7000 SoC 7 Series
Supported User Interfaces	ACE, AXI4-Lite, AXI4-Stream, BRAM, Dynamic Reconfiguration Port (DRP), GPIO, Local Memory Bus (LMB), Interrupt, UART
Resources	Performance and Resource Utilization web page: TMR Manager, TMR Voter, TMR Comparator, TMR Inject, TMR SEM
Provided with Subsystem	
Design Files	RTL
Example Design	VHDL
Test Bench	Not Provided
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver(2)	Standalone
Tested Design Flows(3)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 68483
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72275
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in <install_directory>/Vitis/<release>/data/embeddedsw/doc/xilinx_drivers.htm.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

An increasing number of applications today need a high-reliability processing function to address dependability, safety and security requirements. One such application is use in high-radiation environments, where integrated circuits almost certainly will experience radiation-induced single-event effects.

The Xilinx® Triple Modular Redundancy (TMR) solution for the Vivado® Design Suite is designed for these applications, providing all the necessary building blocks to implement a redundant triplicated MicroBlaze™ subsystem. This processing subsystem is fault tolerant and continues to operate nominally after encountering an error. Together with the capability to detect and recover from errors, the implementation ensures the reliability of the entire subsystem.

In nominal operation all three redundant blocks are working correctly, and outputs are majority voted. When an error is detected in one of the blocks the subsystem enters lockstep mode and keeps operating. Should a second error occur before recovery, it is detected and the subsystem halts with a fatal error.

An essential component of the implementation is the Vivado IP integrator automation, to greatly simplify the creation of a TMR subsystem, which otherwise can be both time consuming and error prone.

Major advantages of the solution are that redundant sub-blocks can be physically separated, that it provides the ability to test error detection, and that it enables rapid software controlled error recovery.

Feature Summary

To implement a complete MicroBlaze TMR subsystem, the following IP cores are provided:

- TMR Manager

  The TMR Manager is responsible for handling the TMR subsystem state, including fault detection and error recovery. The core is triplicated in each of the sub-blocks in the TMR subsystem, and provides majority voting of its internal state.

- TMR Voter

  This core provides the majority voter necessary for all bus interfaces of the triplicated subsystem.

- TMR Comparator

  The comparators implements fault detection of all bus interfaces in the triplicated subsystem, and also includes an optional voter checker to detect faults in the corresponding TMR Voter. The comparators are triplicated in each of the sub-blocks in the TMR subsystem.

- TMR Inject

  The TMR Inject core provides functional fault injection by changing selected MicroBlaze instructions, which provides the possibility to verify that the TMR subsystem error detection and fault recovery logic is working properly.

- TMR SEM

  The TMR Soft Error Mitigation interface implements a wrapper of the Xilinx Soft Error Mitigation functionality to simplify integration in Vivado IP integrator.

In addition to these cores, the I/O Module v3.1 core provides the TMR voting mechanism necessary to preserve its internal state, in order to enable automated error recovery.

Licensing and Ordering

This Xilinx LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

To meet high requirements on availability, safety and tamper protection, the MicroBlaze™ processor can be implemented in a triple-redundant voting scheme. The method is often referred to as Triple Modular Redundancy (TMR). A MicroBlaze TMR subsystem is of Fault Tolerant - Fail Safe (FT-FS) type, which means that it continues to work without degradation after the first failure (FT) and will detect a second failure (FS).

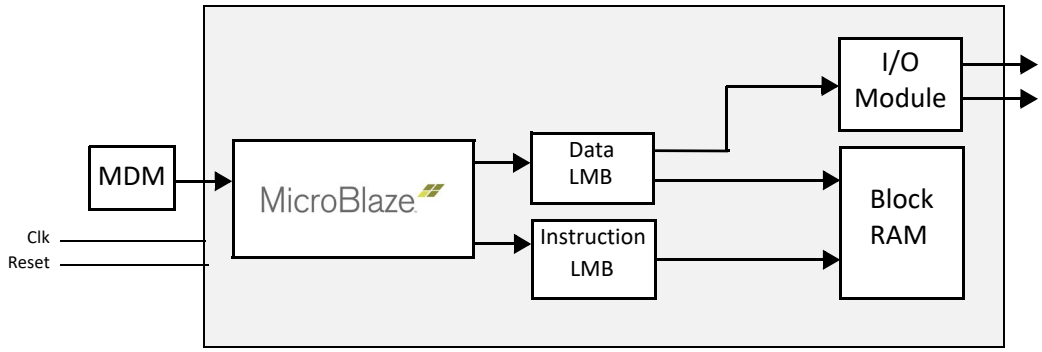

Figure 2-1: **MicroBlaze Micro-controller Subsystem**

The system in Figure 2-1 consists of a MicroBlaze processor with application code and data stored in block RAM. The I/O Module peripherals are connected to MicroBlaze through LMB. To implement an FT-FS scheme for this subsystem, it can be triplicated.

The MicroBlaze Debug Module (MDM) is not triplicated, and all the debug functionality is contained in the first of the three TMR sub-blocks, while the implementation prevents any error in this debug logic from propagating to the other sub-blocks. The triplicated I/O Module cores are interconnected to provide majority voting of their internal state.

TMR Fault Tolerance

The TMR MicroBlaze sub-blocks have triplicated MicroBlaze, LMB memory and I/O Module peripherals, with majority voting of all the interfaces as shown in Figure 2-2. In this configuration the interfaces with voters are:

1. Instruction LMB BRAM Interface Controller: Local memory block RAM
2. Data LMB BRAM Interface Controller: Local memory block RAM
3. I/O Module external interfaces (UART, GPO)

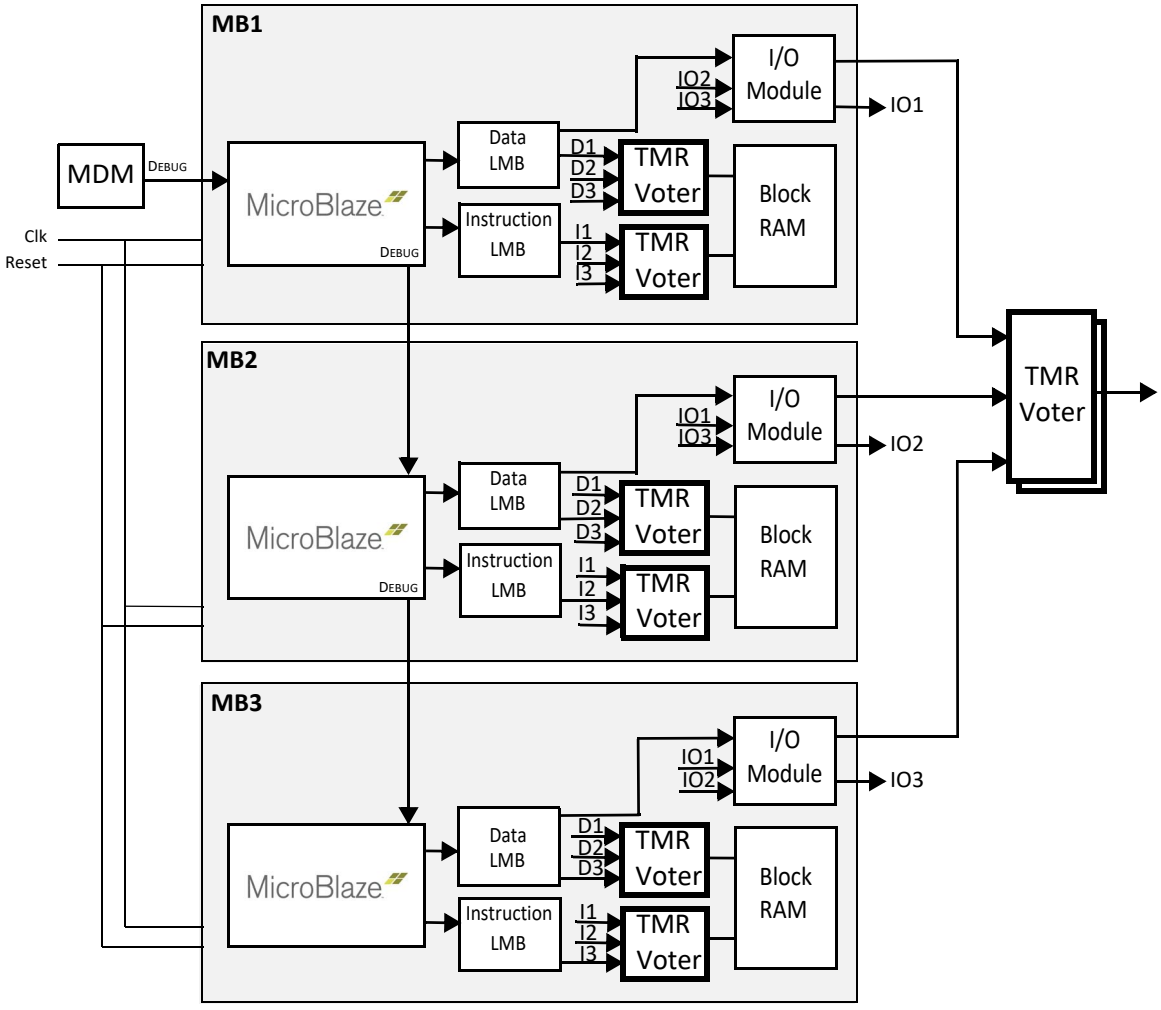

Figure 2-2: **TMR MicroBlaze Fault Tolerant Subsystem - Local Memory**

The voters implement the FT property, with majority voting to ensure that a faulty MicroBlaze sub-block is masked by the two other sub-blocks. This guarantees that the I/O interfaces continue to provide correct output data even in the presence of a fault.

The LMB block RAM is triplicated, with majority voting of the read data to ensure that all three MicroBlaze processors see the correct data. This is necessary to be able to correct any errors in the block RAM.

It is also possible to use a single block RAM protected by Error Correcting Code (ECC) outside the triplicated sub-blocks, as shown in Figure 2-3. The ECC is then generated and checked in the triplicated LMB Interface Controller at the boundary where the two protection schemes overlap. This configuration uses less resources, at the expense of somewhat reduced fault detection.

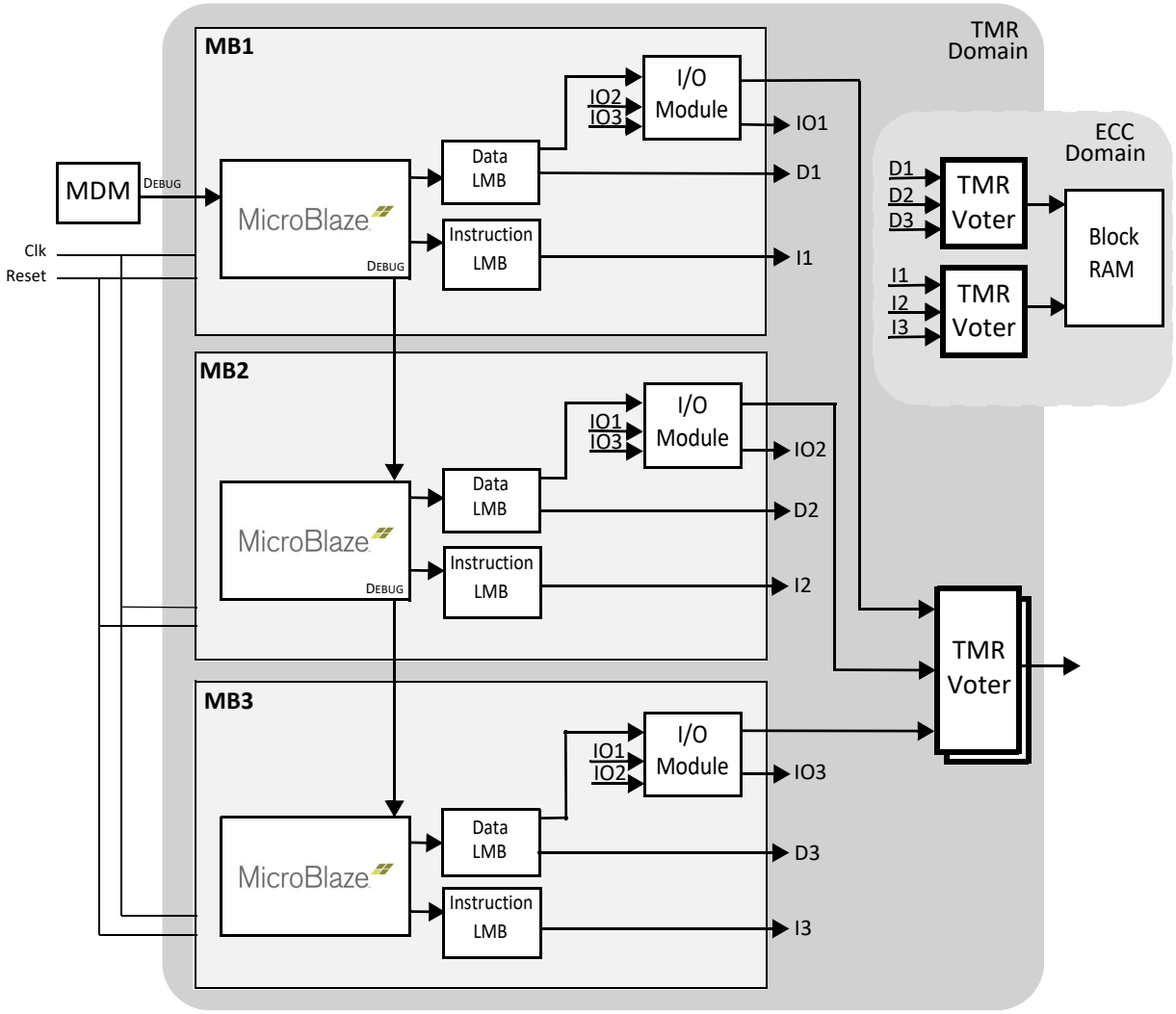

Figure 2-3: **TMR MicroBlaze Fault Tolerant Subsystem - ECC Memory**

To avoid accumulating errors in the block RAM over time, software scrubbing must be implemented with both these configurations.

TMR Fail Safe

After the first failure in a TMR MicroBlaze sub-block, the two remaining healthy sub-blocks operate in lockstep mode, and their outputs need to be compared to detect any difference. The healthy sub-blocks are compared cycle by cycle and if a mismatch occurs, the TMR MicroBlaze subsystem is stopped and the fatal error signal is set to one.

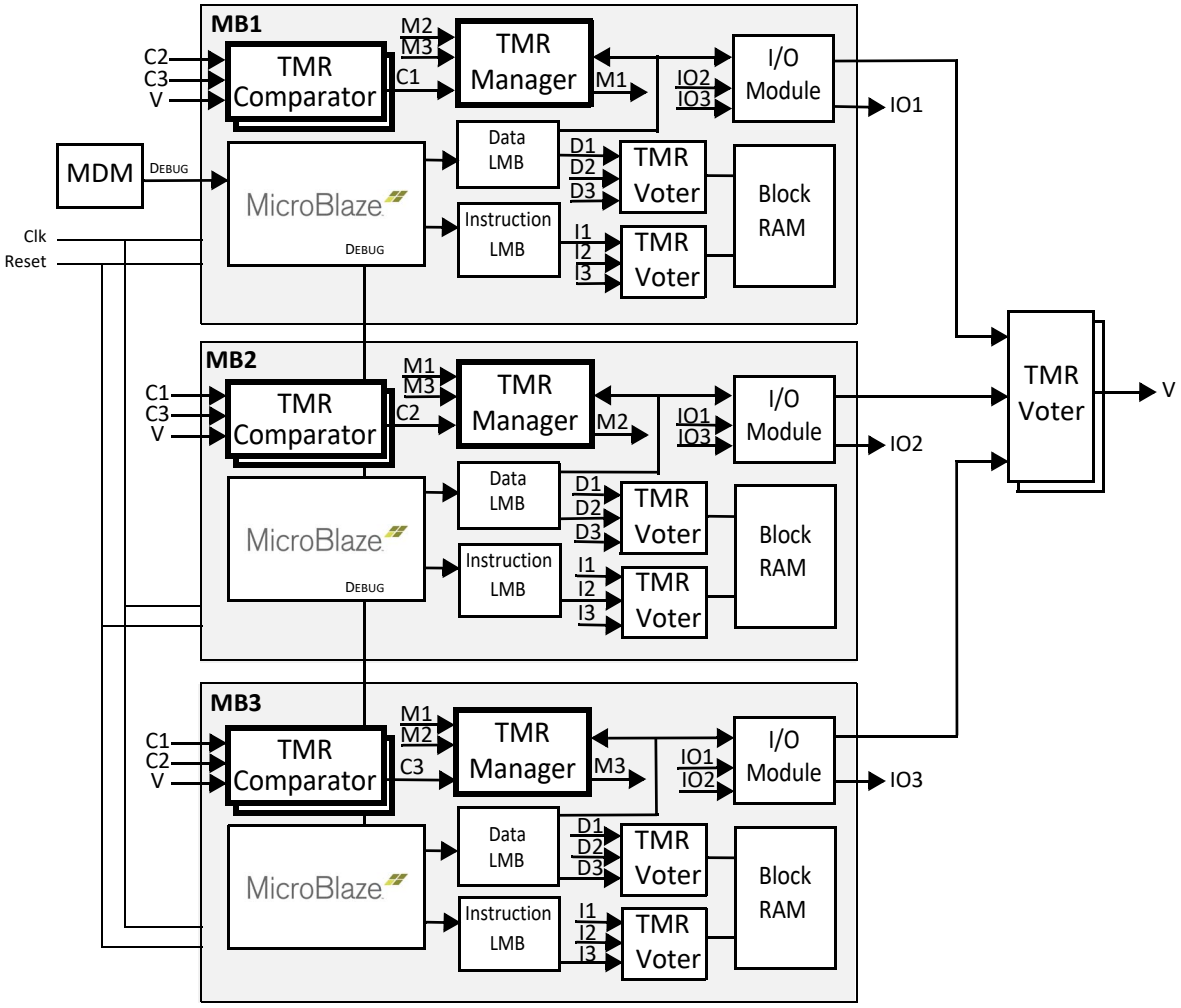

Figure 2-4: **TMR MicroBlaze Fail Safe Subsystem**

While in lockstep mode the TMR voters ensure that the outputs from the faulty sub-block are masked by the healthy ones.

To ensure the integrity of the comparison the TMR comparators are also triplicated in the sub-blocks. This means that there is one voter and three comparators for every external interface. An error in the voter itself is considered a fatal error and needs to be detected. This is done by letting the triplicated comparators also check the voted output.

TMR Operation

The basic voting functionality works without any control or maintenance, but the FS comparison needs to keep track of which MicroBlaze subsystem is considered faulty. This is handled by the TMR Manager IP core. A state machine is also needed to handle recovery of a faulty MicroBlaze sub-block to a healthy status when executing in FS mode.

- Voting (FT-mode) – All three MicroBlaze sub-blocks are healthy.
- Lockstep (FS-mode) – Two MicroBlaze sub-blocks are healthy.
- Fatal (Stop) – The subsystem has detected an unrecoverable error and is stopped.

Figure 2-5 illustrates the three states of the TMR MicroBlaze subsystem.

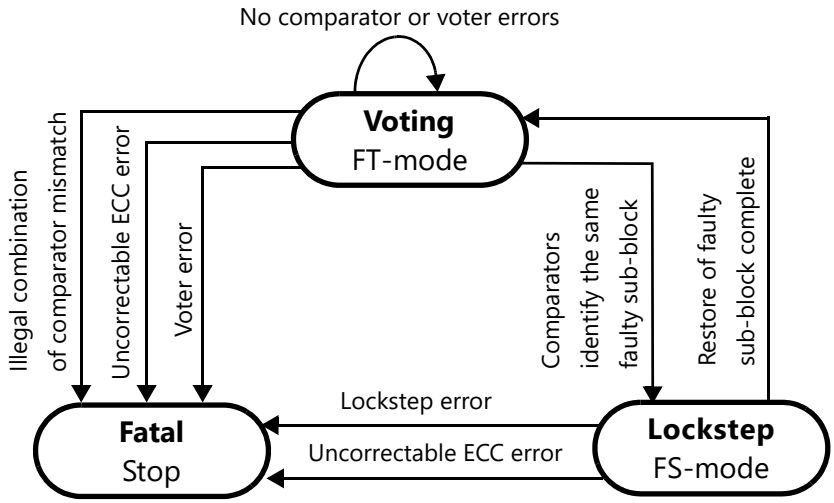

Figure 2-5: **Fault Tolerance State Transition Diagram**

The TMR Manager is also triplicated with one instance in each MicroBlaze subsystem. The TMR Manager implements a voting scheme for all its internal registers and all outputs.

Recovery of the MicroBlaze Subsystem

When the TMR Manager detects a TMR Comparator mismatch with one faulty MicroBlaze uniquely identified, it enters Lockstep state (FS-mode). In this state the two healthy MicroBlaze sub-blocks ensures that the nominal operation of the entire TMR MicroBlaze subsystem continues without degradation. The Lockstep state is signaled to the MicroBlaze processors by asserting a break signal. The software application can handle the break and restore the faulty sub-block by performing the following steps:

1. The executing software is interrupted by the break signal.
2. The software break handler stores all internal MicroBlaze registers in RAM.

3. The software performs a reset of the entire MicroBlaze subsystem excluding the TMR Managers by executing a SUSPEND instruction.
4. The reset restores the TMR Manager to Voting (FT-mode) state.
5. The software starts executing from the reset vector, and reads the TMR Manager First Failing Register (FFR) to determine the actions to perform.
6. If the FFR indicates a cold reset (the Recovery bit is not set), a normal program cold start should be done. If the FFR indicates that one MicroBlaze sub-block is faulty (all of the Fatal bits are cleared, the Recovery bit and two of the three Lockstep mismatch bits are set), a recovery should be done. If the register holds any other value, the software should not attempt a recovery. The action in this case is application dependent, and could for example be entering an infinite loop to allow logic outside the subsystem to handle recovery, or doing a cold reset.
7. The software clears the TMR Manager FFR.
8. The software restores all registers from RAM and execute an RTBD instruction to return from the break handler, to resume nominal execution at the place where the break occurred.

Because restoring the faulty processor is controlled by software, it can postpone recovery until any critical tasks have been completed if necessary. This scheme can be modified to handle permanent errors and run in degraded Lockstep mode using the two healthy processors, by masking the break.

If the system requirements allow a periodic reset of the MicroBlaze subsystem, the software need not perform an explicit restore by handling the break, because a potential Lockstep state would implicitly be restored to the Voting state by the periodic reset. Another advantage of a periodic reset is that any latent faults in the subsystem are removed, which reduces the failure intensity.

Finally, if the system requirements allow, software recovery can be omitted altogether. The subsystem would then run until there is a fatal error condition, which could be resolved by a power-on-reset.

Fault Injection

Fault injection is useful to test the integrity of the TMR subsystem, and ensure that error detection and recovery operates nominally. There are three principal methods of fault injection:

- Functional fault injection, supported by the TMR Inject core.
- Comparator fault injection, supported by the TMR Manager and TMR Comparator cores, and
- Configuration bit fault injection, supported by the TMR SEM core.

Functional fault injection is implemented by injecting a different instruction in one of the triplicated MicroBlaze processors at a certain instruction address (program counter). This eventually causes a mismatch between this processor and the two others, which is detected by a TMR comparator.

To inject a fault in one of the three processors, the software writes the instruction, address and CPU ID to the TMR Inject core. The software can check that the expected comparator mismatch has occurred by reading the TMR Manager First Failing Register.

During testing, it is possible to prevent the TMR Manager from reacting to the fault using the TMR Manager Comparison Mask Register. In the absence of an additional actual fault, the subsystem continues to operate nominally.

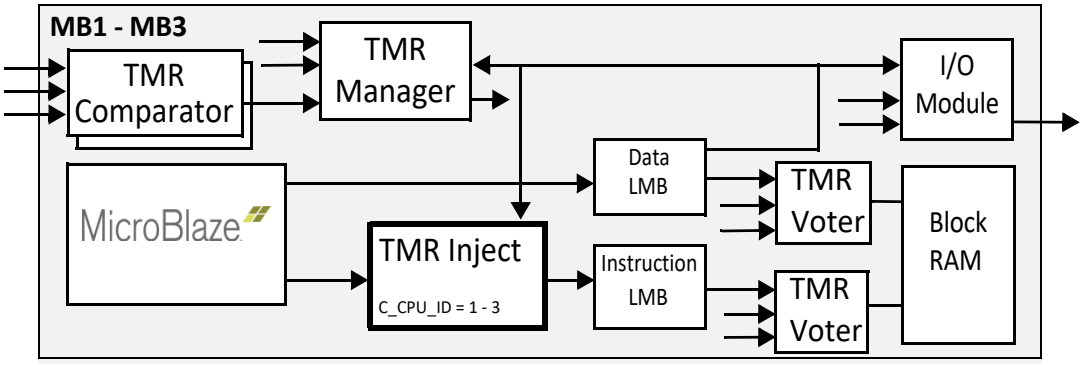

Figure 2-6: **TMR MicroBlaze Functional Fault Injection**

Lockstep Fail Safe

The TMR Manager and TMR Comparator implementation supports a lockstep configuration, which only duplicates the sub-blocks to provide a Fail-Safe (FS) subsystem, as shown in Figure 2-7. In this subsystem, there is no need for voting.

Similar to the TMR case, the block RAM can either be duplicated or use ECC.

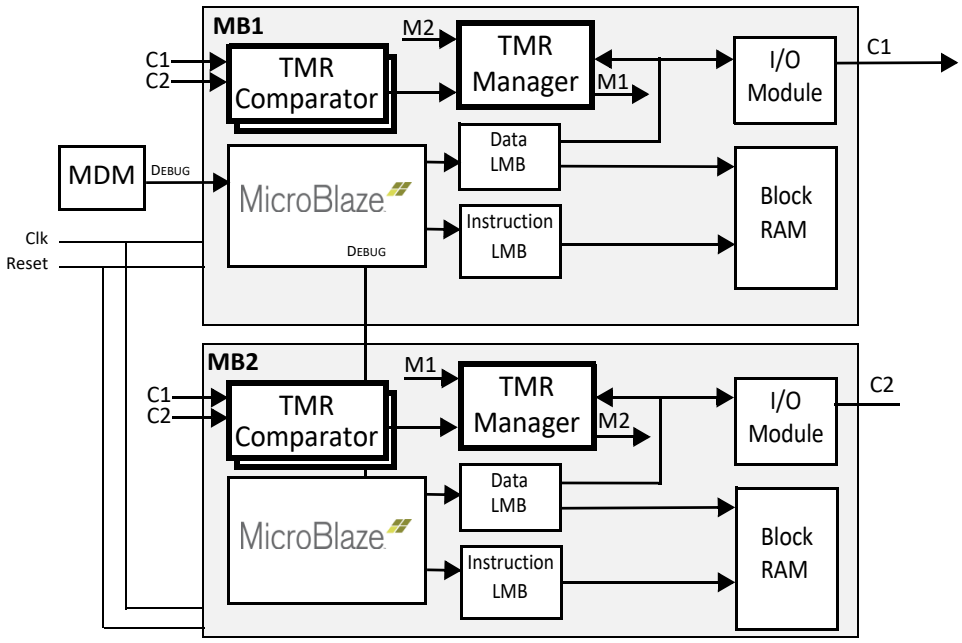

Figure 2-7: **Lockstep MicroBlaze Fail Safe Subsystem**

TMR Design Rules

- All synchronization has to be made outside the TMR region, because asynchronous interfaces inside the TMR region can result in a one cycle jitter of the synchronized signal, which would lead to a comparison error.
- Recovery needs the complete state of all IP cores to be restored.
- All IP cores need to have the identical configuration as their triplicated counterparts.

It might not be possible to include IP cores or RTL modules that do not adhere to these rules in the TMR subsystem. In this case, third-party tools such as the BL-TMR Tool [Ref 16] might be used to implement fine-grained majority voting for these cores or modules.

Supported Interfaces

The MicroBlaze TMR solution supports voting and comparison on all the typical interfaces found in a MicroBlaze subsystem. The supported interfaces are:

- Discrete
- BRAM
- LMB
- AXI4
- AXI4LITE
- AXI4 ACE

- AXI4 Stream Master
- AXI4 Stream Slave
- MicroBlaze Trace
- MicroBlaze Interrupt
- IO Bus (IO Module)
- UART
- GPIO

IP Partitioning

The MicroBlaze TMR solution is implemented using five IP cores. They are:

1. TMR Voter
2. TMR Comparator
3. TMR Manager
4. TMR Inject
5. TMR SEM

TMR Voter

The TMR Voter diagram in Figure 2-8 shows voting of one output bit and distribution of one input bit to all sub-blocks. This logic is duplicated for every output and input in each of the supported bus interfaces.

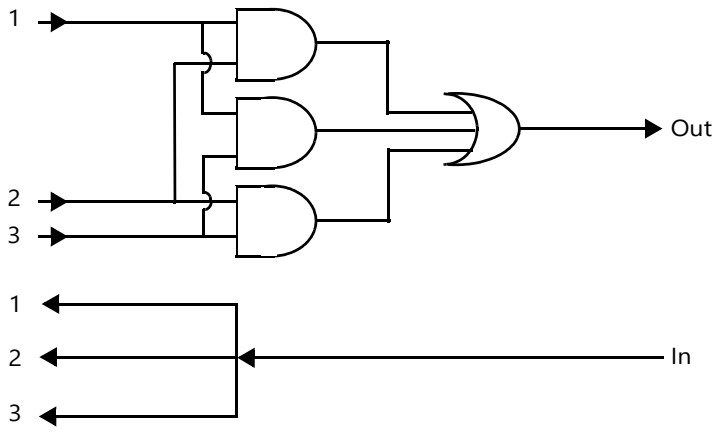

Figure 2-8: **TMR Voter Logic**

TMR Comparator

The TMR Comparator diagram in Figure 2-9 shows comparison of one output from each sub-block. This logic is duplicated for every output in each of the supported bus interfaces.

The TMR comparator has three outputs to allow classification of the detected mismatch according to Table 2-1. The optional voter error check logic provides fault detection of the TMR Voter.

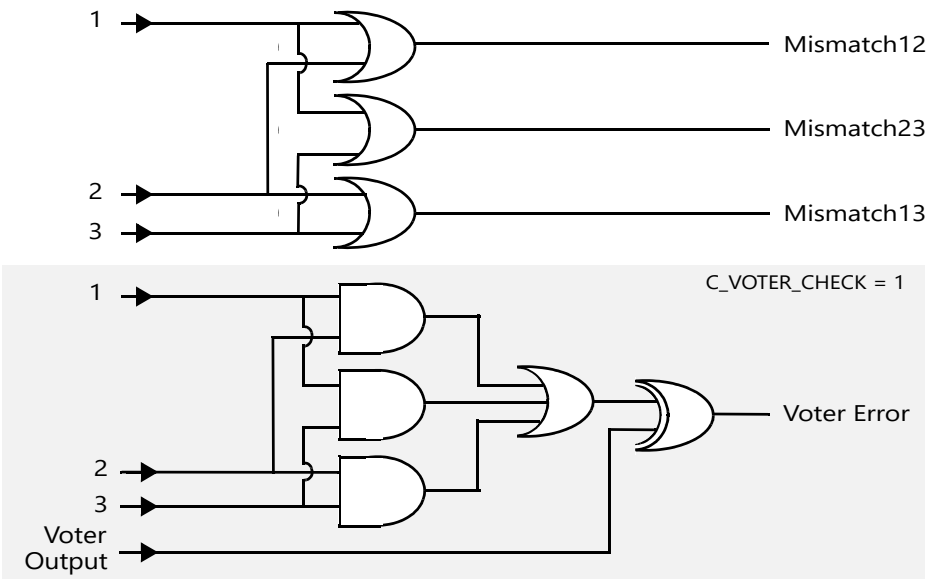

Figure 2-9: **TMR Comparator Logic**

Table 2-1: **Mismatch Classification**

Mismatch23	Mismatch13	Mismatch12	Classification
0	0	0	No fault detected
0	0	1	Fatal error
0	1	0	Fatal error
0	1	1	Sub-block 1 fault detected
1	0	0	Fatal error
1	0	1	Sub-block 2 fault detected
1	1	0	Sub-block 3 fault detected
1	1	1	Fatal error

TMR Manager

The TMR Manager block diagram in Figure 2-10 shows the main components and interfaces of the IP core.

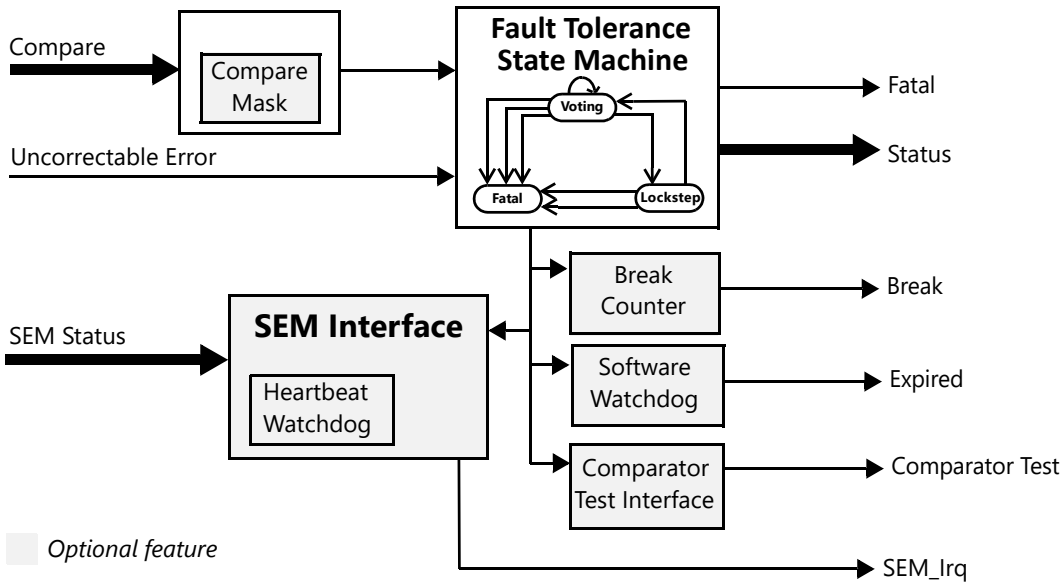

Figure 2-10: **TMR Manager Block Diagram**

Compare and Compare Mask

The compare inputs from all TMR Comparators in the sub-block are used to control the Fault Tolerance state machine, and can be masked for test purposes with the Compare Mask registers.

SEM Interface

The SEM Interface provides the ability to generate interrupts to the processor for any SEM event. It also provides a Heartbeat Watchdog, to monitor the health of the SEM core.

Break Counter

The break counter is used to generate a Break signal to the processor, signaling that a recovery is necessary due to a fault. The signal can be delayed, to ensure that a complete configuration memory scrubbing cycle has been performed by the SEM core before attempting a recovery. This ensures that any potential configuration memory error has been corrected, to avoid that the same error immediately generates a new fault.

Software Watchdog

The internal software watchdog has basic functionality, with a fixed counter width set by the parameter C_WATCHDOG_WIDTH. The watchdog has a `WatchDog_Pause` input, intended to pause the watchdog during debugging. It should normally be directly connected from the MicroBlaze processor MB_Halted output.

For more advanced watchdog functionality, an external watchdog such as the AXI Watchdog Timer (WDT) can be used instead. In this case, the AXI Watchdog Timer `wdt_reset` output pin should be connected to the TMR Manager `watchdog_expired` input. See the *AXI Timebase Watchdog Timer* (PG128) [Ref 1] for details.

Comparator Test Interface

The comparator test interface allows injecting faults in any bit of a comparator, in combination with the TMR Comparator AXI4-Stream test interface. Typically a test sequence would consist of the following steps:

- Set the TMR Manager compare mask to mask faults from the tested comparator.
- Inject all possible faults and verify that they are correctly detected:
  - Use the AXI4-Stream test connection to the TMR Comparator to shift in an erroneous bit into the tested comparator.
  - Set the comparator test interface to inject the fault in the tested comparator.
  - Shift out the result via the AXI4-Stream test connection and verify that the fault has been detected by the comparator.
  - Repeat for all bits in the comparator interface and all possible faults.
- Reset the TMR Manager compare mask for nominal operation.
- Repeat for all comparators.

Optionally, the TMR Manager can be allowed to handle the injected fault provided that debug is enabled, by not setting the TMR Manager compare mask. In that case, the Control Register should be set up to block the Break signal. Return to nominal operation is done by using the Reset Failing State Register.

TMR Inject

The TMR Inject block diagram in Figure 2-11 depicts the implementation of the instruction inject functionality.

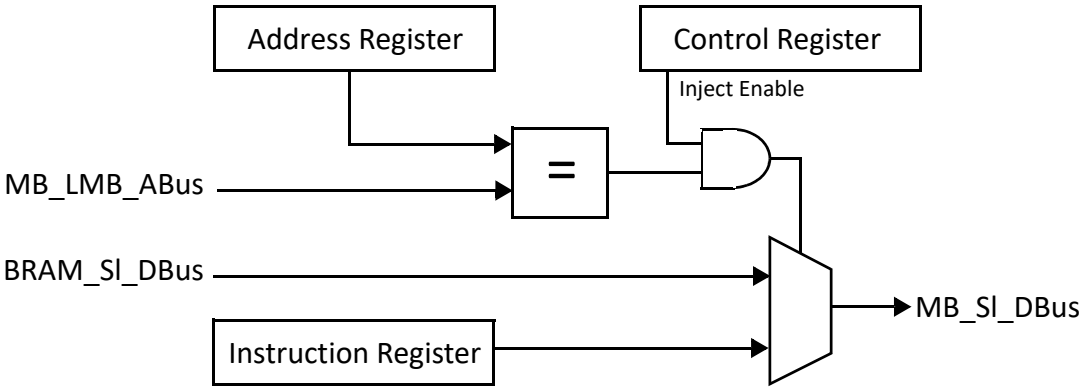

Figure 2-11: **TMR Inject Block Diagram**

TMR Inject has three registers to perform fault injection by replacing a particular instruction fetched by one of the processors. TMR Inject is inserted between the processor and the instruction LMB bus IP core, and responds to all LMB instruction read accesses from the processor.

The Address Register holds the Program Counter of the physical address of the instruction to replace, and is compared with the actual instruction fetch address to trigger a replacement. The Instruction Register holds the instruction, and the Control Register is used to enable injection for a particular processor.

When setting the Address Register, it is necessary to ensure that the processor has not already prefetched the instruction at that Program Counter to ensure that the instruction can be replaced.

Replacing the instruction in one of the three processors provides the possibility of generating internal errors by affecting the internal processor registers, and generating errors on any external interface by affecting load, store, put and get instructions to either change a bit in the address or in the write data.

It is possible to replace the instruction in two of the three processors simultaneously, but that does not result in a fatal error, since the TMR Manager will transition to the Lockstep state in this case.

TMR SEM

The TMR SEM block diagram in Figure 2-12 shows the encapsulation of the SEM core, ICAP and Frame ECC, as well as the control interfaces – AXI4-Lite or UART.

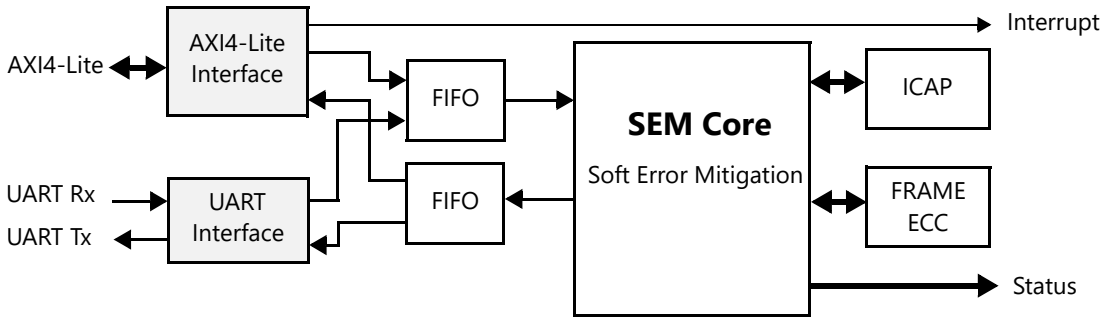

Figure 2-12: **TMR SEM Block Diagram**

The TMR SEM is a wrapper to simplify the most common use cases where the SEM core is used. However, it does not support more advanced configurations. In these case, the SEM functionality must be implemented externally.

Standards

The TMR Comparator and TMR Voter adhere to the AMBA® AXI4, ACE and AXI4-Lite Interface standard (see Arm® *AMBA AXI and ACE Protocol Specification Arm IHI 0022E* [Ref 2]).

The TMR Comparator and TMR Voter adhere to the AMBA AXI4-Stream Interface standard (see Arm AMBA *AXI4-Stream Protocol Specification, Version 1.0* [Ref 3]).

The I/O bus interface provided by the I/O Module core handled by the TMR Comparator and TMR Voter is fully compatible with the Xilinx Dynamic Reconfiguration Port (DRP). For a detailed description of the DRP, see the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 4].

The TMR SEM adheres to the AMBA AXI4-Lite Interface (see Arm *AMBA AXI and ACE Protocol Specification Arm IHI 0022E* [Ref 2]).

Performance and Resource Utilization

For full details about performance and resource utilization of the individual cores, visit the Performance and Resource Utilization web page for each core:

- TMR Manager
- TMR Voter
- TMR Comparator
- TMR Inject
- TMR SEM

Maximum Frequencies

The maximum frequency of a triplicated MicroBlaze subsystem itself is generally not significantly reduced. However, the majority-voted outputs from the subsystem impose an additional logic level, which might affect the overall frequency.

When using the internal SEM option with the TMR SEM core, the frequency is limited to the SEM sub-core maximum frequency. To avoid this limitation, the external SEM option can be used, to allow clocking the SEM with a slower clock.

Latency

There is no additional latency imposed by triplicating the MicroBlaze subsystem. The majority voting on the outputs is combinatorial.

Throughput

There is no difference in throughput or computational performance imposed by triplicating the MicroBlaze subsystem. A triplicated subsystem has exactly the same performance as a single-string subsystem.

Power

The power requirements of a triplicated subsystem is at least three times the equivalent single string subsystem, and can approach four times due to the additional power consumed by the TMR cores.

Triplicated Subsystem Performance and Resource Utilization

Table 2-2 provides approximate performance and resource utilization for typical TMR subsystems, and a comparison with the corresponding single-string subsystem in parentheses. The results were obtained with the IP Characterization and f_{MAX} Margin System Methodology described in the *MicroBlaze Processor Reference Guide* (UG984) [Ref 5]. The resource use of the triplicated sub-systems can be less that three times the single-string, mostly because the designs have debug enabled, which is not triplicated.

See Appendix E, Benchmarks for information on how to recreate the designs.

Table 2-2: **TMR Subsystem Performance and Resource Utilization**

Subsystem Design	Device Family	Device Resources		Frequency
		LUTs	FFs	MHz
Typical I/O Module design: MicroBlaze, MDM, I/O Module UART, external SEM	Virtex-7 XC7VX485T ffg1761-3	7485 (347%)	6432 (414%)	258 (-22%)
	Kintex-7 XC7K325T ffg900-3	7550 (349%)	6432 (414%)	273 (-11%)
	Artix-7 XC7A200T fbg676-3)	7409 (349%)	6432 (414%)	195 (-14%)
	Virtex UltraScale XCVU095 ffvd1924-3	7424 (350%)	6432 (414%)	332 (-9%)
	Kintex UltraScale XCKU040 ffva1156-3	7422 (350%)	6432 (414%)	324 (-12%)
	Virtex UltraScale+ XCVU3P ffvc1517-3	7556 (355%)	6432 (414%)	421 (-12%)
	Kintex UltraScale+ XCKU15P ffva1156-3	7565 (357%)	6432 (414%)	428 (-18%)
	Zynq UltraScale+ XCZU9EG ffvb1156-3	7561 (356%)	6432 (414%)	428 (-11%)
Typical AXI design: MicroBlaze, MDM, AXI Interrupt Controller, AXI Timer, AXI GPIO, AXI UARTLite, external SEM	Virtex-7 XC7VX485T ffg1761-3	9542 (280%)	8985 (317%)	264 (-17%)
	Kintex-7 XC7K325T ffg900-3	9542 (283%)	8985(317%)	266 (-13%)
	Artix-7 XC7A200T fbg676-3)	9470 (292%)	8985 (317%)	206 (-8%)
	Virtex UltraScale XCVU095 ffvd1924-3	9483 (290%)	8985 (317%)	307 (-15%)
	Kintex UltraScale XCKU040 ffva1156-3	9502 (291%)	8985 (317%)	310 (-12%)
	Virtex UltraScale+ XCVU3P ffvc1517-3	9714 (292%)	8985 (317%)	421 (-14%)
	Kintex UltraScale+ XCKU15P ffva1156-3	9667 (291%)	8985 (317%)	420 (-12%)
	Zynq UltraScale+ XCZU9EG ffvb1156-3	9724 (291%)	8985 (317%)	441 (-16%)

Port Descriptions

TMR Manager

The I/O signals for the TMR Manager are listed and described in Table 2-3.

Table 2-3: **TMR Manager I/O Signals**

Signal Name	Interface	I/O	Initial State	Description
Clk		I	-	Clock
Rst		I	-	Synchronous reset, active-High
Fault Tolerance Signals				
TMR_Disable		I	-	TMR disable
Debug_Disable		I	-	Debug disable
Recover		I	-	Recover signal from MicroBlaze
LockStep_Break		O	0	Lockstep Break signal to MicroBlaze
Reset		O	0	Subsystem recovery reset output
Fatal		O	0	Fatal error status output
Status		O	0x0	TMR status output (see Table 2-4)
Compare_*n*		I	-	Comparator inputs (n = 0-63)
UE_LMB		I	-	Uncorrectable Error from LMB
WatchDog_Expired		O	0	Software watchdog expired
WatchDog_Pause		I	-	Internal watchdog pause
To_TMR_Managers		O	0x0	Interconnect to other TMR Managers
From_TMR_Manager_1		I	-	Interconnect from TMR Manager 1
From_TMR_Manager_2		I	-	Interconnect from TMR Manager 2
From_TMR_Manager_3		I	-	Interconnect from TMR Manager 3
SEM Interface Signals(1) (C_SEM_INTERFACE = 1)				
SEM_heartbeat		I	-	Heartbeat status input from SEM
SEM_initialization		I	-	Initialization status input from SEM
SEM_observation		I	-	Observation status input from SEM
SEM_correction		I	-	Correction status input from SEM
SEM_classification		I	-	Classification status input from SEM
SEM_injection		I	-	Injection status input from SEM
SEM_essential		I	-	Essential status input from SEM
SEM_uncorrectable		I	-	Uncorrectable status input from SEM
SEM_diagnostic_scan(2)		I	-	Diagnostic scan status input from SEM

Table 2-3: **TMR Manager I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
SEM_detect_only[(2)]		I	-	Detect only status input from SEM
SEM_heartbeat_expired		O	0	SEM heartbeat watchdog expired
SEM_status_irq		O	0	SEM status interrupt output
Comparator Test Interface (C_TEST_COMPARATOR > 0)				
Test_Comparator		O	0x0	Comparator test interface used to: • Trigger status capture on error • Clear captured status • Inject errors
LMB Slave Interface				
LMB_ABus[0:C_LMB_AWIDTH-1]	LMB	I	-	Data Address
LMB_WriteDBus[0:C_LMB_DWIDTH-1]	LMB	I	-	Data Write Bus
LMB_AddrStrobe	LMB	I	-	Address Strobe
LMB_ReadStrobe	LMB	I	-	Read Strobe
LMB_WriteStrobe	LMB	I	-	Write Strobe
LMB_BE[0:C_LMB_DWIDTH/8-1]	LMB	I	-	Byte Enable
Sl_DBus[0:C_LMB_DWIDTH-1]	LMB	O	0x0	Data Read Bus
Sl_Ready	LMB	O	0	Ready
Sl_Wait	LMB	O	0	Wait
Sl_CE	LMB	O	0	Correctable Error
Sl_UE	LMB	O	0	Uncorrectable Error

Notes:

1. For details see the *Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG036) [Ref 6] and *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187) [Ref 7].
2. Only available on UltraScale architecture devices.

Table 2-4: **Status Output Signal**

Bit	Field	Description
0	Lockstep Mismatch	Lockstep mismatch between processor 1 and 2 causing transition to Lockstep state
1		Lockstep mismatch between processor 1 and 3 causing transition to Lockstep state
2		Lockstep mismatch between processor 2 and 3 causing transition to Lockstep state
3	Recovery	Recovery

Table 2-4: **Status Output Signal *(Cont'd)***

Bit	Field	Description
4	Fatal Errors	Fatal error between processor 1 and 2 causing transition to Fatal state
5		Fatal error between processor 1 and 3 causing transition to Fatal state
6		Fatal error between processor 2 and 3 causing transition to Fatal state
7		Fatal voter error
8		Fatal uncorrectable ECC error
9		Watchdog expired
11-10	FT State	00 = Nominal state 01 = Lockstep 10 = Recover reset 11 = Fatal error
12	Processor Faults	Processor 1 fault, valid when Lockstep Mismatch field is not zero
13		Processor 2 fault, valid when Lockstep Mismatch field is not zero
14		Processor 3 fault, valid when Lockstep Mismatch field is not zero
31-15	Reserved	Reserved

TMR Voter

The I/O signals for the TMR Voter are listed and described in Table 2-5.

Table 2-5: **TMR Voter I/O Signals**

Signal Name	Interface	I/O	Initial State	Description
Clk		I	-	Clock
TMR_Disable		I	-	TMR disable
Compare[1]		O	0	Comparison result
M_BRAM_Mismatch[2]		O	0	Master BRAM Mismatch
Comparator Test Interface (C_TEST_COMPARATOR > 0)				
Rst		I	-	Synchronous reset
Test_Comparator		I	-	Comparator test interface used to: • Trigger status capture on error • Clear captured status • Inject errors
S_AXIS_TDATA_Test [C_TEST_AXIS_DATA_WIDTH-1:0]	AXI4-Stream	I	-	AXI4-Stream TDATA test input
S_AXIS_TLAST_Test	AXI4-Stream	I	-	AXI4-Stream TLAST test input
S_AXIS_TVALID_Test	AXI4-Stream	I	-	AXI4-Stream TVALID test input
S_AXIS_TREADY_Test	AXI4-Stream	O	0	AXI4-Stream TREADY test output
M_AXIS_TDATA_Test [C_TEST_AXIS_DATA_WIDTH-1:0]	AXI4-Stream	O	0x0	AXI4-Stream TDATA test output

Table 2-5: **TMR Voter I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
M_AXIS_TLAST_Test	AXI4-Stream	O	0	AXI4-Stream TLAST test output
M_AXIS_TVALID_Test	AXI4-Stream	O	0	AXI4-Stream TVALID test output
M_AXIS_TREADY_Test	AXI4-Stream	I	-	AXI4-Stream TREADY test input
Discrete Interface (C_INTERFACE = 0)				
Discrete*n*[C_DISCRETE_WIDTH-1:0]		I	-	Discrete voter inputs (n = 1-3)
Discrete		O	-	Discrete voted output
LMB Interface (C_INTERFACE = 1)				
LMB*n*_* (see Table 2-6)	LMB	I	-	LMB voter inputs (n = 1-3)
SI*n*_* (see Table 2-6)	LMB	O	-	SI voter outputs (n = 1-3)
LMB_* (see Table 2-6)	LMB	O	-	LMB voted output
SI_* (see Table 2-6)	LMB	I	-	SI voted input
BRAM Interface (C_INTERFACE = 2)				
BRAM*n*_* (see Table 2-7)	BRAM	I/O	-	BRAM voter inputs (n = 1-3)
BRAM_* (see Table 2-7)	BRAM	I/O	-	BRAM voted output
AXI4 and AXI4-Lite Interface (C_INTERFACE = 3 or 8)				
S_AXI*n*_* (see Table 2-8)	AXI4	I/O	-	S_AXI voter inputs (n = 1-3)
M_AXI_* (see Table 2-8)	AXI4	I/O	-	M_AXI voted output
AXI4 and AXI4-Lite Slave Interface (C_INTERFACE = 15 or 16)				
S_AXI_* (see Table 2-8)	AXI4	I/O	-	S_AXI voter output
M_AXI*n*_* (see Table 2-8)	AXI4	I/O	-	M_AXI voter inputs (n = 1-3)
AXI4 Stream Master Interface (C_INTERFACE = 4)				
S_AXIS*n*_* (see Table 2-9)	AXI4-Stream	I/O	-	S_AXIS voter inputs (n = 1-3)
M_AXIS_* (see Table 2-9)	AXI4-Stream	I/O	-	M_AXIS voted output
AXI4 Stream Slave Interface (C_INTERFACE = 5)				
M_AXIS*n*_* (see Table 2-9)	AXI4-Stream	I/O	-	M_AXIS voter inputs (n = 1-3)
S_AXIS_* (see Table 2-9)	AXI4-Stream	I/O	-	S_AXIS voted output
ACE Interface (C_INTERFACE = 6)				
S_AXI*n*_* (see Table 2-8)	ACE	I/O	-	ACE voter inputs (n = 1-3)
M_AXI_* (see Table 2-8)	ACE	I/O	-	ACE voted output
ACE Slave Interface (C_INTERFACE = 17)				
S_AXI_* (see Table 2-8)	ACE	I/O	-	ACE voter output
M_AXI*n*_* (see Table 2-8)	ACE	I/O	-	ACE voter inputs (n = 1-3)
Trace Interface (C_INTERFACE = 7)				
TRACE*n*_* (see Table 2-10)	Trace	I	-	TRACE voter inputs (n = 1-3)
TRACE_* (see Table 2-10)	Trace	O	-	TRACE voted output

Table 2-5: **TMR Voter I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
Interrupt Interface (C_INTERFACE = 9)				
IRQ*n*	Interrupt	O	-	IRQ voter output (n = 1-3)
IRQ*n*_Address	Interrupt	O	-	IRQ address voter output (n = 1-3)
IRQ*n*_Ack	Interrupt	I	-	IRQ acknowledge voter input (n=1-3)
IRQ	Interrupt	I	-	IRQ voted input
IRQ_Address	Interrupt	I	-	IRQ address voted input
IRQ_Ack	Interrupt	O	-	IRQ acknowledge voted output
Interrupt Slave Interface (C_INTERFACE = 18)				
S_IRQ*n*	Interrupt	I	-	IRQ voter input (n = 1-3)
S_IRQ*n*_Address	Interrupt	I	-	IRQ address voter input (n = 1-3)
S_IRQ*n*_Ack	Interrupt	O	-	IRQ acknowledge voter output (n=1-3)
M_IRQ	Interrupt	O	-	IRQ voter output
M_IRQ_Address	Interrupt	O	-	IRQ address voter output
M_IRQ_Ack	Interrupt	I	-	IRQ acknowledge voter input
IO Bus Interface (C_INTERFACE = 10)				
IO*n*_* (see Table 2-11)	IOBUS	I/O	-	IO voter inputs (n = 1-3)
IO_* (see Table 2-11)	IOBUS	I/O	-	IO voted output
GPIO Interface (C_INTERFACE = 11)				
GPIO*n*_* (see Table 2-12)	GPIO	I/O	-	GPIO voter inputs (n = 1-3)
GPIO_* (see Table 2-12)	GPIO	I/O	-	GPIO voted output
UART Interface (C_INTERFACE = 12)				
UART*n*_* (see Table 2-13)	UART	I/O	-	UART voter inputs (n = 1-3)
UART_* (see Table 2-13)	UART	I/O	-	UART voted output
BRAM Master Interface (C_INTERFACE = 13)				
M_BRAM*n*_* (see Table 2-7)	BRAM	I/O	-	M_BRAM voter inputs (n = 1-3)
S_BRAM_* (see Table 2-7)	BRAM	I/O	-	S_BRAM voted output
LMB Slave Interface (C_INTERFACE = 14)				
S_LMB*n*_* (see Table 2-6)	LMB	O	-	S_LMB voter inputs (n = 1-3)
S_SI*n*_* (see Table 2-6)	LMB	I	-	S_SI voter inputs (n = 1-3)
S_LMB_* (see Table 2-6)	LMB	I	-	S_LMB voted output
S_SI_* (see Table 2-6)	LMB	O	-	S_SI voted output

Notes:

1. Only available when C_COMPARATOR is set to 1 to activate the internal voter comparator.
2. Only available when C_INTERFACE is set to 13 (Master BRAM) and C_COMPARATOR is set to 1 to activate the internal voter comparator.

Common interface signals are listed in Table 2-6 to Table 2-13.

Table 2-6: **LMB Interface Signals**

Signal Name Suffix	Description
LMB Prefix	
ABus[0:C_LMB_AWIDTH-1]	Address Bus
WriteDBus[0:C_LMB_DWIDTH-1]	Data Write Bus
AddrStrobe	Address Strobe
ReadStrobe	Read Strobe
WriteStrobe	Write Strobe
BE[0:C_LMB_DWIDTH/8-1]	Byte Enable
SI Prefix	
DBus[0:C_LMB_DWIDTH-1]	Data Read Bus
Ready	Ready
Wait	Wait
UE	Uncorrectable Error
CE	Correctable Error

Table 2-7: **BRAM Interface Signals**

Signal Name Suffix	BRAM*n* S_BRAM	BRAM M_BRAM*n*	Description
Rst	I	O	Reset
Clk	I	O	Clock
Addr	I	O	Address
EN	I	O	Enable
WE	I	O	Write Enable
Dout	I	O	Data out
Din	O	I	Data in

Table 2-8: **AXI4, AXI4-Lite and ACE Interface Signals**

Signal Name Suffix	Bus	S_AXI*n* S_AXI	M_AXI M_AXI*n*	Description
AWID	AXI4, ACE	I	O	Write Address ID
AWADDR	AXI4, AXI4-Lite, ACE	I	O	Write Address
AWLEN	AXI4, ACE	I	O	Write Address Length
AWSIZE	AXI4, ACE	I	O	Write Address Size
AWBURST	AXI4, ACE	I	O	Write Address Burst
AWLOCK	AXI4, ACE	I	O	Write Address Lock
AWCACHE	AXI4, ACE	I	O	Write Address Cache

Table 2-8: **AXI4, AXI4-Lite and ACE Interface Signals *(Cont'd)***

Signal Name Suffix	Bus	S_AXI*n* S_AXI	M_AXI M_AXI*n*	Description
AWPROT	AXI4, ACE	I	O	Write Address Protection
AWQOS	AXI4, ACE	I	O	Write Address QoS
AWVALID	AXI4, AXI4-Lite, ACE	I	O	Write Address Valid
AWREADY	AXI4, AXI4-Lite, ACE	O	I	Write Address Ready
AWUSER	AXI4, AXI4-Lite, ACE	I	O	Write Address User bits
AWDOMAIN	ACE	I	O	Write Address Domain
AWSNOOP	ACE	I	O	Write Address Transaction Type
AWBAR	ACE	I	O	Write Address Barrier
WDATA	AXI4, AXI4-Lite, ACE	I	O	Write Data
WSTRB	AXI4, AXI4-Lite, ACE	I	O	Write Strobes
WLAST	AXI4, ACE	I	O	Write Last
WVALID	AXI4, AXI4-Lite, ACE	I	O	Write Valid
WREADY	AXI4, AXI4-Lite, ACE	O	I	Write Ready
WUSER	AXI4, AXI4-Lite, ACE	I	O	Write User bits
WACK	ACE	I	O	Write Acknowledge
BRESP	AXI4, AXI4-Lite, ACE	O	I	Write Response
BID	AXI4, ACE	O	I	Write Response ID
BVALID	AXI4, AXI4-Lite, ACE	O	I	Write Response Valid
BREADY	AXI4, AXI4-Lite, ACE	I	O	Write Response Ready
BUSER	AXI4, AXI4-Lite, ACE	O	I	Write Response User bits
ARID	AXI4, ACE	I	O	Read Address ID
ARADDR	AXI4, AXI4-Lite, ACE	I	O	Read Address
ARLEN	AXI4, ACE	I	O	Read Address Length
ARSIZE	AXI4, ACE	I	O	Read Address Size
ARBURST	AXI4, ACE	I	O	Read Address Burst
ARLOCK	AXI4, ACE	I	O	Read Address Lock
ARCACHE	AXI4, ACE	I	O	Read Address Cache
ARPROT	AXI4, ACE	I	O	Read Address Protection
ARQOS	AXI4, ACE	I	O	Read Address QoS
ARVALID	AXI4, AXI4-Lite, ACE	I	O	Read Address Valid
ARREADY	AXI4, AXI4-Lite, ACE	O	I	Read Address Ready
ARUSER	AXI4, AXI4-Lite, ACE	I	O	Read Address User bits
ARDOMAIN	ACE	I	O	Read Address Domain
ARSNOOP	ACE	I	O	Read Address Transaction Type
ARBAR	ACE	I	O	Read Address Barrier

Table 2-8: **AXI4, AXI4-Lite and ACE Interface Signals *(Cont'd)***

Signal Name Suffix	Bus	S_AXI*n* S_AXI	M_AXI M_AXI*n*	Description
RID	AXI4, ACE	O	I	Read ID
RDATA	AXI4, AXI4-Lite, ACE	O	I	Read Data
RRESP	AXI4, AXI4-Lite, ACE	O	I	Read Response
RLAST	AXI4, ACE	O	I	Read Last
RVALID	AXI4, AXI4-Lite, ACE	O	I	Read Valid
RREADY	AXI4, AXI4-Lite, ACE	I	O	Read Ready
RUSER	AXI4, AXI4-Lite, ACE	O	I	Read User bits
RACK	ACE	I	O	Read Acknowledge
ACVALID	ACE	O	I	Snoop Address Valid
ACADDR	ACE	O	I	Snoop Address
ACSNOOP	ACE	O	I	Snoop Address Transaction Type
ACPROT	ACE	O	I	Snoop Address Protection Type
ACREADY	ACE	I	O	Snoop Address Ready
CRVALID	ACE	I	O	Snoop Response Valid
CRRESP	ACE	I	O	Snoop Response
CRREADY	ACE	O	I	Snoop Response Ready
CDVALID	ACE	I	O	Snoop Data Valid
CDDATA	ACE	I	O	Snoop Data
CDLAST	ACE	I	O	Snoop Data Last
CDREADY	ACE	O	I	Snoop Data Ready

Table 2-9: **AXI4-Stream Interface Signals**

Signal Name Suffix	S_AXIS*n* S_AXIS	M_AXIS M_AXIS*n*	Description
TLAST	I	O	Transfer Last
TDATA	I	O	Transfer Valid
TVALID	I	O	Transfer Ready
TREADY	O	I	Transfer Data
TSTRB	I	O	Transfer Strobe
TKEEP	I	O	Transfer Keep
TID	I	O	Transfer ID
TDEST	I	O	Transfer Destination
TUSER	I	O	Transfer User bits

Table 2-10: **Trace Interface Signals**

Signal Name Suffix	Description
Valid_Instr	Valid instruction on trace port
Instruction	Instruction code
PC	Program counter
Reg_Write	Instruction writes to the register file
Reg_Addr	Destination register address
MSR_Reg	Machine status register
PID_Reg	Process identifier register
New_Reg_Value	Destination register update value
Exception_Taken	Instruction results in taken exception
Exception_Kind	Exception type
Jump_Taken	Branch instruction evaluated true
Jump_Hit	Branch Target Cache hit
Delay_Slot	Instruction is in delay slot of taken branch
Data_Access	Valid D-side memory access
Data_Address	Address for D-side memory access
Data_Write_Value	Value for D-side memory access
Data_Byte_Enable	Byte enables for D-side memory access
Data_Read	D-side memory access is read
Data_Write	D-side memory access is write
DCache_Req	Data memory address within D-Cache
DCache_Hit	Data memory address present
DCache_Rdy	Data memory access completed
DCache_Read	D-Cache request is a read
ICache_Req	Instruction memory address within I-Cache
ICache_Hit	Instruction memory address present
ICache_Rdy	Instruction memory access completed
OF_PipeRun	Pipeline advance for Decode stage
EX_PipeRun	Pipeline advance for Execute stage
MEM_PipeRun	Pipeline advance for Memory stage
MB_Halted	Pipeline is halted by debug

Table 2-11: **IOBUS Interface Signals**

Signal Name Suffix	IO*n*	IO	Description
Addr_Strobe	I	O	Address Strobe
Read_Strobe	I	O	Read Strobe
Write_Strobe	I	O	Write Strobe

Table 2-11: **IOBUS Interface Signals *(Cont'd)***

Signal Name Suffix	IO*n*	IO	Description
Address	I	O	Address
Byte_Enable	I	O	Byte Enable
Write_Data	I	O	Write Data
Read_Data	O	I	Read Data
Ready	O	I	Ready

Table 2-12: **GPIO Interface Signals**

Signal Name Suffix	GPIO*n*	GPIO	Description
GPO	I	O	General Purpose Output
GPT	I	O	General Purpose Tristate
GPI	O	I	General Purpose Input

Table 2-13: **UART Interface Signals**

Signal Name Suffix	UART*n*	UART	Description
Rx	O	I	Serial Input
Tx	I	O	Serial Output
RTSn	I	O	Request to Send
CTSn	O	I	Clear to Send
DTRn	I	O	Data Terminal Ready
DSRn	O	I	Data Set Ready
BAUDOUTn	I	O	Baud Out
RCLK	O	I	Receiver Clock
XIN	O	I	External Crystal Input
XOUT	I	O	External Crystal Output
DCDn	O	I	Data Carrier Detect
DDIS	I	O	Driver Disable
OUT1n	I	O	User Controlled Output 1
OUT2n	I	O	User Controlled Output 2
TXRDYn	I	O	Transmitter DMA Control Signal
RXRDYn	I	O	Receiver DMA Control Signal
RI	O	I	Ring Indicator

TMR Comparator

The I/O signals for the TMR Comparator are listed and described in Table 2-14.

Table 2-14: **TMR Comparator I/O Signals**

Signal Name	Interface	I/O	Initial State	Description
Clk		I	-	Clock
TMR_Disable		I	-	TMR disable
Compare		O	0x0	Comparison result
Comparator Test Interface (C_TEST_COMPARATOR > 0)				
Rst		I	-	Synchronous reset
Test_Comparator		I	-	Comparator test interface used to: • Trigger status capture on error • Clear captured status • Inject errors
S_AXIS_TDATA_Test [C_TEST_AXIS_DATA_WIDTH-1:0]	AXI4-Stream	I	-	AXI4-Stream TDATA test input
S_AXIS_TLAST_Test	AXI4-Stream	I	-	AXI4-Stream TLAST test input
S_AXIS_TVALID_Test	AXI4-Stream	I	-	AXI4-Stream TVALID test input
S_AXIS_TREADY_Test	AXI4-Stream	O	0	AXI4-Stream TREADY test output
M_AXIS_TDATA_Test [C_TEST_AXIS_DATA_WIDTH-1:0]	AXI4-Stream	O	0x0	AXI4-Stream TDATA test output
M_AXIS_TLAST_Test	AXI4-Stream	O	0	AXI4-Stream TLAST test output
M_AXIS_TVALID_Test	AXI4-Stream	O	0	AXI4-Stream TVALID test output
M_AXIS_TREADY_Test	AXI4-Stream	I	-	AXI4-Stream TREADY test input
Discrete Interface (C_INTERFACE = 0)				
Discrete*n*[C_DISCRETE_WIDTH-1:0]		I	-	Discrete comparison inputs (n = 1-3)[(1)]
Discrete		I	-	Discrete voter input[(2)]
LMB Interface (C_INTERFACE = 1)				
LMB*n*_* (see Table 2-15)	LMB	I	-	LMB comparison inputs (n = 1-3)[(1)]
LMB_* (see Table 2-15)	LMB	I	-	LMB voter input[(2)]
BRAM Interface (C_INTERFACE = 2)				
BRAM*n*_* (see Table 2-16)	BRAM	I	-	BRAM comparison inputs (n = 1-3)[(1)]
BRAM_* (see Table 2-16)	BRAM	I	-	BRAM voter input[(2)]
AXI4 and AXI4-Lite Interface (C_INTERFACE = 3, 8, 15 or 17)				
S_AXI*n*_* (see Table 2-17)	AXI4	I	-	S_AXI comparison inputs (n = 1-3)[(1)]
M_AXI_* (see Table 2-17)	AXI4	I	-	M_AXI voter input[(2)]
AXI4 Stream Master Interface (C_INTERFACE = 4)				
S_AXIS*n*_* (see Table 2-18)	AXI4-Stream	I	-	S_AXIS comparison inputs (n = 1-3)[(1)]
M_AXIS_* (see Table 2-18)	AXI4-Stream	I	-	M_AXIS voter input[(2)]

Table 2-14: **TMR Comparator I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
AXI4 Stream Slave Interface (C_INTERFACE = 5)				
M_AXIS*n*_* (see Table 2-18)	AXI4-Stream	I	-	M_AXIS comparison inputs (n = 1-3)[1]
S_AXIS_* (see Table 2-18)	AXI4-Stream	I	-	S_AXIS voter input[2]
ACE Interface (C_INTERFACE = 6 or 17)				
S_AXI*n*_* (see Table 2-17)	ACE	I	-	S_ACE comparison inputs (n = 1-3)[1]
M_AXI_* (see Table 2-17)	ACE	I	-	M_ACE voter input[2]
Trace Interface (C_INTERFACE = 7)				
Trace*n*_* (see Table 2-19)	Trace	I	-	Trace comparison inputs (n = 1-3)[1]
Interrupt Interface (C_INTERFACE = 9 or 18)				
IRQ*n*	Interrupt	I	-	IRQ input (n = 1-3)[1], Unused
IRQ*n*_Address	Interrupt	I	-	IRQ address input (n = 1-3)[1], Unused
IRQ*n*_Ack	Interrupt	I	-	IRQ acknowledge input (n = 1-3)[1]
IRQ	Interrupt	I	-	IRQ voter input[2]
IRQ_Address	Interrupt	I	-	IRQ address voter input[2]
IRQ_Ack	Interrupt	I	-	IRQ acknowledge voter input[2], Unused
IO Bus Interface (C_INTERFACE = 10)				
IO*n*_* (see Table 2-20)	IOBUS	I	-	IO comparison inputs (n = 1-3)[1]
IO_* (see Table 2-20)	IOBUS	I	-	IO voter input[2]
GPIO Interface (C_INTERFACE = 11)				
GPIO*n*_* (see Table 2-21)	GPIO	I	-	GPIO comparison inputs (n = 1-3)[1]
GPIO_* (see Table 2-21)	GPIO	I	-	GPIO voter input[2]
UART Interface (C_INTERFACE = 12)				
UART*n*_* (see Table 2-22)	UART	I	-	UART comparison inputs (n = 1-3)[1]
UART_* (see Table 2-22)	UART	I	-	UART voter input[2]
BRAM Master Interface (C_INTERFACE = 13)				
M_BRAM*n*_* (see Table 2-16)	BRAM	I	-	M_BRAM comparison inputs (n = 1-3)[1]
S_BRAM_* (see Table 2-16)	BRAM	I	-	S_BRAM voter input[2]
LMB Slave Interface (C_INTERFACE = 14)				
S_LMB*n*_* (see Table 2-15)	LMB	I	-	S_LMB comparison inputs (n = 1-3)[1]
S_LMB_* (see Table 2-15)	LMB	I	-	S_LMB voter input[2]

Notes:

1. The third interface is only enabled when C_TMR is set to 1.
2. Only enabled when C_VOTER_CHECK is set to 1.

Common interface signals are listed in Table 2-15 to Table 2-22.

Table 2-15: **LMB Interface Input Signals**

Signal Name Suffix	Interface	Description
ABus[0:C_LMB_AWIDTH-1]	LMB, S_LMB unused	Address Bus
WriteDBus[0:C_LMB_DWIDTH-1]	LMB, S_LMB unused	Data Write Bus
AddrStrobe	LMB, S_LMB unused	Address Strobe
ReadStrobe	LMB, S_LMB unused	Read Strobe
WriteStrobe	LMB, S_LMB unused	Write Strobe
BE[0:C_LMB_DWIDTH/8-1]	LMB, S_LMB unused	Byte Enable
DBus[0:C_LMB_DWIDTH-1]	LMB unused, S_LMB	Data Read Bus
Ready	LMB unused, S_LMB	Ready
Wait	S_LMB only	Wait
UE	S_LMB only	Uncorrectable Error
CE	S_LMB only	Correctable Error

Table 2-16: **BRAM Interface Input Signals**

Signal Name Suffix	Interface	Description
Rst	Unused	Reset
Clk	Unused	Clock
Addr	BRAM, M_BRAM unused	Address
EN	BRAM, M_BRAM unused	Enable
WE	BRAM, M_BRAM unused	Write Enable
Dout	BRAM only	Data out
Din	M_BRAM only	Data in

Table 2-17: **AXI4, AXI4-Lite and ACE Input Interface Signals**

Signal Name Suffix	Interface	Description
AWID	AXI4, ACE	Write Address ID
AWADDR	AXI4, AXI4-Lite, ACE	Write Address
AWLEN	AXI4, ACE	Write Address Length
AWSIZE	AXI4, ACE	Write Address Size
AWBURST	AXI4, ACE	Write Address Burst
AWLOCK	AXI4, ACE	Write Address Lock
AWCACHE	AXI4, ACE	Write Address Cache
AWPROT	AXI4, ACE	Write Address Protection
AWQOS	AXI4, ACE	Write Address QoS
AWVALID	AXI4, AXI4-Lite, ACE	Write Address Valid
AWREADY	Unused	Write Address Ready

Table 2-17: **AXI4, AXI4-Lite and ACE Input Interface Signals *(Cont'd)***

Signal Name Suffix	Interface	Description
AWUSER	AXI4, AXI4-Lite, ACE	Write Address User bits
AWDOMAIN	ACE	Write Address Domain
AWSNOOP	ACE	Write Address Transaction Type
AWBAR	ACE	Write Address Barrier
WDATA	AXI4, AXI4-Lite, ACE	Write Data
WSTRB	AXI4, AXI4-Lite, ACE	Write Strobes
WLAST	AXI4, ACE	Write Last
WVALID	AXI4, AXI4-Lite, ACE	Write Valid
WREADY	Unused	Write Ready
WUSER	AXI4, AXI4-Lite, ACE	Write User bits
WACK	ACE	Write Acknowledge
BRESP	Unused	Write Response
BID	Unused	Write Response ID
BVALID	Unused	Write Response Valid
BREADY	AXI4, AXI4-Lite, ACE	Write Response Ready
BUSER	Unused	Write Response User bits
ARID	AXI4, ACE	Read Address ID
ARADDR	AXI4, AXI4-Lite, ACE	Read Address
ARLEN	AXI4, ACE	Read Address Length
ARSIZE	AXI4, ACE	Read Address Size
ARBURST	AXI4, ACE	Read Address Burst
ARLOCK	AXI4, ACE	Read Address Lock
ARCACHE	AXI4, ACE	Read Address Cache
ARPROT	AXI4, ACE	Read Address Protection
ARQOS	AXI4, ACE	Read Address QoS
ARVALID	AXI4, AXI4-Lite, ACE	Read Address Valid
ARREADY	Unused	Read Address Ready
ARUSER	AXI4, AXI4-Lite, ACE	Read Address User bits
ARDOMAIN	ACE	Read Address Domain
ARSNOOP	ACE	Read Address Transaction Type
ARBAR	ACE	Read Address Barrier
RID	Unused	Read ID
RDATA	Unused	Read Data
RRESP	Unused	Read Response
RLAST	AXI4, ACE	Read Last
RVALID	AXI4, AXI4-Lite, ACE	Read Valid

Table 2-17: **AXI4, AXI4-Lite and ACE Input Interface Signals *(Cont'd)***

Signal Name Suffix	Interface	Description
RREADY	Unused	Read Ready
RUSER	Unused	Read User bits
RACK	ACE	Read Acknowledge
ACVALID	Unused	Snoop Address Valid
ACADDR	Unused	Snoop Address
ACSNOOP	Unused	Snoop Address Transaction Type
ACPROT	Unused	Snoop Address Protection Type
ACREADY	ACE	Snoop Address Ready
CRVALID	ACE	Snoop Response Valid
CRRESP	ACE	Snoop Response
CRREADY	Unused	Snoop Response Ready
CDVALID	ACE	Snoop Data Valid
CDDATA	ACE	Snoop Data
CDLAST	ACE	Snoop Data Last
CDREADY	Unused	Snoop Data Ready

Table 2-18: **AXI4-Stream Input Interface Signals**

Signal Name Suffix	Interface	Description
TLAST	S_AXIS	Transfer Last
TDATA	S_AXIS	Transfer Valid
TVALID	S_AXIS	Transfer Ready
TREADY	M_AXIS	Transfer Data
TSTRB	S_AXIS	Transfer Strobe
TKEEP	S_AXIS	Transfer Keep
TID	S_AXIS	Transfer ID
TDEST	S_AXIS	Transfer Destination
TUSER	S_AXIS	Transfer User bits

Table 2-19: **Trace Input Interface Signals**

Signal Name Suffix	Interface	Description
Valid_Instr	C_TRACE_SIZE = 0,1,2	Valid instruction on trace port
Instruction	C_TRACE_SIZE = 0	Instruction code
PC	C_TRACE_SIZE = 0	Program counter
Reg_Write	C_TRACE_SIZE = 0,1,2	Instruction writes to the register file
Reg_Addr	C_TRACE_SIZE = 0,1,2	Destination register address
MSR_Reg	C_TRACE_SIZE = 0	Machine status register

Table 2-19: **Trace Input Interface Signals *(Cont'd)***

Signal Name Suffix	Interface	Description
PID_Reg	C_TRACE_SIZE = 0	Process identifier register
New_Reg_Value	C_TRACE_SIZE = 0,1,2	Destination register update value
Exception_Taken	C_TRACE_SIZE = 0	Instruction results in taken exception
Exception_Kind	C_TRACE_SIZE = 0	Exception type
Jump_Taken	C_TRACE_SIZE = 0	Branch instruction evaluated true
Jump_Hit	C_TRACE_SIZE = 0	Branch Target Cache hit
Delay_Slot	C_TRACE_SIZE = 0	Instruction is in delay slot of taken branch
Data_Access	C_TRACE_SIZE = 0,2	Valid D-side memory access
Data_Address	C_TRACE_SIZE = 0,2	Address for D-side memory access
Data_Write_Value	C_TRACE_SIZE = 0,2	Value for D-side memory access
Data_Byte_Enable	C_TRACE_SIZE = 0,2	Byte enables for D-side memory access
Data_Read	C_TRACE_SIZE = 0,2	D-side memory access is read
Data_Write	C_TRACE_SIZE = 0,2	D-side memory access is write
DCache_Req	C_TRACE_SIZE = 0	Data memory address within D-Cache
DCache_Hit	C_TRACE_SIZE = 0	Data memory address present
DCache_Rdy	C_TRACE_SIZE = 0	Data memory access completed
DCache_Read	C_TRACE_SIZE = 0	D-Cache request is a read
ICache_Req	C_TRACE_SIZE = 0	Instruction memory address within I-Cache
ICache_Hit	C_TRACE_SIZE = 0	Instruction memory address present
ICache_Rdy	C_TRACE_SIZE = 0	Instruction memory access completed
OF_PipeRun	C_TRACE_SIZE = 0	Pipeline advance for Decode stage
EX_PipeRun	C_TRACE_SIZE = 0	Pipeline advance for Execute stage
MEM_PipeRun	C_TRACE_SIZE = 0	Pipeline advance for Memory stage
MB_Halted	C_TRACE_SIZE = 0	Pipeline is halted by debug

Table 2-20: **IOBUS Input Interface Signals**

Signal Name Suffix	Interface	Description
Addr_Strobe	IO	Address Strobe
Read_Strobe	IO	Read Strobe
Write_Strobe	IO	Write Strobe
Address	IO	Address
Byte_Enable	IO	Byte Enable

Table 2-20: **IOBUS Input Interface Signals *(Cont'd)***

Signal Name Suffix	Interface	Description
Write_Data	IO	Write Data
Read_Data	Unused	Read Data
Ready	Unused	Ready

Table 2-21: **GPIO Input Interface Signals**

Signal Name Suffix	Interface	Description
GPO	GPIO	General Purpose Output
GPT	GPIO	General Purpose Tristate
GPI	Unused	General Purpose Input

Table 2-22: **UART Input Interface Signals**

Signal Name Suffix	Interface	Description
Rx	Unused	Serial Input
Tx	UART	Serial Output
RTSn	UART	Request to Send
CTSn	Unused	Clear to Send
DTRn	UART	Data Terminal Ready
DSRn	Unused	Data Set Ready
BAUDOUTn	UART	Baud Out
RCLK	Unused	Receiver Clock
XIN	Unused	External Crystal Input
XOUT	UART	External Crystal Output
DCDn	Unused	Data Carrier Detect
DDIS	UART	Driver Disable
OUT1n	UART	User Controlled Output 1
OUT2n	UART	User Controlled Output 2
TXRDYn	UART	Transmitter DMA Control Signal
RXRDYn	UART	Receiver DMA Control Signal
RI	Unused	Ring Indicator

TMR Inject

The I/O signals for the TMR Inject are listed and described in Table 2-23.

Table 2-23: **TMR Inject I/O Signals**

Signal Name	Interface	I/O	Initial State	Description
Clk		I	-	Clock
Rst		I	-	Synchronous reset
LMB Slave Interface				
LMB_ABus[0:C_LMB_AWIDTH-1]	LMB	I	-	Data Address
LMB_WriteDBus[0:C_LMB_DWIDTH-1]	LMB	I	-	Data Write Bus
LMB_AddrStrobe	LMB	I	-	Address Strobe
LMB_ReadStrobe	LMB	I	-	Read Strobe
LMB_WriteStrobe	LMB	I	-	Write Strobe
LMB_BE[0:C_LMB_DWIDTH/8-1]	LMB	I	-	Byte Enable
Sl_DBus[0:C_LMB_DWIDTH-1]	LMB	O	0x0	Data Read Bus
Sl_Ready	LMB	O	0	Ready
Sl_Wait	LMB	O	0	Wait
Sl_CE	LMB	O	0	Correctable Error
Sl_UE	LMB	O	0	Uncorrectable Error
MicroBlaze LMB Interface				
MB_LMB_ABus[0:C_INJECT_LMB_AWIDTH-1]	MB_LMB	I	-	Data Address
MB_LMB_WriteDBus[0:C_INJECT_LMB_DWIDTH-1]	MB_LMB	I	-	Data Write Bus
MB_LMB_AddrStrobe	MB_LMB	I	-	Address Strobe
MB_LMB_ReadStrobe	MB_LMB	I	-	Read Strobe
MB_LMB_WriteStrobe	MB_LMB	I	-	Write Strobe
MB_LMB_BE[0:C_INJECT_LMB_DWIDTH/8-1]	MB_LMB	I	-	Byte Enable
MB_Sl_DBus[0:C_INJECT_LMB_DWIDTH-1]	MB_LMB	O	0x0	Data Read Bus
MB_Sl_Ready	MB_LMB	O	0	Ready
MB_Sl_Wait	MB_LMB	O	0	Wait
MB_Sl_CE	MB_LMB	O	0	Correctable Error
MB_Sl_UE	MB_LMB	O	0	Uncorrectable Error
LMB BRAM Interface				
BRAM_LMB_ABus[0:C_INJECT_LMB_AWIDTH-1]	BRAM_LMB	I	-	Data Address
BRAM_LMB_WriteDBus[0:C_INJECT_LMB_DWIDTH-1]	BRAM_LMB	I	-	Data Write Bus
BRAM_LMB_AddrStrobe	BRAM_LMB	I	-	Address Strobe
BRAM_LMB_ReadStrobe	BRAM_LMB	I	-	Read Strobe
BRAM_LMB_WriteStrobe	BRAM_LMB	I	-	Write Strobe
BRAM_LMB_BE[0:C_INJECT_LMB_DWIDTH/8-1]	BRAM_LMB	I	-	Byte Enable

Table 2-23: **TMR Inject I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
BRAM_SI_DBus[0:C_INJECT_LMB_DWIDTH-1]	BRAM_LMB	O	0x0	Data Read Bus
BRAM_SI_Ready	BRAM_LMB	O	0	Ready
BRAM_SI_Wait	BRAM_LMB	O	0	Wait
BRAM_SI_CE	BRAM_LMB	O	0	Correctable Error
BRAM_SI_UE	BRAM_LMB	O	0	Uncorrectable Error

TMR SEM

The I/O signals for the TMR SEM are listed and described in Table 2-24.

Table 2-24: **TMR SEM I/O Signals**

Signal Name	Interface	I/O	Initial State	Description
S_AXI_ACLK	S_AXI	I	-	AXI Clock
S_AXI_ARESETN	S_AXI	I	-	AXI Reset, active-Low
Interrupt		O	0	Interrupt when SEM output available
SEM Interface Signals[(1)] (C_SEM_STATUS = 1)				
SEM_heartbeat		O	0	SEM Heartbeat status output
SEM_initialization		O	0	SEM Initialization status output
SEM_observation		O	0	SEM Observation status output
SEM_correction		O	0	SEM Correction status output
SEM_classification		O	0	SEM Classification status output
SEM_injection		O	0	SEM Injection status output
SEM_essential		O	0	SEM Essential status output
SEM_uncorrectable		O	0	SEM Uncorrectable status output
SEM_diagnostic_scan[(2)]		O	0	SEM Diagnostic scan status output
SEM_detect_only[(2)]		O	0	SEM Detect only status output
AXI4-Lite Slave Interface (C_INTERFACE = 0)				
S_AXI_AWADDR [C_S_AXI_ADDR_WIDTH-1:0]	S_AXI	I	-	Write Address
S_AXI_AWVALID	S_AXI	I	-	Write Address Valid
S_AXI_AWREADY	S_AXI	O	0	Write Address Ready
S_AXI_WDATA [C_S_AXI_DATA_WIDTH-1:0]	S_AXI	I	-	Write Data
S_AXI_WSTB [C_S_AXI_DATA_WIDTH/8-1:0]	S_AXI	I	-	Write Strobes
S_AXI_WVALID	S_AXI	I	-	Write Valid

Table 2-24: **TMR SEM I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Initial State	Description
S_AXI_WREADY	S_AXI	O	0	Write Ready
S_AXI_BRESP[1:0]	S_AXI	O	0x0	Write Response
S_AXI_BVALID	S_AXI	O	0	Write Response Valid
S_AXI_BREADY	S_AXI	I	-	Write Response Ready
S_AXI_ARADDR [C_S_AXI_ADDR_WIDTH-1:0]	S_AXI	I	-	Read Address
S_AXI_ARVALID	S_AXI	I	-	Read Address Valid
S_AXI_ARREADY	S_AXI	O	0	Read Address Ready
S_AXI_RDATA [C_S_AXI_DATA_WIDTH-1:0]	S_AXI	I	-	Read Data
S_AXI_RRESP[1:0]	S_AXI	O	0x0	Read Response
S_AXI_RVALID	S_AXI	O	0	Read Valid
S_AXI_RREADY	S_AXI	I	-	Read Ready
UART Signals (C_INTERFACE = 1)				
UART_Rx	UART	I	-	UART SEM monitor receive data
UART_Tx	UART	O	1	UART SEM monitor transmit data

Notes:

1. For details see *Soft Error Mitigation Controller Product Guide* (PG036) [Ref 6] and *UltraScale Architecture Soft Error Mitigation Controller Product Guide* (PG187) [Ref 7].
2. Only available on UltraScale architecture devices.

Register Space

TMR Manager

Table 2-25 describes the TMR Manager registers accessible through the LMB slave interface.

Table 2-25: **TMR Manager Registers**

Register Name	Size (bits)	Address Offset	R/W	Description
CR	20	0x00	W	Control Register
FFR	22	0x04	R/W	First Failing Register
CMR0	32	0x08	W	Comparison Mask Register 0
CMR1	32	0x0C	W	Comparison Mask Register 1
BDIR	1-32	0x10	W	Break Delay Initialization Register
SEMSR	11	0x14	R	SEM Status Register
SEMSSR	11	0x18	R/W	SEM Sticky Status Register
SEMIMR	11	0x1C	W	SEM Interrupt Mask Register
WR	32	0x20	W	Watchdog Register
RFSR	32	0x24	W	Reset Failing State Register
CSCR	32	0x28	W	Comparator Status Clear Register
CFIR	4	0x2C	W	Comparator Fault Inject Register

Control Register (CR)

The control register determines how the TMR manager handles any detected faults. This is write only register. Issuing a read request to the control register generates the read acknowledgment with zero data. The register bit assignment is shown in Table 2-26 and described in Table 2-27.

Table 2-26: **Control Register (CR)**

Reserved		BFO	BB	BFR	RIR	MAGIC2		MAGIC1	
31	20	19	18	17	16	15	8	7	0

Table 2-27: **Control Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-20	Reserved	N/A	0	Reserved
19	Block Fatal Output	W	0	Determine if the Fatal output signal is blocked: 0 = Fatal output signal is set on fatal error. 1 = Block Fatal output signal.

Table 2-27: **Control Register Bit Definitions *(Cont'd)***

Bits	Name	Access	Reset Value	Description
18	Block Break	W	0	Determine if Break signal is sent to MicroBlaze: 0 = Break signal is sent. 1 = Block break signal.
17	Block Fatal Reset	W	0	Determine if a fatal error causes a reset: 0 = Reset on fatal error. 1 = Block reset on fatal error.
16	Recover Is Reset	W	0	Treat MicroBlaze Recover signal as reset: 0 = No reset on recover. 1 = Reset on recover.
15-8	Magic Byte 2	W	0x00	Magic Byte 2 Must be set to the value of parameter C_MAGIC2 to: • Enable masking of comparators defined in CMR0-1 • Allow the BFR bit to block reset on fatal error • Allow the BFO bit to block fatal output signal • Reset failing status by writing any value to RFSR
7-0	Magic Byte 1	W	0x00	Magic Byte 1 Must be set to the value of parameter C_MAGIC1 to: • Allow the RIR bit to generate reset on recovery • Allow the BB bit to block the break signal • Clear FFR by writing any value

First Failing Register (FFR)

The first failing register contains the TMR subsystem fault status. This is a read/write register. Issuing a write request to the first failing register with any data clears the entire register, provided that the MAGIC1 field in the Control Register is correctly set. The register bit assignment is shown in Table 2-28 and described in Table 2-29.

Table 2-28: **First Failing Register (FFR)**

Reserved		WE	FAT UE	FAT V	FAT 23	FAT 13	FAT 12	Reserved		REC	LM 23	LM 13	LM 12
31	22	21	20	19	18	17	16	15	4	3	2	0	0

Table 2-29: **First Failing Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-22	Reserved	N/A	0	Reserved
21	Watchdog Expired	R/W	0	Indicates if the watchdog has expired: 0 - The watchdog has not expired. 1 - The watchdog has expired.

Table 2-29: **First Failing Register Bit Definitions *(Cont'd)***

Bits	Name	Access	Reset Value	Description
20	Fatal Uncorrectable Error	R/W	0	Fatal ECC uncorrectable error has occurred: 0 = No fatal error. 1 = A fatal error has occurred.
19	Fatal Voter Error	R/W	0	Fatal voter error has occurred: 0 = No fatal error. 1 = A fatal error has occurred.
18	Fatal 2-3	R/W	0	Fatal error has occurred for processors 2 and 3 causing transition to Fatal state: 0 = No fatal error. 1 = A fatal error has occurred.
17	Fatal 1-3	R/W	0	Fatal error has occurred for processors 1 and 3 causing transition to Fatal state: 0 = No fatal error. 1 = A fatal error has occurred.
16	Fatal 1-2	R/W	0	Fatal error has occurred for processors 1 and 2 causing transition to Fatal state: 0 = No fatal error. 1 = A fatal error has occurred.
15-4	Reserved	N/A	0	Reserved
3	Recovery	R/W	0	Indicates if a recovery has been performed: 0 = No recovery. 1 = A recovery has occurred.
2[(1)]	Lockstep mismatch 2-3	R/W	0	Lockstep mismatch between processor 2 and 3 causing transition to Lockstep state: 0 = No lockstep mismatch. 1 = Lockstep mismatch has occurred.
1[(1)]	Lockstep mismatch 1-3	R/W	0	Lockstep mismatch between processor 1 and 3 causing transition to Lockstep state: 0 = No lockstep mismatch. 1 = Lockstep mismatch has occurred.
0[(1)]	Lockstep mismatch 1-2	R/W	0	Lockstep mismatch between processor 1 and 2 causing transition to Lockstep state: 0 = No lockstep mismatch. 1 = Lockstep mismatch has occurred.

Notes:

1. The Lockstep mismatch bits indicate which processors have detected a mismatch, and if one processor is faulty the two others will detect a mismatch. Consequently, two of the three bits will be set when one processor can be identified as faulty. For example, if processor 1 is faulty, both processors 2 and 3 will detect a mismatch, and the bits will have the binary value 011. In case only one or all three of the bits are set, the processors do not agree on which one is faulty, and recovery is not possible.

Comparison Mask Registers (CMR0, CMR1)

These two registers contain individual bits to mask the corresponding Compare input signals during test. CMR0 masks Compare_0 to Compare_31, and CMR1 masks Compare_32 to Compare_63. The register definitions are shown in Table 2-30. The registers are write-only. Issuing a read request generates the read acknowledgment with zero data. The registers are only implemented if C_COMPARATORS_MASK is set to 1, and the available number of bits is determined by C_NO_OF_COMPARATORS. The initial value after reset is determined by C_MASK_RST_VALUE.

Table 2-30: **Comparison Mask Registers (CMR0, CMR1)**

Comparison Mask	
31	0

Break Delay Initialization Register (BDIR)

This register contains the counter defining the delay from a fatal error has been detected until the MicroBlaze break signal is set to 1. The register definition is shown in Table 2-31. The register is write-only. Issuing a read request generates the read acknowledgment with zero data. The register is only implemented if C_BRK_DELAY_WIDTH is greater than 0, and the initial value after reset is determined by C_BRK_DELAY_RST_VALUE.

Table 2-31: **Break Delay Initialization Register (BDIR)**

Reserved		Break Delay	
31	C_BRK_DELAY_WIDTH	C_BRK_DELAY_WIDTH-1	0

SEM Status Register (SEMSR)

The SEM status register contains the status signals from the TMR SEM core. This is a read only register. If a write request is issued to the SEM status register it does nothing but generate a write acknowledgment. The register bit assignment is shown in Table 2-32 and described in Table 2-33. The register is only implemented if C_SEM_INTERFACE is set to 1.

Table 2-32: **SEM Status Register (SEMSR)**

Reserved		DS	DO	ESS	UNC	INJ	CLA	CORR	OBS	INI	HB	HBWE
31	11	10	9	8	7	6	5	4	3	2	1	0

Table 2-33: **SEM Status Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-11	Reserved	N/A	0	Reserved
10	Diagnostic Scan	R	0	SEM Diagnostic Scan input signal value.
9	Detect Only	R	0	SEM Detect Only input signal value.
8	Essential	R	0	SEM Essential input signal value.

Table 2-33: **SEM Status Register Bit Definitions *(Cont'd)***

Bits	Name	Access	Reset Value	Description
7	Uncorrectable	R	0	SEM Uncorrectable input signal value.
6	Injection	R	0	SEM Injection input signal value.
5	Classification	R	0	SEM Classification input signal value.
4	Correction	R	0	SEM Correction input signal value.
3	Observation	R	0	SEM Observation input signal value.
2	Initialization	R	0	SEM Initialization input signal value.
1	Heartbeat	R	0	SEM Heartbeat input signal value.
0	Heartbeat Watchdog Expired	R	0	The SEM heartbeat watchdog has expired: 0 = The watchdog has not expired. 1 = The watchdog has expired.

SEM Sticky Status Register (SEMSSR)

The SEM sticky status register saves the status signals from the TMR SEM core. This is a read/write register. If a write request is issued to the SEM sticky status register bits set to one in the written data are cleared. The register bit assignment is shown in Table 2-34 and described in Table 2-35. The register is only implemented if C_SEM_INTERFACE is set to 1.

Table 2-34: **SEM Sticky Status Register (SEMSSR)**

Reserved		DS	DO	ESS	UNC	INJ	CLA	CORR	OBS	INI	HB	HBWE
31	11	10	9	8	7	6	5	4	3	2	1	0

Table 2-35: **SEM Sticky Status Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-11	Reserved	N/A	0	Reserved
10	Diagnostic Scan	R	0	Set to 1 by the SEM Diagnostic Scan input. Cleared by writing 1 to the bit.
9	Detect Only	R	0	Set to 1 by the SEM Detect Only input. Cleared by writing 1 to the bit.
8	Essential	R	0	Set to 1 by the SEM Essential input. Cleared by writing 1 to the bit.
7	Uncorrectable	R	0	Set to 1 by the SEM Uncorrectable input. Cleared by writing 1 to the bit.
6	Injection	R	0	Set to 1 by the SEM Injection input. Cleared by writing 1 to the bit.
5	Classification	R	0	Set to 1 by the SEM Classification input. Cleared by writing 1 to the bit.
4	Correction	R	0	Set to 1 by the SEM Correction input. Cleared by writing 1 to the bit.

Table 2-35: **SEM Sticky Status Register Bit Definitions *(Cont'd)***

Bits	Name	Access	Reset Value	Description
3	Observation	R	0	Set to 1 by the SEM Observation input. Cleared by writing 1 to the bit.
2	Initialization	R	0	Set to 1 by the SEM Initialization input. Cleared by writing 1 to the bit.
1	Heartbeat	R	0	Set to 1 by the SEM Heartbeat input signal. Cleared by writing 1 to the bit.
0	Heartbeat Watchdog Expired	R	0	The SEM heartbeat watchdog has expired since writing this register to clear the status: 0 = The watchdog has not expired. 1 = The watchdog has expired. Cleared by writing 1 to the bit.

SEM Interrupt Mask Register (SEMIMR)

The SEM interrupt mask register enables interrupt generation from signals from the TMR SEM interface. This is a write only register.Issuing a read request generates the read acknowledgment with zero data. The register bit assignment is shown in Table 2-36 and described in Table 2-37. The register is only implemented if C_SEM_INTERFACE is set to 1.

Table 2-36: **SEM Interrupt Mask Register (SEMIMR)**

Reserved		DS	DO	ESS	UNC	INJ	CLA	CORR	OBS	INI	HB	HBWE
31	11	10	9	8	7	6	5	4	3	2	1	0

Table 2-37: **SEM Interrupt Mask Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-11	Reserved	N/A	0	Reserved
10	Diagnostic Scan	R	0	The SEM Diagnostic Scan input causes an interrupt.
9	Detect Only	R	0	The SEM Detect Only input causes an interrupt.
8	Essential	R	0	The SEM Essential input causes an interrupt.
7	Uncorrectable	R	0	The SEM Uncorrectable input causes an interrupt.
6	Injection	R	0	The SEM Injection input causes an interrupt.
5	Classification	R	0	The SEM Classification input causes an interrupt.
4	Correction	R	0	The SEM Correction input causes an interrupt.
3	Observation	R	0	The SEM Observation input causes an interrupt.
2	Initialization	R	0	The SEM Initialization input causes an interrupt.
1	Heartbeat	R	0	The SEM Heartbeat input causes an interrupt.
0	Heartbeat Watchdog Expired	R	0	The SEM heartbeat watchdog causes an interrupt: 0 = No interrupt. 1 = An interrupt occurs when the watchdog expires.

Watchdog Register (WR)

This register is used to restart the TMR manager internal software watchdog. Issuing a write request to the watchdog register with any data restarts the watchdog, preventing it from expiring. The register definition is shown in Table 2-38. The register is write-only. Issuing a read request generates the read acknowledgment with zero data. The register is only implemented if C_WATCHDOG is set to 1.

Table 2-38: **Watchdog Register (WR)**

Watchdog	
31	0

Reset Failing State Register (RFSR)

This register is used to reset the failing state, to return to nominal operation. Issuing a write request to the reset failing state register with any data clears the failing state, provided that the MAGIC2 field in the Control Register is correctly set, and the Debug_Disable input signal is cleared to zero. The register definition is shown in Table 2-39. The register is write-only. Issuing a read request generates the read acknowledgment with zero data.

Table 2-39: **Reset Failing State Register (RFSR)**

Reset Failing State	
31	0

Comparator Status Clear Register (CSCR)

This register is used to control comparator testing. Issuing a write request to the comparator status clear register with any data clears the comparator status. The register definition is shown in Table 2-40. The register is write-only. Issuing a read request generates the read acknowledgment with zero data. The register is only implemented if C_TEST_COMPARATOR is greater than 0.

Table 2-40: **Comparator Status Clear Register (CSCR)**

31	0

Comparator Fault Inject Register (CFIR)

This register is used to control comparator fault injection. The register definition is shown in Table 2-41 and described in Table 2-42. The register is write-only. Issuing a read request generates the read acknowledgment with zero data. The register is only implemented if C_TEST_COMPARATOR is set to 2.

Table 2-41: **Comparator Fault Inject Register (CFIR)**

		IVE	IE23	IE13	IE12
31	4	3	2	1	0

Table 2-42: **Comparator Fault Inject Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-4	Reserved	N/A	0	Reserved
3	Inject Voter Error	W	0	Inject a comparison error in the voter checker: 0 = No error. 1 = An error is injected.
2	Inject Error 2-3	W	0	Inject a comparison error between processor 2 and 3: 0 = No error. 1 = An error is injected.
1	Inject Error 1-3	W	0	Inject a comparison error between processor 1 and 3: 0 = No error. 1 = An error is injected.
0	Inject Error 1-2	W	0	Inject a comparison error between processor 1 and 2: 0 = No error. 1 = An error is injected.

TMR Inject

Table 2-43 describes the TMR Inject registers accessible through the LMB slave interface.

Table 2-43: **TMR Inject Registers**

Register Name	Size (bits)	Address Offset	R/W	Description
CR	20	0x0	W	Control Register
AIR	32	0x4	W	Address Inject Register
IIR	32	0x8	W	Instruction Inject Register
EAIR	64	0x14-0x10	W	Extended Address Inject Register

Control Register (CR)

The control register contains the CPU ID and enables fault injection. This is write only register. Issuing a read request to the control register generates the read acknowledgment with zero data. The register bit assignment is shown in Table 2-44 and described in Table 2-45.

Table 2-44: **Control Register (CR)**

Reserved		INJ	CPU		MAGIC	
31	11	10	9	8	7	0

Table 2-45: **Control Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-11	Reserved	N/A	0	Reserved
10	Interrupt Enabled	W	0	Enables fault injection. Automatically cleared after a fault has been injected: 0 = No fault is injected. 1 = Inject fault when AIR is equal to PC.
9-8	CPU ID	W	00	CPU identifier that must match parameter C_CPU_ID to enable injection.
7-0	Magic byte	W	0x00	Magic byte that must match parameter C_MAGIC to enable injection.

Address Inject Register (AIR)

This register is used to define the address (MicroBlaze Program Counter value) where a fault is injected. The register definition is shown in Table 2-46. The register is write-only. Issuing a read request generates the read acknowledgment with zero data. When setting the Address Inject Register, it is necessary to ensure that the processor has not already prefetched the instruction at that Program Counter to ensure that the instruction can be replaced.

Note: The AIR register should be used when the Program Counter has 32 bits.

Table 2-46: **Address Inject Register (AIR)**

Address Inject	
31	0

Instruction Inject Register (IIR)

This register is used set the instruction replacing the actual instruction when a fault is injected. The register definition is shown in Table 2-47. The register is write-only. Issuing a read request generates the read acknowledgment with zero data.

Table 2-47: **Instruction Inject Register (IIR)**

Instruction Inject	
31	0

Extended Address Inject Register (EAIR)

This register is used to define the address (MicroBlaze Program Counter value) where a fault is injected. The register definition is shown in Table 2-48. The register is write-only. Issuing a read request generates the read acknowledgment with zero data. When setting the Extended Address Inject Register, it is necessary to ensure that the processor has not already prefetched the instruction at that Program Counter to ensure that the instruction can be replaced.

Note: The EAIR register should be used when the Program Counter has more than 32 bits.

Table 2-48: **Extended Address Inject Register (EAIR)**

+0	Address Inject[31:0]	
	31	0

+4	Address Inject[C_INJECT_LMB_AWIDTH-1:32]	
	C_INJECT_LMB_AWIDTH-1	32

TMR SEM

Table 2-49 describes the TMR SEM registers accessible through the AXI4-Lite slave interface. The TMR SEM registers are software compatible with the AXI UART Lite and MDM IP cores.

Table 2-49: **TMR SEM Monitor Registers**

Register Name	Size (bits)	Address Offset	R/W	Description
C_INTERFACE = 0				
MON_RECEIVE	8	0x0	R	Monitor receive data
MON_TRANSMIT	8	0x4	W	Monitor transmit data
MON_STATUS	8	0x8	R	Monitor status
MON_CTRL	8	0xC	W	Monitor control

The monitor registers are identical to the AXI UART Lite registers (see the *AXI UART Lite LogiCORE IP Product Guide* (PG142) [Ref 8]), except that the Status register bits 5–7 (Overrun Error, Frame Error, Parity Error) are never set.

Monitor Receive Register (MON_RECEIVE)

This 32-entry-deep FIFO contains data received from the SEM monitor. The FIFO bit definitions are shown in Table 2-50. Reading this register results in reading the data word from the top of the FIFO. When a read request is issued to an empty FIFO, a bus error (SLVERR) is generated and the result is undefined. The register is a read-only register. Issuing a write request to the receive data FIFO does nothing but generates a successful write acknowledgment. Table 2-51 shows the location for data on the AXI slave interface. The register is only implemented if C_INTERFACE is set to 0.

Table 2-50: **Monitor Receive Register (MON_RECEIVE)**

Reserved		MON_RX	
31	8	7	0

Table 2-51: **Monitor Receive Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31:8	-	R	0	Reserved
7:0	MON_RX	R	0	Monitor Receive Data

Monitor Transmit Register (MON_TRANSMIT)

This 32-entry-deep FIFO contains data transmitted to the SEM monitor. The FIFO bit definitions are shown in Table 2-52. Data to be transmitted is written into this register. When a write request is issued while the FIFO is full, a bus error (SLVERR) is generated and the data is not written into the FIFO. This is a write-only location. Issuing a read request to the transmit data FIFO generates the read acknowledgment with zero data. Table 2-53 shows the location for data on the AXI interface. The register is only implemented if C_INTERFACE is set to 0.

Table 2-52: **Monitor Transmit Register (MON_TRANSMIT)**

Reserved		MON_TX	
31	8	7	0

Table 2-53: **Monitor Transmit FIFO Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31:8	-	R	0	Reserved
7:0	MON_TX	R	0	Monitor Transmit Data

Monitor Status Register (MON_STATUS)

The status register contains the status of the receive and transmit data FIFOs, and when interrupt is enabled. This is a read only register. If a write request is issued to the status register it does nothing but generate a write acknowledgment. The register bit assignment is shown in Table 2-54 and described in Table 2-55. The register is only implemented if C_INTERFACE is set to 0.

Table 2-54: **Monitor Status Register (MON_STATUS)**

Reserved		MON_STATUS	
31	5	4	0

Table 2-55: **Monitor Status Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-5	Reserved	N/A	0	Reserved
4	Interrupt Enabled	R	0	Indicates that interrupt is enabled. 0 = Interrupt is disabled 1 = Interrupt is enabled
3	TX FIFO Full	R	0	Indicates if the transmit FIFO is full. 0 = Transmit FIFO is not full 1 = Transmit FIFO is full
2	TX FIFO Empty	R	1	Indicates if the transmit FIFO is empty. 0 = Transmit FIFO is not empty 1 = Transmit FIFO is empty
1	RX FIFO Full	R	0	Indicates if the receive FIFO is full. 0 = Receive FIFO is not full 1 = Receive FIFO is full
0	RX FIFO Valid Data	R	0	Indicates if the receive FIFO has valid data. 0 = Receive FIFO is empty 1 = Receive FIFO has valid data

Monitor Control Register (MON_CTRL)

The control register contains the enable interrupt bit and reset for the receive and transmit data FIFO. This is write only register. Issuing a read request to the control register generates the read acknowledgment with zero data. The register bit assignment is shown in Table 2-56 and described in Table 2-57. The register is only implemented if C_INTERFACE is set to 0.

Table 2-56: **Monitor Control Register (MON_CTRL)**

Reserved		MON_CTRL	
31	5	4	0

Table 2-57: **Monitor Control Register Bit Definitions**

Bits	Name	Access	Reset Value	Description
31-5	Reserved	N/A	0	Reserved
4	Interrupt Enabled	W	0	Enable interrupt for the MDM JTAG Monitor 0 = Disable interrupt signal 1 = Enable interrupt signal
3-2	Reserved	N/A	0	Reserved

Table 2-57: **Monitor Control Register Bit Definitions *(Cont'd)***

Bits	Name	Access	Reset Value	Description
1	Reset RX FIFO	W	1	Reset/clear the receive FIFO Writing a 1 to this bit position clears the receive FIFO 0 = Do nothing 1 = Clear the receive FIFO
0	Reset TX FIFO	W	1	Reset/clear the transmit FIFO Writing a 1 to this bit position clears the transmit FIFO 0 = Do nothing 1 = Clear the transmit FIFO

Designing with the Subsystem

This chapter includes guidelines and additional information to facilitate designing with the subsystem.

General Design Guidelines

- All synchronization has to be made outside the TMR region, because asynchronous interfaces inside the TMR region might result in one cycle of jitter of the synchronized signal.
- All IP cores have to have identical configurations to their triplicated counterparts.
- MicroBlaze Debug is implemented by one of the MicroBlaze processors, with the other two connected as debug slaves.
- Recovery requires the complete state of all IP cores to be restored.

Clocking

The TMR region must be clocked with a single clock, since the triplicated sub-blocks must execute in lockstep.

The TMR SEM clock should be directly provided by the most reliable source possible. For additional details see "Clocking" in Chapter 3 of the *Soft Error Mitigation Controller Product Guide* (PG036) [Ref 6] or the *UltraScale Architecture Soft Error Mitigation Controller Product Guide* (PG187) [Ref 7].

Resets

The TMR region must have a single reset, synchronous to the single TMR region clock since the triplicated sub-blocks must be reset to start executing in lockstep.

Tool Flow

It is highly recommended to use the Vivado IP integrator Block Automation provided for the TMR Manager to create the TMR subsystem.

IMPORTANT: *Ensure that you have a complete single-string MicroBlaze Vivado Block Design that has passed validation in Vivado IP integrator before attempting to triplicate it.*

To triplicate a subsystem the following tool flow can be used:

- Select all the IP cores to be included in the subsystem in the Block Design, ensuring that the selected cores share a single common clock and that their reset signals have a common source.
- Right-click on the selected cores, and select **Create Hierarchy...** in the menu.
- Enter the desired TMR subsystem name, and press **OK**.
- Validate the design, for example by using **Validate Design** in the context menu.
- Double-click on the created hierarchical sub-block.
- Add a TMR Manager IP core inside the sub-block, for example by using **Add IP...** in the context menu. This shows the Designer Assistance banner with the TMR Manager Block Automation.

TIP: *It is a good idea to save a copy of the project at this point, in case any changes of the triplicated subsystem, such as in the IP configuration, are necessary later on. Rather than performing such changes in all three sub-blocks manually, it is usually better to do the changes in the saved single-string project and then re-run the block automation.*

- Run the TMR Manager Block Automation by clicking on the **Run Block Automation** link, and select desired configuration in the dialog shown in Figure 3-1 to perform the triplication:
  - **Mode:** Select TMR or Lockstep mode. Use TMR to triplicate the sub-block and Lockstep to duplicate it.
  - **LMB Memory Configuration**: Select **Local** to triplicate the local memory for the best possible error detection, or select **Common with ECC** to use a common local memory with error correcting codes to reduce block RAM use.
  - **Software Watchdog**: Select **None** or **Internal** to enable the TMR Manager internal watchdog.
  - **SEM Interface**: Select **None** or **Internal** to add a TMR SEM interface in the triplicated subsystem. Use **External** to implement the functionality manually outside the subsystem in case the device is not supported by the TMR SEM core,

when SEM error classification is desired, when the ICAP and/or FRAME_ECC need to be shared, or when required by the SEM clock frequency restrictions.

- **SEM Heartbeat Watchdog**: Enable the TMR Manager SEM heartbeat watchdog to detect SEM issues.
- **Reconfiguration Delay**: Enable the TMR Manager Break delay functionality, to ensure that a reconfiguration is not attempted until after the SEM has scrubbed the complete configuration memory.
- **Enable Fault Injection Test**: Add TMR Inject cores to allow functional test of the error detection logic.
- **Enable Comparator Test:** Enable an exhaustive test of the comparator functionality.

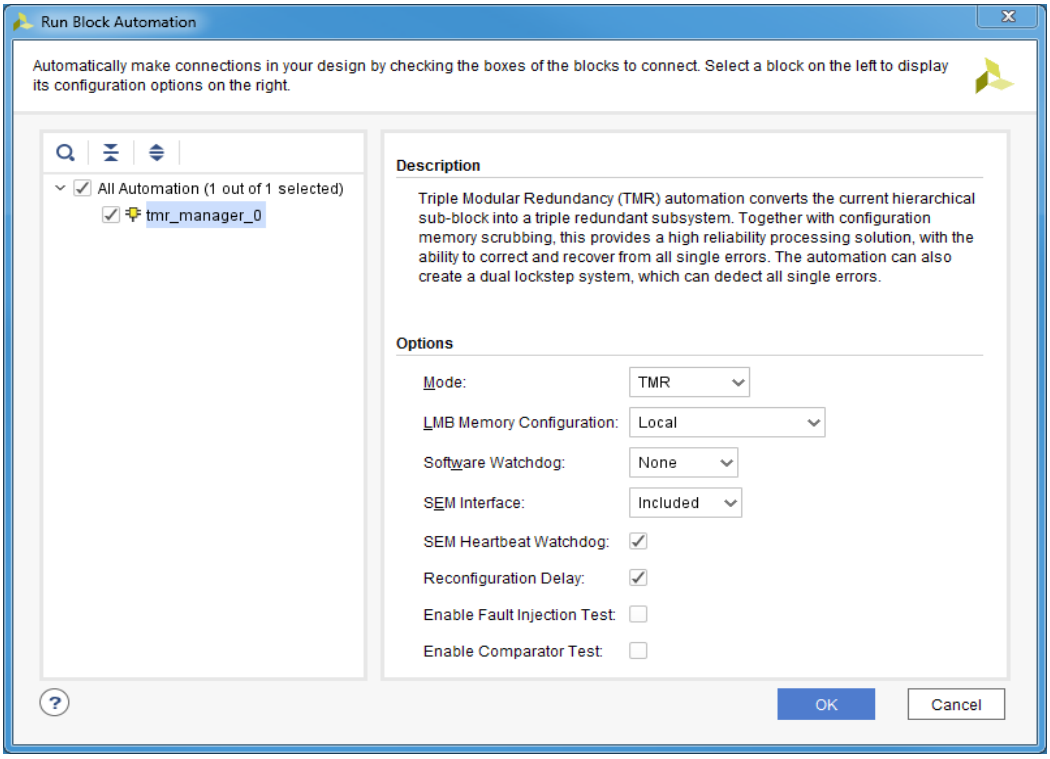

Figure 3-1: **TMR Manager Block Automation Dialog**

If the Block Automation detects an invalid configuration, a note is shown in the dialog. If you attempt to run the automation with an invalid configuration, a warning is issued, and the automation is aborted.

Design Flow Steps

This chapter describes customizing and generating the cores, constraining the cores, and the simulation, synthesis and implementation steps that are specific to these IP cores. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 9]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 11]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12]

Customizing and Generating the Cores

This section includes information about using Xilinx tools to customize and generate the cores in the Vivado Design Suite.

If you are customizing and generating the cores in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 9] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize each IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 11].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

TMR Manager

The TMR Manager core parameters are divided into three tabs in the Vivado Integrated Design Environment (IDE): System, Advanced, and LMB. When using Vivado IP integrator, the LMB addresses and masks are auto generated.

The TMR Manager System tab is shown in Figure 4-1.

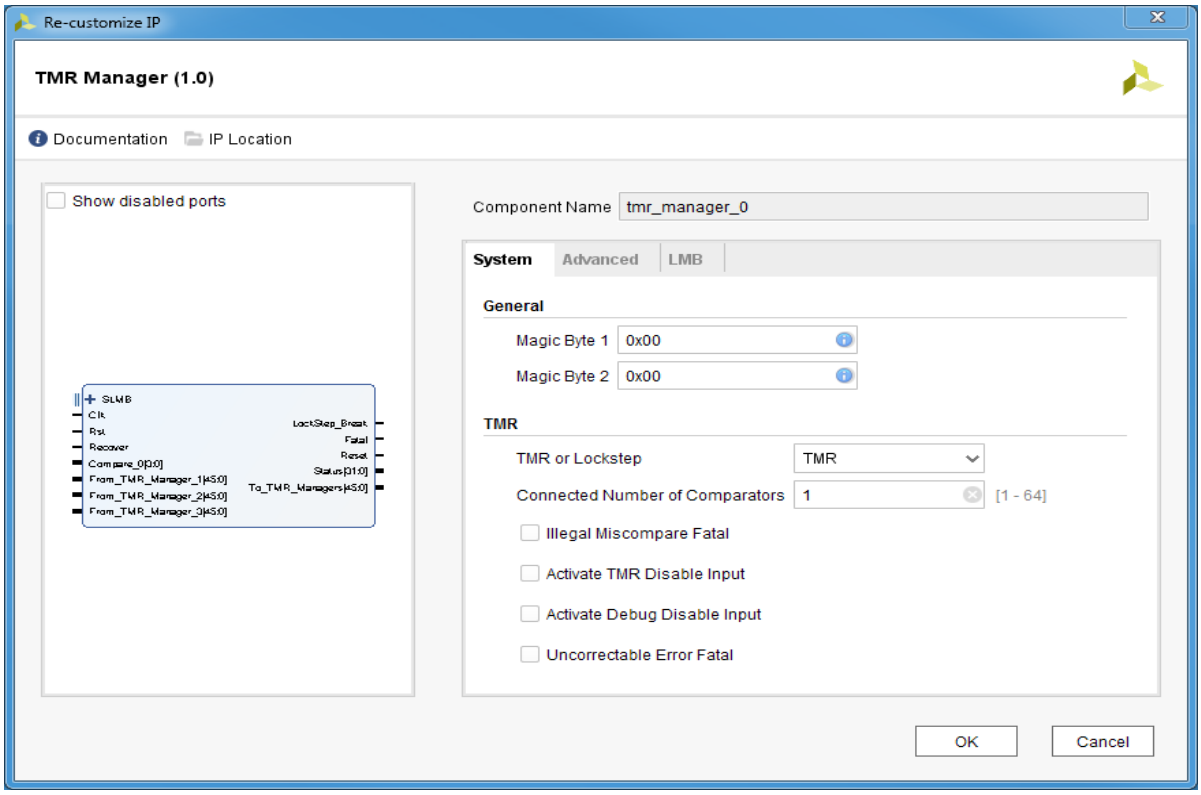

Figure 4-1: **TMR Manager System Tab**

Magic Byte 1 - Magic number 1. When writing to the control register the first write data byte (bits 7:0) must match this value in order to have any effect on the nominal recovery function.

Magic Byte 2 - Magic number 2. When writing to the control register the second write data byte (bits 15:8) must match this value in order to have any effect on the test function.

TMR or Lockstep - Select Triple Modular Redundancy (TMR) or Lockstep mode.

Connected Number of Comparators - Number of connected comparators, ranging from 1 to 64.

Illegal Miscompare Fatal - All illegal comparison failures (miscompare) are treated as fatal errors. An illegal miscompare occurs when only one of the three miscompare signals is set.

Activate TMR Disable Input - Activate TMR disable functionality. When activated and the `TMR_Disable` input is set to one, all processors except the first are disregarded in the TMR Manager and TMR voters.

Activate Debug Disable Input - Activate debug disable functionality. When activated and the `Debug_Disable` input is set to one, the comparator mask is used, fatal reset is blocked and reset of the TMR state machine is allowed.

Uncorrectable Error Fatal - Any uncorrectable error from LMB causes a fatal error. When enabled, the `UE_LMB` input is activated for connection of the UE signals from LMB.

The TMR Manager Advanced tab is shown in Figure 4-2.

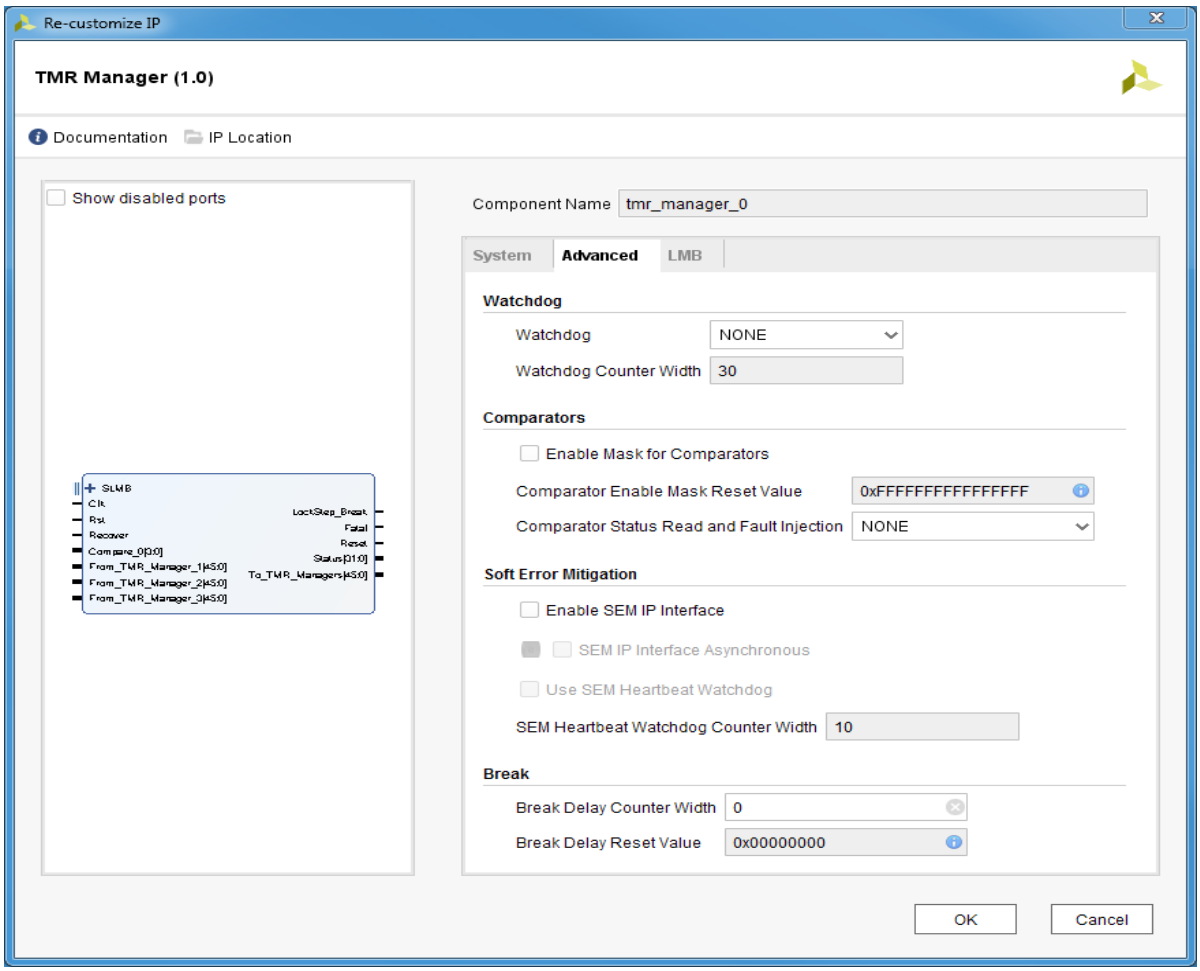

Figure 4-2: **TMR Manager Advanced Tab**

Watchdog - Enable internal or external watchdog. The watchdog is intended to monitor that software is behaving as expected. Software must periodically write to the watchdog to ensure that it does not elapse. Should the watchdog elapse, a fatal error occurs in the TMR Manager. When the external watchdog is selected, the `WatchDog_Expired` input is enabled.

Watchdog Counter Width - Number of bits in the internal watchdog.

Enable Mask for Comparators - Activate enable mask for comparators. The mask is used to define which comparators are active, to be able to test the functionality of individual comparators. The mask can be changed by writing to the TMR Manager Mask Registers.

Comparator Enable Mask Reset Value - The initial value of the comparator enable mask after reset. The 64-bit mask is used to set the reset values of the two TMR Manager Mask Registers.

Comparator Status Read and Fault Injection - Enable functional test of individual comparators. READ enables status read of comparison results, and INJECT also enables fault injection.

Enable SEM IP Interface - Enable Soft Error Mitigation (SEM) interface, for connection of the Soft Error Mitigation IP core.

SEM IP Interface Asynchronous - Use asynchronous Soft Error Mitigation (SEM) interface. Must be set when the Soft Error Mitigation IP core clock is different from the TMR Manager clock, to treat input signals as asynchronous. When using Vivado IP integrator, this value is auto generated.

Use SEM Heartbeat Watchdog - Use Soft Error Mitigation (SEM) Heartbeat Watchdog. If the watchdog elapses, the output signal `SEM_heartbeat_expired` is set to one and the SEM status is updated.

SEM Heartbeat Watchdog Counter Width - Number of bits in Soft Error Mitigation (SEM) Heartbeat Watchdog counter.

Break Delay Counter Width - Number of bits in the Break delay counter. The break delay counter defines a delay from an error occurs until the `LockStep_Break` output signal is set. The purpose of this counter is to wait until a complete scrubbing cycle of the configuration memory has been performed, before attempting a recovery. Setting the width to 0 disables the counter.

Break Delay Reset Value - Break delay counter initial value after reset.

The TMR Manager LMB tab is shown in Figure 4-3.

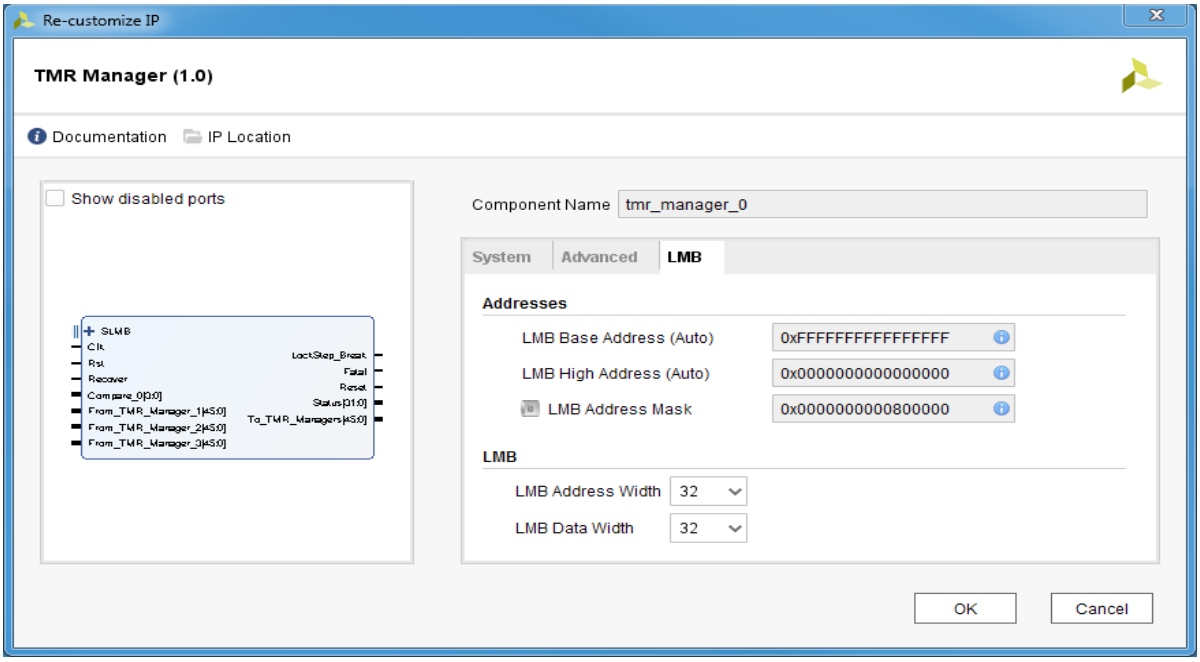

Figure 4-3: **TMR Manager LMB Tab**

LMB Base Address - This address specifies where the TMR Manager's register address space starts. This should never be set to the value of the MicroBlaze parameter C_BASE_VECTORS (normally 0x0000000000000000), since this is where MicroBlaze boots from. By default this value is larger than the high address so that an error is generated if this value is not specified. If IP integrator is used an address is automatically assigned.

LMB High Address - This address specifies where the TMR Manager's register address space ends. By default this value is smaller than the base address so that an error is generated if this value is not specified. If IP integrator is used an address is automatically assigned.

LMB Address Mask - IP integrator automatically sets this value to the mask of bits used to decode this peripheral on the LMB. Any bits that are set to one in the mask indicate that the address bit in that position is used to decode a valid LMB access. All other bits are considered don't cares for the purpose of decoding LMB accesses.

LMB Address Width - This parameter sets the LMB address width. The supported values range from 32 to 64 bits.

LMB Data Width - This parameter sets the LMB data width. The only supported value is 32.

TMR Voter

The TMR Voter System tab is exemplified with the Discrete interface in Figure 4-4.

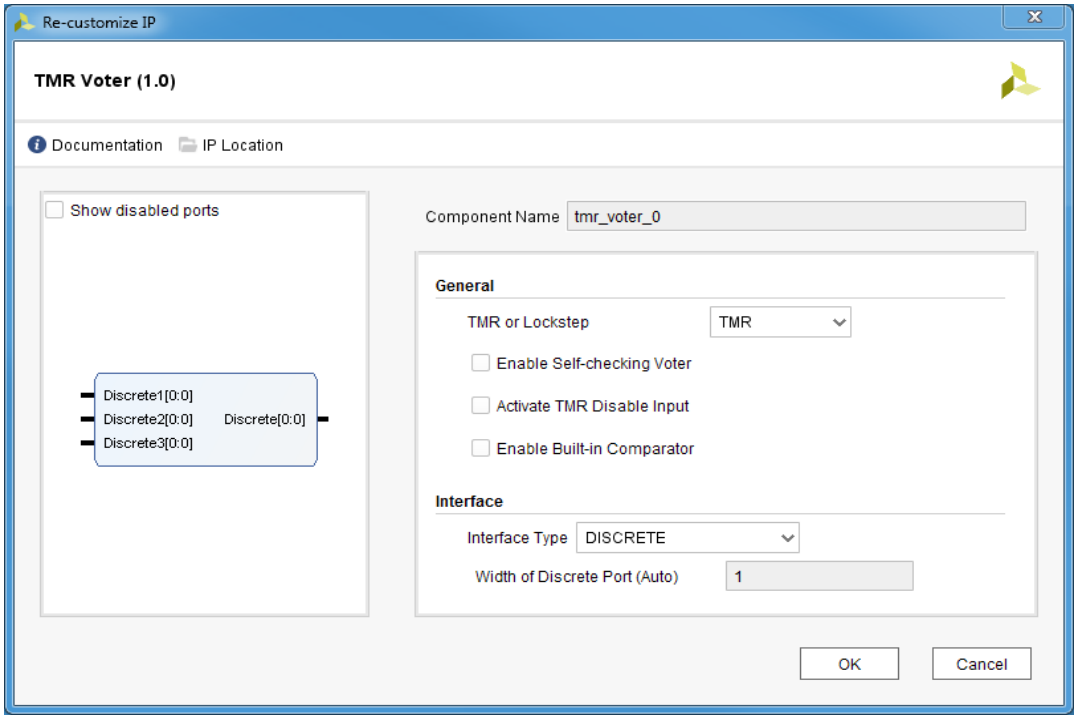

Figure 4-4: **TMR Voter System Tab**

TMR or Lockstep - Select Triple Modular Redundancy (TMR) or Lockstep mode.

Enable Self-checking Voter - Enable self-checking voter.

Activate TMR Disable Input - Activate TMR disable functionality.

Enable Build-in Comparator - Enable built-in comparator, useful with a triplicated LMB BRAM, where the read data must be voted and compared.

Interface Type - Select comparator interface. Depending on the selected type, the following interface-specific parameters are shown. With Vivado IP integrator these parameters are auto generated. The available interfaces are:

- Discrete

  Width of Discrete Port - Number of bits in the discrete port vector.

- LMB and LMB Slave - xilinx.com:interface:lmb:1.0

  LMB Address Width - LMB address width.

  LMB Data Width - LMB data width.

ECC Enabled - ECC enabled on LMB.

- BRAM and BRAM Master - xilinx.com:interface:bram:1.0
- AXI4, AXI4-Lite, ACE - xilinx.com:interface:aximm:1.0, xilinx.com:interface.acemm:1.0

  Address Width - AXI address width.

  Data Width - AXI data width.

  ID Width - AXI ID width.

  ARUSER, AWUSER, RUSER, WUSER, BUSER Width - AXI channel user widths.
- AXI4-Stream Master, AXI4-Stream Slave - xilinx.com:interface.axis:1.0

  Data Width - AXI-Stream data width in bits.

  ID Width - AXI-Stream ID width in bits.

  DEST Width - AXI-Stream destination width in bits.

  USER Width - AXI-Stream user width in bits.
- MicroBlaze Trace - xilinx.com:interface:mbtrace:2.0

  Data Size - Trace Data Size.
- MicroBlaze Interrupt - xilinx.com:interface:mbinterrupt:2.0

  Interrupt Latency - Low latency interrupt.
- I/O Bus - xilinx.com:interface:iobus:1.0
- GPIO - xilinx.com:interface:gpio:1.0

  GPIO Board Interface - Defines the board interface that GPIO is connected to.

  GPO Size - Bit size of the GPO and GPT signals.

  GPI Size - Bit size of the GPI signal.
- UART - xilinx.com:interface:uart:1.0

  UART Board Interface - Defines the board interface that the UART is connected to.

The TMR Voter Advanced tab is shown in Figure 4-5. This tab is only activated when **Enable Build-in Comparator** is selected, and is identical to the equivalent tab on the TMR Comparator.

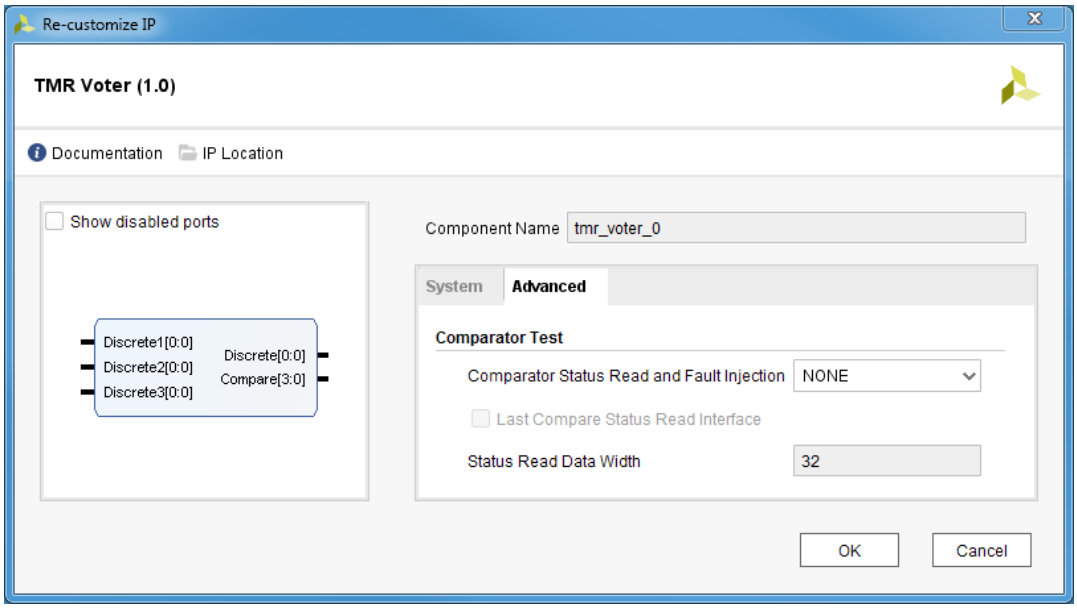

Figure 4-5: **TMR Voter Advanced Tab**

Comparator Status Read and Fault Injection - Enable functional test of individual comparators via AXI4-Stream interfaces. READ enables status read of comparison results, and INJECT also enables fault injection.

Last Compare Status Read Interface - Set for the last comparison status read interface, furthest from the processor reading the status.

Status Read Data Width - Define the AXI-Stream data width for comparison status read.

TMR Comparator

The TMR Comparator System tab is exemplified with the Discrete interface in Figure 4-6.

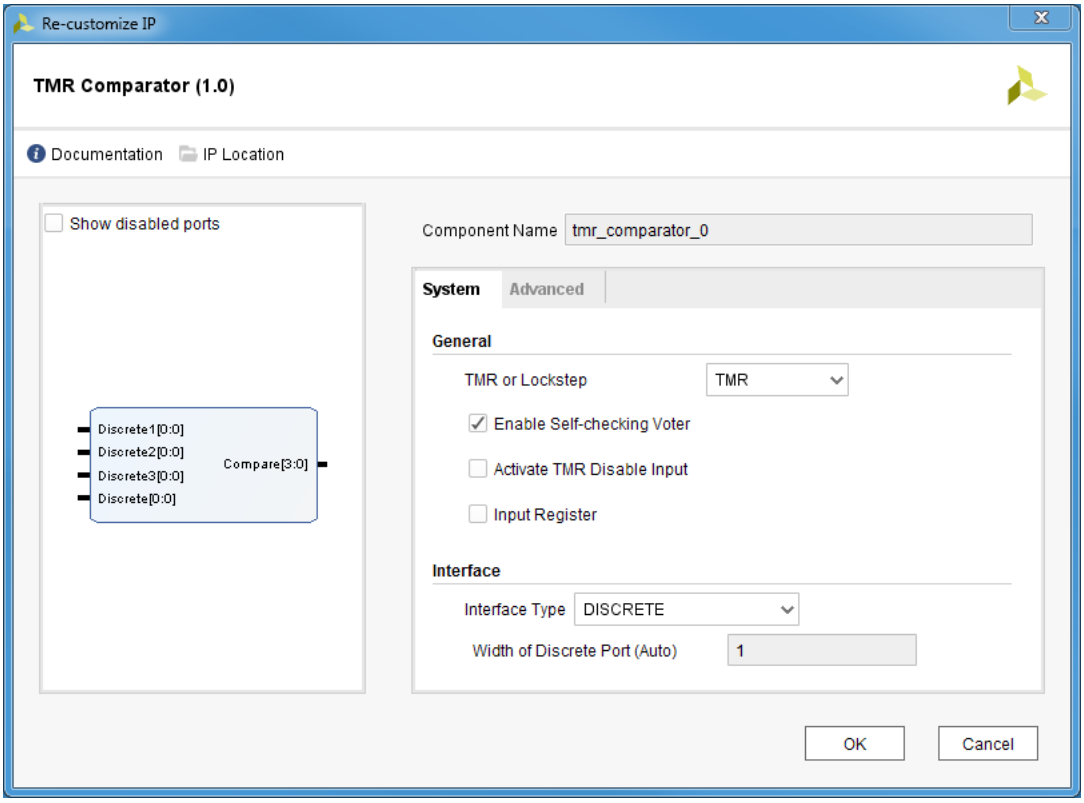

Figure 4-6: **TMR Comparator System Tab**

TMR or Lockstep - Select Triple Modular Redundancy (TMR) or Lockstep mode.

Enable Self-checking Voter - Enable self-checking voter.

Activate TMR Disable Input - Activate TMR disable functionality.

Input Register - Clock input signals before comparison to improve maximum frequency.

Interface Type - Select comparator interface. Depending on the selected type, interface specific parameters are shown. The available interfaces and parameters are common with the TMR Voter (see page 64 for details), except for the following additional parameters only applicable for the TMR Comparator:

LMB1, LMB2 Bus Interface Type - Defines if the LMB bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

BRAM1, BRAM2 Bus Interface Type - Defines if the BRAM bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

AXI1, AXI2 Bus Interface Type - Defines if the AXI4 or ACE bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

AXIS1, AXIS2 Bus Interface Type - Defines if the AXI4-Stream bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

Trace Included in Comparator - Specifies the Trace bus signals included in the comparison:

- FULL: All signals.
- REGWR: Trace_Valid_Instruction, Trace_Reg_Write, Trace_Reg_Addr, and Trace_New_Reg_Value.
- REGWR DATA: Trace_Data_Access, Trace_Data_Address, Trace_Data_Write_Value, Trace_Data_Byte_Enable, Trace_Data_Read, and Trace_Data_Write in addition to all REGWR signals.

Trace1, Trace2, Trace3, Trace Bus Interface Type - Defines if the Trace bus interface is a Slave or Monitor interface.

IRQ1, IRQ2 Bus Interface Type - Defines if the Interrupt bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

IO1, IO2 Bus Interface Type - Defines if the I/O Bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

UART1, UART2 Bus Interface Type - Defines if the UART bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

GPIO1, GPIO2 Bus Interface Type - Defines if the GPIO bus interface is a Slave or Monitor interface, useful in dual lockstep configurations because a single slave interface is necessary in this case.

The TMR Comparator Advanced tab is shown in Figure 4-7.

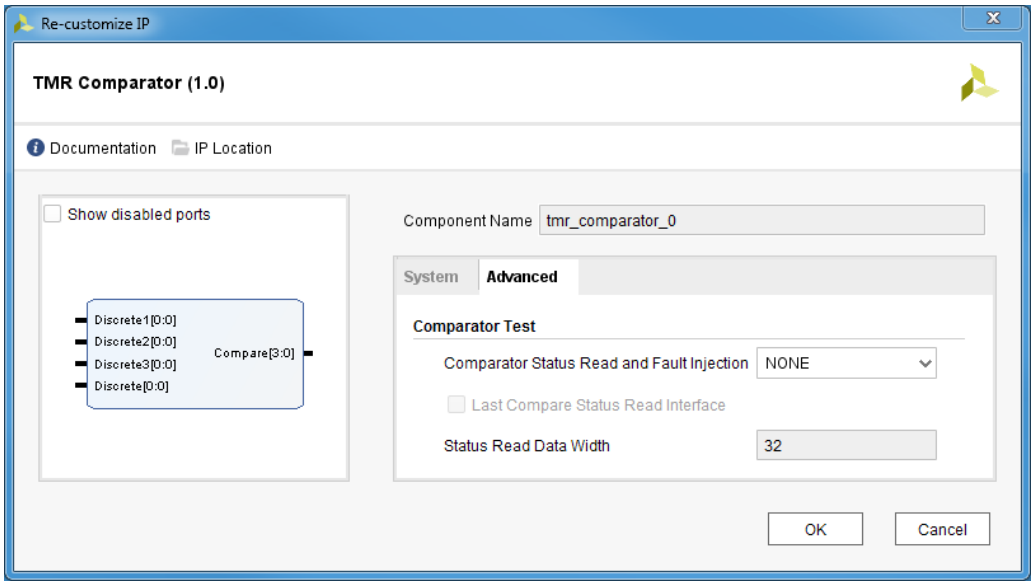

Figure 4-7: **TMR Comparator Advanced Tab**

Comparator Status Read and Fault Injection - Enable functional test of individual comparators via AXI4-Stream interfaces. READ enables status read of comparison results, and INJECT also enables fault injection.

Last Compare Status Read Interface - Set for the last comparison status read interface, furthest from the processor reading the status.

Status Read Data Width - Define the AXI-Stream data width for comparison status read.

TMR Inject

The TMR Inject core parameters are shown in the Vivado Integrated Design Environment (IDE) customization dialog. When using Vivado IP integrator, the addresses and masks are auto generated.

The customization dialog is shown in Figure 4-8.

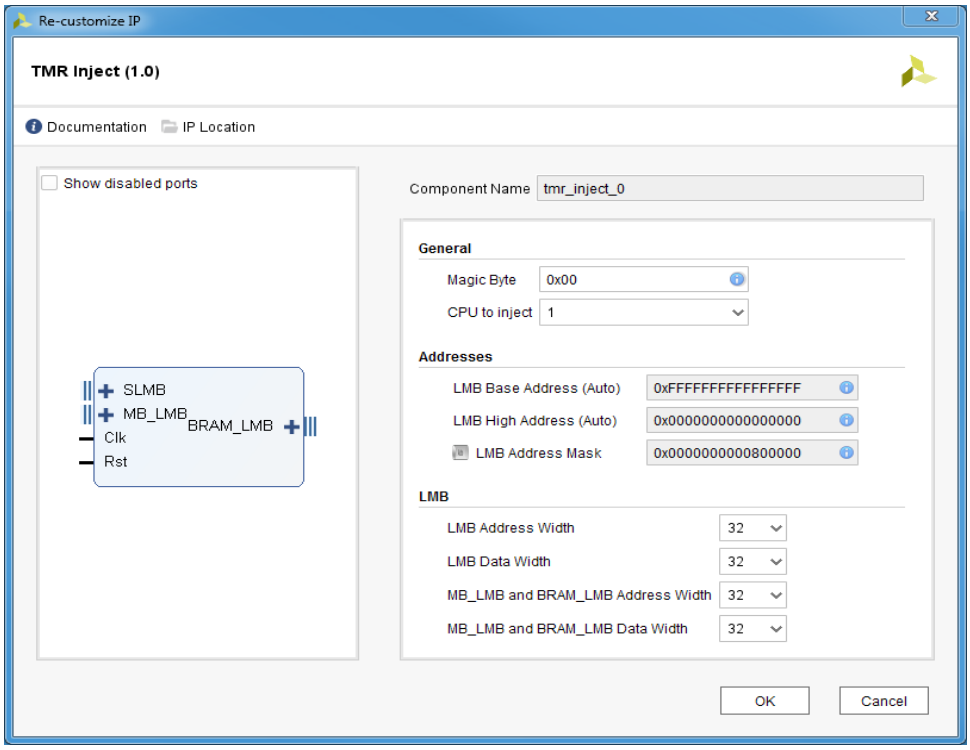

Figure 4-8: **TMR Inject Customization Dialog**

Magic Byte - Magic number used when injecting faults. The fault inject write data least significant byte (bits 7:0) must match this number to have any effect.

CPU to Inject - CPU Identifier used when injecting faults. The fault inject write data bits 9:8 must match this identifier to have any effect.

LMB Base Address - This address specifies where the TMR Inject register address space starts. This should never be set to the value of the MicroBlaze parameter C_BASE_VECTORS (normally 0x0000000000000000), since this is where MicroBlaze boots from. By default this value is larger than the high address so that an error is generated if this value is not specified. If IP integrator is used an address is automatically assigned.

LMB High Address - This address specifies where the TMR Inject register address space ends. By default this value is smaller than the base address so that an error is generated if this value is not specified. If IP integrator is used an address is automatically assigned.

LMB Address Mask - IP integrator automatically sets this value to the mask of bits used to decode this peripheral on the LMB. Any bits that are set to one in the mask indicate that the address bit in that position is used to decode a valid LMB access. All other bits are considered don't cares for the purpose of decoding LMB accesses.

LMB Address Width - This parameter sets the LMB address width. The supported values range from 32 to 64 bits.

LMB Data Width - This parameter sets the LMB data width. The only supported value is 32.

MB_LMB and BRAM_LMB Address Width - This parameter sets the MB_LMB and BRAM_LMB address width. The supported values range from 32 to 64 bits.

MB_LMB and BRAM_LMB Data Width - This parameter sets the MB_LMB and BRAM_LMB data width. The only supported value is 32.

TMR SEM

The customization dialog for 7 Series devices is shown in Figure 4-9.

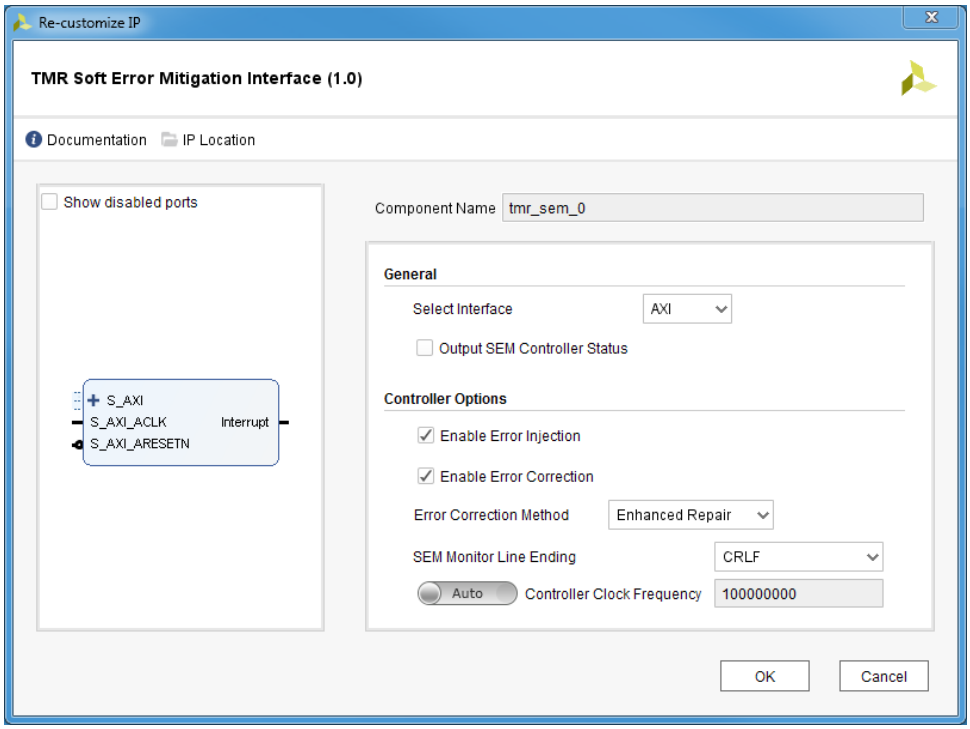

Figure 4-9: **TMR SEM Customization Dialog (7 Series)**

Select Interface - Select control interface. AXI is intended for direct control by a processor in the FPGA. UART is intended for external control from a host computer.

Output SEM Controller Status - Show the Soft Error Mitigation (SEM) discrete status outputs.

Enable Error Injection - Controls inclusion of error injection capability.

Enable Error Correction - Controls inclusion of error correction capability.

Error Correction Method - Selects error correction method, if error correction enabled.

SEM Monitor Line Ending - Select line ending output by the controller. CRLF represents a carriage return followed by a line feed, CR represents just a carriage return, and LF represents just a line feed.

Controller Clock Frequency - Sets controller clock frequency in Hz. The supported clock frequency depends on the FPGA device and speed grade.

For details on the **Enable Error Injection**, **Enable Error Correction, Error Correction Method**, and **Controller Clock Frequency** see Soft Error Mitigation Controller Product Guide (PG036) [Ref 6].

The customization dialog for UltraScale™ devices is shown in Figure 4-10.

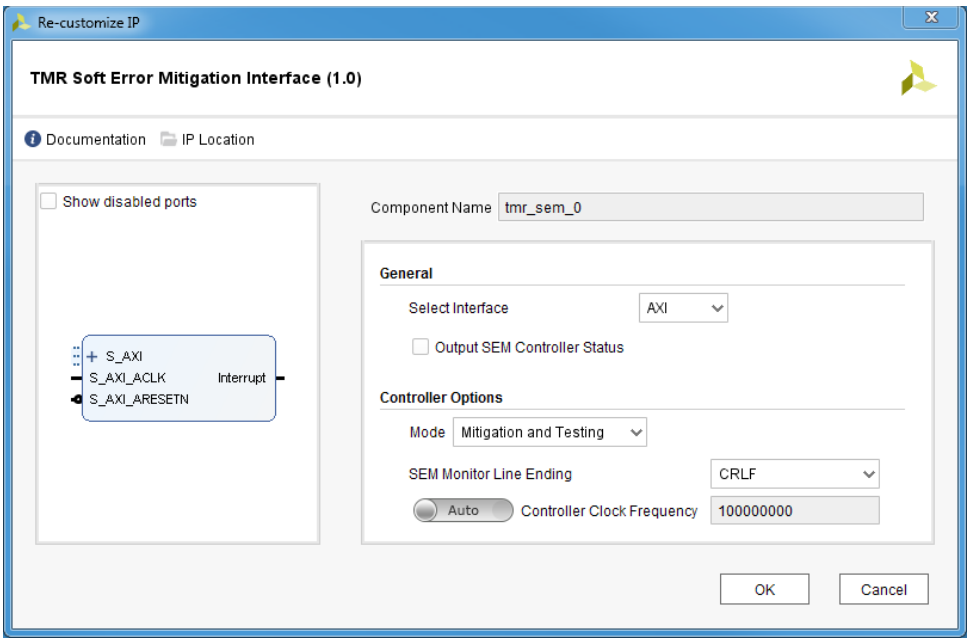

Figure 4-10: **TMR SEM Customization Dialog (UltraScale)**

All parameters are identical to the dialog in Figure 4-9, except for:

Mode - Selects SEM Controller mode.

For details on the **Mode** and **Controller Clock Frequency** parameters, see UltraScale Architecture Soft Error Mitigation Controller Product Guide (PG187) [Ref 7].

User Parameters

To allow the user to obtain a TMR subsystem that is uniquely tailored a specific use case, certain features can be parameterized in the TMR cores. This allows you to configure a design that only uses the resources required by the system. The features that can be parameterized in TMR cores are shown in Table 4-1 to Table 4-5.

Table 4-1: **TMR Manager Design Parameters**

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
LMB Parameters				
TMR Manager base address	C_BASEADDR	Valid address range	0xFFFFFFFF FFFFFFFF	std_logic_ vector
TMR Manager high address	C_HIGHADDR	Valid address range	0x00000000 00000000	std_logic_ vector
LMB decode mask	C_MASK	Valid decode mask for LMB	0x00000000 00800000	std_logic_ vector
LMB address width	C_LMB_AWIDTH	32 - 64	32	integer
LMB data width	C_LMB_DWIDTH	32	32	integer
TMR Parameters				
Magic byte 1	C_MAGIC1	0x00 - 0xFF	0x00	std_logic_ vector
Magic byte 2	C_MAGIC2	0x00 - 0xFF	0x00	std_logic_ vector
Number of Comparators	C_NO_OF_COMPARATORS	1 - 64	1	integer
Uncorrectable error fatal	C_UE_IS_FATAL	0 = false, 1 = true	0	integer
Uncorrectable error width	C_UE_WIDTH	> 0	3	integer
Illegal miscompare fatal	C_STRICT_MISCOMPARE	0 = false, 1 = true	0	integer
Use Debug Disable input	C_USE_DEBUG_DISABLE	0 = false, 1 = true	0	integer
Use TMR Disable input	C_USE_TMR_DISABLE	0 = false, 1 = true	0	integer
Software Watchdog Parameters				
Watchdog enabled	C_WATCHDOG	0 = None 1 = Internal 2 = External	0	integer
Watchdog counter width	C_WATCHDOG_WIDTH	8 - 32	30	integer
SEM Interface Parameters				
Enable SEM IP interface	C_SEM_INTERFACE	0 = false, 1 = true	0	integer
SEM interface asynchronous	C_SEM_ASYNC	0 = false, 1 = true	0	integer
Use SEM heartbeat watchdog	C_SEM_HEARTBEAT_ WATCHDOG	0 = false, 1 = true	0	integer
SEM heartbeat watchdog counter width	C_SEM_HEARTBEAT_ WATCHDOG_WIDTH	1 - 32	10	integer
SEM interface type	C_SEM_INTERFACE_TYPE	0 = Unknown 1 = 7 Series 2 = UltraScale 3 = UltraScale+	0	integer
Break delay counter width	C_BRK_DELAY_WIDTH	0 - 32	0	integer
Break delay counter reset value	C_BRK_DELAY_RST_VALUE	Any	0x00000000	std_logic_ vector

Table 4-1: **TMR Manager Design Parameters *(Cont'd)***

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Test Parameters				
Comparator mask enabled	C_COMPARATORS_MASK	0 = false, 1 = true	0	integer
Comparator enable mask reset value	C_MASK_RST_VALUE	Any	0xFFFFFFFF FFFFFFFF	std_logic_ vector
Comparator status read and fault inject	C_TEST_COMPARATOR	0 = NONE 1 = READ 2 = INJECT	0	integer

Table 4-2: **TMR Voter Design Parameters**

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Interface type	C_INTERFACE	0 = Discrete 1 = LMB 2 = BRAM 3 = AXI4 4 = AXI4-Stream Master 5 = AXI4-Stream Slave 6 = ACE 7 = Trace 8 = AXI4-Lite 9 = Interrupt 10 = I/O Bus 11 = GPIO 12 = UART 13 = BRAM Master 14 = LMB Slave 15 = AXI4 Slave 16 = AXI4-Lite Slave 17 = ACE Slave 18 = Interrupt Slave	0	integer
Activate TMR Disable input	C_USE_TMR_DISABLE	0 = false, 1 = true	0	integer
TMR or Lockstep	C_TMR	0 = Lockstep, 1 = TMR	1	integer
Enable built-in comparator	C_COMPARATOR	0 = false, 1 = true	0	integer
Enable self-checking voter	C_VOTER_CHECK	0 = false, 1 = true	0	integer
Include comparison mask	C_INCLUDE_MASK	Any	0xFFFFFFFF FFFFFFFF	std_logic_ vector
Test Parameters				
Comparator status read and fault inject	C_TEST_COMPARATOR	0 = NONE 1 = READ 2 = INJECT	0	integer
Last compare status read interface	C_TEST_LAST_INTERFACE	0 = false, 1 = true	0	integer
Status read data width	C_TEST_AXIS_DATA_WIDTH	32	32	integer

Table 4-2: **TMR Voter Design Parameters *(Cont'd)***

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Interface Parameters (C_INTERFACE dependent)				
Discrete interface width	C_DISCRETE_WIDTH	>0	1	integer
LMB, S_LMB address width	C_LMB_AWIDTH	32 - 64	32	integer
LMB, S_LMB data width	C_LMB_DWIDTH	32	32	integer
LMB, S_LMB has ECC	C_ECC	0 = false, 1 = true	0	integer
AXI4, ACE ID width	C_AXI_ID_WIDTH	>0	1	integer
AXI4, ACE data width	C_AXI_DATA_WIDTH	32,64,128,256,512,1024	32	integer
AXI4, ACE address width	C_AXI_ADDR_WIDTH	32 - 64	32	integer
AXI4, ACE address width	C_AXI_AWUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_ARUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_WUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_RUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_BUSER_WIDTH	>0	1	integer
AXI4-Stream data width	C_AXIS_DATA_WIDTH	8 - 512	32	integer
AXI4-Stream ID width	C_AXIS_ID_WIDTH	1 - 32	1	integer
AXI4-Stream dest width	C_AXIS_DEST_WIDTH	1 - 32	1	integer
AXI4-Stream user width	C_AXIS_USER_WIDTH	1 - 4096	1	integer
Interrupt low latency	C_LOW_LATENCY	0 = false, 1 = true	0	integer
GPIO interface output use	C_USE_GPO	0 = None 1 = Output 2 = Tristate	2	integer
GPIO interface output size	C_GPO_SIZE		32	integer
GPIO interface input use	C_USE_GPI	0 = None, 1 = Input	1	integer
GPIO interface input size	C_GPI_SIZE		32	integer

Table 4-3: **TMR Comparator Design Parameters**

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Interface type	C_INTERFACE	0 = Discrete 1 = LMB 2 = BRAM 3 = AXI4 4 = AXI4-Stream Master 5 = AXI4-Stream Slave 6 = ACE 7 = Trace 8 = AXI4-Lite 9 = Interrupt 10 = I/O Bus 11 = GPIO 12 = UART 13 = BRAM Master 14 = LMB Slave 15 = AXI4 Slave 16 = AXI4-Lite Slave 17 = ACE Slave 18 = Interrupt Slave	0	integer
Activate TMR Disable input	C_USE_TMR_DISABLE	0 = false, 1 = true	0	integer
TMR or Lockstep	C_TMR	0 = Lockstep, 1 = TMR	1	integer
Enable self-checking voter	C_VOTER_CHECK	0 = false, 1 = true	0	integer
Input Register	C_INPUT_REGISTER	0 = false, 1 = true	0	integer
Include comparison mask	C_INCLUDE_MASK	Any	0xFFFFFFFF FFFFFFFF	integer
Test Parameters				
Comparator status read and fault inject	C_TEST_COMPARATOR	0 = NONE 1 = READ 2 = INJECT	0	integer
Last compare status read interface	C_TEST_LAST_ INTERFACE	0 = false, 1 = true	0	integer
Status read data width	C_TEST_AXIS_DATA_ WIDTH	32	32	integer
Interface Parameters (C_INTERFACE dependent)				
Discrete interface width	C_DISCRETE_WIDTH	>0	1	integer
LMB, S_LMB address width	C_LMB_AWIDTH	32 - 64	32	integer
LMB, S_LMB data width	C_LMB_DWIDTH	32	32	integer
LMB, S_LMB has ECC	C_ECC	0 = false 1 = true	0	integer
LMB, S_LMB bus interface type	C_LMB1, C_LMB2	0 = Monitor 1 = Slave	0	integer

Table 4-3: **TMR Comparator Design Parameters *(Cont'd)***

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
AXI4, ACE ID width	C_AXI_ID_WIDTH	>0	1	integer
AXI4, ACE data width	C_AXI_DATA_WIDTH	32,64,128,256,512,1024	32	integer
AXI4, ACE address width	C_AXI_ADDR_WIDTH	32 - 64	32	integer
AXI4, ACE address width	C_AXI_AWUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_ARUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_WUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_RUSER_WIDTH	>0	1	integer
AXI4, ACE address width	C_AXI_BUSER_WIDTH	>0	1	integer
AXI4, ACE, bus interface type	C_AXI1, C_AXI2	0 = Monitor 1 = Slave	0	integer
AXI4-Stream data width	C_AXIS_DATA_WIDTH	8 - 512	32	integer
AXI4-Stream ID width	C_AXIS_ID_WIDTH	1 - 32	1	integer
AXI4-Stream dest width	C_AXIS_DEST_WIDTH	1 - 32	1	integer
AXI4-Stream user width	C_AXIS_USER_WIDTH	1 - 4096	1	integer
AXI4-Stream bus interface type	C_AXIS1, C_AXIS2	0 = Monitor 1 = Slave	0	integer
Trace interface data size	C_DATA_SIZE	32	32	integer
Trace size	C_TRACE_SIZE	0 = Full 1 = RegWr 2 = RegWr Data	0	integer
Trace bus interface type	C_TRACE C_TRACE*n* (n = 1-3)	0 = Monitor 1 = Slave	0	integer
Interrupt low latency	C_LOW_LATENCY	0 = false, 1 = true	0	integer
Interrupt bus interface type	C_IRQ1, C_IRQ2	0 = Monitor 1 = Slave	0	integer
I/O bus interface type	C_IO1, C_IO2	0 = Monitor 1 = Slave	0	integer
UART bus interface type	C_UART1, C_UART2	0 = Monitor 1 = Slave	0	integer
GPIO interface output use	C_USE_GPO	0 = None 1 = Output 2 = Tristate	2	integer
GPIO interface output size	C_GPO_SIZE		32	integer
GPIO interface input use	C_USE_GPI	0 = None, 1 = Input	1	integer
GPIO interface input size	C_GPI_SIZE		32	integer
GPIO bus interface type	C_GPIO1, C_GPIO2	0 = Monitor 1 = Slave	0	integer

Table 4-4: **TMR Inject Design Parameters**

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
LMB Parameters				
TMR Inject base address	C_BASEADDR	Valid address range	0xFFFFFFFF FFFFFFFF	std_logic_ vector
TMR Inject high address	C_HIGHADDR	Valid address range	0x00000000 00000000	std_logic_ vector
LMB Decode Mask	C_MASK	Valid decode mask for LMB	0x00000000 00800000	std_logic_ vector
LMB address width	C_LMB_AWIDTH	32 - 64	32	integer
LMB data width	C_LMB_DWIDTH	32	32	integer
MB_LMB and BRAM_LMB address width	C_INJECT_LMB_AWIDTH	32 - 64	32	integer
MB_LMB and BRAM_LMB data width	C_INJECT_LMB_DWIDTH	32	32	integer
TMR Parameters				
Magic Byte	C_MAGIC	0x00 - 0xFF	0x00	std_logic_ vector
CPU Identifier	C_CPU_ID	1 = CPU number 1 2 = CPU number 2 3 = CPU number 3	1	integer

Table 4-5: **TMR SEM Design Parameters**

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Selected interface	C_INTERFACE	0 = AXI 1 = UART	0	integer
Output SEM status	C_SEM_STATUS	0 = false, 1 = true	0	integer
SEM interface type	C_SEM_INTERFACE_TYPE	0 = Unknown 1 = 7 Series 2 = UltraScale 3 = UltraScale+	0	integer
SEM monitor line ending	C_LINE_ENDING	0 = CRLF 1 = CR 2 = LF	0	integer
Controller clock frequency	C_S_AXI_ACLK_FREQ_HZ	Any valid frequency	100000000	integer
AXI4-Lite Parameters (C_INTERFACE = 0)				
AXI slave address width	C_S_AXI_ADDR_WIDTH	4	4	integer
AXI slave data width	C_S_AXI_DATA_WIDTH	32	32	integer

Table 4-5: **TMR SEM Design Parameters *(Cont'd)***

Feature / Description	Parameter Name	Allowable Values	Default Value	VHDL Type
UART Parameters (C_INTERFACE = 1)				
Defines baud rate	C_UART_BAUDRATE	110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 128000, 230400, 460800, 921600	9600	integer
Use parity	C_USE_UART_PARITY	0 = false, 1 = true	0	integer
Even or odd parity	C_UART_ODD_PARITY	0 = Even 1 = Odd	0	integer
SEM Controller Parameters				
Enable injection(1)	ENABLE_INJECTION	false, true	true	Boolean
Enable correction(1)	ENABLE_CORRECTION	false, true	true	Boolean
Correction method(1)	CORRECTION_METHOD	repair, enhanced repair	enhanced repair	string
Mode(2)	MODE	mitigation and testing, mitigation only, detect and testing, detect only, emulation, monitoring	mitigation and testing	string

Notes:
1. Only available when C_SEM_INTERFACE_TYPE is set to 1.
2. Only available when C_SEM_INTERFACE_TYPE is set to 2 or 3.

The relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl Console) are shown in Table 4-1 to Table 4-5. The Vivado IDE parameter is defined in the Feature/Description column, and the User Parameter is defined in the Parameter Name column.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10].

Constraining the Cores

This section contains information about constraining the cores in the Vivado Design Suite.

Required Constraints

This section is not applicable for these IP cores.

Device, Package, and Speed Grade Selections

This section is not applicable for these IP cores.

Clock Frequencies

This section is not applicable for these IP cores.

Clock Management

This section is not applicable for these IP cores.

Clock Placement

This section is not applicable for these IP cores.

Banking

This section is not applicable for these IP cores.

Transceiver Placement

This section is not applicable for these IP cores.

I/O Standard and Placement

This section is not applicable for these IP cores.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 12].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 10].

To ensure physical separation of the triplicated sub-blocks, floorplanning can be used. Because the resources required by the sub-blocks differ between designs, the actual location is design-dependent. Floorplanning constraints for a KC705 board design with an I/O Module and 32K Block-RAM local memory is shown in the following example. In this design, MicroBlaze is configured to use hardware multiplication, which necessitates the DSP48 constraints.

```
# MB1 sub-block
create_pblock MB1
add_cells_to_pblock [get_pblocks MB1] [get_cells design_1_i/tmr_0/MB1/*]
add_cells_to_pblock [get_pblocks MB1] [get_cells design_1_i/tmr_0/MB1]
resize_pblock [get_pblocks MB1] -add {SLICE_X50Y100:SLICE_X61Y169
  SLICE_X48Y150:SLICE_X49Y169 SLICE_X48Y100:SLICE_X49Y119}
resize_pblock [get_pblocks MB1] -add {DSP48_X2Y40:DSP48_X2Y67}
resize_pblock [get_pblocks MB1] -add {RAMB36_X2Y30:RAMB36_X2Y33
  RAMB36_X2Y20:RAMB36_X2Y23}
set_property CONTAIN_ROUTING 1 [get_pblocks MB1]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks MB1]

# MB2 sub-block
create_pblock MB2
add_cells_to_pblock [get_pblocks MB2] [get_cells design_1_i/tmr_0/MB2/*]
add_cells_to_pblock [get_pblocks MB2] [get_cells design_1_i/tmr_0/MB2]
resize_pblock [get_pblocks MB2] -add {SLICE_X36Y100:SLICE_X47Y119
  SLICE_X0Y100:SLICE_X23Y134}
resize_pblock [get_pblocks MB2] -add {DSP48_X0Y40:DSP48_X1Y53}
resize_pblock [get_pblocks MB2] -add {RAMB36_X1Y20:RAMB36_X1Y23
  RAMB36_X0Y20:RAMB36_X0Y26}
set_property CONTAIN_ROUTING 1 [get_pblocks MB2]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks MB2]

# MB3 sub-block
create_pblock MB3
add_cells_to_pblock [get_pblocks MB3] [get_cells design_1_i/tmr_0/MB3/*]
add_cells_to_pblock [get_pblocks MB3] [get_cells design_1_i/tmr_0/MB3]
resize_pblock [get_pblocks MB3] -add {SLICE_X36Y150:SLICE_X47Y169
  SLICE_X0Y135:SLICE_X23Y169}
resize_pblock [get_pblocks MB3] -add {DSP48_X0Y54:DSP48_X1Y67}
resize_pblock [get_pblocks MB3] -add {RAMB36_X1Y30:RAMB36_X1Y33
  RAMB36_X0Y27:RAMB36_X0Y33}
set_property CONTAIN_ROUTING 1 [get_pblocks MB3]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks MB3]
```

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite.

The TMR Manager has a system-level example design, which includes the other four IP cores. This example consists of a Vivado IPI Block Design, with a MicroBlaze TMR subsystem that closely resembles the design in Figure 2-4.

IMPORTANT: *The example design is currently only available for the boards AC701, KC705, KCU105, VC707, VC709, VCU108, VCU110, and VCU118. If any other board is selected in the project from which the example is generated, the example uses the KCU105 board instead.*

Functions

The finished system-level design example has a Triple Modular Redundancy (TMR) MicroBlaze subsystem in the hierarchical sub-block tmr_0. This sub-block contains the triplicated sub-blocks MB1, MB2, and MB3. It also contains the TMR SEM, providing access to Soft Error Mitigation for configuration bit scrubbing, and the voter for the UART output. Each triplicated sub-block contains the TMR Manager, a MicroBlaze processor with local memory, an I/O Module, and the necessary TMR Comparators for fault detection.

Figure 5-1 shows the functional block diagram of the system-level design example. Only one of the three identical triplicated blocks is expanded, to simplify the diagram.

The example design has the ports:

- Clock
- Reset
- UART

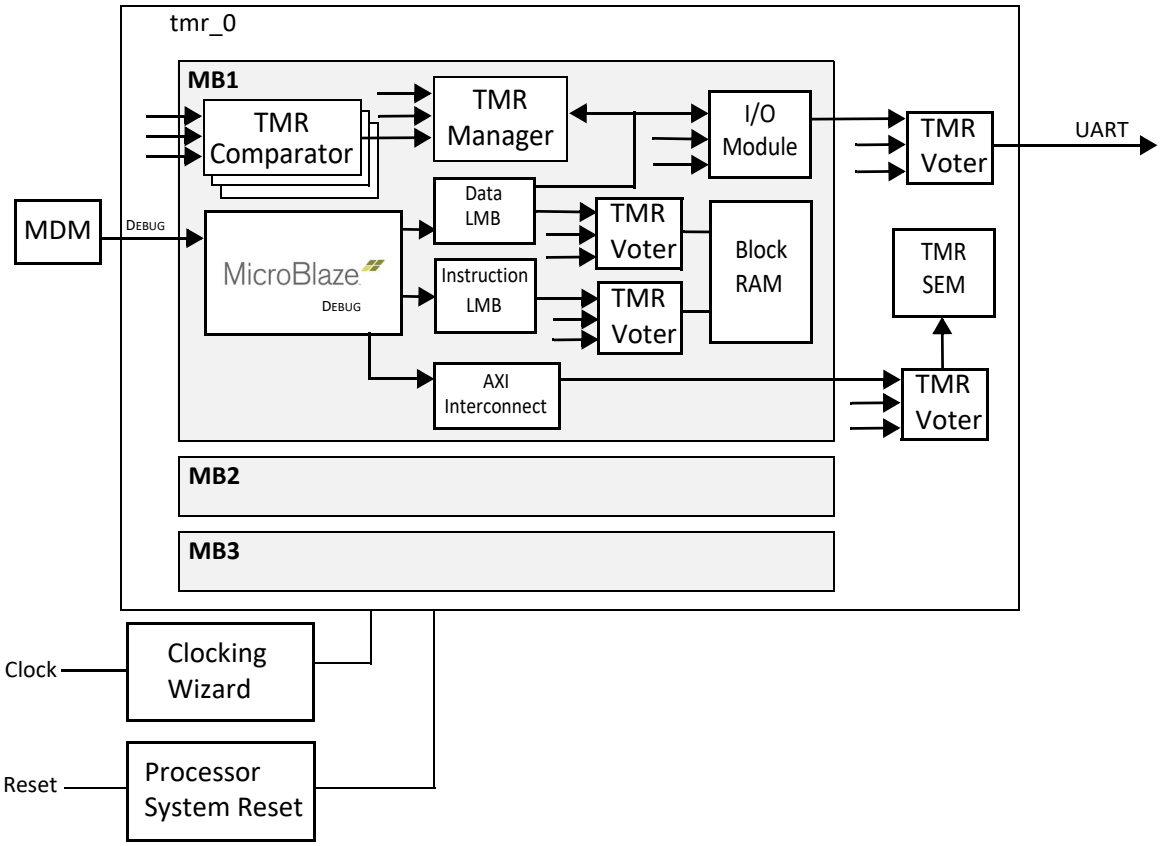

Figure 5-1: **Example Design Functional Block Diagram**

Implementation

The example design is not generated by default. The example design is generated by user request and can be opened in a new instance of Vivado. This allows you to view and modify the example of various cores being used without touching their own design.

To generate the example design, right-click on a TMR Manager IP instance in a Block Design or on the `XCI` file under **Design Sources** and select **Open IP Example Design**.

TIP: *Take a look at the `Tcl` file `instancename_exdes_bd.tcl` under **Design Sources** in the example to examine the script used to generate the entire Block Design.*

Software Generation

The example design is a fully functional MicroBlaze design, but to make the processor do anything useful, software needs to be added.

- Generate the design by clicking **Generate Block Design** in the left side menu.
- Select **Export - Export Hardware** in the **File** menu to export the hardware design.
- Start the Vitis™ unified software platform from the **Tools** menu, to open the development environment.
- In the Vitis software platform, all three processors in the TMR subsystem show up. Because they all have identical hardware configuration it does not matter which one is used. Leaving the default processor selected is usually sufficient.

Synthesis and Implementation

Synthesis and implementation can be run separately by clicking on the appropriate option in the left side menu.

Simulation

When generating the example design, a simulation test bench is automatically created. This provides the stimulus for the input clock and reset.

To run a meaningful simulation, an ELF file created in the Vitis software platform must be associated with the MicroBlaze processors. This can be done in Vivado by selecting **Associate ELF Files...** in the **Tools** menu, and adding a new ELF file in the dialog to all three MicroBlaze processors.

IMPORTANT: *The ELF file must be associated with all three MicroBlaze processors, otherwise they will not execute the same program, and the fault detection immediately detects a fatal error.*

XILINX

Appendix A

Verification, Compliance, and Interoperability

This appendix provides details about how these IP cores were tested for compliance with the protocol to which they were designed.

Simulation

Complete simulation flow regression tests based on a golden simulation of an entire TMR subsystem has been performed for each core, in all supported simulators. This includes verification of all supported interfaces.

Hardware Testing

Random based and directed testing of entire TMR subsystems has been performed on supported boards, verifying nominal operation, fault injection and SEM interface support. This includes ensuring that all supported interfaces operate nominally.

Upgrading

This appendix contains information about upgrading to a more recent version of the IP core. For customers upgrading in the Vivado® Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

The following parameters have been added to the cores:

- **TMR Voter**: C_TMR
- **TMR Comparator**: C_INPUT_REGISTER
- **TMR Inject**: C_ INJECT_LMB_AWIDTH, C_ INJECT_LMB_DWIDTH

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the TMR IP cores, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the TMR IP cores. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this subsystem can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the TMR IP Cores

AR: 68483

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

There are many tools available to address TMR IP subsystem design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 13].

Reference Boards

Various Xilinx development boards support the TMR IP cores. These boards can be used to prototype designs and establish that the TMR implementation operates as intended.

- 7 series FPGA evaluation boards

  AC701, KC705, VC707, VC709, ZC702, ZC706

- UltraScale™ FPGA evaluation boards

  KCU105, VCU108, VCU110, VCU118

- UltraScale+™ FPGA evaluation board

  ZCU102

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the subsystem were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `locked` port.
- If your outputs go to 0, check your licensing.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `s_axi_aclk` and `aclk` inputs are connected and toggling.
- The interface is not being held in reset, and `s_axi_areset` is an active-Low reset.
- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main subsystem clocks are toggling and that the enables are also asserted.

- If the simulation has been run, verify in simulation and/or a debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `<interface_name>_tready` is stuck Low following the `<interface_name>_tvalid` input being asserted, the subsystem cannot send data.
- If the receive `<interface_name>_tvalid` is stuck Low, the subsystem is not receiving data.
- Check that the `aclk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed.
- Check subsystem configuration.

Application Software Development

Device Drivers

Device drivers are available for the TMR Manager and TMR Inject. Additionally, the TMR SEM wrapper uses the *uartlite* driver, when configured with an AXI interface,

TMR Inject

The TMR Inject driver, *tmr_inject*, provides the ability to inject faults in a TMR subsystem.

TMR Manager

The TMR Manager driver, *tmr_manager*, main functions are:

- Access TMR and SEM status.
- Support recovery, including the ability to associate user functions to save state before recovery reset and restore state afterwards.

TMR SEM

The TMR SEM uses the *uartlite* driver, which can be used in either polled or interrupt mode, like any normal UART. For details on the SEM protocol, see the *Soft Error Mitigation Controller Product Guide* (PG036) [Ref 6] and the *UltraScale Architecture Soft Error Mitigation Controller Product Guide* (PG187) [Ref 7].

Benchmarks

This appendix includes Vivado® Tcl code to recreate the benchmark designs used to generate the TMR Subsystem Performance and Resource Utilization results in Table 2-2. This code creates a Kintex UltraScale project, but it can easily be changed to any other device.

Typical I/O Module Design

The single-string design is comprised of MicroBlaze with local memory, MDM, and an I/O Module configured to enable the UART.

```
create_project project_1 project_1 -part xcku040-ffva1156-3-e
create_bd_design benchmark

set mb [create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze microblaze_0]
set_property -dict [list CONFIG.C_USE_BARREL 1 CONFIG.C_USE_HW_MUL 1 \
  CONFIG.C_USE_MSR_INSTR 1 CONFIG.C_USE_PCMP_INSTR 1] $mb
apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config \
  {local_mem "32KB" debug_module "Debug Only" axi_periph "Enabled" axi_intc "0"
   clk "New External Port (100 MHz)"} $mb

set io [create_bd_cell -type ip -vlnv xilinx.com:ip:iomodule iomodule_0]
set_property -dict [list CONFIG.C_USE_UART_RX 1 CONFIG.C_USE_UART_TX 1] $io
apply_bd_automation -rule xilinx.com:bd_rule:iomodule -config \
  {processor "/microblaze_0"} $io

make_bd_pins_external [get_bd_pins $io/UART_Rx] [get_bd_pins $io/UART_Tx]
make_bd_pins_external [get_bd_pins rst_Clk_100M/ext_reset_in]

group_bd_cells tmr_0 $mb $io [get_bd_cells microblaze_0_local_memory]

assign_bd_address
regenerate_bd_layout
save_bd_design
```

Save the Tcl code listed here in a file named benchmark_1.tcl, and then use the following command to recreate the single-string design:

```
% vivado -source benchmark_1.tcl
```

When the single-string design has been created, the TMR Block Automation can be run to triplicate it by executing this Tcl code in the Vivado Tcl console:

```
validate_bd_design
create_bd_cell -type ip -vlnv xilinx.com:ip:tmr_manager:1.0 tmr_0/tmr_manager_0
apply_bd_automation -rule xilinx.com:bd_rule:tmr -config \
  {bram "Local" wd "None" sem_if "External" sem_wd "1" brk "1" mask "0"
   inject "0"} [get_bd_cells tmr_0/tmr_manager_0]
save_bd_design
```

Typical AXI Design

The single-string design is comprised of MicroBlaze with local memory, MDM, AXI Interrupt Controller, AXI Timer, AXI GPIO, and AXI UARTLite.

```
create_project project_2 project_2 -part xcku040-ffva1156-3-e
create_bd_design benchmark

set mb [create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze microblaze_0]
set_property -dict [list CONFIG.C_USE_BARREL 1 CONFIG.C_USE_HW_MUL 1 \
  CONFIG.C_USE_MSR_INSTR 1 CONFIG.C_USE_PCMP_INSTR 1] $mb
apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config \
  {local_mem "32KB" debug_module "Debug Only" axi_periph "Enabled" axi_intc "1"
   clk "New External Port (100 MHz)"} $mb

set prefix xilinx.com:ip
set tim  [create_bd_cell -type ip -vlnv $prefix:axi_timer:2.0 axi_timer_0]
set gpio [create_bd_cell -type ip -vlnv $prefix:axi_gpio:2.0 axi_gpio_0]
set uart [create_bd_cell -type ip -vlnv $prefix:axi_uartlite:2.0 axi_uartlite_0]

set_property -dict [list CONFIG.C_GPIO_WIDTH {4}] $gpio
connect_bd_net [get_bd_pins $gpio/gpio_io_i] [get_bd_pins $gpio/gpio_io_o]
set config {Master "/microblaze_0 (Periph)" intc_ip "/microblaze_0_axi_periph"
  Clk_xbar "Auto" Clk_master "Auto" Clk_slave "Auto"}

set rule xilinx.com:bd_rule:axi4
apply_bd_automation -rule $rule -config $config [get_bd_intf_pins $tim/S_AXI]
apply_bd_automation -rule $rule -config $config [get_bd_intf_pins $gpio/S_AXI]
apply_bd_automation -rule $rule -config $config [get_bd_intf_pins $uart/S_AXI]

set xlconcat [get_bd_cells microblaze_0_xlconcat]
connect_bd_net [get_bd_pins $uart/interrupt] [get_bd_pins $xlconcat/In0]
connect_bd_net [get_bd_pins $tim/interrupt] [get_bd_pins $xlconcat/In1]

make_bd_pins_external [get_bd_pins $uart/rx] [get_bd_pins $uart/tx]
make_bd_pins_external [get_bd_pins rst_Clk_100M/ext_reset_in]

group_bd_cells tmr_0 $mb $tim $gpio $uart $xlconcat \
  [get_bd_cells microblaze_0_axi_intc] [get_bd_cells microblaze_0_axi_periph] \
  [get_bd_cells microblaze_0_local_memory]

regenerate_bd_layout
save_bd_design
```

Save the Tcl code listed here in a file named benchmark_2.tcl, and then use the following command to recreate the single-string design:

```
% vivado -source benchmark_2.tcl
```

Once the single-string design has been created, the TMR Block Automation can be run to triplicate it by using the Tcl code listed for the Typical I/O Module Design.

Appendix F

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. *AXI Timebase Watchdog Timer LogiCORE IP Product Guide* (PG128)
2. *AMBA AXI and ACE Protocol Specification* (Arm IHI 0022E)
3. *AMBA AXI4-Stream Protocol Specification. Version 1.0* (Arm IHI 0051A)
4. *7 Series FPGAs Configuration User Guide* (UG470)
5. *MicroBlaze Processor Reference Guide* (UG984)
6. *Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG036)
7. *UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide* (PG187)
8. *AXI UART Lite LogiCORE IP Product Guide* (PG142)
9. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
10. *Vivado Design Suite User Guide: Designing with IP* (UG896)
11. *Vivado Design Suite User Guide: Getting Started* (UG910)
12. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
13. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
14. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
15. *LogiCORE IP LMB BRAM Interface Controller Product Guide* (PG112)

The following lists additional resources you can access directly using the provided URLs:

16. Brigham Young University, Configurable Computing Lab: http://reliability.ee.byu.edu/

Training Resources

1. Vivado Design Suite Hands-on Introductory Workshop
2. Vivado Design Suite Tool Flow

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/22/2019	1.0	Updated to replace SDK with Vitis™ unified software platform.
11/14/2018	1.0	Updated TMR Inject with extended address inject register, to support up to 64-bit program counter.
10/04/2017	1.0	Updated due to adding support for: • Lockstep, including TMR Voter lockstep mode • Additional TMR comparator and TMR Voter bus interfaces • TMR Comparator input register • Added "MicroBlaze" to Doc title
04/05/2017	1.0	Initial Xilinx release.

Please Read: Important Legal Notices