UltraScale Architecture Soft Error Mitigation Controller v2.0

LogiCORE IP Product Guide

Vivado Design Suite

PG187 April 1, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The LogiCORE™ IP UltraScale™ architecture Soft Error Mitigation (SEM) controller is an automatically configured, pre-verified solution to detect and correct soft errors in Configuration Memory of Xilinx FPGAs. Soft errors are unintended changes to the values stored in state elements caused by ionizing radiation.

The SEM controller does not prevent soft errors; however, it provides a method to better manage the system-level effects of soft errors. Proper management of these events can increase reliability and availability, and reduce system maintenance and downtime costs.

Features

- Typical detection latency of 13 ms for KU040.
- Integration of built-in silicon primitives to fully leverage and improve upon the inherent error detection capability of the FPGA.
- Four convenient modes:
  - Mitigation and testing
  - Mitigation only
  - Emulation
  - Monitoring
- Optional error correction based on ECC algorithm with expedited correction time for multi-bit errors across adjacent frames.
- Using Xilinx Essential Bits technology, optional error classification to determine if a soft error has affected the function of the user design.
  - Increases uptime by avoiding disruptive recovery approaches for errors that have no effect on design operation.
  - Reduces effective failures-in-time (FIT).
- Optional error injection and convenient debug feature to support evaluation of SEM controller applications.
- ICAP arbitration interface available to ease ICAP primitive sharing.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[(1)]	UltraScale Architecture[(1)]
Supported User Interfaces	RS-232, SPI
Resources	See Table 2-9 to Table 2-10.
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	N/A
Constraints File	XDC
Simulation Model	N/A
Supported S/W Driver	N/A
Tested Design Flows	
Design Entry	Vivado® Design Suite
Simulation[(2)]	N/A
Synthesis[(2)]	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog. KU040, VU095, KU060, and KU115 have been verified in hardware. Other devices have not been verified in hardware and have limited feature supported (no error injection and linear frame addressing support).
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Ionizing radiation is capable of inducing undesired effects in most silicon devices. Broadly, an undesired effect resulting from a single event is called a single event effect (SEE). In most cases, these events do not permanently damage the silicon device; SEEs that result in no permanent damage to the device are called soft errors. However, soft errors have the potential to reduce reliability.

Xilinx® devices are designed to have an inherently low susceptibility to soft errors. However, Xilinx also recognizes that soft errors are unavoidable within commercial and practical constraints. As a result, Xilinx has integrated soft error detection and correction capability into many device families.

In many applications, soft errors can be ignored. In applications where higher reliability is desired, the integrated soft error detection and correction capability is usually sufficient. In demanding applications, the UltraScale architecture SEM controller can ensure an even higher level of reliability.

Memory Types

If a soft error occurs, one or more memory bits are corrupted. The memory bits affected can be in the device configuration memory (which determines the behavior of the design), or might be in design memory elements (which determine the state of the design). The following four memory categories represent a majority of the memory in a device:

- **Configuration Memory** – Storage elements used to configure the function of the design loaded into the device. This includes function block behavior and function block connectivity. This memory is physically distributed across the entire device and represents the largest number of bits. Only a fraction of the bits are essential to the proper operation of any specific design loaded into the device.
- **Block Memory** – High capacity storage elements used to store design state. As the name implies, the bits are clustered into a physical block, with several blocks distributed across the entire device. Block Memory represents the second largest number of bits.

- **Distributed Memory** – Medium capacity storage elements used to store design state. This type of memory is present in certain configurable logic blocks (CLBs) and is distributed across the entire device. Distributed Memory represents the third largest number of bits.
- **Flip-Flops** – Low capacity storage elements used to store design state. This type of memory is present in all configurable logic blocks (CLBs) and is distributed across the entire device. Flip-Flops represent the fourth largest number of bits.

An extremely small number of additional memory bits exist as internal device control registers and state elements. Soft errors occurring in these areas can result in regional or device-wide interference that is referred to as a single-event functional interrupt (SEFI). Due to the small number of these memory bits, the frequency of SEFI events is considered negligible in this discussion, and these infrequent events are not addressed by the SEM controller.

Mitigation Approaches

Soft error mitigation for design state in Block Memory, Distributed Memory, and Flip-Flops can be performed in the design itself, by applying standard techniques such as error detection and correction codes or redundancy. Soft errors in unused design state resources (those physically present in the device, but unused by the design) are ignored. Designers concerned about reliability must assess risk areas in the design and incorporate mitigation techniques for the design state as warranted.

Soft error mitigation for the design function in Configuration Memory is performed using error detection and correction codes.

Configuration Memory is organized as an array of frames, much like a wide static RAM. In many device families, each frame is protected by ECC, with the entire array of frames protected by CRC in all device families. The two techniques are complementary; CRC is incredibly robust for error detection, while ECC provides high resolution of error location.

The SEM controller builds upon the robust capability of the integrated logic by adding optional capability to classify Configuration Memory errors as either "essential" or "non-essential." This leverages the fact that only a fraction of the Configuration Memory bits are essential to the proper operation of any specific design.

Without error classification, all Configuration Memory errors must be considered "essential." With error classification, most errors will be assessed "non-essential" which eliminates false alarms and reduces the frequency of errors that require a potentially disruptive system-level mitigation response.

Additionally, the SEM controller extends the built-in correction capability to accelerate error detection and provides the optional capability to handle multi-bit errors.

Reliability Estimation

As a starting point, the specification for system reliability should highlight critical sections of the system design and provide a value for the required reliability of each subsection. Reliability requirements are typically expressed as failures in time (FIT), which is the number of design failures that can be expected in 10^9 hours (approximately 114,155 years).

When more than one instance of a design is deployed, the probability of a soft error affecting any one of them increases proportionately. For example, if the design is shipped in 1,000 units of product, the nominal FIT across all deployed units is 1,000 times greater. This is an important consideration because the nominal FIT of the total deployment can grow large and can represent a service or maintenance burden.

The nominal FIT of the total deployment is different from the probability of an individual unit being affected. Also, the probability of a specific unit incurring a second soft error is determined by the FIT of the individual design and not the deployment. This is an important consideration when assessing suitable soft error mitigation strategies for an application.

The FIT associated with soft errors must not be confused with that of product life expectancy, which considers the replacement or physical repair of some part of a system.

Xilinx device FIT data is reported in the *Xilinx Device Reliability Report* (UG116) [Ref 1]. The data reveals the overall infrequency of soft errors.

TIP: *The failure rates involved are so small that most designs does not need to include any form of soft error mitigation.*

The contribution to FIT from flip-flops is negligible based on the flip-flop's very low FIT and small quantity. However, this does not discount the importance of protecting the design state stored in flip-flops. If any state stored in flip-flops is highly important to design operation, the design must contain logic to detect, correct, and recover from soft errors in a manner appropriate to the application.

The contribution to FIT from Distributed Memory and Block Memory can be large in designs where these resources are highly utilized. As previously noted, the FIT contribution can be substantially decreased by using soft error mitigation techniques in the design. For example, Block Memory resources include built-in error detection and correction circuits that can be used in certain Block Memory configurations. For all Block Memory and Distributed Memory configurations, soft error mitigation techniques can be applied using programmable logic resources.

The contribution to FIT from Configuration Memory is large. Without using an error classification technique, all soft errors in Configuration Memory must be considered "essential," and the resulting contribution to FIT eclipses all other sources combined.

Use of error classification reduces the contribution to FIT by no longer considering most soft errors as failures; if a soft error has no effect, it can be corrected without any disruption.

In designs requiring the highest level of reliability, classification of soft errors in Configuration Memory is essential. This capability is provided by the SEM controller.

Feature Summary

The SEM controller can be generated in four different modes dependent on the design requirements:

- Mitigation and testing
- Mitigation only
- Emulation
- Monitoring only

The mitigation modes, Mitigation and testing and Mitigation only, enables error detection, error correction, and error classification (optional) functions. Error injection is not available in the Mitigation only mode.

The other two modes, Emulation and Monitoring only, enable you to use the SEM controller to assess and monitor your system behavior when an SEU event occurs without enabling the error detection, error correction, and error classification functions. Error injection is not available in the Monitoring only mode.

For all the modes, the IP performs an initialization function that brings the integrated soft error detection capability of the FPGA into a known state after the FPGA enters the user mode. After this initialization, depending on the chosen mode, the SEM controller can either observe the integrated soft error detection status (mitigation modes) or transition to idle state where you can give commands to the IP through the Command or Monitor Interface (emulation and monitoring modes).

In the mitigation modes, when an ECC or CRC error is detected, the SEM controller evaluates the situation to identify the Configuration Memory location involved.

If the location can be identified, the SEM controller corrects the soft error. The correction method uses active partial reconfiguration to perform a localized correction of the Configuration Memory using a read-modify-write scheme. This method uses algorithms to identify the error in need of correction.

The SEM controller optionally classifies the soft error as essential or non-essential using a lookup table. Information is fetched as needed during execution of error classification. This data is also provided by the implementation tools and stored outside the SEM controller.

When the SEM controller is idle, it optionally accepts input from you to inject errors into the Configuration Memory (for Mitigation and testing and emulation modes). In the Mitigation and testing mode, this feature is useful for testing the integration of the SEM controller into a larger system design.

In the Emulation mode, this feature is useful to evaluate the effects of SEU events on the system design. Using the error injection feature, system verification and validation engineers can construct test cases to ensure the complete system responds to soft error events as expected.

Other features available in idle include configuration frame reads, configuration register reads, external memory reads, and frame address translations. These features are useful for testing and debugging.

Most will use the SEM controller in the Mitigation and Testing mode and in its default configuration as it enables SEU event detection and correction with the ability to inject errors, and has access to all the other convenience features in idle. Others might opt to migrate to the Mitigation only mode during production to disable any error injection capabilities. Other modes and features are targeted for systems that require advanced or user-controlled SEU mitigation solutions.

Applications

Although the SEM controller can operate autonomously, most applications use the solution in conjunction with an application-level supervisory function. This supervisory function monitors the event reporting from the SEM controller and determines if additional actions are necessary (for example, reconfigure the device or reset the application).

System designers are encouraged to carefully consider each design reliability requirements and system-level supervisory functions to make informed decisions.

Is an error mitigation solution even required? If the SEM controller is required, what features should be used?

When the SEM controller is the best choice for the application, Xilinx recommends that the SEM controller is used as provided, including the system-level design helper blocks for interfacing with external devices. However, these interfaces can be modified if required for the application.

Unsupported Features

The SEM controller does not operate on soft errors in Block Memory, Distributed Memory, or Flip-Flops. Soft error mitigation in these memory resources must be addressed by the user logic through preventive measures such as redundancy or error detection and correction codes.

Other considerations when you are using the SEM controller in your design include:

- SEM controller initializes and manages the FPGA integrated silicon features for soft error mitigation and when included in a design, do not include any design constraints or options that would enable the built-in detection functions. For example, do not set POST_CRC, POST_CONFIG_CRC, or any other related constraints. Similarly, do not include options to modify GLUTMASK.
- Software computed ECC and CRC values are not supported.
- Design simulations that instantiate the controller are supported. However, it is not possible to observe the controller behaviors in simulation. Design simulation including the controller compiles, but the controller does not exit the initialization state. Hardware-based evaluation of the controller behaviors is required.
- Use of bitstream security (encryption and authentication) is currently not supported by the controller.
- Use of SelectMAP persistence is not supported by the controller.
- Only a single ICAP instance is supported for each SEM controller and it must reside at the primary/top physical location.

Licensing and Ordering Information

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

This chapter contains the specification of the LogiCORE IP UltraScale™ architecture SEM controller. This configurable controller for mitigation of soft errors in configuration memory also comes with a system-level example design showing use of the controller in a system.

Features

The SEM controller includes:

- Four IP modes to align with your SEU mitigation goals:
    - Mitigation and testing
    - Mitigation only
    - Emulation
    - Monitoring only
- Features specific to mitigation modes:
    - Integration of silicon features to leverage built-in error detection capability for mitigation modes.
    - Implementation of error correction capability to support correction of soft errors.
    - ECC algorithm-based correction that supports correction of configuration memory frames with up to 4-bit errors.
    - Minimal latency in detecting and correcting multi-bit errors due to a single SEU event that is spread across adjacent frames.
    - Implementation of error classification capability to determine if corrected errors have affected configuration memory in locations essential to the function of the design.
    - Provision for error injection to support verification of the controller and evaluation of applications of the controller.
- Variety of debug and test feature during Idle:
    - Configuration frame reads (`Query` command).

- Xilinx recommends reading configuration frame before and after error injection to filter out mask bits and set expectations of IP behavior.
- Configuration register reads (`Peek` command).
- Frame address translation (`Translate` command) translates configuration Physical Frame Address (PFA) to Linear Frame Address (LFA) and vice-versa.
- External SPI flash memory reads (`Xmem` command).

- SPI flash master helper block provides an interface between the controller and external storage. This is required when the controller is configured to perform error classification.
- UART helper block provides an interface between the controller and an external processor for ease of use when logging the controller status and performing error injection.
- Flexibility to control the location of the helper blocks and configuration primitives to be in the IP boundary or example design.
- ICAP arbitration interface to ease sharing of ICAP with other blocks and enable safe hand-off.

Table 2-1 summarizes the feature of each of the modes.

Table 2-1: **Mode Features**

Features	**Modes**			
	Mitigation and Testing	**Mitigation Only**	**Emulation**	**Monitoring**
IP state after initialization	OBSV	OBSV	IDLE	IDLE
Correction (Repair)	Yes	Yes	N/A	N/A
Classification	Optional	Optional	N/A	N/A
Error injection	Yes	N/A	Yes	N/A
Interface: Standalone	Yes	Yes	Yes	Yes
Debugging features: • Transition to Idle state • Configuration frames and register reads • External memory reads • Address translations	Yes	Yes	Yes	Yes

Standards

No standards compliance or certification testing is defined. The SEM controller is exposed to a beam of accelerated particles as part of an extensive hardware validation process.

Performance

Performance metrics for the SEM controller are derived from silicon specifications and direct measurement, and are for budgetary purposes only. Actual performance might vary.

Maximum Frequency

The maximum frequency of operation of the SEM controller is not guaranteed. In no case can the maximum frequency of operation exceed the ICAP F_{Max} specified in the relevant FPGA data sheet as configuration interface AC timing parameter F_{ICAPCK}. Table 2-2 provides a summary of ICAP F_{Max} values.

Table 2-2: **ICAP Maximum Frequency**

Device		ICAP F_{Max} (MHz)
UltraScale FPGAs	Kintex	200
	Virtex	200
	Kintex SSI	200
	Virtex SSI	200
	All Devices (0.9V, -1L)	175

Other maximum frequency limitations might apply. For more details on determining the maximum frequency of operation for the SEM controller, see System Clock Interface in Chapter 3.

Solution Reliability

The system-level design example is analyzed in the following section to provide an estimate of the FIT of the solution itself, as implemented in the FPGA. This analysis method is also appropriate for generating estimates of other circuits implemented in the FPGA.

In this analysis, all features are considered enabled with all signals brought to I/O pins. VIO core is specifically excluded from analysis, as it is unlikely a production design will include this interactive debug and experimentation capability. As a result, the estimate represents an upper bound.

Estimation Data

To calculate the reliability estimation of a design (including SEM IP), use the pre-design (spreadsheet-based) SEU FIT estimation tool. The maximum estimated FIT rate for the SEM IP solution (with all features enabled including all of the helper blocks) is in Table 2-3.

Table 2-3: **Maximum Estimated FIT Rate**

Device	FIT
Monolithic devices	9
KU115 (SSI example)	TBD

When including the contribution of block RAMs to SEM controller FIT rate calculation, only one of the three block RAMs are not protected using ECC. This unprotected 36 Kb block RAM is used as an internal data buffer. In the data array, 10,800 bits are allocated to data buffers used in error correction and classification; a soft error here would only cause an issue if it occurred during mitigation activity. No permanent data resides here and errors are therefore ignored. Another 16,400 bits are allocated to constant storage; errors in these locations are highly likely to break the controller and must be considered in the analysis. The remaining 9,664 bits are unused.

When including the contribution of the block RAMs to the helper block FIT rate calculations, the following block RAMs should be included. For SSI device solutions, the UART helper block contains two blocks RAMs (18 Kb).

Solution Latency

The error mitigation latency of the solution is defined as the total time that elapses between the creation of an error condition and the conclusion of the mitigation process. The mitigation process consists of detection, correction, and classification.

Estimation Data

The solution behaviors are based on processing of FPGA configuration memory frames. Single-bit errors always reside in a single frame. Generally, an *N*-bit error can present in several ways, ranging from one frame containing all bit errors, to *N* frames each containing 1-bit error. When multiple frames are affected by an error, the sequence of detection, correction, and classification is repeated for each affected frame.

The solution properly mitigates an arbitrary workload of errors. The error mitigation latency estimation of an arbitrary workload is complex. This section focuses on the common case involving a single frame, but provides insight into the controller behavior to aid in understanding other scenarios.

Start-Up Latency

Start-up latency is the delay between the end of FPGA configuration and the completion of the controller initialization, as marked by entry into the Observation state. This latency is a function of the FPGA size (frame count) and the solution clock frequency.

The start-up latency is incurred only once. It is not part of the mitigation process. Table 2-4 illustrates start-up latency, decomposed into sub-steps of boot and initialization.

Table 2-4: **Maximum Start-up Latency at ICAP F_{Max}**

Device		Boot Time at ICAP F_{Max} (ms)	Initialization Time at ICAP F_{Max} (ms)
UltraScale	XCKU035	127	52
	XCKU040	127	52
	XCKU060	127	75
	XCKU075	127	91
	XCKU100	127	75
	XCKU115	127	75
	XCVU065	127	75
	XCVU080	127	109
	XCVU095	127	109
	XCVU125	127	75
	XCVU160	127	75
	XCVU190	127	75
	XCVU440	127	156

The start-up latency is the sum of the boot and initialization latency. The start-up latency at the actual frequency of operation can be estimated using data from Table 2-4 and Equation 2-1.

$$StartUpLatency_{ACTUAL} = StartUpLatency_{ICAP\_F_{Max}} \cdot \left[\frac{ICAP\_F_{Max}}{Frequency_{ACTUAL}}\right]$$

Equation 2-1

Error Detection Latency

Error detection latency is the major component of the total error mitigation latency. Error detection latency is a function of the FPGA size (frame count) and the solution clock frequency. It is also a function of the type of error and the relative position of the error with respect to the position of the silicon readback process. Table 2-5 illustrates IP error detection times.

Table 2-5: **Maximum IP Error Detection Times at ICAP F_{Max}**

Device		Detection Time at ICAP F_{Max} (ms)
UltraScale	XCKU035	22
	XCKU040	22
	XCKU060	30
	XCKU075	35
	XCKU100	30
	XCKU115	30
	XCVU065	30
	XCVU080	41
	XCVU095	41
	XCVU125	30
	XCVU160	30
	XCVU190	30
	XCVU440	58

The IP error detection time for the target device, at the actual frequency of operation, can be estimated using data from Table 2-5 and Equation 2-2.

$$DetectionTime_{ACTUAL} = DetectionTime_{\text{ICAP\_F}_{Max}} \cdot \left[\frac{\text{ICAP\_F}_{Max}}{Frequency_{ACTUAL}}\right] \qquad \text{Equation 2-2}$$

The error detection latency can be bounded as follows:

- Maximum error detection latency for detection by ECC is $\text{DetectionTime}_{\text{ACTUAL}}$
- Absolute maximum error detection latency for detection by CRC alone is 2.0 × $\text{DetectionTime}_{\text{ACTUAL}}$

In the case of multi-bit errors caused by a single SEU event that spread across four adjacent frames, the SEM controller algorithm has been optimized to reduce the detection time of the errors in the subsequent frames to the minimum. This algorithm enables the controller to detect and correct up to 16-bit errors across four adjacent frames in a single pass. The worst case accumulative detection time for multi-bit errors are to multiply the number of single-bit error with the worst case detection time for a single-bit error.

Error Correction Latency

After detecting an error, the solution attempts correction. Errors are correctable depending on the selected correction mode and error type. Table 2-6 provides error correction latency for a configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-6: **Error Correction Latency, No Throttling on Monitor Interface**

Device	Correction Mode	Correctability	Error Correction Latency at ICAP_F$_{Max}$ (µs)
UltraScale	Repair	Correctable[1]	41
		Uncorrectable	21
	Any	CRC-only (Uncorrectable)	9

Notes:

1. IP could correct up to four bits of error in frame depending on its physical adjacency.

The error correction latency at the actual frequency of operation can be estimated using data from Table 2-6 and Equation 2-3.

$$CorrectionLatency_{ACTUAL} = CorrectionLatency_{\mathrm{ICAP\_F}_{Max}} \cdot \left[\frac{\mathrm{ICAP\_F}_{Max}}{Frequency_{ACTUAL}}\right] \quad \textit{Equation 2-3}$$

Error Classification Latency

After attempting correction of an error, the solution classifies the error. The classification result depends on the correction mode, error type, error location, and selected classification mode. Table 2-7 provides error classification latency for a configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-7: **Error Classification Latency, No Throttling on Monitor Interface**

Device Family	Errors in Frame (Correctability)	Classification Mode	Error Classification Latency at ICAP_F$_{Max}$ (µs)
UltraScale	Correctable	Enabled	185
	Uncorrectable	Disabled	5
	Uncorrectable	Any	5

The error classification latency at the actual frequency of operation can be estimated using data from Table 2-7 and Equation 2-4.

$$ClassificationLatency_{ACTUAL} = ClassificationLatency_{\mathrm{ICAP\_F}_{Max}} \cdot \left[\frac{\mathrm{ICAP\_F}_{Max}}{Frequency_{ACTUAL}}\right] \quad \textit{Equation 2-4}$$

Error Injection Latency

Table 2-8 provides error injection latency for a 1-bit configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-8: **Error Injection Latency, No Throttling on Monitor Interface**

Device Family	Error Injection Latency at ICAP_F_{Max} (µs)
UltraScale KU040	50

Sources of Additional Latency

It is highly desirable to avoid throttling on the Monitor Interface, because it increases the total error mitigation latency:

- After an attempted error correction, but before exiting the Error Correction state (at which time the `status_uncorrectable` flag is updated), the controller issues a detection and correction report through the Monitor Interface. If the UART helper block transmit FIFO becomes full during this report generation, the controller dwells in this state until it has written the entire report into the UART helper block transmit FIFO. When this happens, the error correction latency increases.
- After classifying an error, but before exiting the Error Classification state (at which time the `status_essential` flag is updated), the controller issues a classification report through the Monitor Interface. If the UART helper block transmit FIFO becomes full during this report generation, the controller dwells in this state until it has written the entire report into the UART helper block transmit FIFO. When this happens, the error classification latency increases.

For helper block or peripherals where the potential bottleneck is a concern, it can be mitigated. This is accomplished by adjusting the transmit FIFO size to accommodate the longest burst of status messages that are anticipated so that the transmit FIFO never goes full during error mitigation.

If a transmit FIFO full condition does occur, the increase in the total error mitigation latency is roughly estimated as shown in Equation 2-5.

$$AdditionalLatency = \frac{MessageLength - BufferDepth}{TransmissionRate}$$ *Equation 2-5*

In Equation 2-5, MessageLength – BufferDepth is in message bytes, and the Transmission Rate is in bytes per unit of time.

Sample Latency Estimation

The first sample estimation illustrates the calculation of error mitigation latency for a single-bit error by the solution implemented in an XCKU40 device with a 90 MHz clock. The solution is configured for the mitigation and testing mode with error classification disabled. The initial assumption is that no throttling occurs on the Monitor Interface.

$$DetectionLatency = 22ms \cdot \left[\frac{200MHz}{90MHz}\right] = 48.889ms$$ *Equation 2-6*

$$CorrectionLatency = 41\mu s \cdot \left[\frac{200MHz}{90MHz}\right] = 0.091ms$$ *Equation 2-7*

$$ClassificationLatency = 5\mu s \cdot \left[\frac{200MHz}{90MHz}\right] = 0.011ms$$ *Equation 2-8*

$$MitigationLatency = 48.889ms + 0.091ms + 0.011ms = 48.991ms$$ *Equation 2-9*

The final sample estimation illustrates an assessment of the additional latency that would result from throttling on the Monitor Interface. Assume the message length in both the first and second samples is approximately 80 bytes, but the buffer depth of the UART helper block is 32 bytes. Further, the UART helper block has been modified to raise the bit rate from 9600 baud to 460800 baud. The standard 8-N-1 protocol used requires 10-bit times on the serial link to transmit a 1-byte payload:

$$AdditionalLatency = \frac{80bytes - 32bytes}{\left[\frac{460800bittimes}{s} \cdot \frac{byte}{10bittimes} \cdot \frac{s}{1000ms}\right]} = 1.042ms$$ *Equation 2-10*

This result illustrates that the additional latency resulting from throttling on the Monitor Interface can become significant, especially when the data transmission is serialized and the data rate is low.

Throughput

The throughput metrics of the SEM controller are not specified.

Power

The power metrics of the SEM controller are not specified.

Resource Utilization

Resource utilization metrics for the SEM controller are derived from post-synthesis reports and are for budgetary purposes only. Actual resource utilization might vary.

Table 2-9: **Device Utilization – Monolithic Kintex and Virtex UltraScale Devices (Non-SSI)[(1)(2)]**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs	DSP48
Kintex UltraScale (All devices)	Complete solution in Mitigation and Testing mode with no optional features.	425	490	59	3 RAMB36	1

Notes:

1. The complete solution is the SEM controller and the logic included within the support wrapper hierarchy, which are intended to be used together. The IP is configured to its default GUI option where the mode is set to Mitigation and Testing, interface option is set to standalone, and error classification is disabled.
2. The Vivado Lab Edition IPs delivered in the top-level example design is not included. Using the Vivado Lab Edition increases LUTs/FFs, but decreases I/Os.

Table 2-10: **Device Utilization – Multi-SLR UltraScale Devices (SSI)[(1)(2)]**

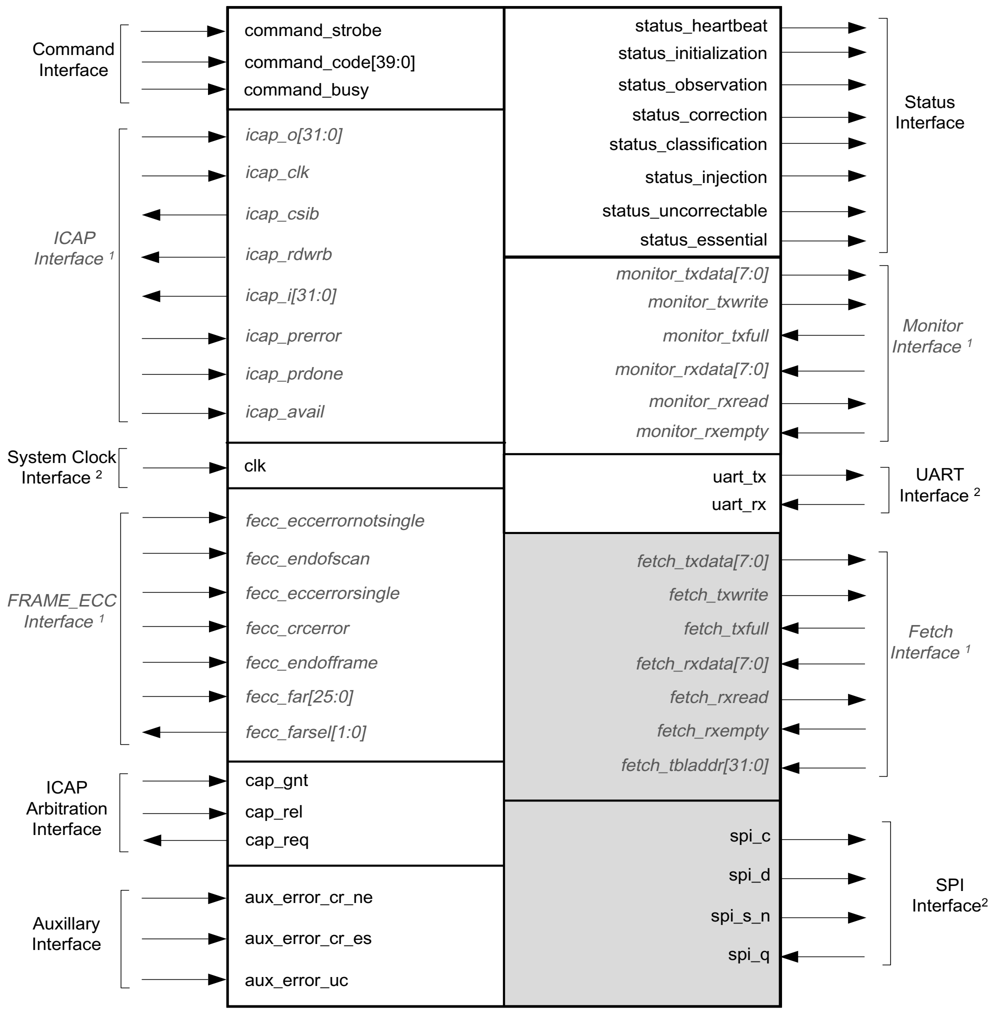
Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs	DSP48
KU115	Complete solution in Mitigation and Testing mode with no optional features.	1,060	1,280	70	6 RAMB36 and 2 RAMB18	2

Notes:

1. The complete solution is the SEM controller and the logic included within the support wrapper hierarchy, which are intended to be used together. The IP is configured to its default GUI option where the mode is set to Mitigation and Testing, interface option is set to standalone, and error classification is disabled.
2. The Vivado Lab Edition IPs delivered in the top-level example design is not included. Using the Vivado Lab Edition increases LUTs/FFs, but decreases I/Os.

Port Descriptions

The SEM controller is the kernel of the Soft Error Mitigation solution. When integrating the controller into a design, Xilinx recommends that the SEM controller is used as provided, including the system-level design that includes configuration primitives, UART, and SPI flash master helper blocks when relevant. Figure 2-1 shows the SEM controller and the system-level example design ports. Shading indicates port groups that only exist when error classification is enabled. Unless indicated, ports are available at the core and all levels of the example design hierarchy (see Figure 5-1).



Legend:
[1] Ports only available on the SEM controller
[2] Ports on the system-level example design solution

Figure 2-1: **SEM Controller Ports**

Table 2-11 provides a brief summary of the interface, dependencies on the IP feature, and the level of hierarchy where the interface is exposed. Figure 5-1 provides an example design block diagram that includes the IP hierarchy and its interface connections. For more information on the different levels of hierarchy, see Structural Options in Chapter 3.

Table 2-11: **Interfaces with Hierarchy Level**

Interfaces	**Description**	**Hierarchy Level**	
		Configuration Primitive Location = Example Design	**Configuration Primitive Location = Core**
Command	Interface to interact with SEM controller through minimal set of commands.	SEM controller	SEM controller
ICAP	Interface to ICAP to access configuration memory system.	SEM controller. ICAP Interface connected to primitive in support level.	Not exposed. ICAP Interface connected to primitive within the core.
System Clock	Interface to supply system clock to the solution.	Support Wrapper level	Support Wrapper level
FRAME_ECC	Interface to FRAME_ECC to access information from the native configuration readback mechanism on the silicon.	SEM controller. FRAME_ECC Interface connected to primitive in support level.	Not exposed. FRAME_ECC Interface connected to primitive within the core.
ICAP Arbitration	Interface to manage the sharing of ICAP with other blocks. Enables a simple way to manage graceful hand-off and resumption of the SEM controller use of the ICAP.	SEM controller	SEM controller
Auxiliary	Interface provides mechanism to notify the controller of soft error events not directly observable to the controller.	SEM controller	SEM controller
Status	Interface provides updates on the state of the IP including its health.	SEM controller	SEM controller
Monitor	Interface provides a mechanism to the user to interact with the controller that also provides comprehensive information about the controller behavior and current state.	SEM controller	SEM controller
UART	Interface to the UART helper block that serializes and de-serializes the byte-stream ASCII codes used by Monitor Interface.	Support level	Support wrapper level

Table 2-11: **Interfaces with Hierarchy Level *(Cont'd)***

Interfaces	Description	Hierarchy Level	
		Configuration Primitive Location = Example Design	Configuration Primitive Location = Core
Fetch	Interface provides a mechanism for the controller to request data from external source. ***Note:*** Only available when error classification feature is enabled.	SEM controller	SEM controller
SPI	Interface to the SPI flash master helper block to retrieve essential bits data from an external SPI flash. ***Note:*** Only available when error classification feature is enabled.	Support level	Support wrapper level

The SEM controller has no reset input or output. It automatically initializes itself with an internal synchronous reset derived from the deassertion of the global GSR signal.

The SEM controller is a fully synchronous design using `icap_clk` as the single clock. All state elements are synchronous to the rising edge of this clock. As a result, all interfaces are also synchronous to the rising edge of this clock.

ICAP Interface

The ICAP Interface is a point-to-point connection between the SEM controller and the ICAP primitive. The ICAP primitive enables read and write access to the registers inside the FPGA configuration system. The ICAP primitive and the behavior of the signals on this interface are described in the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 2].

This interface is exposed at the core level when the configuration primitives used by the IP (ICAP and FRAME_ECC) are located in the example design.

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

Name	Sense	Direction	Description
icap_o[icap_width – 1:0]	High	In	Receives O output of ICAP. The variable icap_width is equal to 32.
icap_csib	Low	Out	Drives CSIB input of ICAP.
icap_rdwrb	Low	Out	Drives RDWRB input of ICAP. Read (Active-High) or Write Active-Low) Select input.
icap_i[icap_width – 1:0]	High	Out	Drives I input of ICAP. The variable icap_width is equal to 32.
icap_clk	Edge	In	Receives the clock for the design. This same clock also must be applied to the CLK input of ICAP. The clock frequency must comply with the ICAP input clock requirements as specified in the target device data sheet.

Table 2-12: **ICAP Interface Signals *(Cont'd)***

Name	Sense	Direction	Description
icap_prdone	Low	In	Receives PRDONE output of ICAP.
icap_prerror	High	In	Receives PRERROR output of ICAP.
icap_avail	High	In	Receives AVAIL output of ICAP.

System Clock Interface

The System Clock Interface is used to provide a system-level clock to the ICAP and SEM controller. Internally the clock signal is distributed on a global clock buffer to all the synchronous logic element. This interface is available at the support wrapper level.

For more information on this interface, see System Clock Interface in Chapter 3.

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

Name	Sense	Direction	Description
clk	High	In	Clock input that drives the ICAP and SEM controller.

ICAP Arbitration Interface

The ICAP Arbitration Interface simplifies the ability of your design to share the ICAP with other blocks and ensure safer hand-off of the ICAP controls. It is expected that this interface is used with an ICAP arbiter.

For more information on this interface, see ICAP Arbitration Interface in Chapter 3.

Table 2-14: **ICAP Arbitration Interface Signals**

Name	Sense	Direction	Description
cap_gnt	High	In	For use with an ICAP arbiter. This signal is asserted by the arbiter to inform the SEM controller that it has permission to access the ICAP. After cap_gnt is asserted, it should remain asserted until cap_req is deasserted. If arbitration is not required, tie this signal to a constant 1.
cap_rel	High	In	For use with an ICAP arbiter. This signal should be asserted by the arbiter on every clock cycle where something else is requesting access to the ICAP. When set to 1, the signal should remain at 1 until cap_req returns to 0. This signal indicates to the IP that it should relinquish control of the ICAP at the earliest safe opportunity. If arbitration is not required, tie this signal to a constant 0.
cap_req	High	Out	For use with an ICAP arbiter. This signal is asserted by the IP on every clock cycle where it has data to transfer to the ICAP.

FRAME_ECC Interface

The FRAME_ECC Interface is a point-to-point connection between the SEM controller and the FRAME_ECC primitive. The FRAME_ECC primitive provides a window into the soft error detection function in the FPGA configuration system.

Table 2-15: **FRAME_ECC Interface Signals**

Name	Sense	Direction	Description
fecc_eccerrornotsingle	High	In	Receives ECCERRORNOTSINGLE output of FRAME_ECC.
fecc_eccerrorsingle	High	In	Receive ECCERRORSINGLE output of FRAME_ECC.
fecc_endofframe	High	In	Receive ENDOFFRRAME output of FRAME_ECC.
fecc_endofscan	High	In	Receive ENDOFSCAN output of FRAME_ECC.
fecc_crcerror	High	In	Receive CRCERROR output of FRAME_ECC.
fecc_farsel[1:0]	High	Out	Send FARSEL input of FRAME_ECC.
fecc_far[far_width – 1:0]	High	In	Receives FAR output of FRAME_ECC. The variable far_width is equal to 26.

Status Interface

The Status Interface provides a convenient set of decoded outputs that indicate, at a high level, what the controller is doing.

For more information on this interface, see Status Interface in Chapter 3.

Table 2-16: **Status Interface Signals**

Name	Sense	Direction	Description
status_heartbeat	High	Out	The heartbeat signal is active while status_observation is asserted. This output issues a single-cycle high pulse at least once every TBD clock cycles. This signal can be used to implement an external watchdog timer to detect "controller stop" scenarios that can occur if the controller or clock distribution is disabled by soft errors. When status_observation is deasserted, the behavior of the heartbeat signal is unspecified.
status_initialization	High	Out	The initialization signal is active during controller initialization, which occurs one time after the design begins operation.
status_observation	High	Out	The observation signal is active during controller observation of bit upsets. This signal remains active after an error detection while the controller queries the hardware for information.
status_correction	High	Out	The correction signal is active during controller correction of an error or during transition through this controller state if correction is disabled.

Table 2-16: **Status Interface Signals *(Cont'd)***

Name	Sense	Direction	Description
status_classification	High	Out	The classification signal is active during controller classification of an error or during transition through this controller state if error classification is disabled.
status_injection	High	Out	The injection signal is active during controller injection of an error. When an error injection is complete, and the controller is ready to inject another error or return to observation, this signal returns inactive.
status_essential	High	Out	The essential signal is an error classification status signal. Prior to exiting the Classification state, the controller sets this signal to reflect whether the error occurred on an essential bit(s). Then, the controller exits Classification state.
status_uncorrectable	High	Out	The uncorrectable signal is an error correction status signal. Prior to exiting the Correction state, the controller sets this signal to reflect the correctability of the error. Then, the controller exits Correction state.

The `status_heartbeat` output provides an indication that the controller is active. Although the controller mitigates soft errors, it can also be disrupted by soft errors. For example, the controller clock can be disabled by a soft error. If the `status_heartbeat` signal stops, you can take remedial action.

TIP: *See Systems in Chapter 3 for more details about remedial action available if there is a soft error upset.*

The `status_initialization`, `status_observation`, `status_correction`, `status_classification`, and `status_injection` outputs indicate the current controller state. The `status_uncorrectable` and `status_essential` outputs qualify the nature of detected errors.

Two additional controller states can be decoded from the five controller state outputs. If all five signals are low, the controller is idle (inactive but ready to resume). If all five signals are high, the controller is halted (inactive due to fatal error).

Command Interface

The Command Interface provides a convenient set of inputs to command the controller to inject a bit error into configuration memory.

For more information on this interface, see Command Interface in Chapter 3.

Table 2-17: **Command Interface Signals**

Name	Sense	Direction	Description
command_strobe	High	In	The command control is used to indicate a command request. The command_strobe signal should be pulsed High for one cycle, synchronous to icap_clk, when command_busy is low concurrent with the application of a valid code to the command_code input.
command_code[39:0]	High	In	The command_code bus is used to specify the command the controller should perform. The value on this bus is captured at the same time command_strobe is sampled active.
command_busy	High	Out	The busy signal is used to indicate whether the controller is ready to process a command. Only assert command_strobe when command_busy is Low.

The Command Interface provides a simple interface for you to perform error injection and software reset. For more information on how to use the interface and valid command codes, see Command Interface in Chapter 3.

The use of this interface is entirely optional. If not used, all inputs can be tied to Low.

Monitor Interface

The Monitor Interface provides a mechanism for you to receive detailed status of the controller and to interact with it. The controller is designed to read commands and write status information to this interface as ASCII strings. The status and command capability of the Monitor Interface is a superset of the Status Interface and the Command Interface. This interface is always available at the core level. In the example design, this interface is connected to a UART helper block.

At the minimum, Xilinx recommends that you store the status sent on this interface into a FIFO as it assists in debugging any future behavior if and when it is needed. For more information, see Monitor Interface in Chapter 3.

Table 2-18: **Monitor Interface Signals**

Name	Sense	Direction	Description
monitor_txdata[7:0]	High	Out	Parallel transmit data from controller.
monitor_txwrite	High	Out	Write strobe, qualifies validity of parallel transmit data.
monitor_txfull	High	In	This signal implements flow control on the transmit channel, from the helper block to the controller.
monitor_rxdata[7:0]	High	In	Parallel receive data from the helper block.

Table 2-18: **Monitor Interface Signals *(Cont'd)***

Name	Sense	Direction	Description
monitor_rxread	High	Out	Read strobe, acknowledges receipt of parallel receive data.
monitor_rxempty	High	In	This signal implements flow control on the receive channel, from the helper block to the controller.

UART Interface

As a convenience, the system-level example design provides a UART interface to ease integration of the Monitor Interface to a processor block by connecting the Monitor Interface to a UART helper block. You can use this UART interface to receive status information from the IP and send commands to inject errors to the IP and confirm that the error injected is detected and corrected.

The UART helper block serializes status information generated by controllers (a byte stream of ASCII codes) for serial transmission. Similarly, the UART helper block de-serializes command information presented to controllers (a bitstream of ASCII codes) for parallel presentation to controllers.

The UART helper block uses a standard serial communication protocol. The helper block contains synchronization and over sampling logic to support asynchronous serial devices that operate at the same nominal baud rate. For more information, see UART Interface in Chapter 3.

The resulting interface is directly compatible with a wide array of devices ranging from embedded microcontrollers to desktop computers. External level translators might be necessary depending on system requirements.

Table 2-19: **UART Interface Signals**

Name	Sense	Direction	Description
uart_tx	Low	Out	Serial transmit data.
uart_rx	Low	In	Serial receive data.

Fetch Interface

The Fetch Interface provides a mechanism for the controller to request data from an external source.

During error classification, the controller might need a frame of essential bit data. The controller is designed to write a command describing the desired data to the Fetch Interface in binary. The external source must use the information to fetch the data and return it to the Fetch Interface.

When error classification is enabled, this interface is always available at the core level. In the example design, this interface is connected to a SPI flash master helper block and a SPI Interface is used to connect this interface to an external SPI flash that contains the essential bits data.

For more information on this interface, see Fetch Interface in Chapter 3.

Table 2-20: **Fetch Interface Signals**

Name	Sense	Direction	Description
fetch_txdata[7:0]	High	Out	Parallel transmit data from controller.
fetch_txwrite	High	Out	Write strobe, qualifies validity of parallel transmit data.
fetch_txfull	High	In	This signal implements flow control on the transmit channel, from the helper block to the controller.
fetch_rxdata[7:0]	High	In	Parallel receive data from the helper block.
fetch_rxread	High	Out	Read strobe, acknowledges receipt of parallel receive data.
fetch_rxempty	High	In	This signal implements flow control on the receive channel, from the helper block to the controller.
fetch_tbladdr[31:0]	High	In	Used to specify the starting address of the controller data table in the external source.

SPI Interface

As a convenience, the system-level example design provides a SPI Interface to connect the Fetch Interface to SPI flash master helper block to retrieve essential bits data from an external SPI flash. When present, the SPI flash master helper block in the system-level design example is a fixed-function SPI bus master.

This helper block accepts commands from the controller through the fetch that consists of an address and a byte count. The helper block generates SPI bus transactions to fetch the requested data from an external SPI flash. The helper block formats the returned data for controllers to pick up.

The helper block uses standard SPI bus protocol, implementing the most common mode (CPOL = 0, CPHA = 0, often referred to as "Mode 0"). The SPI bus clock frequency is locked to one half of the system clock for the system-level design example. For more information, see SPI Interface in Chapter 3.

The resulting interface is directly compatible with a wide array of standard SPI flash. External level translators might be necessary depending on system requirements.

Table 2-21: **SPI Interface Signals**

Name	Sense	Direction	Description
spi_c	Edge	Out	SPI bus clock for an external SPI flash.
spi_d	High	Out	SPI bus "master out, slave in" signal for an external SPI flash.

Table 2-21: **SPI Interface Signals *(Cont'd)***

Name	Sense	Direction	Description
spi_s_n	Low	Out	SPI bus select signal for an external SPI flash.
spi_q	High	In	SPI bus "master in, slave out" signal for an external SPI flash.

Auxiliary Interface

The Auxiliary Interface provides a mechanism to notify the controller of soft error events that take place in areas not directly observable to the controller through the scanning of the configuration memory. For more information, see Auxiliary Interface in Chapter 3.

Table 2-22: **Auxiliary Interface Signals**

Name	Sense	Direction	Description
aux_error_cr_ne	High	In	Auxiliary input indicating a correctable non-essential error.
aux_error_cr_es	High	In	Auxiliary input indicating a correctable essential error.
aux_error_uc	High	In	Auxiliary input indicating uncorrectable error.

Designing with the Core

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

General Design Guidelines

This section describes the steps required to turn the UltraScale™ architecture SEM controller into a fully-functioning design with user-application logic.

IMPORTANT: *Not all implementations require all the design steps listed in this chapter. Follow the logic design guidelines in this manual carefully.*

Use the Example Design as a Starting Point

The SEM controller core has an example design that can be implemented in an FPGA and used to understand the behavior of the IP.

The system-level example design encapsulates the SEM controller and various primitives and helper blocks that serve to interface the Controller to other devices, as shown in Figure 5-1, page 85.

For designs targeting SSI devices, the system-level example design provides the example of how to mitigate soft errors in each SLR and should be used at the minimum as a guidance. For more information on the delivered example design, see Chapter 5, Detailed Example Design and for the system-level port solutions, see Port Descriptions in Chapter 2.

TIP: *Xilinx recommends integrating the `<component_name>_support_wrapper.v` and all of its submodules into the user design, as this contains all integral logic of the total Soft Error Mitigation solution. The solution has been fully verified as delivered. See Structural Options, page 33.*

Know the Degree of Difficulty

The SEM controller design can be challenging to implement, and the degree of difficulty is further influenced by:

- Maximum system clock frequency
- Targeted device architecture
- Nature of your application
- Level of device congestion

All SEM implementations need careful attention to system performance requirements. Hence, it is strongly recommended that the IP is integrated in the early stages of the design cycle. Pipelining, logic mapping, placement constraints, and logic duplication are all methods that help boost system performance.

IMPORTANT: *Do not add any pipeline registers between the SEM controller and ICAP and FRAME_ECC primitives as it will alter and corrupt the behavior of the SEM controller.*

Keep It Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between your application and the core. This means that all inputs and outputs from your application should come from, or connect to a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx® tools to place and route the design.

Recognize Timing Critical Signals

The XDC provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. For further information, see Constraining the Core in Chapter 4.

Make Only Allowed Modifications

The SEM controller core is not user-modifiable. Do not make modifications as they can have adverse effects on system timing. Supported user configurations of the SEM controller core can only be made by selecting the options from within the Vivado® Design Suite tool when the core is generated. See Customizing and Generating the Core in Chapter 4.

Structural Options

The **IP Customization** dialog box includes **Structural Options** which provide location choices for the configuration primitives and the available helper blocks. These options specify different permutations of the IP boundary that enable the delivered IP solution to function either as a standalone core with a simplified interface or as a part of a larger design with greater access to the SEM controller interfaces. These options minimize the scope of HDL modifications required when integrating the solution into the user design, while also providing flexibility for different uses of the IP core.

There are two levels of hierarchy which are called:

- `<component_name>_support`
- `<component_name>_support_wrapper`

Figure 3-1 and Figure 3-2 show two hierarchies where the configuration primitive is either in the core or in the example design. In these figures, `<component_name>` is the name of the generated core.

The difference between the two hierarchies is the boundary of the core and where the helper blocks are instantiated. It is controlled using the **Structural Options** in the SEM controller customization Vivado IDE.

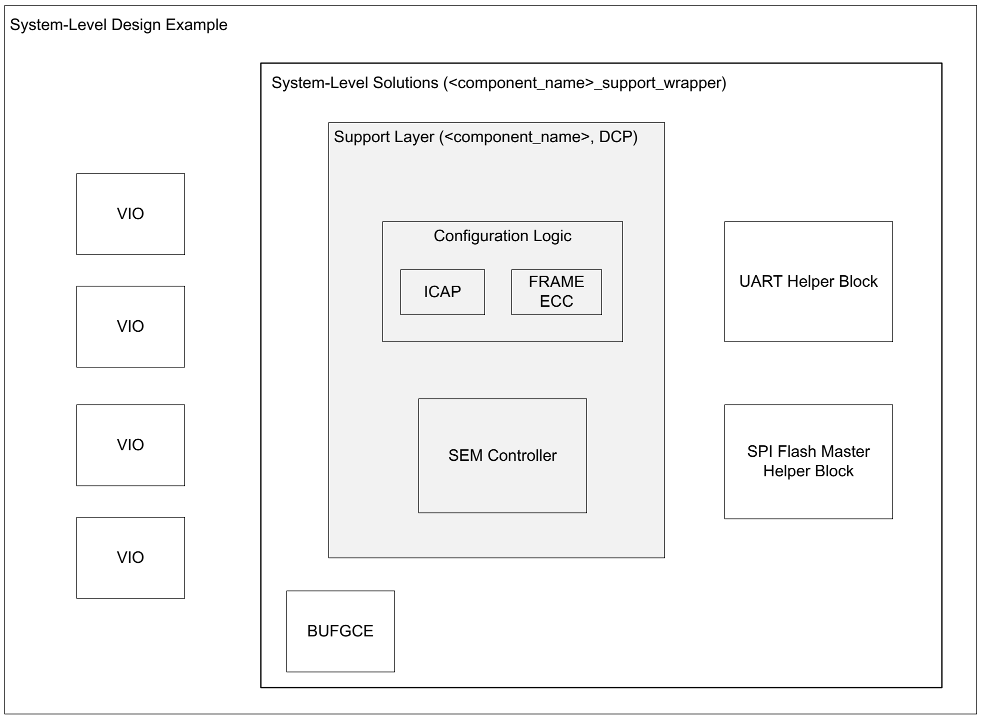

System-Level Design Example
System-Level Solutions (<component_name>_support_wrapper)
Support Layer (<component_name>, DCP)
VIO
VIO
VIO
VIO
Configuration Logic
ICAP
FRAME ECC
UART Helper Block
SEM Controller
SPI Flash Master Helper Block
BUFGCE

Figure 3-1: **Configuration Primitive Included in Core**

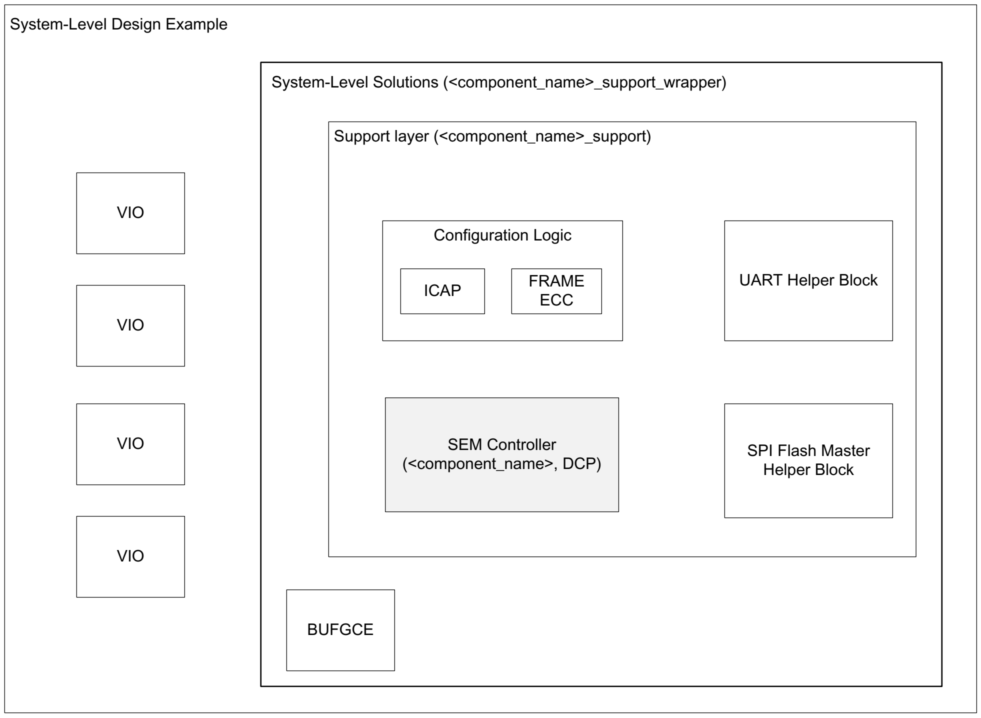

System-Level Design Example
System-Level Solutions (<component_name>_support_wrapper)
Support layer (<component_name>_support)
VIO
VIO
VIO
VIO
Configuration Logic
ICAP
FRAME ECC
UART Helper Block
SEM Controller (<component_name>, DCP)
SPI Flash Master Helper Block
BUFGCE

Figure 3-2: **Configuration Primitive Included in Example Design**

TIP: *Xilinx recommends integrating the* `<component_name>_support_wrapper` *level hierarchy into the design as it contains all integral logic of the total Soft Error Mitigation solution and has been fully verified as delivered.*

IMPORTANT: *If you choose not to integrate the example design, attention needs to be paid to the system-level requirements and recommendation for each interface in the following sections.*

Configuration Primitive Included in Core

Select this option if you do not have other logic requiring access to ICAP and FRAME_ECC primitives. These primitives are then included in the core, and connection to and from the primitives and the SEM controller are automatically connected and not visible as core ports.

Configuration Primitive Included in Example Design

Select this option if you have other logic requiring access to ICAP and FRAME_ECC primitives. These primitives are then instantiated in the example design hierarchy.

For guidance on sharing ICAP between the user design and the SEM controller, contact Xilinx.

Interfaces

This section provides details on how to apply the core in your design and is organized based on the different interfaces available for the IP. Each section discusses the purpose of the interface, how to use it, its behavior, and how to integrate it into a larger system.

ICAP Interface

The ICAP Interface should be connected to the ICAP primitive as described in ICAP Interface in Chapter 2. The `icap_clk` port of this interface is also the main input clock to the IP. The following System Clock Interface section describes the requirements for this clock.

System Clock Interface

The following recommendations exist for the system input clock. These recommendations are derived from the FPGA data sheet requirements for clock signals applied to the FPGA configuration system:

- Duty Cycle: 45% minimum, 55% maximum

The higher the frequency of the input clock, the lower the mitigation latency of the solution. Therefore, faster is better. There are several important factors that must be considered in determination of the maximum input clock frequency:

- Frequency must not exceed FPGA configuration system maximum clock frequency. Consult the device data sheet for the target device for this information.
- Frequency must not exceed the maximum clock frequency as reported in the static timing analyzer. This is generally not a limiting constraint.

Based on the fully synchronous design methodology, additional considerations arise in clock frequency selections that relate to the timing of external interfaces, if the system-level design example is used:

- For the SPI flash master block and SPI interface:
  - The SPI bus timing budget must be evaluated to determine the maximum SPI bus clock frequency; a sample analysis will be provided in a future version of this document.
  - The SPI bus clock is the input clock divided by two; therefore, the input clock cannot exceed twice the maximum SPI bus clock frequency.
- For the UART helper block and UART interface:
  - The input clock and the serial interface baud rate are related by an integer multiple of 16. For very high baud rates or very low input clock frequencies, the solution space might be limited if standard baud rates are desired.

- ◦ A sample analysis will be provided in a future version of this document.

System-Level Considerations

The system clock is absolutely critical to the controller and therefore needs to be provided from the most reliable source possible. To achieve the very highest reliability, the clock must be connected as directly as possible to the controller. This means the use of an external oscillator of the desired frequency, connected directly to a pin associated directly with a global clock buffer.

The inclusion of any additional logic or interconnect into the clock path results in additional configuration memory being used to control the connection of the clock to the controller. This additional memory has a negative effect on the estimated controller FIT. Although the impact is small, it is best to strive for high reliability unless it poses a significant burden.

When additional logic exists on the clock path (for example, clock management blocks, or logic-based clock division), care must be taken to guarantee by design that the maximum clock frequency of the FPGA configuration system and the maximum clock frequency of the system-level design example and controller are not transiently violated.

For example, the clock output of a DLL or PLL might be “out of specification” while those functions lock. One method of handling this is to use a global clock buffer with enable (BUFGCE). Only enable the global clock buffer after lock is achieved. As an example, the system-level clock in the example design is distributed using a BUFGCE where the clock enable is tied High.

The system-level design example, the controller, and the configuration system are all static. This means, any clock frequency can be used up to the specified maximum allowed by the FPGA configuration system or the maximum clock frequency of the system-level design example and controller (whichever is lower). However, higher clock rates result in faster mitigation of errors, which is desirable.

After considering the factors, select an input clock frequency that satisfies all requirements.

FRAME_ECC Interface

The FRAME_ECC Interface should be connected to the FRAME_ECC primitive as described in FRAME_ECC Interface, page 25 and has no additional requirements.

ICAP Arbitration Interface

This interface is used in systems where the ICAP primitive is shared between multiple functions. This interface should be driven by an arbiter that controls when the different functions have ICAP control. For more information of the mechanism to sharing the ICAP with the SEM controller, contact Xilinx.

If the ICAP sharing function is not used, tie off the interface inputs in the following method:

- `cap_gnt` = High
- `cap_rel` = Low

Switching Behavior

The switching characteristics are illustrated in Figure 3-3. In the first section of the diagram, the SEM controller is booting up and asserts the `cap_req` signal to request access to the ICAP. It continuously asserts this signal as long as it needs access to the ICAP. At some point in time, the ICAP arbiter asserts the `cap_gnt` signal which informs the controller that it now has ICAP control. The IP enters the Initialization state when `cap_gnt` is asserted. After initialization completes, the IP enters the Observation state.

In the second section of the diagram, the ICAP arbiter asserts the `cap_rel` signal, indicating to the IP that it needs to release the ICAP. Subsequently the SEM controller automatically transitions to the Idle state and deasserts its `cap_req` signal. The ICAP arbiter then deasserts the `cap_gnt` and `cap_rel` signals. Now, the IP asserts the `cap_req` signal and starts the boot process. When `cap_gnt` is asserted, the IP enters the Initialization state. After the initialization completes, the IP enters the Observation state.

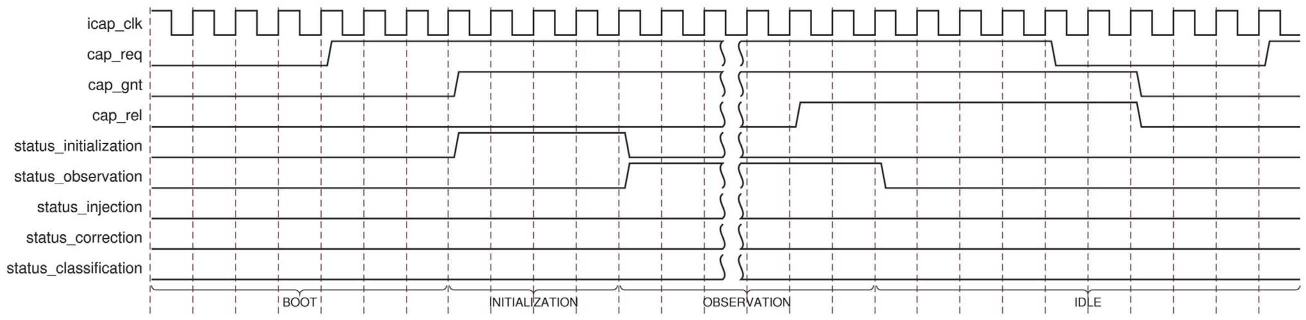


Figure 3-3: **ICAP Arbitration Switching Behavior**

System-Level Requirements

This interface should be used in a system that has more than one block that needs to access the ICAP. For such systems, an arbiter should be designed to manage and control the hand-off of the ICAP between different blocks.

Status Interface

Direct, logic-signal-based event reporting is available from the Status Interface. The Status Interface can be used for many purposes, but its use is entirely optional. This interface reports three different types of information:

- **State –** Indicates what a controller is doing.
- **Flags –** Identifies the type of error detected.
- **Heartbeat –** Indicates scanning is active.

For SSI implementations of the system-level example design, there is a controller instance on each SLR and therefore an independent Status Interface per SLR. In most cases, desired signals from the Status Interface should be brought to I/O pins on the FPGA. The system-level design example brings all of the signals to I/O pins.

Externally, the status signals can be connected to indicators for viewing, or to another device for observation. To properly capture event reporting, the switching behavior of the Status Interface must be accounted for when interfacing to another device.

The Status Interface can become unwieldy, especially in SSI implementations, due to the number of signals. Only the heartbeat event is unique to the Status Interface. The other information is also available on the Monitor Interface.

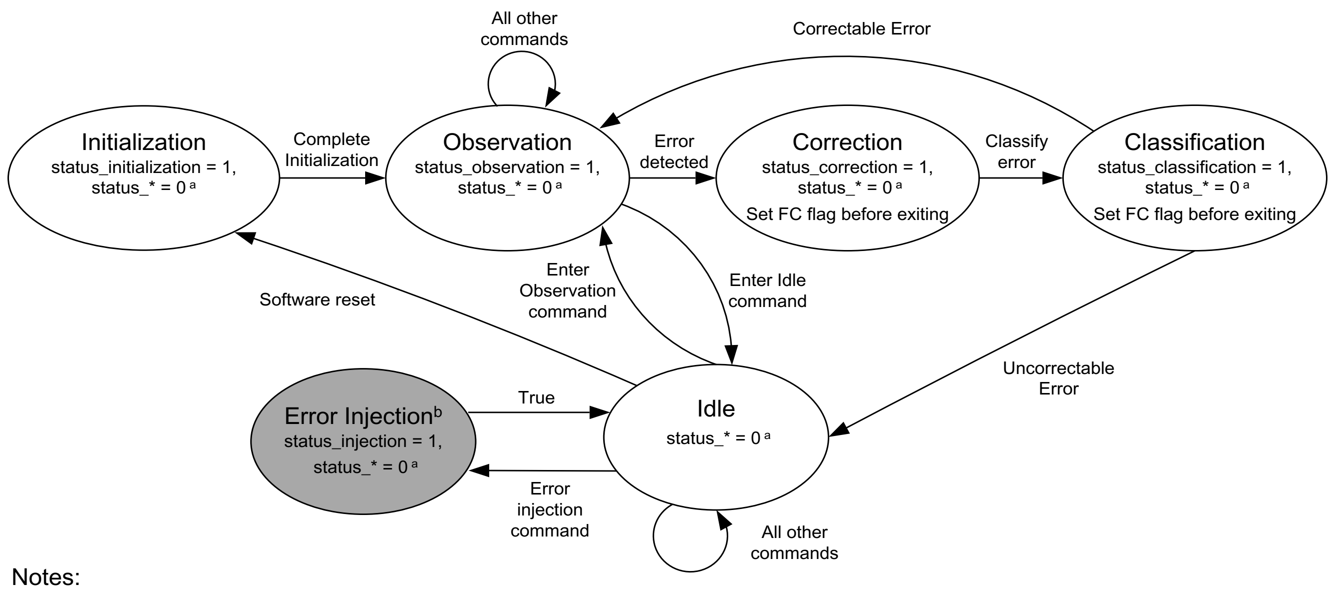
The signals in the Status Interface are generated by sequential logic processes in controllers using the clock supplied to the system-level design example. As a result, the pulse widths are always an integer number of clock cycles.

States

The SEM controller has seven valid states:

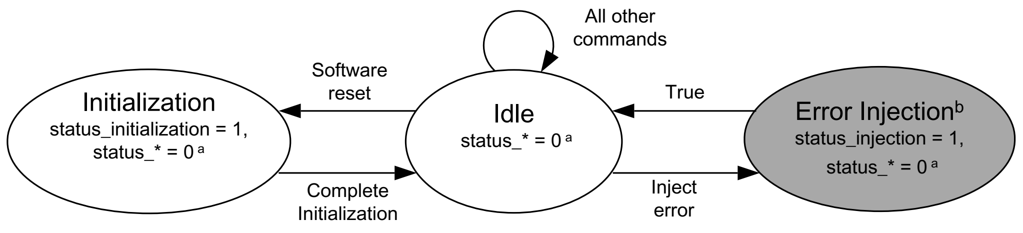
- Initialization
- Observation
- Correction
- Classification
- Idle
- Injection
- Fatal Error

Figure 3-4 and Figure 3-5 illustrate the valid states and state transitions for each IP mode.



Notes:
[a] status_* consists of status_observation, initialization, correction, injection and classification (omitting the signal that has already been defined in the state).
[b] Only available in Mitigation and testing mode.
SC flag is updated before IP exits a state.
Transition to Fatal Error state is not shown.

Figure 3-4: **Valid State Transition Diagram for Mitigation Modes**



Notes:
[a] status_* consists of status_observation, initialization, correction, injection and classification (omitting the signal that has already been defined in the state).
[b] Only available in Emulation mode.
SC flag is updated before IP exits a state.
Transition to Fatal Error state is not shown.

Figure 3-5: **Valid State Transition Diagram for Other Modes**

Note that these state changes are reported in more detail by the Monitor Interface. For more information, see Monitor Interface.

The detailed description of each state is given in the following sections.

Initialization

The controller is held inactive by the FPGA global set/reset signal. At the completion of configuration, the FPGA configuration system deasserts the global set/reset signal and the controller boots. The controller maintains all five state bits on the Status Interface deasserted through the boot process.

The controller polls its `cap_gnt` and `cap_rel` input during boot to determine if it has been granted permission to enter the Initialization state and begin using ICAP. Unless an ICAP arbiter is used to drive these signals, the `cap_gnt` signal should be High and `cap_rel` should be Low.

During the Initialization state, `status_initialization` is asserted High. Initialization includes some internal housekeeping, as well as directly observable events such as the generation of an initialization report on the Monitor Interface. The specific activities include:

- First readback cycle during which the frame-level ECC checksums are computed.
- Second readback cycle during which the device-level CRC checksum is computed.

At the completion of initialization, the controller transitions to the Observation or Idle state depending on the IP mode.

Observation (Mitigation Modes Only)

The controller spends virtually all of its time in the Observation state. During the Observation state, `status_observation` is asserted High and the controller observes the FPGA configuration system for indication of error conditions.

If no error exists, and the controller receives a command (from either the Command Interface or the Monitor Interface), then the controller processes the received command. Only two commands are supported in the Observation state, the **Enter Idle** and **Status Report** commands. The controller ignores all other commands.

- **Enter Idle** – This command can be applied through either the Command Interface or the Monitor Interface. This is used to idle the controller so that other commands can be performed. This command causes the controller to transition to the Idle state.
- **Status Report** – This command provides some diagnostic information, and can be helpful as a mechanism to “ping” the controller. This command is only supported on the Monitor Interface.

In the event an error is detected, the controller reads additional information from the configuration logic in preparation for a correction attempt. After the controller has gathered the available information, it transitions to the Correction state.

Correction (Mitigation Modes Only)

The controller attempts to correct errors in the Correction state. The controller always passes through the Correction state, even if correction is not successful. During the correction state, `status_correction` is asserted High.

If the error is a CRC-only error, the controller sets `status_uncorrectable` and generates a report on the Monitor Interface. It then transitions to the Classification state. If the error is not a CRC-only error, then it attempts to correct the error using algorithmic methods.

If the error is correctable, the controller performs active partial reconfiguration to rewrite the frame with the corrected contents and clears `status_uncorrectable`. Otherwise, the controller sets `status_uncorrectable`. In either case, the controller generates a correction report on the Monitor Interface and then transitions to the Classification state.

TIP: *`status_uncorrectable` should be sampled at the falling edge of `status_correction`.*

Classification (Mitigation Modes Only)

The controller classifies errors in the Classification state. The controller always passes through the Classification state, even if error classification is disabled. During the Classification state, `status_classification` is asserted High.

All errors signaled as uncorrectable during the Correction state are signaled as essential. The only reason an error can be uncorrectable is because it cannot be located. In this circumstance, the controller cannot look up the error to determine whether it is essential. The controller asserts `status_essential`, generates a classification report on the Monitor Interface, and then transitions to the Idle state. After an uncorrectable error is encountered, the controller does not continue looking for errors. Now, the FPGA must be reconfigured.

TIP: *`status_essential` should be sampled at the falling edge of `status_classification`.*

The treatment of errors signaled as correctable during the Correction state depends on the controller option setting. If error classification is disabled, all correctable errors are unconditionally signaled as essential. If error classification is enabled, the controller generates an essential bits data request on the Fetch Interface.

In the system-level design example, the SPI flash master helper block translates this essential bits data request into a read of the external memory. The returned data is provided to the controller by the SPI flash master helper block. With this data, the controller then determines whether it is essential. In all cases, the controller generates a classification report on the Monitor Interface, changes `status_essential` as appropriate, and then transitions to the Observation state to resume looking for errors.

Idle

When the controller enters the Idle state, it disables the built-in configuration memory scan and checks. Therefore, SEU events are not detected or corrected by the IP. This state is used for testing and debugging purposes. The Idle state is indicated by the deassertion of all five state bits on the Status Interface. The following table summarizes the commands available from this state and which interface these commands can be applied through.

Table 3-1: **Idle State – Available Commands**

Command	Description	Availability
Enter observation	This command is used to return the controller to the Observation state so that errors can be detected. It is only valid for IP configured for mitigation modes. This command is ignored by the IP in other modes.	Command and Monitor Interface
Error injection	These commands direct the controller to perform error injections. Multi-bit errors can be constructed by injecting multiple single bit errors.	Command and Monitor Interface
Software reset	This command directs the controller to perform a software reset (reboots and re-initializes the SEM controller).	Command and Monitor Interface
Full status report	This command provides comprehensive diagnostic information, and can be helpful as a mechanism to "ping" the controller. This command is only supported on the Monitor Interface.	Monitor Interface Only
Configuration Frame Reads (`Query` command)	This command provides the ability to read the contents of the configuration memory. Xilinx recommends performing a `Query` command before and after performing an error injection on a configuration address to set the expectations of the IP behavior. For more information, see Configuration Memory Masking, page 68.	Monitor Interface Only
Configuration Register Reads (`Peek` command)	This command provides the ability to read the contents of the Configuration registers.	Monitor Interface Only
Frame Address Translation (`Translate` command)	Convert LFA addresses to PFA addresses and vice-versa.	Monitor Interface Only
External Memory Reads (`Xmem` command)	This command provides the ability to read the contents of an external memory device.	Monitor Interface Only

For more information of how to generate the above commands, see Command Interface and Monitor Interface.

Injection (Mitigation and Testing or Emulation Modes Only)

The controller performs error injections in the Injection state. The controller always passes through the Injection state in response to a valid error injection command issued from the Idle state, even if error injection is disabled. During the Injection state, `status_injection` is asserted High.

The error injection process is a simple read-modify-write to invert one configuration memory bit at an address specified as part of the error injection command. The controller always transitions from the Injection state back to the Idle state. Multi-bit errors can be constructed by repeated error injection commands, each resulting in a transition through the Injection state.

Fatal Error

The controller enters the Fatal Error state when it detects an internal inconsistency. Although very unlikely, it is possible for the controller to halt due to soft errors that affect the controller-related configuration memory or the controller design state elements. The Fatal Error state is indicated by the assertion of all five state bits on the Status Interface, along with a fatal error report message. This condition is non-recoverable and the FPGA must be reconfigured.

Switching Behavior

The collective switching behavior of the state signals `status_initialization`, `status_observation`, `status_correction`, `status_classification`, and `status_injection` is illustrated in Figure 3-6. In the figure, the `status_[`*`state`*`]` signal represents the five state signals, as a group, which can be considered an indication of the controller state.

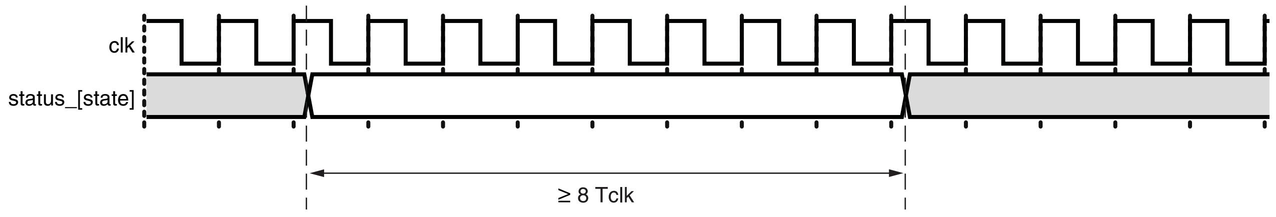


Figure 3-6: **Status Interface State Signals Switching Characteristics**

System-Level Requirements

Monitoring the status signals for completion of the Initialization state and transition to the Observation/Idle state (depending on the mode) and fatal errors are good practices in ensuring the IP behaves as expected. For other system-level recommendations to monitor the health and state of the SEM controller IP, see Systems.

Flags

The `status_uncorrectable` and `status_essential` signals are flags used to indicate whether the error detected is correctable and/or essential. If the Classification feature is disabled, the `status_essential` signal asserts and stays asserted after the first error is detected and corrected. These signals are only valid in mitigation modes.

Switching Behavior

The switching behavior of the flag signals, `status_uncorrectable` and `status_essential`, is relative to the exit from the states where these flags are updated, as illustrated in Figure 3-7 and Figure 3-8. The figures illustrate a window of time when the flags are valid with respect to transitions out of the state in which they can be updated. Specific flag values are not shown in the waveform.

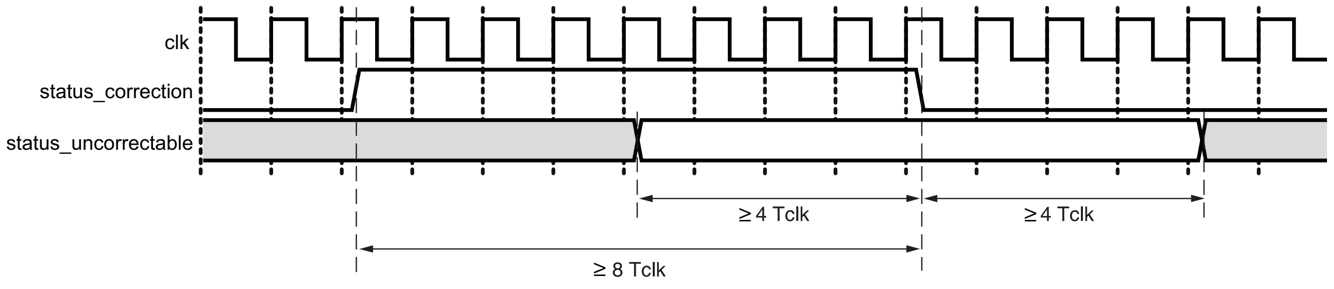


Figure 3-7: **Status Interface Uncorrectable Flag Switching Characteristics**

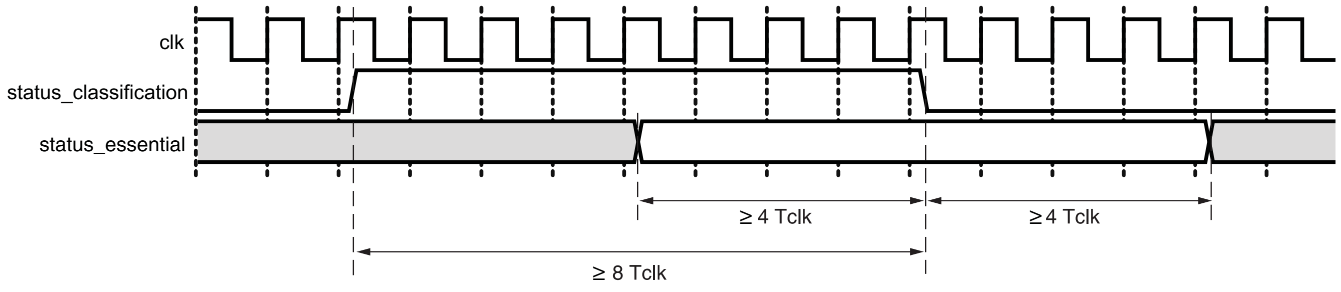


Figure 3-8: **Status Interface Essential Flag Switching Characteristics**

System- Level Requirements

In the case where the IP detects an uncorrectable error, the system-level design must assess what action must be taken. If the Classification feature is enabled, the system-level design should also consider what action must be taken if an essential error has been detected and corrected. System-level action taken is dependent on the overall SEU mitigation goal of the design. At a minimum, Xilinx recommends that a system log is generated indicating these conditions have occurred.

TIP: *Note that when the SEM controller detects a uncorrectable error, it stops scanning the configuration memory for SEUs and transition into the Idle state.*

For other system-level recommendations to monitor the health and state of the SEM controller IP, see Systems.

Heartbeat

This signal is used to monitor the status of the configuration readback system. It is a direct output from the readback process and is active during the Observation state. After entering the Observation state, the heartbeat signal becomes active when the readback process is scanning for errors. The first heartbeat pulse observed during the Observation state must be used to arm any circuit that monitors for loss of heartbeat. In all other states, the heartbeat signal behavior is unspecified.

Switching Behavior

The switching behavior of the heartbeat signal, `status_heartbeat`, is illustrated in Figure 3-9. This signal is a direct output from the readback process, and is active during the Observation state. After entering the Observation state, the heartbeat signal becomes active when the readback process is scanning for errors. The first heartbeat pulse observed during the Observation state must be used to arm any circuit that monitors for loss of heartbeat. In all other states, the heartbeat signal behavior is unspecified.

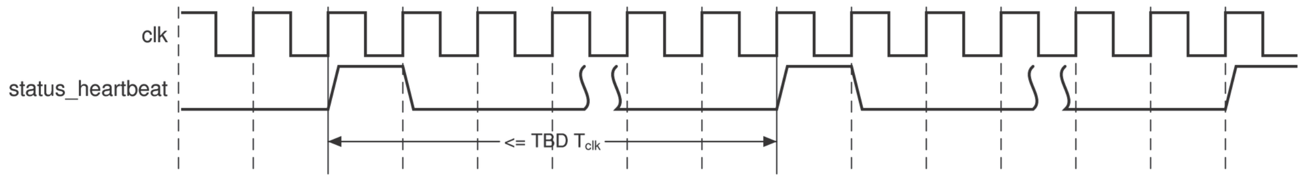


Figure 3-9: **Status Interface Heartbeat Switching Characteristics**

Due to the small pulse widths involved, approaches such as sampling the Status Interface signals through embedded processor GPIO using software polling are not likely to work. Instead, use other approaches such as using counter/timer inputs, edge sensitive interrupt inputs, or inputs with event capture capability.

System-Level Requirements

Monitoring the heartbeat signal while the controller is in the Observation state is a good indicator of the overall health and function of the IP. For other system-level recommendations to monitor the health and state of the SEM controller IP, see Systems.

Command Interface

The Command Interface provides a simple interface if you want to transition to the Idle or Observation states, perform error injections or software resets. The Command Interface consists of an input command, input strobe, and an output busy signal implementing a simple parallel input port with a busy output. This interface is optional. If not used, all inputs can be tied Low.

This interface accepts the following types of commands:

- Command to enter Idle state (suspend normal scanning)
- Command to enter Observation state (resume normal scanning)
- Command to perform a software reset (reboot and initialize)
- Inject error at a frame address

To execute the commands, you must monitor the Status Interface as some of the commands only execute if the IP is in Observation or Idle state.

Note that each command generated through the Command Interface is reflected and reported on the Monitor Interface, the comprehensive interface for user interaction with the IP.

For SSI implementations of the system-level example design, there is a controller instance on each SLR with all controller instances receiving the signals from the Command Interface. In many cases, signals from the Command Interface can be brought to I/O pins on the FPGA.

This interface can be driven both from internal to the FPGA (for example connected to the Vivado Lab Edition analyzer) or connected to another device for control through I/O pins. In the latter case, the timing requirements of the Command Interface must be accounted for to properly capture supplied commands when interfacing to another device. See Switching Behavior.

Commands

Commands are presented by applying a value to the `command_code` bus, and then pulsing the `command_strobe` signal. After a command is presented, the `command_busy` signal asserts and stays asserted until the command has been processed and executed. Do not present another command until the `command_busy` signal deasserts. Table 3-2 lists all the commands and how to generate it. The following sections describe the commands in detail.

Table 3-2: **Command Format and Usage**

Command	Command_code[39:0] Format
Directed State Change to Idle	`1110 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Observation state. Valid for mitigation modes only.
Error Injection Using LFA	`1100 0000 0ssl llll llll llll llll wwww wwwb bbbb` **Binary Value** — **Equals to** `ss` — Hardware slr number (2-bit) Valid range: 0..2 `lllllllllllllllll` — Linear frame address (17-bit) Valid range: 0..Max Frame `wwwwwww` — Word address (7-bit) Valid range: 0..122 `bbbbb` — Bit address (5-bit) Valid range: 0..31 Valid when controller in Idle state. Valid for Mitigation and testing or Emulation modes only.
Directed State Change to Observation	`1010 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Idle state. Valid for mitigation modes only.
Software Reset	`1011 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Idle state.
Error Injection Using PFA	`0sst trrr rrrc cccc cccc cmmm mmmm wwww wwwb bbbb` **Binary Value** — **Equals to** `ss` — Hardware slr number (2-bit) `tt` — Block type (2-bit) `rrrrrr` — Row address (6-bit) `cccccccccc` — Column address (10-bit) `mmmmmmm` — Minor address (7-bit) `wwwwwww` — Word address (7-bit) `bbbbb` — Bit address (5-bit) Valid when controller in Idle state. Valid for Mitigation and testing or Emulation modes only.

Directed State Change to Idle

This command is only effective if the IP is in the Observation state. Monitor the `status_observation` signal before executing this command.

Error injection Using Linear Frame Addressing

This command is only effective if the IP is in Idle state. If the IP is not in the Idle state, as indicated by the `status_*` signals being deasserted (Low), use the Directed State Change to Idle command to transition the IP to the Idle state. In the Idle state, this command can be executed.

Software Reset

This command is only effective if the IP is in Idle state. If the IP is not in the Idle state, as indicated by the `status_*` signals being deasserted (Low), use the Directed State Change to Idle command to transition the IP to the Idle state. In the Idle state, this command can be executed.

Directed State Change to Observation

This command is only effective if the IP is in Idle state. It sends the IP to the Observation state to continue the scanning of the configuration memories and soft error detection, correction, and classification.

Error Injection Using Physical Frame Addressing

This command is only effective if the IP is in Idle state. If the IP is not in the Idle state, as indicated by the `status_*` signals being deasserted (Low), use the Directed State Change to Idle command to transition the IP to the Idle state. In the Idle state, this command can be executed.

Switching Behavior

The signals in the Command Interface are received by a sequential logic process in controllers using the strobe to enable an Input register. The timing requirements shown in Figure 3-10 must be observed to ensure successful data capture. The `command_strobe` is synchronous to the `icap_clk` and has to be a one clock cycle pulse for each command issued.

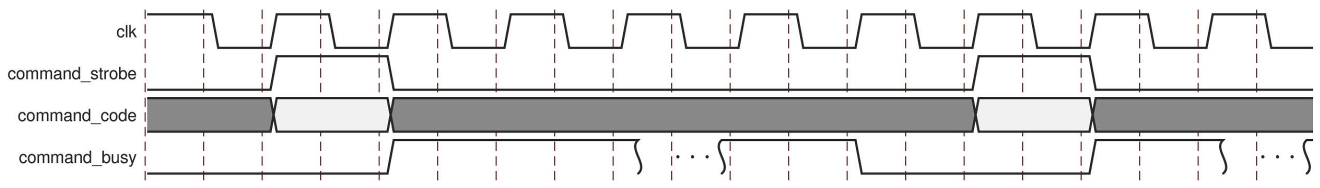


Figure 3-10: **Command Interface Switching Behavior**

If an error is injected into a frame that is masked or beyond the supported address range for a device or SEU coverage, the error injection command is ignored and no error is detected in the Observation state.

If an invalid command is received, the IP ignores the command.

System-Level Requirements

There is no system-level requirements for this interface.

Monitor Interface

The Monitor Interface provides a mechanism to interact with the controller that also provides a comprehensive view into the controller behavior and current state. This interface supports commands and information that is a superset of the Command and Status Interface combined.

The controller is designed to read commands and write status information to this interface as ASCII strings. See Table 2-18 for the port list and definition of this interface.

Although you can use the Status Interface to monitor the state, state transitions, and errors detected by the IP, the Monitor Interface provides more detailed insight of the IP state and information of the errors encountered by the IP (including the configuration frame address). For a list of state and state transitions of the controller, see Status Interface.

In terms of commands with the addition to the directed state changes and error injection commands, this interface also provides the mechanism to query the content of Configuration registers and frames, translate physical configuration frame addresses to linear frame addresses (and vice-versa), and the ability to read the external memory connected to the design. See Table 3-1 to compare the commands available on the Command and Monitor Interfaces, respectively.

There are two main use cases for this interface:

- Monitoring and debug
- Testing the controller and user design

The following section describes these use cases.

Monitoring and Debug

RECOMMENDED: *This interface provides the most insight into the IP state and behavior. Xilinx recommends that at the minimum, connect this interface to a FIFO to store the output of the interface because it provides information that is crucial to the understanding of what is taking place in your design and often necessary if you submit a support case to Xilinx technical support.*

The size of the FIFO varies based on what you intend to do with the FIFO. If the goal is to use the FIFO data for debugging purposes and it is not periodically retrieved, Xilinx recommends that the FIFO should at least be able to capture a full initialization report and two full error reports. This requires at the minimum a FIFO that is 512 x 8 in size.

If you aim to retrieve this information periodically, the size of the FIFO depends on how often data is retrieved. In this use case, you can tie off the inputs of the interface in the following manner to disable any write commands to the controller:

```
monitor_txfull = 0 or to the FIFO full flag
monitor_rxdata = to all 0s
monitor_rxempty = 1
```

Note that the monitor port provides more details of the errors it detects including its address location and timestamp. This additional error information is useful to log the SEU events that occur in a system.

Testing Controller and User Design

As a convenience, the system-level example design provides a UART helper block to connect to the Monitor Interface to ease integration of this interface to a processor block. You can use this UART Interface to receive status information from the IP, send commands to inject errors to the IP, and confirm that the error injected is detected and corrected by the IP.

The following section discusses the UART Interface behavior.

UART Interface

The UART Interface is a UART helper block that serializes and de-serializes the byte-stream ASCII codes used by the Monitor Interface. The helper block uses a standard serial communication protocol and is compatible to a standard RS-232 port, or to USB through a USB-to-UART bridge. For more information, see UART Interface in Chapter 2.

The following section describes the different messages and commands available on this interface.

UART Interface Messages

The UART Interface messages define what messages you can expect from the controller through the Monitor Interface. This message set is intended to offer a superset of the "reporting" available from the Status Interface.

Initialization Report

As the controller performs the initialization sequence, it generates the initialization report. This report contains diagnostic information and is generated when the controller first starts and at subsequent software resets.

```
SEM_ULTRA_V2_0            Name and version
SC 01                     State transition to Initialization state
FS {2 digit hex value}    Core Configuration Information
AF {2 digit hex value}    Additional Core Configuration Information
ICAP OK                   Status: ICAP Available
```

```
RDBK OK                     Status: Readback Active
INIT OK                     Status: Completed Setup
SC {00, 02}                 State transition to Idle or Observation state
```

Command Prompt

The command prompt issued by the controller is one of two characters, depending on the controller state. If the controller is in the Observation state (the default state after initialization completes for all mitigation modes) the prompt issued is O>. If the controller is in the Idle state (the default state after initialization for Monitoring only or Emulation mode), the prompt issued is I>.

State Change Report

Any time the controller changes state, the controller also issues a state change report. The report is a single line with the following format:

```
SC {2-digit hex value}
```

The 2-digit hex value is the representation of the Status Interface outputs.

Table 3-3: **State Change Report Decoding**

Report String	State Name
SC 00	Idle
SC 01	Initialization
SC 02	Observation
SC 04	Correction
SC 08	Classification
SC 10	Injection
SC 1F	Fatal Error

Entry into the Fatal Error state can occur at anytime, even without an explicit state change report. Upon entering this state, the controller issues the following fatal error message:

```
HLT
```

Flag Change Report

Any time the controller changes flags, the controller also issues a flag change report. The report is a single line with the following format:

```
FC {2-digit hex value}
```

The 2-digit hex value is the representation of the Status Interface outputs.

Table 3-4: **Flag Change Report Decoding**

Report String	Condition Name
FC 00	Correctable, Non-Essential
FC 20	Uncorrectable, Non-Essential
FC 40	Correctable, Essential
FC 60	Uncorrectable, Essential

Error Detection Report

Upon detection of an error condition, the controller corrects the error as quickly as possible. Therefore, the report information is actually generated after the correction has taken place, assuming it is possible to correct the error. The following scenarios exist:

Diagnosis: CRC error only [cannot identify location or number of bits in error]

```
SC 04                     State Transition to Correction state
CRC                       CRC error detected
TS {8-digit hex value} Timestamp
```

Diagnosis: ECC-based error – Uncorrectable

```
SC 04                     State Transition to Correction state
ECC                       ECC Error Detected
TS {8-digit hex value} Timestamp
PA {7-digit hex value} PFA of Detected Error
LA {7-digit hex value} LFA of Detected Error
```

Diagnosis: ECC-based error – Correctable

```
SC 04                     State Transition to Correction state
ECC                       ECC error detected
TS {8-digit hex value} Timestamp
PA {7-digit hex value} PFA of Detected Error
LA {7-digit hex value} LFA of Detected Error
```

Diagnosis: Auxiliary-based error (error detected from auxiliary input)

```
SC 04                     State Transition to Correction state
AUX                       AUX Error
TS {8-digit hex value} Timestamp
```

Error Correction Report

The error correction process varies depending on the controller configuration and the nature of what has been detected and what can be corrected.

The general form of the report for an uncorrectable error, or when correction is disabled is:

```
COR
END
```

Followed by:

```
FC 20         Bit 5, uncorrectable set (stale essential flag)
```

or

```
FC 60         Bit 5, uncorrectable set (stale essential flag)
```

The general form of the report for a correctable error is:

```
COR
{correction list}
END
```

Followed by:

```
FC 00         Bit 5, uncorrectable cleared (stale essential flag)
```

or

```
FC 40         Bit 5, uncorrectable cleared (stale essential flag)
```

The {correction list} is one or more lines providing the word in frame and bit in word of each corrected bit. The list can potentially be thousands of lines. This is the same notation used for the error detection report. Each line of the list is formatted as follows:

```
WD {2-digit hex value} BT {2-digit hex value}
```

Error Classification Report

The error classification process involves looking up each of the errors in a frame to determine if any of them are essential. If one or more are identified as essential, the entire event is considered essential.

With error classification enabled, the general form of the report for a correctable, non-essential event is:

```
SC 08
CLA
END
FC 00          Bit 6, essential is cleared
```

With error classification enabled, the general form of the report for a correctable, essential event is:

```
SC 08
CLA
{classification list}
END
```

```
FC 40          Bit 6, essential is set
```

The {classification list} is one or more lines providing the word in frame and bit in word of each essential bit. The list can potentially be thousands of lines. This is the same notation used for the error detection report. Each line of the list is formatted as follows:

```
WD {2-digit hex value} BT {2-digit hex value}
```

With error classification disabled, no detailed classification list is generated. All errors must be considered essential because the controller has no basis to indicate otherwise. The general form of the report for a correctable event is:

```
SC 08
FC 40          Bit 6, essential is set
```

All uncorrectable errors must be considered essential because the controller has no basis to indicate otherwise. The general form of the report for an uncorrectable event is:

```
SC 08
FC 60          Bit 6, essential is set
```

Status Report

A status report provides more information about the controller state. It is a multiple-line report that is generated in response to the "S" command, provided the controller is in the Observation or Idle states.

The length of the status report varies depending on what state the IP is in. When the core is in Idle state, a complete status report is given. An abbreviated report is given in the Observation state. The non-SSI device status report during Idle state has the following format:

```
SN {2-digit hex value} SLR Number
SC {2-digit hex value} Current State
FC {2-digit hex value} Current Flags
RI {2-digit hex value} Reserved information
MF {8-digit hex value} Maximum Linear Frame Count
TS {8-digit hex value} Timestamp
TB {8-digit hex value} Table Base (valid when error classification is enabled;
otherwise X's)
CB {8-digit hex value} Classification Base (valid when error classification is
enabled; otherwise X's)
CL {3-digit hex value} Classification Level
```

The non-SSI device status report during the Observation state omits the Maximum Linear Frame Count (MF), Timestamp (TS), Table Base (TB), Classification Base (CB), and Classification Level (CL).

The SSI device status report is similar to the non-SSI device status report, but with a sub-report per SLR. The sub-reports are sorted by hardware SLR number.

For example, a device with three SLRs generates a report in this format:

```
SN {2-digit hex value} SLR Number
SC {2-digit hex value} Current State
FC {2-digit hex value} Current Flags
RI {2-digit hex value} Reserved information
MF {8-digit hex value} Maximum Linear Frame Count
TS {8-digit hex value} Timestamp
TB {8-digit hex value} Table Base (valid when classification is enabled; otherwise
X's)
CB {8-digit hex value} Classification Base (valid when classification is enabled;
otherwise X's)
CL {3-digit hex value} Classification Level
SN {2-digit hex value} SLR Number
SC {2-digit hex value} Current State
FC {2-digit hex value} Current Flags
RI {2-digit hex value} Reserved information
MF {8-digit hex value} Maximum Linear Frame Count
TS {8-digit hex value} Timestamp
TB {8-digit hex value} Table Base (valid when classification is enabled; otherwise
X's)
CB {8-digit hex value} Classification Base (valid when classification is enabled;
otherwise X's)
CL {3-digit hex value} Classification Level
SN {2-digit hex value} SLR Number
SC {2-digit hex value} Current State
FC {2-digit hex value} Current Flags
RI {2-digit hex value} Reserved information
MF {8-digit hex value} Maximum Linear Frame Count
TS {8-digit hex value} Timestamp
TB {8-digit hex value} Table Base (valid when classification is enabled; otherwise
X's)
CB {8-digit hex value} Classification Base (valid when classification is enabled;
otherwise X's)
CL {3-digit hex value} Classification Level
```

UART Interface Commands

The Monitor Interface commands define what you can send to the controller through the Monitor Interface. This command set is intended to offer a superset of the “command capability” available from the Command Interface.

Table 3-5: **UART Commands and Usage**

Command	UART Command Set
Directed State Changes	“O” = Enter Observation. Valid in Idle state. “I” = Enter Idle. Valid in Observation state. Valid for mitigation modes only.
Status Report	“S” Valid in Idle and Observation state.

Table 3-5: **UART Commands and Usage *(Cont'd)***

Command	UART Command Set
Error Injection Using LFA	"N {10-digit hex value}" Binary value = `1100 0000 0ssl 1111 1111 1111 1111 wwww wwwb bbbb` **Binary Value** — **Equals to** `ss` — Hardware slr number (2-bit) Valid range: 0..2 `11111111111111111` — Linear frame address (17-bit) Valid range: 0..Max Frame `wwwwwww` — Word address (7-bit) Valid range: 0..122 `bbbbb` — Bit address (5-bit) Valid range: 0..31 Valid in Idle state. Valid for Mitigation and testing or Emulation modes only.
Configuration Frames Reads (`Query` command)	"Q {10-digit hex value}" 10-digit hex value can be either the PFA or LFA address format used in the error injection command. Valid in Idle state.
Configuration Register Reads (`Peek` command)	"P {2-digit hex value}" Binary value = `0ssr rrrr` **Binary Value** — **Equals to** `ss` — Hardware slr (2-bit) `rrrrr` — Register address (5-bit) See Configuration Register Reads (`Peek` Command), page 58. Valid in Idle state.
External Memory Reads (`Xmem` command)	"X {8-digit hex value)" 8-digit hex value or 32-bit value is the address used to read a byte from the external memory. Valid in Idle state when Error Classification is available.
Software Reset	"R `xxxx xxxx`" `xxxx xxxx` = Don't care Valid in Idle state.

Table 3-5: **UART Commands and Usage *(Cont'd)***

Command	UART Command Set
Frame Address Translation LFA <-> PFA (`Translate` command)	"T {10-digit hex value}" 10-digit hex value can be either the PFA or LFA address format used in the error injection command. Valid in Idle state.
Error Injection Using PFA	"N {10-digit hex value}" Binary value = `0sst trrr rrrc cccc cccc cmmm mmmm wwww wwwb bbbb` **Binary Value** / **Equals to**: `ss` – Hardware slr number (2-bit) `tt` – Block type (2-bit) `rrrrrr` – Row address (6-bit) `cccccccccc` – Column address (10-bit) `mmmmmmm` – Minor address (7-bit) `wwwwwww` – Word address (7-bit) `bbbbb` – Bit address (5-bit) Valid in Idle state. Valid for Mitigation and testing or Emulation modes only.

Nested table (Error Injection Using PFA):

Binary Value	Equals to
`ss`	Hardware slr number (2-bit)
`tt`	Block type (2-bit)
`rrrrrr`	Row address (6-bit)
`cccccccccc`	Column address (10-bit)
`mmmmmmm`	Minor address (7-bit)
`wwwwwww`	Word address (7-bit)
`bbbbb`	Bit address (5-bit)

Directed State Changes

The controller can be moved between Observation and Idle states by a directed state change. "I" command is used to enter Idle state. "O" command is used to enter Observation state.

Configuration Frame Reads (`Query` Command)

The Configuration Frame Read command is used to read the contents of Configuration Frames. The hex value supplied with this command represents the same address value used for error injections in Table 3-5. The format of this command is:

```
Q {10-bit hex value}
```

The ability to read the configuration memory content is especially useful to debug the behavior of the IP when error injections are performed.

RECOMMENDED: *Xilinx recommends reading the configuration memory content before and after injecting errors. This is to confirm that the error injection is successful in altering the configuration memory and whether the IP detects and corrects the error injected.*

Configuration Register Reads (`Peek` Command)

The Configuration Register Read command reads certain configuration registers and report its content. The format of the command is:

```
P {2-digit hex value}
```

Table 3-6 lists all the registers that the SEM controller support reads from.

Table 3-6: **Configuration Registers Readable Through `Peek` Command**

Name	Address
CRC	00000
FAR	00001
CMD	00100
CTL0	00101
MASK	00110
STAT	00111
COR0	01001
IDCODE	01100
AXSS	01101
COR1	01110
WBSTAR	10000
TIMER	10001
BOOTSTS	10110

For more information on this register, see the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 2].

External Memory Reads (`Xmem` Command)

The External Memory Read command is used to read the contents of an external memory. One byte of data is returned for each address read. The format of this command is:

```
X {8-bit hex value}
```

Error Injection

"N" command is used to perform an error injection by either LFA or PFA. The controller only accepts this command when in the Idle state. The format of the command is:

```
N {10-digit hex value}
```

Issuing this command is analogous to presenting an error injection command on the Command Interface. The hex value supplied with this command depends on if a PFA or LFA injection will be performed. See Table 3-5.

Status Report

"S" command is used to request a status report from the controller. The status report format is detailed in the previous section which describes status messages generated by the controller. The controller accepts this command when in Idle or Observation states.

Frame Address Translation (`Translate` Command)

This command is a useful debugging command to translate LFA addresses to PFA addresses and vice-versa. The hex value supplied with this command represents the same value used for error injections in Table 3-5. The format of the command is:

```
T {10-digit hex value}
```

Software Reset

"R" command is used to perform a software reset. The controller only accepts this command when in the Idle state. The format of the command is:

```
R {2-digit hex value}
```

For details on issuing this command, see Table 3-5.

Switching Behavior

The Monitor Interface consists of two signals implementing an RS-232 protocol compatible, full duplex serial port for exchange of commands and status. The following configuration is used:

- **Baud**: 115200
- **Settings**: 8-N-1
- **Flow Control**: None
- **Terminal Setup**: VT100
  - **TX Newline**: CR (Terminal transmits CR [0x0D] as end of line)
  - **RX Newline**: CR+LF (Terminal receives CR [0x0D] as end of line, and expands to CR+LF [0x0D, 0x0A])
  - **Local Echo**: NO

Any external device connected to the UART Interface must support this configuration. Figure 3-11 shows the switching behavior, and is representative of both transmit and receive.

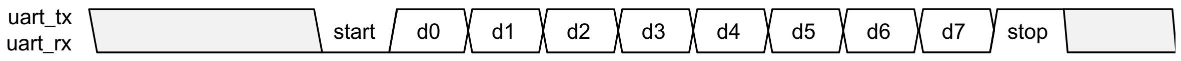


Figure 3-11: **UART Interface Switching Characteristics**

Transmit and receive timing is derived from a 16x bit rate enable signal which is created inside the system-level example design using a counter. The behavior of this counter is to start counting from zero, up to and including a terminal count (a condition detected and used to synchronously reset the counter). The terminal count output is also supplied to transmit and receive processes as a time base.

From a compatibility perspective, the advantages of 115200 baud are that it is a standard bit rate, and it is also realizable with a broad range of input clock frequencies. It is used in the system-level design example for these reasons.

From a practical perspective, the baud rate should be selected to meet the desired communication performance (both data rate and latency). This performance can throttle the controller. For this reason, use of a high bit rate is strongly encouraged. A wide variety of other bit rates are possible, including standard bit rates: 9600, 230400, 460800, and 921600 baud.

In the UART helper block system-level example design module, the parameter V_ENABLETIME sets the communication bit rate. The value for V_ENABLETIME is calculated using:

$$\text{V\_ENABLETIME} = \text{round to integer}\left[\frac{\mathit{input\ clock\ frequency}}{16 \times \mathit{nominal\ \ bitrate}}\right] - 1 \qquad \textit{Equation 3-1}$$

A rounding error as great as ±0.5 can result from the computation of V_ENABLETIME. This error produces a bit rate that is slightly different than the nominal bit rate. A difference of 2% between RS-232 devices is considered acceptable, which suggests a bit rate tolerance of ±1% for each device.

Example: The input clock is 66 MHz, and the desired bit rate is 115200 baud.

$$\text{V\_ENABLETIME} = \text{round to integer}\left[\frac{66000000}{16 \times 115200}\right] - 1 = 35 \qquad \textit{Equation 3-2}$$

The actual bit rate that results is approximately 114583 baud, which deviates -0.54% from the nominal bit rate of 115200 baud. This is acceptable because the difference is within ±1%.

When exploring bit rates, if the difference from nominal exceeds the ±1% tolerance, select another combination of bit rate and input clock frequency that yields less error. No additional switching characteristics are specified.

Electrically, the I/O pins used by the UART Interface use LVCMOS signaling, which is suitable for interfacing with other devices. No specific I/O mode is required. When full electrical compatibility with RS-232 is desired, an external level translator must be used.

System-Level Requirements

No further system-level requirements are required beyond what has been discussed.

Fetch Interface

The Fetch Interface provides a mechanism for the controller to request data from an external source. This interface is only present when the error classification feature is enabled. See Table 2-20, page 29 for the port list and definition of this interface.

As a convenience, the system-level example design provides an example SPI flash master helper block to connect to the Fetch Interface to retrieve the data from an external SPI flash. The following section discusses the behavior of the SPI flash master helper block in the SPI Interface.

SPI Interface

The SPI Interface consists of four signals implementing a SPI bus protocol compatible, full duplex serial port. This interface is only present when the Error Classification feature is enabled. The implementation of this function requires external storage. The system-level design example provides a fixed-function SPI bus master in the SPI flash master helper block to fetch data from a single external SPI flash device. Table 3-7 provides the SPI flash density requirement for each supported FPGA.

Table 3-7: **External Storage Requirements**

Device		Storage Requirements for Error Classification
UltraScale	XCKU035	TBD
	XCKU040	TBD
	XCKU060	TBD
	XCKU075	TBD
	XCKU100	TBD
	XCKU115	TBD
	XCVU065	TBD
	XCVU080	TBD
	XCVU095	TBD
	XCVU125	TBD
	XCVU160	TBD
	XCVU190	TBD
	XCVU440	TBD

The SPI flash master helper block uses the fast read command (0x0B) and can be configured to support one of several different families of SPI flash. The family supported by default depends on the external storage requirements shown in Table 3-7. In the SPI flash master helper block system-level example design module, there are three parameters that control the command sequence sent to the SPI flash device.

- **B_ISSUE_WREN** – Indicates if a write enable command (0x06) must be issued prior to any other commands that modify the device behavior. Must be set to "1" for N25Q devices, but generally set to "0" for other devices.
- **B_ISSUE_WVCR** – Indicates if a write volatile configuration register command (0x81) must be issued to explicitly set the fast read dummy cycle count to eight cycles. The state machine in the SPI flash master helper block is byte-oriented and expects the fast read dummy cycle count to be eight. The volatile configuration register data is overwritten (0x8B). Must be set to "1" for N25Q devices, but generally set to "0" for other devices.
- **B_ISSUE_EN4B** – Indicates if an enable 4-byte addressing command (0xB7) must be issued to explicitly enter the four-byte addressing mode. Must be set to "1" for devices > 128 Mbits.

For storage requirements ≤ to 128 Mbits, the SPI flash master helper block supports M25P devices by default (B_ISSUE_WREN = 0, B_ISSUE_WVCR = 0, B_ISSUE_EN4B = 0). These devices are not capable of 4-byte addressing mode.

For storage requirements > 128 Mbits, the SPI flash master helper block supports higher-density N25Q devices by default (B_ISSUE_WREN = 1, B_ISSUE_WVCR = 1, B_ISSUE_EN4B = 1). These devices are capable of 4-byte addressing mode.

Other supported devices include lower-density N25Q devices for storage requirements ≤ 128 Mbits (B_ISSUE_WREN = 1, B_ISSUE_WVCR = 1, B_ISSUE_EN4B = 0) and higher-density MX25 devices for storage requirements > 128 Mbits (B_ISSUE_WREN = 0, B_ISSUE_WVCR = 0, B_ISSUE_EN4B = 1).

Note: The SPI flash master helper block implementation supports only one SPI flash read command (fast read) in SPI Mode 0 (CPOL = 0, CPHA = 0) to a single SPI flash device.

Figure 3-12 shows the connectivity between an FPGA and SPI flash device. Note the presence of level translators (marked "LT"). These are required because commonly available SPI flash devices use 3.3V I/O, which might not be available depending on the selected FPGA or I/O bank voltage.

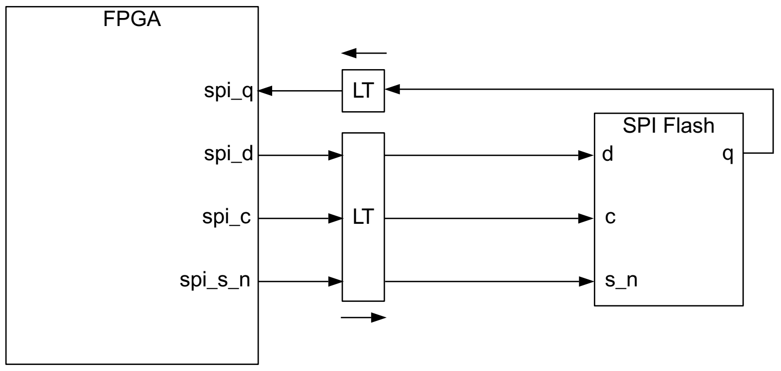


Figure 3-12: **SPI Flash Device Connection, Including Level Translators**

The level translators must exhibit low propagation delay to maximize the SPI bus performance. The SPI bus performance can potentially affect the maximum frequency of operation of the entire system-level design example.

Note: Information on the switching behavior, system-level requirements, and example SPI bus timing budgets are added in the next revision of the document. If you want information prior to this update, contact Xilinx.

Data Consistency

When using the error classification feature, the controller requires access to externally stored data. This data is created by `write_bitstream` at the same time the programming file for the FPGA is created.

Any time the FPGA design is changed and a new programming file is created, the additional data files used by the controller must also be updated. When the hardware design is updated with the new programming file, the externally stored data must also be updated.

IMPORTANT: *Failure to maintain data consistency can result in unpredictable controller behavior. Xilinx recommends use of an update methodology which ensures that the programming file and the additional data files are always synchronized.*

Auxiliary Interface

This interface provides a mechanism to notify the controller of soft error events that take place in areas not directly observable to the controller through the scanning of the configuration memory. For example, if the design uses BRAM ECC function, the errors detected by these function can be inputs to this interface. Errors notified through this interface are detected when the IP is in the Observation state and affects both the behavior of the Monitor and Status Interfaces. Each reported error should only be pulsed for 1 clock cycle to avoid the same error reported multiple times.

If this interface is not used, tie off the interface inputs to Low.

Switching Behavior

The switching characteristics is illustrated in Figure 3-13. In the first section of the waveform, a correctable essential error is reported through the Auxiliary Interface causing the `status_essential` signal to be asserted High.

This causes the IP to report the error to the Monitor Interface in the following manner:

```
aux_error_cr_es
  SC 04
  AUX
  TS {8-digit hex value}
  FC 00
  SC 08
```

```
    FC 40
    SC 02
  aux_error_cr_ne
    SC 04
    AUX
    TS {8-digit hex value}
    FC 40
    SC 08
    FC 00
    SC 02
```

Note that the IP only reports these errors when it is in the Observation state.

In the second section of the waveform, an uncorrectable error is reported through the Auxiliary Interface. This causes the IP to go to Idle, behaving the same way as if an uncorrectable error is detected by the controller. Error state is reflected in both the Monitor and Status Interfaces. This causes the IP to report the error to the Monitor Interface in the following manner:

```
  aux_error_uc
    SC 04
    AUX
    TS {8-digit hex value}
    FC 20
    SC 08
    FC 60
    SC 00
```

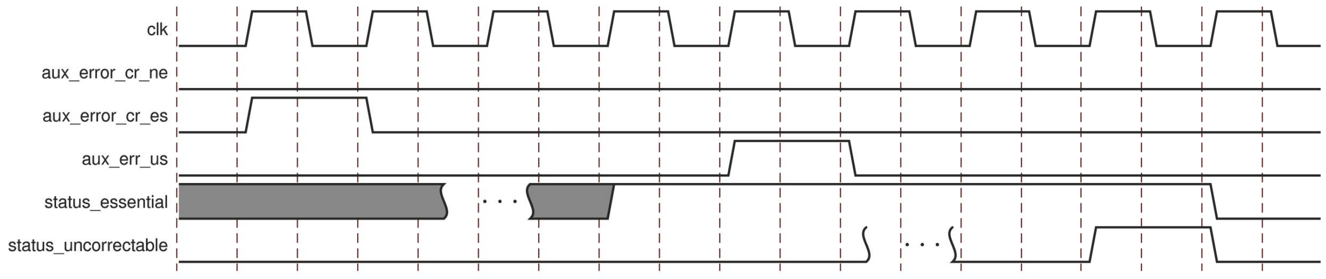


Figure 3-13: **Switching Behavior for Auxiliary Interface**

System-Level Requirements

This interface should be used if the SEM controller is used to combine all reporting of soft error events in the device and its Status and Monitor Interface is used by the overall system to take action on errors. Alternatively, you can create this type of function external to the IP and manage the system reaction to these errors independently.

Systems

Although the Soft Error Mitigation solution can operate autonomously, many applications of this solution are used with a system-level supervisory function. The decision to implement a system-level supervisory function and the scope of the responsibilities of this function are system-specific.

Note: The recommendations below also includes system-level recommendations described in each interface.

The following points illustrate methods by which a system-level supervisory function can monitor the Soft Error Mitigation solution.

- Monitor the Soft Error Mitigation solution to determine if additional system-level actions are necessary in response to a soft error event. This action can be as simple as logging each soft error event that is detected, or it might involve a more complex determination of the appropriate system-level response based on factors such as the classification value of the error or whether the error is correctable. Analysis of these and other factors could result in system-level actions including, but not limited to, resetting the design, reconfiguring the FPGA, or rebooting the system.

  To monitor the Soft Error Mitigation solution event reporting, use the Status Interface `status_correction` and `status_uncorrectable` signals, the Status Interface `status_classification` and `status_essential` signals, or the Monitor Interface `monitor_tx` signal for error detection, correction, and classification reports.

- Monitor the Soft Error Mitigation solution to confirm it is healthy. As discussed and quantified in the Solution Reliability in Chapter 2, there is a very small possibility of failure of the Soft Error Mitigation solution. Statistically, such failures might occur during any state of the controller:

  - **Boot and Initialization States** – Monitor the Soft Error Mitigation solution to confirm it boots, initializes, and enters the Observation state. Xilinx specifies the Soft Error Mitigation solution will boot, initialize, and enter the Observation state within the time specified through Table 2-4 and Equation 2-1, provided that the `cap_gnt` signal is asserted, the FPGA configuration logic is available to the Soft Error Mitigation solution through the ICAP primitive, and there is no throttling on the Monitor Interface.

    Reasons the Soft Error Mitigation solution could fail to initialize and/or fail to enter the Observation state are usually design errors (versus soft error events) and include incorrect control of the `cap_gnt` signal, incorrect implementation of ICAP sharing, and general unavailability of the FPGA configuration logic to the Soft Error Mitigation solution through the ICAP primitive. This last issue can occur for several reasons, ranging from use of bitstream options documented to be incompatible with the Soft Error Mitigation solution, to the failure of a system-level JTAG controller to properly complete and/or clear FPGA configuration instructions issued through JTAG to the FPGA.

To confirm the solution initializes and enters the Observation state, the system-level supervisory function can observe the Status Interface `status_initialization` and `status_observation` signals for assertion (see state diagrams Figure 3-4 and Figure 3-5), or the Monitor Interface `monitor_tx` signal for the expected initialization report.

- **Observation State** – The controller spends virtually all of its time in this state. There are at least three methods for monitoring the controller in this state, each provides slightly different information about the health of the controller:

  - **Controller Heartbeat, `status_heartbeat`** – This signal is a direct output from the Soft Error Mitigation solution. This signal exhibits pulses, specified in the Status Interface, which indicate the readback process is active. If, during the Observation state, these pulses become out-of-specification, the system-level supervisory function should conclude that the readback process has experienced a fault. This condition is an uncorrectable, essential error. In SSI implementations, which have a `status_heartbeat` output per SLR, it is necessary to monitor the heartbeat from all SLRs.

    Note that `status_heartbeat` is undefined in other controller states and should only be observed during the Observation state.

  - **CRC Failure Indicator, `INIT_B`** – This signal is a direct output from the readback process. If the readback process detects a CRC failure, it asserts `INIT_B`. If, during the Observation state, `INIT_B` indicates an error and the controller does not respond with a state transition to correction within one second, then the controller has experienced a fault. State transition can be determined using the Status Interface `status_correction` signal or the Monitor Interface state change report. This condition is an uncorrectable, essential error. In SSI implementations, which have an internal CRC failure indicator per SLR, the indicators are wire-ORed to form the single `INIT_B` device pin, but the Status Interface has a `status_correction` signal for each SLR.

    Note that the CRC failure indicator, `INIT_B`, is undefined in other controller states and should only be observed during the Observation state.

  - **Controller Status Command and Report** – Using the Monitor Interface `monitor_rx` and `monitor_tx` signals, the system-level supervisory function can periodically transmit a status command and confirm receipt of the expected status report. Provided the controller has not changed state, the system-level supervisory function should conclude that the controller has experienced a fault if the expected status report is not received within one second. This condition is an uncorrectable, essential error.

In the use of this method, care should be taken to select the lowest frequency of status command transmission that yields acceptable detection time of a "controller unresponsive" condition.

Status command and report processing by the controller can be an undesirable source of additional latency. For example, a status command transmission period of 60 seconds might be a reasonable trade-off to guard against rare "controller unresponsive" conditions while not adding significant additional latency to general operation. As a counter example, one second would be a poor choice. In this counter example, the status reports could keep the UART helper block transmit buffer frequently non-empty, possibly resulting in throttling on the Monitor Interface, adding latency to error detection, correction, and classification activities.

Note that the controller status command and report method only functions in the Observation and Idle states. Assuming the UART helper block receive buffer is not in an overflow condition, status commands sent during other states are buffered and processed upon return to the Observation or Idle state.

- **Correction and Classification States** – The Soft Error Mitigation solution transitions through the Correction and Classification states within the time specified in Table 2-6/Equation 2-3 and Table 2-7/Equation 2-4, provided there is no throttling on the Monitor Interface. Due to the infrequency of soft errors, the controller spends very little time in these states and normally transitions back to the Observation state, or less frequently, the Idle state. If the controller dwells continuously in either the Correction or Classification states in excess of one second, as observed on the Status Interface `status_correction` and `status_classification` signals, or on the Monitor Interface as indicated by the state change reports, then the system-level supervisory function should conclude that the controller has experienced a fault. This is an uncorrectable, essential error.

  Independently, the system-level supervisory function might elect to monitor for conditions where the Soft Error Mitigation solution repeatedly corrects the same address. Many rare issues might generate this symptom, ranging from soft errors in the controller to hard errors in the device itself.

- **Idle and Injection States** – The controller only enters the Idle state as a result of an uncorrectable error, or if specifically directed. In the event of an uncorrectable error, see the section above about monitoring event reporting. Directed entry to the Idle state is generally for the purpose of issuing other commands for error injection or ICAP sharing. It is inadvisable to implement the “Observation State” point above for status command and report monitoring during the Idle state as it might conflict with commands issued by other processes at the application level. Instead, the application-level processes should test that any issued command completes and generates a response within one second. Otherwise, an uncorrectable, essential error has occurred and the application should report this to the system.

- **Halt State** – The controller only enters this state when it has detected an inconsistent internal state. This condition is observable on the Status Interface as the assertion of all five state indicators, and on the Monitor Interface as a HLT message. In SSI implementations, where more than one controller instance exists, the solution is considered halted if one or more of the controller instances halts. This is an uncorrectable, essential error.

Even though it is optional to implement any system-level supervisory function that is described above, Xilinx recommends that at the minimum implement the following system-level supervisory function to ensure that the IP is healthy and functional when using the IP in mitigation modes:

1. Confirm that IP has completed Boot and Initialization states and successfully transitions into Observation state after device configuration as discussed in the Boot and Initialization.
2. Monitor `status_heartbeat` signal during Observation to ensure that it is within the specification as discussed in the Heartbeat.
3. Ensure that IP has NOT halted or gone to Idle when it is deployed in any Mitigation mode. If either of these states occur, the IP has stopped any mitigation activity and will no longer detect or correct any SEU that might occur. This can be done by monitoring the `status_*` signals.
4. Monitor the `INIT_B` signal when the SEM controller is in the Observation state. If `INIT_B` remains asserted for longer than one second and the controller has not transitioned to the Correction state, this is an indication that a non-correctable error has occurred or that the IP is no longer responsive to mitigate errors as discussed in the CRC Failure Indicator, `INIT_B`.
5. Buffer `monitor_tx` output into a FIFO to ease debugging of the IP behavior if required at a future point. This is recommended especially if the Monitor Interface is not used by the system. See the Monitor Interface.

Configuration Memory Masking

By design, certain configuration memory bits can change value during design operation. This is frequently the case where logic slice resources are configured to implement LUTRAM functions such as Distributed RAM or Shift Registers. It also occurs when other resource types with Dynamic Reconfiguration Ports are updated during design operation.

The memory bits associated with these resources must be masked so that they are excluded from CRC and ECC calculations to prevent false error detections. Xilinx FPGA devices implement configuration memory masking to prevent these false error detections. A global control signal, `GLUTMASK_B`, selects if masking is enabled or disabled. The controller always enables configuration memory masking.

UltraScale architecture FPGAs implement fine grain masking at a resource level. This means individual resources, when configured for dynamic operation, have their configuration memory bits masked. Only the required memory bits are masked, without impacting unrelated memory bits. The masked bits are no longer monitored by the controller.

Configuration memory reads of bits associated with masked resources return constant values (either logic one or logic zero). This prevents false error detections. Configuration memory writes to bits associated with masked resources are discarded. This prevents over-writing the contents of dynamic state elements with stale data. A side effect is that error injections into masked resources do not result in error detections.

In many cases (for example, LUTRAM functions) it is possible for the user design to implement data protection on these bits for purposes of soft error mitigation. Another approach is to modify the user design to eliminate the use of features that introduce configuration memory masking.

Resets

There is deliberately no reset for the controller because the entire configuration of the device cannot be reset. The controller is a monitor of the device configuration from the point when the device is configured until the power is removed (or it is reconfigured). The task of the SEM controller is to monitor and maintain the original configuration state and not restart from some interim (potentially erroneous) state.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. 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 with IP* (UG896) [Ref 5]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 3]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite. To customize and generate the core, locate the IP core in the Vivado IP catalog at **FPGA Features and Design** > **Soft Error Mitigation** > **UltraScale Soft Error Mitigation** and click it once to select it. Important information regarding the solution is displayed in the **Details** pane of the **Project Manager** window. Review this information before proceeding.

Double-click the IP core in the IP catalog to open the customization dialog box, shown in Figure 4-1.

Note: The screen captures in this chapter are an illustration of the Vivado Integrated Design Environment (IDE). They might not represent the most recent version.

Customization Vivado IDE Organization

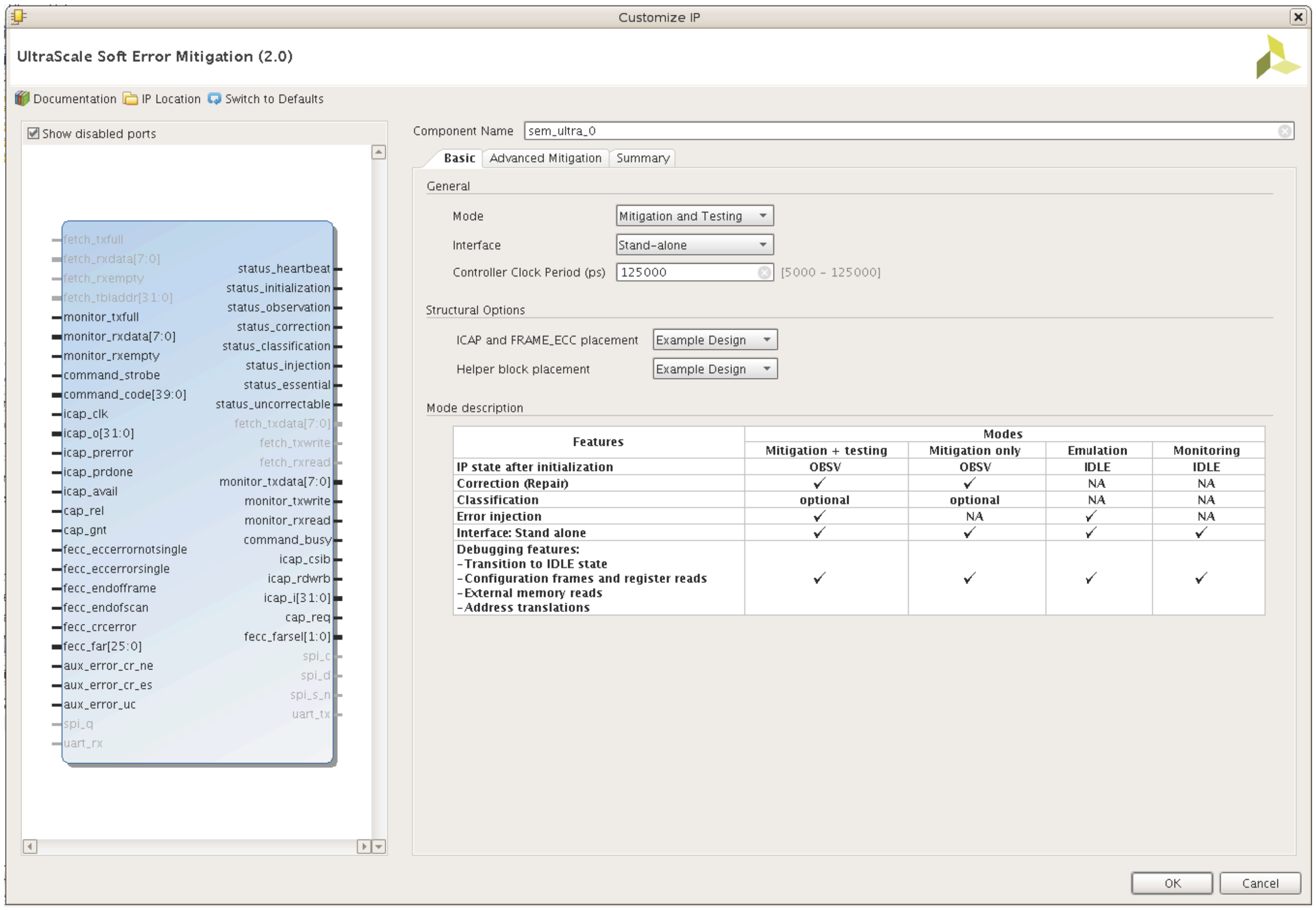
The SEM controller IP customization Vivado IDE is organized in three tabs:

- **Basic** – Provides customization options for the elementary SEU mitigation features including IP mode, interface, target clock period, and structural options for the configuration primitives and helper blocks.
- **Advanced Mitigation** – Provides customization options for more complex SEU mitigation features including error classification and reference data retrieval options.

- **Summary** – Provides a summary of the selected IP configuration specified in the **Basic** and **Advanced Mitigation** tabs. Review the selected configuration before generating the IP.

Review each of the available options, and modify them as desired so that the SEM controller solution meets the requirements of the larger project into which it will be integrated. The following subsections discuss the options in detail to serve as a guide.

Basic Tab



Component Name: sem_ultra_0

General

Mode: Mitigation and Testing

Interface: Stand-alone

Controller Clock Period (ps): 125000 [5000 – 125000]

Structural Options

ICAP and FRAME_ECC placement: Example Design

Helper block placement: Example Design

Mode description

Features	Modes			
	Mitigation + testing	Mitigation only	Emulation	Monitoring
IP state after initialization	OBSV	OBSV	IDLE	IDLE
Correction (Repair)	✓	✓	NA	NA
Classification	optional	optional	NA	NA
Error injection	✓	NA	✓	NA
Interface: Stand alone	✓	✓	✓	✓
Debugging features: -Transition to IDLE state -Configuration frames and register reads -External memory reads -Address translations	✓	✓	✓	✓

Figure 4-1: **SEM Controller Basic Tab**

Component Name and Symbol

The name of the generated component is set by the Component Name field. The name "`sem_ultra_0`" is used in this example. The Component Symbol occupies the left half of the dialog box and provides a visual indication of the ports that exist on the component, given the current option settings. This diagram is automatically updated when the option settings are modified.

Mode

The SEM controller IP is available in four modes:

- Mitigation and testing
- Mitigation only
- Emulation
- Monitoring

Pick the mode based on your usage and application of the IP. A comparison table is provided at the bottom of the GUI to help you understand the feature differences between each mode.

If error correction capability is desired, select either the Mitigation and Testing or Mitigation only modes. In these modes, the controller monitors the error detection circuits and reports error conditions. Additionally, it attempts to correct errors that are detected. In general most errors are correctable, and after successful correction, the controller signals that a correctable error has occurred and was corrected. For errors that are not correctable, the controller signals that an uncorrectable error has occurred and goes to Idle state.

The error correction method uses the ECC syndrome to identify the exact location of the error in a frame. The frame containing the error is read, the relevant bit inverted, and the frame is written back. This is signaled as correctable.

If the built-in error correction capability is not desired, select either the Emulation or Monitoring modes. In these modes, the IP is in the Idle state after initialization completes and you will not be able to enable continuous scan of the configuration memory to detect and correct SEU errors (cannot transition to Observation state).

Error injection is a design verification function that provides a mechanism for you to create errors in Configuration Memory that model a soft error event. This is useful during integration or system-level testing to verify that the controller has been properly interfaced with system supervisor y logic and that the system responds as desired when a soft error event occurs. The error injection feature is not available in the Mitigation only or Monitoring modes.

TIP: *The Monitor Interface continues to exist even if the Error Injection feature is disabled. If error injection commands are applied to the Monitor Interface, the controller parses the commands but other wise ignores them.*

For a summary of features available in each mode, see Table 2-1, page 12.

Interface

The Interface drop-down option is used to define the type of interface available on the IP user ports. Currently, the only option is a standalone interface which is backwards compatible to the SEM controller in prior architectures.

Controller Clock Period

The controller clock period is set by the Clock Period field. The error mitigation time decreases as the controller clock period decreases or as frequency increases. Therefore, the clock period should be as small as practical. The dialog box warns if the desired period exceeds the capability of the target device.

For designs that require a data retrieval interface to fetch external data for error classification, an additional consideration exists. The example design implements an external memory interface that is synchronous to the controller. The controller clock frequency therefore also determines the external memory cycle time. The external memory system must be analyzed to determine its minimum cycle time, as it can limit the maximum controller clock frequency.

Instructions on how to perform this analysis are located in Interfaces in Chapter 3. However, this analysis requires timing data from implementation results. Therefore, Xilinx recommends the following:

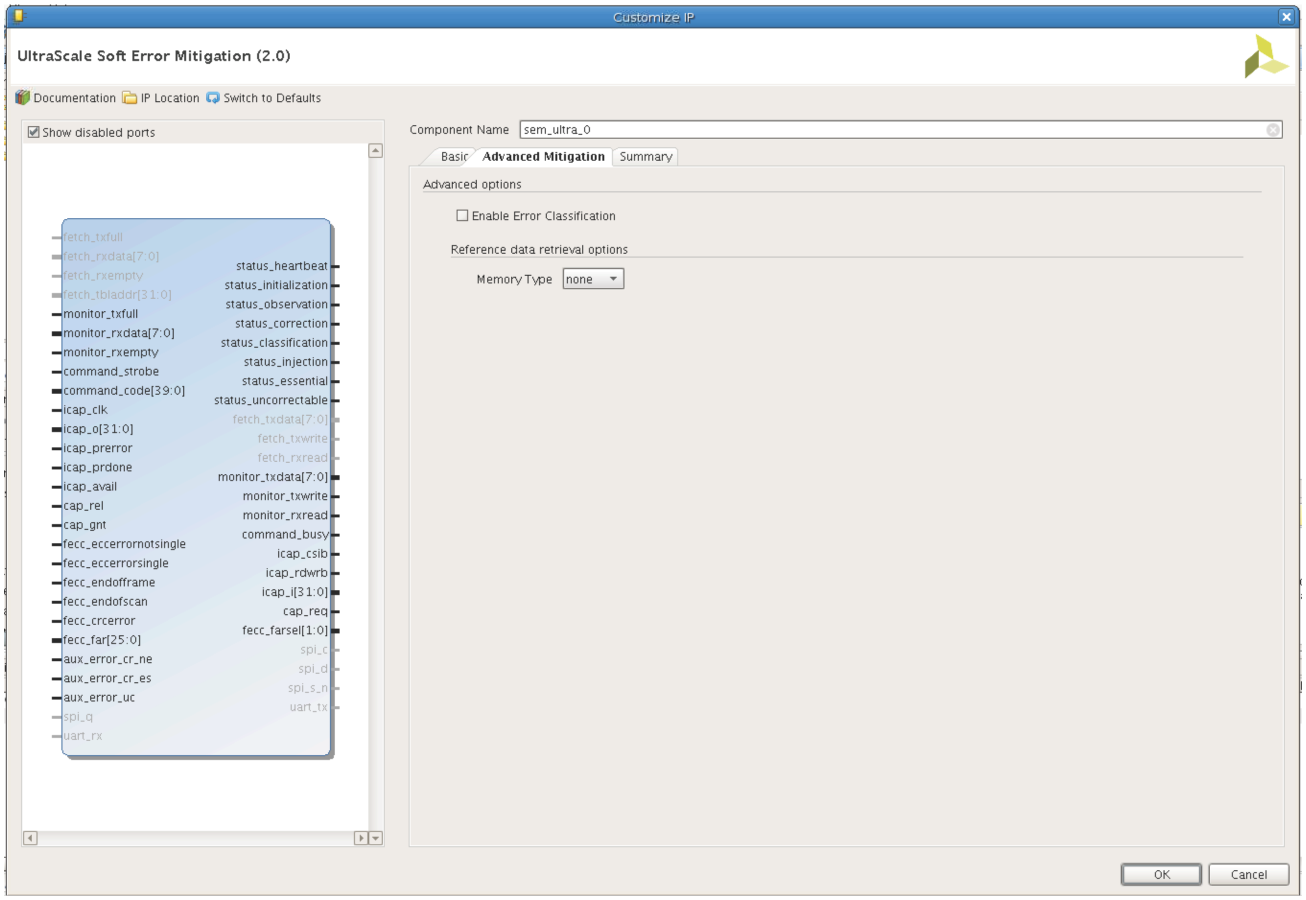
1. Generate the solution using the desired frequency or clock period setting.
2. Extract the required timing data from the implementation results.
3. Complete the timing budget analysis to determine maximum frequency.
4. Re-generate the solution with a frequency at or below the calculated maximum frequency of operation.

Structural Options

The structural options are used to determine the organization of the required ICAP and FRAME_ECC primitives and the IP helper blocks (SPI flash master and UART blocks) relative to the IP. You have the option to include the configuration primitive blocks either in the IP example design (generated with the example design of the IP) or as part of the scope of the IP (included in the OOC design check point).

The verification of the SEM controller IP includes the use of IP helper blocks and primitives that are delivered. For more information, see Structural Options in Chapter 3.

Advanced Mitigation Tab (Monolithic Devices Only)



Component Name: sem_ultra_0

Advanced options

☐ Enable Error Classification

Reference data retrieval options

Memory Type: none

Figure 4-2: **SEM Controller Advanced Mitigation Tab**

Enable Error Classification

The Enable Error Classification check box is used to enable or disable the error classification feature. Error classification is available in the mitigation modes (mitigation and testing or mitigation only) where error correction is enabled. In this version of the IP, this feature is not supported for SSI devices.

The error classification feature uses the Xilinx Essential Bits technology to determine whether a detected and corrected soft error has affected the function of a user design.

Essential Bits are those bits that have an association with the circuitry of the design. If an Essential Bit changes, it changes the design circuitry. However, it might not necessarily affect the function of the design.

Without knowing which bits are essential, the system must assume any detected soft error has compromised the correctness of the design. The system-level mitigation behavior often results in disruption or degradation of service until the FPGA configuration is repaired and the design is reset or restarted.

For example, if the Vivado Bitstream Generator reports that 20% of the configuration memory is essential to an operation of a design, then only two out of every 10 soft errors (on average) actually merits a system-level mitigation response. The error classification feature is a table lookup to determine if a soft error event has affected essential configuration memory locations. Use of this feature reduces the effective FIT of the design. The cost of enabling this feature is the external storage required to hold the lookup table.

When error classification is enabled, the Fetch Interface is generated (as indicated by the Component Symbol) so that the controller has an interface through which it can retrieve external data.

If error classification is enabled, and a detected error has been corrected, the controller looks up the error location. Depending on the information in the table, the controller either reports the error as essential or non-essential. If a detected error cannot be corrected, this is because the error cannot be located. Therefore, the controller conservatively reports the error as essential because it has no way to look up data to indicate otherwise.

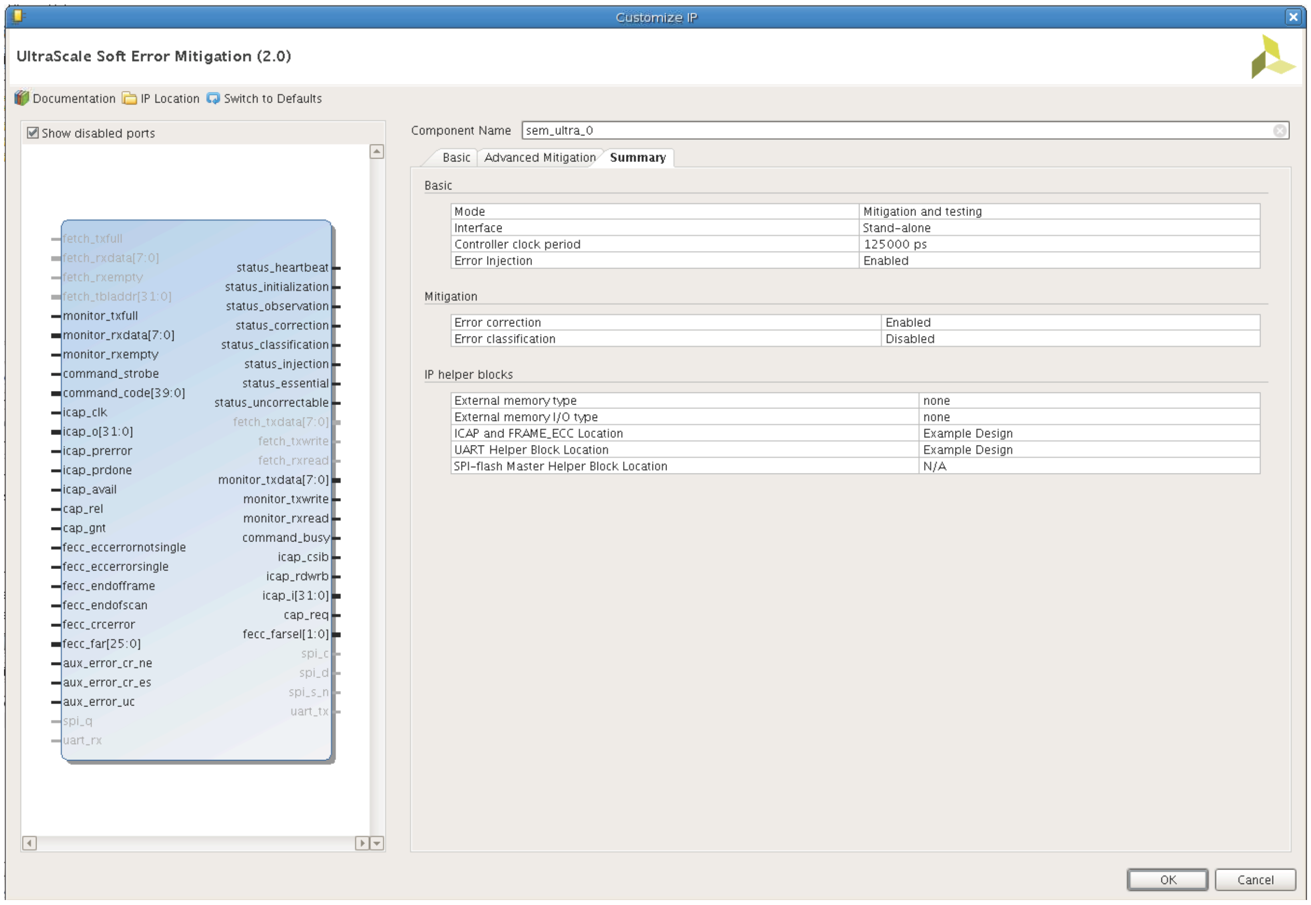
If error classification is disabled, the controller unconditionally reports all errors as essential because it has no data to indicate otherwise.

TIP: *Error classification does not have to be performed by the controller. It is possible to disable error classification by the controller and implement it elsewhere in the system using location of the errors and the essential bit data provided by the implementation tools. The error report messages including their location are issued by the controller through the Monitor Interface.*

Memory Type

When the error classification feature is enabled, the customized IP requires a data retrieval interface to fetch external essential bits data. This requires the Fetch Interface to be bridged to an external storage device and the delivered design includes a SPI flash master helper block to enable connection to an external SPI flash device.

Summary Tab



Component Name sem_ultra_0

Basic

Mode	Mitigation and testing
Interface	Stand-alone
Controller clock period	125000 ps
Error Injection	Enabled

Mitigation

Error correction	Enabled
Error classification	Disabled

IP helper blocks

External memory type	none
External memory I/O type	none
ICAP and FRAME_ECC Location	Example Design
UART Helper Block Location	Example Design
SPI-flash Master Helper Block Location	N/A

Figure 4-3: **SEM Controller Summary Tab**

Review the configuration summary of the IP in the **Summary** tab to confirm each option is correct. Return to the previous tab, if necessary, to correct or change the selected options. After the options are reviewed and correct, click **OK** to complete the IP customization.

User Parameter

Table 4-1 shows the relationship between the GUI fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter	User Parameter	Default Value
Mode	c_feature_set, c_has_error injection	Mitigation and testing
Interface	N/A	Standalone
Clock period	N/A	125,000 ps
Locate Config Prim	c_config_prim_loc	Example design
Locate Helper Blocks	c_helper_block_loc	Example design

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Enable Classification	c_feature_set	FALSE
Memory Type	N/A	None

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Constraining the Core

This chapter contains details about applicable constraints.

Required Constraints

The SEM controller and the system-level design example require the specification of physical implementation constraints to yield a functional result that meets performance requirements. These constraints are provided with the system-level design example in an XDC file.

To achieve consistent implementation results, the XDC provided with the solution must be used. For additional details on the definition and use of a XDC or specific constraints, see the Constraints Guide available through the documentation page for the Vivado Design Suite.

Constraints might require modification to integrate the solution into a larger project, or as a result of changes made to the system-level design example. Modifications should only be made with a thorough understanding of the effect of each constraint. Additionally, support is not provided for designs that deviate from the provided constraints.

Contents of the Xilinx Design Constraints File

Although the XDC delivered with each generated solution shares the same overall structure and sequence of constraints, the contents might vary based on options set at generation. The sections that follow define the structure and sequence of constraints using a Kintex™ UltraScale architecture device implementation as an example.

Controller Constraints

The controller, considered in isolation and regardless of options at generation, is a fully synchronous design. Fundamentally, it only requires a clock period constraint on the system clock input. In the generic XDC, this constraint is placed on the system-level design example clock input and propagated into the controller. The constraint is discussed in Example Design Constraints.

The signal paths between the controller and the FPGA configuration system primitives must be considered as synchronous paths. By default, the paths between the ICAP or FRAME_ECC primitives and the controller are analyzed as part of a clock period constraint on the system clock, because the ICAP and FRAME_ECC clock pin is required to be connected to the same system clock signal.

The exception to this is the following asynchronous ICAP signal outputs: PRERROR, PRDONE, and AVAIL pins. These signals are synchronized internally to the IP and hence additional constraints are needed to ignore these timing paths. See the `set_false_path` constraints listed in Example Design Constraints.

Example Design Constraints

The example design constraints are organized by constraint type and interface.

Timing Constraints

The first constraint in this group is for the system clock input for the entire design. The period constraint value is based on options set at generation:

```
create_clock -name clk -period 125.0 [get_ports clk]
```

This is followed by constraints to ignore the asynchronous output signals of the ICAP ports:

```
set_false_path -from [get_pins {example_support_wrapper/example_support/example_cfg/
cfg_icape3/CLK}] -to [get_pins {example_support_wrapper/example_support/sem_controller/
inst/controller_synchro_icap_prerror/sync_a/D}]
set_false_path -from [get_pins {example_support_wrapper/example_support/example_cfg/
cfg_icape3/CLK}] -to [get_pins {example_support_wrapper/example_support/sem_controller/
inst/controller_synchro_icap_prdone/sync_a/D}]
set_false_path -from [get_pins {example_support_wrapper/example_support/example_cfg/
cfg_icape3/CLK}] -to [get_pins {example_support_wrapper/example_support/sem_controller/
inst/controller_synchro_icap_avail/sync_a/D}]
```

The second set of constraints in this group are for the UART helper block, applying input/ output timing constraints the interface. The input and output timing constraints are set at a single clock period.

```
set_input_delay -clock clk -max -125.0 [get_ports uart_rx]
set_input_delay -clock clk -min 250.0 [get_ports uart_rx]
set_output_delay -clock clk -125.0 [get_ports uart_tx] -max
set_output_delay -clock clk 0 [get_ports uart_tx] -min
```

The third set of constraints in this group are for the SPI flash master helper block, and is only present when error classification is enabled and this helper block is generated. It applies input/output timing constraints to the interface. The input and output timing is of considerable importance, as the actually timing must be used in the analysis of the SPI bus timing budget. However, there is no hard requirement for input and output timing of the FPGA implemented. It varies based on selected device and speed grade.

As such, the input and output timing constraints are arbitrarily set at two times the period constraint. The additional constraint to use IOB flip-flops yields substantially better input and output timing than the constraint values suggest. It is the actual timing obtained from the timing report that should be used in the analysis of the SPI bus timing budget, not the constraint value.

```
set_input_delay -clock clk -max -125.0 [get_ports spi_q]
set_input_delay -clock clk -min 250.0 [get_ports spi_q]

set_output_delay -clock clk -125.0 [get_ports spi_c] -max
set_output_delay -clock clk 0 [get_ports spi_c] -min

set_output_delay -clock clk -125.0 [get_ports spi_d] -max
set_output_delay -clock clk 0 [get_ports spi_d] -min

set_output_delay -clock clk -125.0 [get_ports spi_s_n] -max
set_output_delay -clock clk 0 [get_ports spi_s_n] -min

set_property IOB TRUE [get_cells example_support_wrapper/example_support/
example_spi/example_spi_byte/spi_c_ofd]
set_property IOB TRUE [get_cells example_support_wrapper/example_support/
example_spi/example_spi_byte/spi_d_ofd]
set_property IOB TRUE [get_cells example_support_wrapper/example_support/
example_spi/example_spi_byte/spi_q_ifd]
set_property IOB TRUE [get_cells example_support_wrapper/example_support/
example_spi/spi_s_ofd]
```

Placement Constraints

The following constraints in the XDC implement a pblock to place portions of the system-level design example into a bounded region of the selected device. The instances included in the pblock depend on the options set at generation. The range values vary depending on device selection.

The pblock forces packing of the soft error mitigation logic into an area physically adjacent to the ICAP site in the device. Most importantly, this maintains reproducibility in timing results. It also improves resource usage; the pblock forces tighter packing. The delivered pblock is provided as an example. You are encouraged to further tighten the size of the pblock to improve the resource usage and reduce physical footprint of the SEM controller.

```
create_pblock sem
resize_pblock [get_pblocks sem] -add {SLICE_X82Y75:SLICE_X87Y89}
resize_pblock [get_pblocks sem] -add {RAMB36_X8Y15:RAMB36_X8Y17}
resize_pblock [get_pblocks sem] -add {DSP48E2_X15Y30:DSP48E2_X15Y35}
add_cells_to_pblock -pblock sem -cells [get_cells example_support_wrapper/
example_support/example_spi/*]
```

```
add_cells_to_pblock -pblock sem -cells [get_cells example_support_wrapper/
example_support/example_uart/*]
add_cells_to_pblock -pblock sem -cells [get_cells example_support_wrapper/
example_support/*]
set_property LOC RAMB36_X8Y17 [get_cells example_support_wrapper/example_support/
sem_controller/inst/controller_dbuffer/dbuffer_dbuffer_mem/storage]
set_property LOC RAMB36_X8Y16 [get_cells example_support_wrapper/example_support/
sem_controller/inst/controller_instrom/firmware0]
set_property LOC RAMB36_X8Y15 [get_cells example_support_wrapper/example_support/
sem_controller/inst/controller_instrom/firmware1]

create_pblock dbg
resize_pblock [get_pblocks dbg] -add {SLICE_X88Y75:SLICE_X100Y89}
resize_pblock [get_pblocks dbg] -add {RAMB36_X9Y15:RAMB36_X9Y17}
add_cells_to_pblock -pblock dbg -cells [get_cells example_vio_si8/*]
add_cells_to_pblock -pblock dbg -cells [get_cells example_vio_so32/*]
add_cells_to_pblock -pblock dbg -cells [get_cells example_vio_si1_so5/*]
add_cells_to_pblock -pblock dbg -cells [get_cells example_vio_si1_so41/*]
```

Pin Constraints

The following constraints in the XDC are a template for assigning I/O pin locations to the top-level ports of the system-level example design. These assignments are board-specific and therefore cannot be automatically generated. The exception to this is when the design is targeted to support evaluation boards (KCU105 evaluation board). In these circumstances, the pin locations are compatible to the targeted board.

In other cases, apply these constraints by assigning valid I/O pin locations and standards for the target board:

```
set_property IOSTANDARD <io standard> [get_ports clk]
set_property PACKAGE_PIN <package pin> [get_ports clk]

set_property IOSTANDARD <io standard> [get_ports uart_rx]
set_property PACKAGE_PIN <package pin> [get_ports uart_rx]

set_property IOSTANDARD <io standard> [get_ports uart_tx]
set_property PACKAGE_PIN <package pin> [get_ports uart_tx]

set_property IOSTANDARD <io standard> [get_ports spi_q]
set_property PACKAGE_PIN <package pin> [get_ports spi_q]

set_property IOSTANDARD <io standard> [get_ports spi_c]
set_property PACKAGE_PIN <package pin> [get_ports spi_c]

set_property IOSTANDARD <io standard> [get_ports spi_d]
set_property PACKAGE_PIN <package pin> [get_ports spi_d]

set_property IOSTANDARD <io standard>8 [get_ports spi_s_n]
set_property PACKAGE_PIN <package pin> [get_ports spi_s_n]
```

Essential Bit Generation

The following constraint is necessary to generate the essential bit information. This constraint is only available when Error Classification is enabled.

```
set_property bitstream.seu.essentialbits yes [current_design]
```

Constraints for SSI Devices

In the system-level design example, two to three controller instances are generated depending on the device. The controller instances are named to include an identification number ranging from 0 to 2. The controller instance numbering matches the hardware SLR numbering.

An area constraint is applied to each controller to keep controllers centrally located near their associated configuration logic primitives on each SLR. The configuration logic primitives also have placement constraints. This is very similar to the constraints for non-SSI devices as they are repeated on a per-SLR basis.

The shared blocks such as the UART and SPI flash master helper blocks also have area constraints to locate them in the Master SLR, which is centrally located in the device.

Device, Package, and Speed Grade

This section is not applicable for this IP core.

Clock Frequency

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Simulation

This section contains information about simulating IP in the Vivado Design Suite. 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 3].

Design simulations that instantiate the controller is supported. In other words, including the controller in a larger project does not adversely affect ability to run simulations of functionality unrelated to the controller. However, it is not possible to observe the controller behaviors in simulation. Design simulation including the controller compiles, but the controller does not exit the Initialization state. Hardware-based evaluation of the controller behaviors is required.

Synthesis and Implementation

The SEM core should be synthesized and implemented in conjunction with the provided example design. For more details, see Implementation in Chapter 5.

For general details about synthesis and implementation using the Vivado Design Suite, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

Detailed Example Design

This section provides an overview of the UltraScale™ architecture SEM controller system-level example design and the interfaces it exposes. The system-level example design encapsulates the controller and various helper blocks that serve to interface the controller to other devices. These helper blocks can include I/O Pins, I/O Interfaces, Memory Controllers, or application-specific system management interfaces.

The system-level example design is verified along with the controller. As delivered, the system-level example design is not a "reference design," but an integral part of the total solution. While users do have the flexibility to modify the system-level example design, the recommended approach is to use it as delivered.

Functions

The system-level example design can be divided into two general functional groups:

- Support layer (`<component_name>_support`)
- Example layer (`<component_name>_example_design`)

The support layer and its sub-layers, contains all the integral logic of the total Soft Error Mitigation solution. This includes, the instantiations of the following logic:

- The helper blocks required to connect the IP to external devices:
    - The UART helper block, a bridge between controllers and a standard RS-232 port. The resulting interface can be used to exchange commands and status with controllers. This interface is designed for connection to processors.
    - The SPI flash master helper block, a bridge between controllers and a standard SPI bus. The resulting interface can be used to fetch data by controllers. This helper block is only present when the classification feature is enabled and is designed for connection to standard SPI flash.
- Configuration system primitives instantiation required by the IP.
- Clocking primitive used to distribute the core system clock.

The verification of the SEM controller IP includes these blocks and integrating of this logic into a design is recommended and is fully supported.

The example layer contains instantiation of the support layer and several VIO cores that eases the ability to visually inspect the IP status and dynamically drive inputs to the IP that do not require connection to external devices.

For monolithic devices, there are four VIO IP cores instantiated:

- `<component_name>_vio_si8` – Displays SEM controller Status Interface.
- `<component_name>_vio_so32` – Defines the `fetch_tbladdr` input for the Fetch Interface. Only available when error classification is enabled.
- `<component_name>_vio_si1_so5` – Displays output and define input for the ICAP Arbitration and Auxiliary Interfaces.
- `<component_name>_vio_si1_so41` – Displays output and define input for the Command Interface.

For SSI devices, there is a VIO core for each SLR and another VIO core to drive inputs that are shared by all SLRs:

- `<component_name>_vio_slr_si10_so2` – Displays all outputs and define inputs for the Status and ICAP Arbitration Interfaces for a single SLR.
- `<component_name>_vio_generic_so44` – Drives shared SLR inputs for Auxiliary, Command, and Fetch Interfaces. Inputs for the Fetch Interface is only available when error classification is enabled.

Combined, the support and example layer provides a complete SEU mitigation solution for you to evaluate. The system-level design example is also provided as RTL source code unlike the controller itself to allow flexibility in system-level interfacing.

Figure 5-1 shows a block diagram of the system-level design example for non-SSI devices. The blocks drawn in gray only exist in certain configurations.

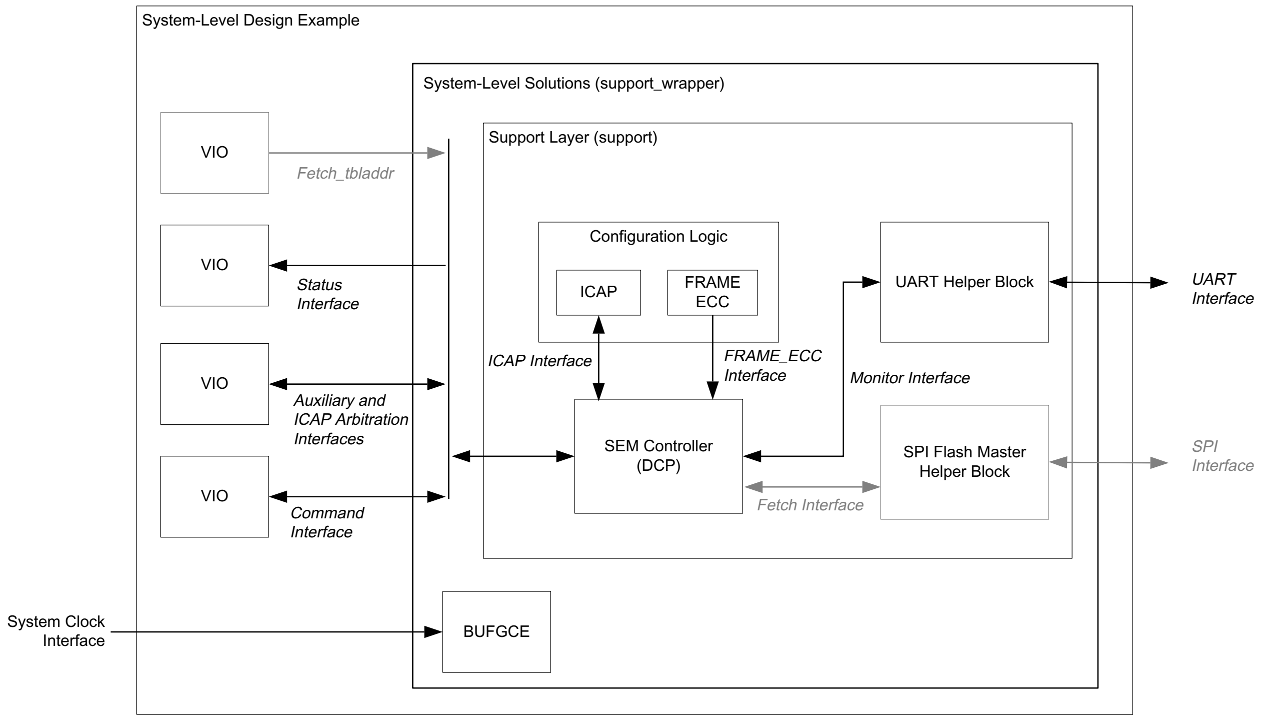


Figure 5-1: **Example Design Block Diagram for Non-SSI Devices**

Figure 5-2 shows a block diagram of the system-level design example for SSI devices. The blocks drawn in gray only exist in certain configurations.

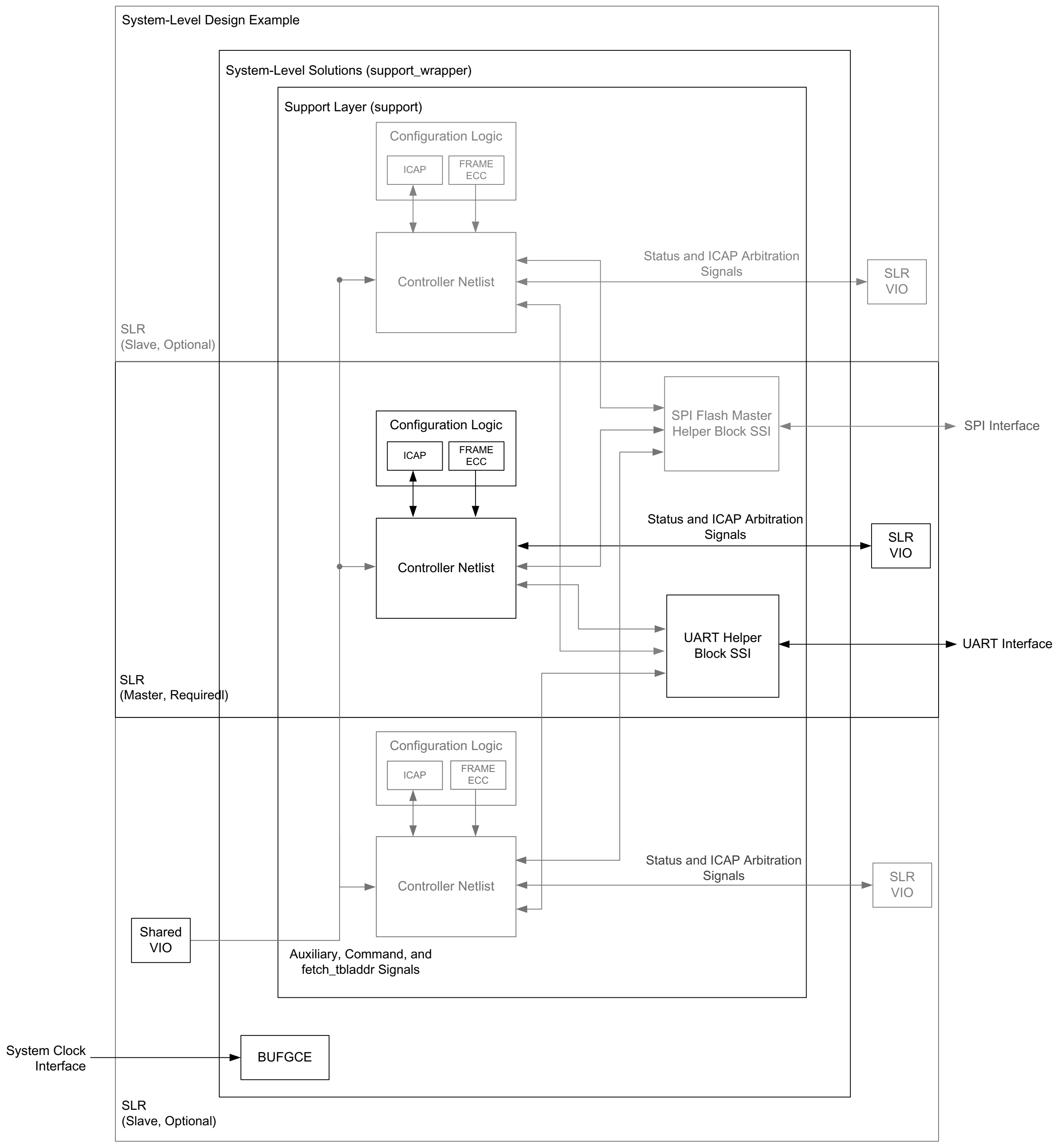


Figure 5-2: **Example Design Block Diagram for SSI Devices**

Port Descriptions

Figure 5-3 shows the example design ports for all devices. The ports are clustered into three groups. The groups shaded in gray only exist when error classification feature is enabled.

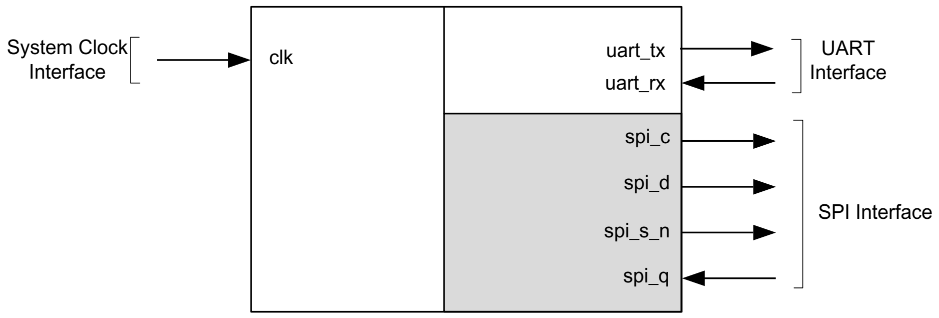


Figure 5-3: **Example Design Ports**

In an SSI device, each SLR is numbered. There are two numbering methods: hardware SLR numbering and software SLR numbering.

A hardware SLR number represents the configuration order of the SLR in the device. The Master SLR, which is always present, is hardware SLR 0. The hardware SLR numbers of additional Slave SLRs are approximately assigned radially outward from the Master SLR.

A controller instance located in an SLR determines the hardware SLR number at runtime by reading the IDCODE register through the ICAP on the SLR. In all command and status exchanges with controllers implemented in an SSI device, hardware SLR numbering is used.

A software SLR number represents the bottom-to-top physical order of the SLR in the device. The Master SLR, which is always present, has a software SLR number that varies by device. The software SLR numbers are prominently visible in the device view presented by the Xilinx development software.

Table 5-1 details the mapping between hardware SLR numbers and software SLR numbers.

Table 5-1: **Device Number SLR**

Device	Software SLR Number	Hardware SLR Number	SLR Type
KU115	1	1	Slave
	0	0	Master
VU125[(1)]	1	1	Slave
	0	0	Master
VU190[(1)]	2	2	Slave
	1	0	Master
	0	1	Slave

Table 5-1: **Device Number SLR *(Cont'd)***

Device	Software SLR Number	Hardware SLR Number	SLR Type
VU440[1]	2	2	Slave
	1	0	Master
	0	1	Slave

Notes:

1. Not supported in the 2015.1 release.

TIP: *Understanding and translating the SLR numbering methods is not necessary for successful implementation of controllers in an SSI device. However, this information might be useful in conjunction with error injection if it is desired to direct an injected error to a specific SLR.*

The example design ports for SSI devices are the same as monolithic devices because the delivered UART and SPI flash master helper blocks combine all the Fetch and Monitor Interfaces of each controller (per SLR) to provide a single interface to manage and interact with IPs. However, if the ports on the `support_wrapper` hierarchy are inspected, you will find that some interface ports (for example, Status Interface) become buses, where the bus width is determined by the number of SLRs in the SSI devices.

The system-level design example does not have a reset port. The controller automatically initializes itself. The controller then initializes the helper blocks, as required.

The system-level design example is a fully synchronous design using `clk` as the single clock. All state elements are synchronous to the rising edge of this clock. As a result, the interfaces are generally synchronous to the rising edge of this clock.

All interfaces available in the system-level design example including their system requirements has been discussed in Chapter 2, Product Specification and Chapter 3, Designing with the Core in detail. The following links are given as a convenience:

- **System Clock Interface** – Port Description (System Clock Interface in Chapter 2) and Usage (System Clock Interface in Chapter 3).
- **Status Interface** – Port Description (Status Interface in Chapter 2) and Usage (Status Interface in Chapter 3).
- **Command Interface** – Port Description (Command Interface in Chapter 2) and Usage (Command Interface in Chapter 3).
- **UART Interface** – Port Description (UART Interface in Chapter 2) and Usage (UART Interface in Chapter 3).
- **SPI Interface** – Port Description (SPI Interface in Chapter 2) and Usage (SPI Interface in Chapter 3).

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 the `XCI` file under **Design Sources** and select **Open IP Example Design**.

Run Synthesis and Implementation

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

Generate the Bitstream

When configured to use the optional error classification, a Tcl property must be set prior to bitstream generation to enable essential bits. For Tcl property syntax, see Essential Bit Generation in Chapter 4.

For these configurations, bitstream generation requires a large amount of system RAM.

Creating the External Memory Programming File for Non-SSI Devices

If the solution requires external data storage to support error classification, an additional Tcl script is called to post-process special `write_bitstream` output files into a SPI flash programming file.

The `makedata.tcl` script is generated in the example project. After bitstream generation is complete, locate the script on disk, and source this script in the Tcl Console.

```
source <path to example project>/<component_name>_example.srcs/sources_1/imports/
<component_name>/<component_name>/implement/makedata.tcl

sem_ultra_0_example/sem_ultra_0_example.srcs/sources_1/imports/sem_ultra_0/
sem_ultra_0/implement/makedata.tcl
```

Next, locate the implementation results directory. Click the **Design Runs** tab, select the implementation run, and note the Run directory indicated in the **Implementation Run Properties** window. From the Tcl Console, navigate to the implementation results directory to run the `makedata` script over the `write_bitstream` output files.

```
cd <component_name>_example.runs/impl_1
```

Then execute the commands outlined in these steps:

```
makedata -ebd <ebd filename> datafile
makedata -ebd sem_ultra_0_example_design.ebd datafile
```

The command creates the VMF, BIN and MCS files.

External Memory Programming File

When error classification is enabled, an image of the essential bit lookup data is required. The size of the data set is a function of the target device. The data sets are generated by the `write_bitstream` application.

The format of the data is required to be binary, using the full data set generated by `write_bitstream`. The external storage must be byte addressable. A small table is required at the address specified to the SEM controller through `fetch_tbladdr[31:0]`. By default, `fetch_tbladdr[31:0]` is zero.

For non-SSI devices, the table format is:

- **Byte 0** – 32-bit pointer to start of essential bit data, byte 0 (least significant byte)
- **Byte 1** – 32-bit pointer to start of essential bit data, byte 1
- **Byte 2** – 32-bit pointer to start of essential bit data, byte 2
- **Byte 3** – 32-bit pointer to start of essential bit data, byte 3 (most significant byte)
- Remaining bytes are reserved, filled with ones

A pointer value of 0xFFFFFFFF is used if a particular block of data is not present. The essential bit data can be located at any addresses provided each data block is contiguous and it is possible to perform a read burst through each data block. For SPI flash that does not support read burst across device boundaries, data blocks must be located so that they do not straddle any of these device boundaries. For example, many SPI flash of a density greater than 256 Mbit do not allow read burst across 256 Mbit boundaries.

The Tcl script, which post processes the `write_bitstream` output files, generates three outputs:

- An Intel hex data file (MCS) for programming SPI flash devices
- A raw binary data file (BIN) for programming SPI flash devices
- An initialization file (VMF) for loading SPI flash simulation models

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

A simulation test harness is provided with the example design. This enables functional and timing simulation of designs that include the UltraScale architecture SEM controller using standard Xilinx simulation flows. However, it is not possible to observe the SEM controller behaviors in simulation. Hardware-based evaluation is required.

Verification, Compliance, and Interoperability

The controller and example design are verified together, using several methods including an automated hardware test bench. The controller and example design will also be validated together, in an accelerated particle beam, to ensure the solution responds correctly to naturally injected, random error events.

Verification

The SEM verification objectives are derived from the functional specification of the product. Verification is performed to ensure a high-quality product that uses a hardware verification methodology. The techniques and tools used were:

- Dynamic checks, through a hardware test bench
    - Functional Coverage: Compares design behavior against expected behavior
- Static checks, through a checking tool suite
    - Linting
    - Clock Domain Crossing

The SPI flash devices used in the hardware verification platform were:

- M25P128 (ST Microelectronics / Numonyx)
- M25L25635E (Macronix)
- N25Q512 (Micron)
- N25Q00 (Micron)

Validation

Hardware validation is a key final test and gates the release of the product. Hardware validation adds value by conducting the following tests:

- External Interface Evaluation
  - Timing budget evaluation for external memory system
- Integration and Implementation
  - Hardware testing of all possible generated core netlists
- Operating Environment Robustness
  - Sample testing across the supported device list for all sub-families

Conformance Testing

No industry standard certification testing is defined. The generated core netlists will be put through testing while exposed to a beam of accelerated particles. This testing:

- Validates that detection, correction, and classification take place separate from injection. The verification methodology relies on error injection by the solution itself, and does not test detection, correction, and classification processes separate from injection. Errors during beam testing occur separate from injection by the solution itself.
- Validates that the solution exhibits normal and expected behaviors, including detection, correction, and classification of errors.

Migrating and Upgrading

There are no port or parameter changes for upgrading the core in the Vivado® Design Suite at this time.

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 UltraScale architecture SEM controller, the Xilinx Support web page (www.xilinx.com/support) 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 SEM controller. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx® Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). 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 core 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 UltraScale architecture SEM controller

AR 63609

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Additional Resources.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Note: Access to WebCase is not available in all cases. Log in to the WebCase tool to see your specific support options.

Debug Tools

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

Vivado Lab Edition

Vivado® Lab Edition inserts logic analyzer and virtual I/O cores directly into your design. Vivado Lab Edition 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.

The Vivado logic analyzer is used with the logic debug LogiCORE 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 7].

Reference Boards

Various Xilinx development boards support the SEM controller core. These boards can be used to prototype designs and establish that the core can communicate with the system. The UltraScale architecture evaluation board used is KCU105.

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. Vivado Lab Edition is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using Vivado Lab Edition for debugging the specific problems.

General Checks

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

If using any clock management blocks in the design, ensure they have obtained lock by monitoring their status.

Interface Debug

Monitor Interface

While using the Monitor Interface is optional, Xilinx strongly recommends having a method in place to connect the Monitor Interface. The Monitor Interface provides information that is crucial in debugging potential problems or answering questions that might arise. The UART helper block provided in the example design is a UART that can be connected to a standard RS232 port, or to USB through a USB-to-UART bridge.

If this connection is not available in the system, at the minimum, Xilinx strongly recommends you to buffer the output of Monitor Interface into a FIFO and the data can be post-processed for debugging purposes.

If neither of these options are possible, the Monitor Interface needs to be correctly tied-off for the IP to complete initialization correctly. See Monitor Interface in Chapter 3 for guidance. If this is not done correctly, the IP hangs in the initialization state.

To confirm the SEM controller is operational, observe the initialization report issued by the SEM controller over the Monitor Interface. It generally has the following form:

```
SEM_ULTRA_V2_0
SC 01
FS 03
AF 01
ICAP OK
RDBK OK
INIT OK
SC 02
O>
```

The first line lists the core version. The third line indicates the SEM controller feature set, which is a summary of the SEM controller core options selected when the core was generated.

If the UART helper block is used, and the initialization report appears scrambled or garbage characters appear, verify that the terminal program communication settings match those listed in Monitor Interface in Chapter 3.

Also, verify that the frequency of the actual clock provided to the SEM controller coupled with the UART helper block V_ENABLETIME parameter value, yield a standard baud rate and that the terminal program communication settings match the bit rate. This is described in Equation 3-1 and Equation 3-2.

If the SEM controller cannot achieve communication with the FPGA configuration logic through the ICAP primitive, the initialization report does not get past the ICAP line, and OK is not present because the controller cannot communicate with the FPGA configuration logic. In such a scenario, the initialization report will look like this:

```
SEM_ULTRA_V2_0
SC 01
FS 03
AF 01
ICAP
```

If this happens, it is necessary to determine why the ICAP is not responding. Some possible items to check:

- Ensure the instantiation of the ICAP is correct for the device being used.
- Ensure that no other process is blocking the ICAP.
    - Verify no JTAG access is occurring and that SelectMAP persist is not set.
- The connection between the SEM controller and the ICAP must be direct, unless the ICAP sharing is implemented. Never add pipelining between the SEM controller and the ICAP.

As documented in Unsupported Features in Chapter 1, the SEM controller is not compatible with bitstream authentication nor is it compatible with POST_CRC, POST_CONFIG_CRC, or any other related constraints. If the initialization report does not get past the ICAP, RDBK, or INIT lines, verify that none of these have been used.

Status Interface

If the Monitor Interface is not available for debugging, you should also verify that the IP completes its initialization state correctly and the behavior is normal. The valid state transitions of the IP can be found in Figure 3-4, page 39 and Figure 3-5, page 39.

Clocking

Xilinx recommends the clock to be sourced from an oscillator and brought in from a pin directly to the SEM controller. While the likelihood of an SEU event hitting the configuration cells associated with creating the clock internally from a PLL or MMCM is very small, it is best to strive for the highest reliability possible. However, if a PLL or MMCM output or other logic is used to generate the clock, ensure the clock never violates the SEM controller minimum period at any time, including during design start-up or prior to PLL/MMCM lock.

When clock management is used, suppress the clock toggling to the SEM controller until after the clock is stable. For example use a BUFGMUX or BUFGCE to keep the SEM controller clock from toggling until PLL/MMCM lock is achieved.

If an unstable clock is provided to the IP, the SEM controller might behave in the following method:

- Boot and initialization sequences not consistently completed
- IP completes initialization but reports fatal error soon after entering observation

Other Incompatibilities

The SEM controller initializes and manages the FPGA integrated silicon features for soft error mitigation and when included in a design, do not include any design constraints or options that would enable the built-in detection functions. For example, do not set POST_CRC, POST_CONFIG_CRC, or any other related constraints. Similarly, do not include options to disable GLUTMASK. The default value of YES is required to prevent false error detections by the SEM controller.

Additional Resources and Legal Notices

Xilinx Resources

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

References

These documents provide supplemental material useful with this user guide:

1. *Xilinx Device Reliability Report (*UG116)
2. *UltraScale™ Architecture Configuration User Guide* (UG570)
3. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
4. *Vivado Design Suite User Guide: Implementation* (UG904)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *ISE® to Vivado Design Suite Migration Methodology Guide* (UG911)
7. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
8. *Vivado Design Suite User Guide: Getting Started* (UG910)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/01/2015	2.0	Initial Xilinx release.

Please Read: Important Legal Notices