UltraScale Architecture Soft Error Mitigation Controller v3.1

LogiCORE IP Product Guide

Vivado Design Suite

PG187 November 5, 2022

Table of Contents

IP Facts

Chapter 1: Overview

Navigating Content by Design Process 6
Memory Types 7
Mitigation Approaches 7
Reliability Estimation 8
Feature Summary 9
Applications 12
Key Considerations for SEM IP Adoption 12
Partial Reconfiguration Support 16
Encryption and Authentication Support 16
Unsupported Features 17
Licensing and Ordering 18

Chapter 2: Product Specification

Features 19
Standards 23
Performance 23
Resource Utilization 35
Port Descriptions 37

Chapter 3: Designing with the Core

General Design Guidelines 49
Structural Options 51
Interfaces 53
Systems 99
Configuration Memory Masking 105
Resets 105

Chapter 4: Design Flow Steps

Customizing and Generating the Core 106
Constraining the Core 113

Simulation 119
Synthesis and Implementation 119
Integration and Validation 120

Chapter 5: Example Design

Functions 122
Port Descriptions 127
Implementation 130
External Memory Programming File 131

Chapter 6: Test Bench

Appendix A: Verification, Compliance, and Interoperability

Verification 135
Validation 136
Conformance Testing 136

Appendix B: Upgrading

Upgrading in the Vivado Design Suite 137

Appendix C: Utilizing an Evaluation Board to Demonstrate SEM Controller Behavior

Generating Bitstream 139
Hardware and Software Setup 143

Appendix D: Example UART Logs

Initialization Report 148
Status Report 148
Injection Commands 150
Enter Observation Command 150
Enter Idle Command 150
Enter Detect Only Command. 150
Software Reset Command 150
Frame Address Translation Command 151
Query Command 151
External Memory Reads Command (Xmem) 151
Peek Command 151
Observation Error Reports 151
Detect Only Error Reports 154
Diagnostic Scan 155

Appendix E: Error Injection Guidance

Appendix F: IP Design Checklist

Appendix G: SPI Bus Timing Budget

SPI Bus Clock Waveform and Timing Budget 161
SPI Bus Transmit Waveform and Timing Budget 163
SPI Bus Receive Waveform and Timing Budget 165
SPI Bus Timing Budget Conclusions 174

Appendix H: Monitor Interface Signaling and Protocol

Monolithic and UltraScale+ SSI Devices 175
UltraScale SSI Devices 178

Appendix I: Fetch Interface Signaling and Protocol

Appendix J: Debugging

Finding Help on Xilinx.com 183
Debug Tools 184
Hardware Debug 185
Interface Debug 185
Additional Error Injection Options 187
Clocking. 188
Device Dependency 189
Design Properties and Constraints 189

Appendix K: Additional Resources and Legal Notices

Xilinx Resources 190
Documentation Navigator and Design Hubs 190
References 191
Revision History 192
Please Read: Important Legal Notices 197

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.
- Six convenient modes:
  - Mitigation and Testing
  - Mitigation only
  - Detect and Testing
  - Detect only
  - Emulation
  - Monitoring
- Optional error correction based on error-correction code (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 internal configuration access port (ICAP) primitive sharing.
- Ability to continuously monitor the Configuration Memory and report the first error (ECC or cyclic redundancy check (CRC)) without correcting the error (Detect only)
- Ability to perform a diagnostic scan of the Configuration Memory and report all errors detected by frame level ECC (Diagnostic Scan)
- An IP Design Checklist is provided to assist in the use and integration of the SEM controller.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	UltraScale+™ [1], UltraScale
Supported User Interfaces	RS-232, SPI
Resources	See Table 2-12 to Table 2-13
Provided with Core	
Design Files	Encrypted register transfer level (RTL)
Example Design	Verilog
Test Bench	N/A
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	N/A
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation	N/A
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 63609
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see AR 63609.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Chapter 1

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 UltraScale™ architecture SEM controller can ensure a higher level of reliability.

If your application benefits from using the SEM controller, the IP Design Checklist in Appendix F provides a list of guidance and considerations when using and integrating the SEM controller.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development**: Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Interfaces
  - Customizing and Generating the Core
  - Example Design

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 determines 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 do 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 six different modes dependent on the design requirements:

- Mitigation and Testing
- Mitigation only
- Detect and Testing
- Detect 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 next two modes, Detect and Testing and Detect only, enables error detection without the ability to correct or classify. Error injection is not available in Detect only mode.

The last two modes, Emulation and Monitoring only, enable you to use the SEM controller to assess and monitor your system behavior when a single event upset (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 and detect 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.

TIP: *Although the out-of-the-box solution for error classification requires an external SPI flash to store the essential bits data, you can also implement the classification feature outside the IP by storing the essential bits data in the system memory and performing the look up based on the error location reported by the SEM controller Monitor/UART Interface.*

In the detect modes, when an ECC or CRC error is detected, the SEM controller evaluates the situation to identify the Configuration Memory location involved and reports it if possible. After the error detect report is completed, the IP transitions to idle.

When the SEM controller is idle, it optionally accepts input from you to inject errors into the Configuration Memory (for Mitigation and Testing, Detect and Testing, and Emulation modes). In the Mitigation and Testing and Detect and Testing modes, 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.

In addition to error injection, there are other useful testing and debugging features provided in the Idle state including frame reads, configuration register reads, external memory reads, and frame address translations.

Finally, there are two other types of error detection: Detect only and Diagnostic scan. These commands are issued to the SEM controller from the Idle state (independent of mode):

- **Detect Only** – This command causes the SEM controller to continuously monitor the configuration memory until it detects an ECC or CRC error. After an error is detected, the SEM controller reports the error and goes to the Idle state. Unlike the mitigation mode, this type of monitoring does not include error correction.
- **Diagnostic Scan** – This command causes the SEM controller to perform a single scan of the configuration memory and report all ECC errors that are detected. After completing a single pass, the SEM controller returns to the Idle state. The error detection mechanism used in this feature does not leverage the built-in error detection capability of the device. No error correction is performed in this type of scan.

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.

One example of a mitigation strategy that requires a user-controlled mitigation solution is error logging with no correction. One method to implement this is by configuring the SEM controller in the Detect mode.

For example, you could use the IP configured in Detect mode which automatically enables the SEM controller to perform Detect only scan right after it completes initialization. After an error is detected, you can take any action on the errors found (reconfigure the device, reset the logic, etc.) if you choose to.

After detecting the first error or first uncorrectable error, you can then issue a Diagnostic Scan of the device periodically to log all accumulative ECC errors resident on the device. The frame level ECC errors that are detected and logged can then be used for off-line post processing.

IMPORTANT: *Note that the Diagnostic Scan feature is not designed for mitigation purposes and should not be used for real-time mitigation. Xilinx does not guarantee the functionality of the FPGA if the device is left to accumulate errors in its configuration memory. The error detection latency for this feature is also significantly longer than the Detect or Mitigation modes, hence it should only be used for diagnostic purposes.*

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.

RECOMMENDED: *Xilinx recommends integrating the SEM IP core as early as possible, ideally at the start of the project. For more information, see Integration and Validation, page 120.*

Key Considerations for SEM IP Adoption

This section is designed to assess whether the SEM IP helps meet the soft error mitigation goals of your product's deployment and what mitigation approach should be chosen. The concepts used in this section were covered in prior sections.

There are two main considerations:

- Understanding the soft error mitigation requirement for your design
- What actions need to be taken if a soft error occurs

Understanding the Soft Error Mitigation Requirement

If the effects of soft errors are a concern for your product's deployment, each component in the system design will usually need a FIT budget. To calculate the FIT for a Xilinx FPGA, use the SEU FIT rate estimator available in the SEU lounge and see the XAPP472 SEU Estimator on how to use the FIT rate estimator.

To calculate FIT for a deployment, at the minimum you need:

- Target device

- Estimated number of the device to be deployed

Besides providing the estimated FIT for the device, the estimator also predicts the number of soft errors expected for a given time frame.

By using an implemented design, a more accurate estimation of the FIT is achieved. This is possible by entering the number of block RAM used, whether the block RAM ECC feature is employed to detect and correct soft errors, as well as the percentage of essential bits in the design. Essential bits are defined as configuration RAM bits that are used to define the function on the FPGA. If an essential bit is changed unintentionally by a soft error, it is possible that the function in the FPGA does not behave as intended.

On the other hand, if a non-essential bit is changed, there is no impact to the function. The following steps determine the percentage of essential bits in a design:

1. Set the following property in Vivado:

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

2. Regenerate the bitstream for your design.
3. The essential bits percentage is printed in the Vivado Tcl console and consequently the Vivado log while the bitstream and essential bits data are being generated. Here is an example:

   ```
   Writing bitstream ./sem_ultrap_v3_1_example.bit...
   Creating bitstream...
   Writing bitstream ./sem_ultrap_v3_1_example.ebc...
   Creating essential bits data...
   This design has 707717 essential bits out of 143015456 total (0.49%).
   ```

After a FIT estimation has been completed, it must be checked to determine if the FIT requirement for the deployment is met. Other design approaches might need to be implemented to reduce the FIT.

If there is not a FIT target, then it is unclear what design changes (and trade-offs that come with them) need to be made to mitigate the soft error effects, including whether a deployment benefits from using the SEM IP. If that is the case, it is important to identify the benefits obtained using SEM IP before integrating it.

Actions to Consider When a Soft Error Occurs

The SEM IP can be configured to detect or detect and correct soft errors. Therefore, consideration must be given to the system-level actions, if any, that must be taken in reaction to soft errors. If no action is taken when a soft error is detected in a design, the benefits of having the SEM IP in the design should be assessed and understood. While this is a valid use case, the effects of soft errors on the design should be known and this mitigation approach should be well understood to ensure it meets the soft error mitigation goals.

RECOMMENDED: *When using the SEM IP in a design, it is strongly recommended to log the output of the Monitor interface.*

Figure 1-1 shows an example of a decision tree that iterates what a system could do when a soft error occurs. By understanding the possibilities and the system-level considerations, a decision can be made to determine whether or not to use the SEM IP in a design.

Note: This diagram is provided as an example and it does not list all the possible considerations.

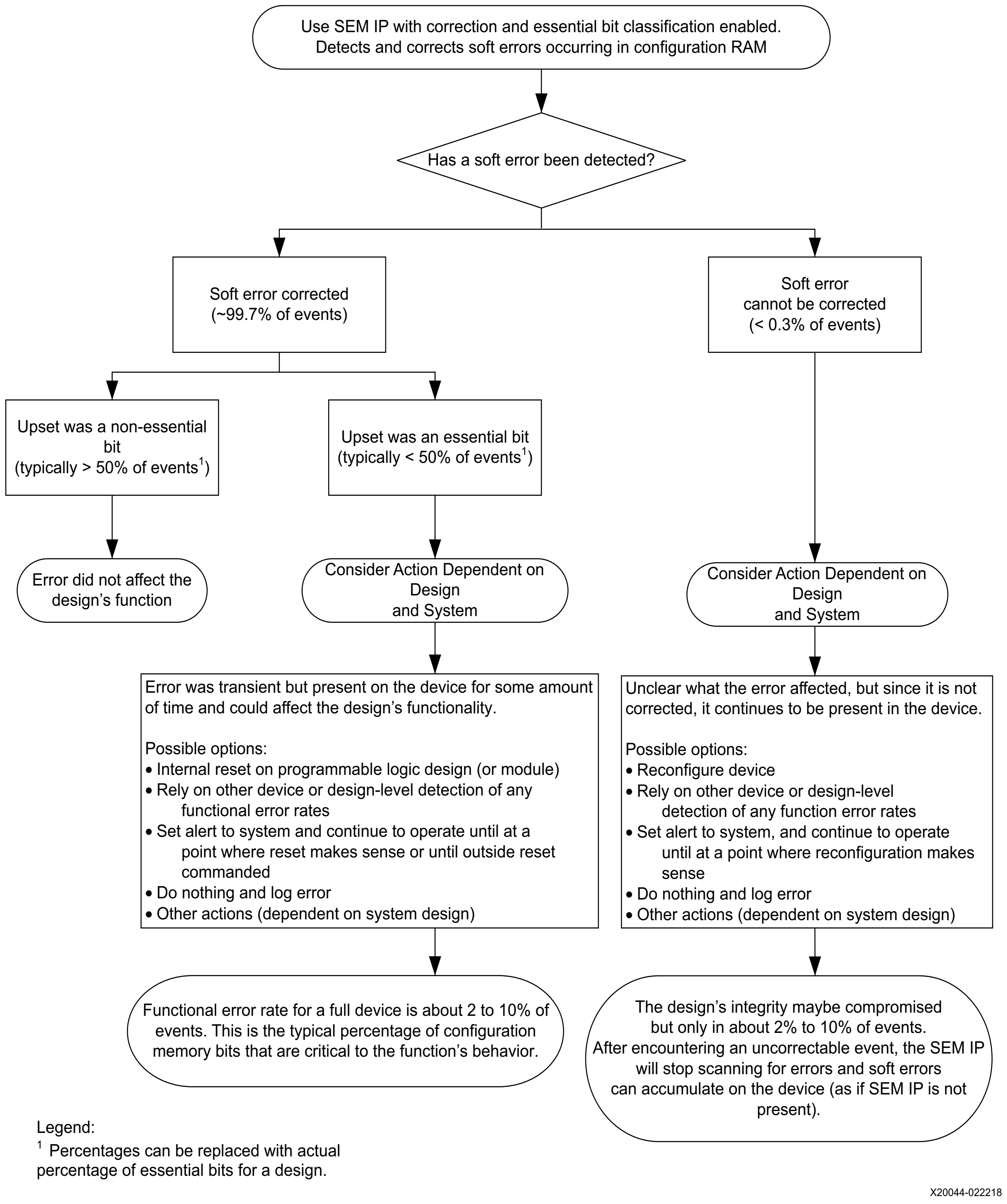

Use SEM IP with correction and essential bit classification enabled.
Detects and corrects soft errors occurring in configuration RAM
Has a soft error been detected?
Soft error corrected (~99.7% of events)
Soft error cannot be corrected (< 0.3% of events)
Upset was a non-essential bit (typically > 50% of events¹)
Upset was an essential bit (typically < 50% of events¹)
Error did not affect the design's function
Consider Action Dependent on Design and System
Consider Action Dependent on Design and System
Error was transient but present on the device for some amount of time and could affect the design's functionality.
Possible options:
• Internal reset on programmable logic design (or module)
• Rely on other device or design-level detection of any functional error rates
• Set alert to system and continue to operate until at a point where reset makes sense or until outside reset commanded
• Do nothing and log error
• Other actions (dependent on system design)
Unclear what the error affected, but since it is not corrected, it continues to be present in the device.
Possible options:
• Reconfigure device
• Rely on other device or design-level detection of any function error rates
• Set alert to system, and continue to operate until at a point where reconfiguration makes sense
• Do nothing and log error
• Other actions (dependent on system design)
Functional error rate for a full device is about 2 to 10% of events. This is the typical percentage of configuration memory bits that are critical to the function's behavior.
The design's integrity maybe compromised but only in about 2% to 10% of events.
After encountering an uncorrectable event, the SEM IP will stop scanning for errors and soft errors can accumulate on the device (as if SEM IP is not present).
Legend:
¹ Percentages can be replaced with actual percentage of essential bits for a design.
X20044-022218

Figure 1-1: **Decision Tree When a Soft Error is Detected**

Partial Reconfiguration Support

For UltraScale devices, SEM IP and Partial Reconfiguration (PR) are supported only when monolithic devices are used. When using stacked silicon interconnect (SSI) devices, SEM IP and PR are not supported.

For UltraScale+ devices, SEM IP and Partial Reconfiguration is supported with both monolithic and SSI devices.

Although the implementation of a design that contains SEM IP and Partial Reconfiguration is outside the scope of this document, there is an application note that can provide guidance. See *Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolithic Devices* (XAPP1261) [Ref 6]. While XAPP1261 demonstrates compatibility of SEM IP and monolithic devices, the same guidance can be used for UltraScale+ SSI devices.

Encryption and Authentication Support

In UltraScale devices, private key encryption of the bitstream using AES-GCM-256 (a self-authenticating AES algorithm) is supported with the SEM IP. Public key authentication using RSA-2048 is supported with the SEM IP.

In UltraScale+ FPGAs, private key encryption of the bitstream using AES-GCM-256 (a self-authenticating AES algorithm) has been verified and is supported with the SEM IP. Public key authentication using RSA-2048 has also been verified and is supported with the SEM IP.

In UltraScale+ MPSoC, private key encryption of the bitstream using AES-GCM-256 (a self-authenticating AES algorithm) has been verified and is supported with the SEM IP. Public key authentication using RSA-4096 has also been verified and is supported with the SEM IP.

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 SelectMAP persistence is not supported by the controller.
- For UltraScale architecture, each device die or super logic region (SLR) requires a single instance of the SEM controller and ICAP. The ICAP must be placed in the primary/top physical location in that die. If another logic requires access of the configuration memory through the ICAP, the access of the ICAP must be MUXed and shared. Do NOT use BITSTREAM.Readback_ICAP_Select or ICAP_AUTO_SWITCH because these methods are used for switching between the top/bottom ICAP. They will not MUX or share the ICAP with other logic.
- The SEM controller does not operate when a golden or fallback bitstream is loaded by a configuration error and fallback condition from a SPI/BPI flash. See AR 67645.
- If there are multiple logics accessing the configuration memory, an arbitration logic must be created to manage access to this memory. See ICAP Arbitration Interface.
- The SEM IP was not developed nor tested for use in space radiation environments. Therefore Xilinx does not support or answer questions specific to the use of this IP in this environment. If you choose to use the SEM IP in space radiation environments, do so at your own risk.

Licensing and Ordering

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

Information about 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 is device dependent and includes the following features:

- Six IP modes to align with your SEU mitigation goals:
  - Mitigation and Testing
  - Mitigation only
  - Detect and Testing
  - Detect 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.

    - Provision to command the SEM controller to perform Detect only monitoring of soft errors. In this state, the SEM controller continuously monitors the configuration memory until it detects an error. After an error is detected, the SEM controller reports the error and goes to the Idle state. This command does not include error correction.

- Variety of debug and test features 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 the configuration of the Physical Frame Address (PFA) to the Linear Frame Address (LFA) and vice-versa.
    - External SPI flash memory reads (`Xmem` command) available when Error Classification feature is enabled.

- Option to continuously monitor and report errors without performing error correction (Detect modes and Detect only command). In this state, the SEM controller transitions to Idle state when it detects and reports the first error it encounters.
- Provision to command the SEM controller to perform a Diagnostic Scan. In this state, the SEM controller performs a single scan of the configuration memory and reports all detected ECC errors. After completing a single pass of the configuration memory, the SEM controller returns to the Idle state. The error detection mechanism used in this feature does not leverage the built-in error detection capability of the device. Error correction is not performed in this type of scan.
- 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 features of each of the modes.

Table 2-1: **Mode Features**

Features	Modes					
	Mitigation and Testing	Mitigation Only	Detect and Testing	Detect Only	Emulation	Monitoring
IP state after initialization	OBSV	OBSV	DETECT	DETECT	IDLE	IDLE
Correction (Repair)	Yes	Yes	N/A	N/A	N/A	N/A
Classification	Optional	Optional	N/A	N/A	N/A	N/A
Error injection	Yes	N/A	Yes	N/A	Yes	N/A
Debugging features: • Transition to Idle state • Configuration frames and register reads • External memory reads • Address translations	Yes	Yes	Yes	Yes	Yes	Yes
On-demand detect features: • Detect only • Diagnostic scan	Yes	Yes	Yes	Yes	Yes	Yes

Table 2-2 lists all the supported devices and the maximum number of configuration frames it scans, which is also equivalent to the maximum number of linear frames that is reported by the Status command (MF {8-digit hex value}).

Table 2-2: **Maximum Number of Configuration Frames**

Device		MF (Dec)	MF (Hex)
UltraScale	XCKU035	26179	00006643
	XCKU040	26179	00006643
	XCKU060	37651	00009313
	XCKU085	37651	00009313
	XCKU095	54559	0000D51F
	XCKU115	37651	00009313
	XCVU065	37706	0000934A
	XCVU080	54559	0000D51F
	XCVU095	54559	0000D51F
	XCVU125	37706	0000934A
	XCVU160	37706	0000934A
	XCVU190	37706	0000934A
	XCVU440	78555	000132DB

Table 2-2: **Maximum Number of Configuration Frames *(Cont'd)***

Device		MF (Dec)	MF (Hex)
UltraScale+	XCZU1EG	5684	00001634
	XCZU1CG	5684	00001634
	XQZU55DR	40794	00009F5A
	XQZU57DR	40794	00009F5A
	XQZU65DR	40794	00009F5A
	XQZU67DR	40794	00009F5A
	XCKU15P	75283	00012613
	XCVU3P	56601	0000DD19
	XCVU5P	56601	0000DD19
	XCVU7P	56601	0000DD19
	XCVU9P	56601	0000DD19
	XCVU11P	61963	0000F20B
	XCVU13P	61963	0000F20B
	XCVU27P	61963	0000F20B
	XCVU29P	61963	0000F20B
	XCVU31P	61929	0000F1E9
	XCVU33P	61929	0000F1E9
	XCVU35P (SLR0)	61929	0000F1E9
	XCVU35P (SLR1)	61963	0000F20B
	XCVU37P (SLR0)	61929	0000F1E9
	XCVU37P (SLR1 and SLR2)	61963	0000F20B
	XCVU45P (SLR0)	61929	0000F1E9
	XCVU45P (SLR1)	61963	0000F20B
	XCVU47P (SLR0)	61929	0000F1E9
	XCVU47P (SLR1 and SLR2)	61963	0000F20B
	XCZU1	10373	00002885
	XCZU2	10373	00002885
	XCZU3	10373	00002885
	XCZU4	17907	000045F3
	XCZU5	17907	000045F3
	XCZU6	48054	0000BBB6
	XCZU7	44273	0000ACF1
	XCZU9	48054	0000BBB6
	XCZU11	49127	0000BFE7
	XCZU15	57532	0000E0BC

Table 2-2: **Maximum Number of Configuration Frames *(Cont'd)***

Device		MF (Dec)	MF (Hex)
UltraScale+ (continued)	XCZU17	75283	00012613
	XCZU29DR	68119	00010A17
	XCZU46DR	68119	00010A17
	XCZU47DR	68119	00010A17
	XCZU48DR	68119	00010A17
	XCZU49DR	68119	00010A17
	U55N (SLR1)	61963	0000F20B
	U55N (SLR0)	61929	0000F1E9
	U55C (SLR1 and SLR2)	61963	0000F20B
	U55C (SLR0)	61929	0000F1E9
	XCVU57P (SLR1 and SLR2)	61963	0000F20B
	XCVU57P (SLR0)	61929	0000F1E9
	XCVU19P	122412	0001DE2C
	XCVU15P	122412	0001DE2C
	XCVU23P	130547	0001FDF3
	XCK26	17907	000045F3
	XCUX35	130547	0001FDF3
	XCAU25P	31287	00007A37
	XCAU20P	31287	00007A37
	XCAU15P	31287	00007A37
	XCAU10P	31287	00007A37

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 Frequencies

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 device data sheet as configuration interface AC timing parameter F_{ICAPCK}. Table 2-3 provides a summary of ICAP F_{Max} values.

Table 2-3: **ICAP Maximum Frequencies**

Device		ICAP F_{Max} (MHz)
UltraScale	Kintex	200
	Virtex	200
	Kintex® SSI	200
	Virtex® SSI	200
	All Devices (0.9V, -1L)	175
UltraScale+	Kintex	200
	Virtex	200
	Virtex SSI	125
	Zynq® UltraScale+™ devices	200
	All Devices (0.72V, -1L, -2L)	150

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. Virtual Input Output (VIO) core is specifically excluded from analysis, as it is unlikely a production design includes this interactive debug and experimentation capability. As a result, the estimate represents an upper bound.

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 2-4.

Table 2-4: **Maximum Estimated FIT Rate**

Device	FIT
UltraScale Monolithic devices	9
UltraScale KU115 (SSI example)	23
UltraScale+ Monolithic devices	3
UltraScale+ SSI devices	6

The estimations above includes the contribution of the configuration RAM and block RAM used by the SEM controller and its system-level design example.

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 the processing of FPGA configuration memory frames. Single-bit errors always reside in a single frame. Generally, an *N*-bit error can be 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 2-5 and Table 2-6 illustrate start-up latency, decomposed into sub-steps of boot and initialization.

Table 2-5: **Maximum Start-up Latency at ICAP F_{Max} for UltraScale Devices**

Device		Boot Time at ICAP F_{Max} (ms)	Initialization Time at ICAP F_{Max} (ms)
UltraScale	XCKU035	127	52
	XCKU040(1)	127	52
	XCKU060	127	75
	XCKU085	127	75
	XCKU095	127	109
	XCKU115	127	75
	XCVU065	127	75
	XCVU080	127	109
	XCVU095	127	109
	XCVU125	127	75
	XCVU160	127	75
	XCVU190	127	75
	XCVU440	127	156

Notes:

1. Measured in hardware. Data for other devices in this architecture are extrapolated based on this measurement.

Table 2-6: **Maximum Start-up Latency at ICAP F_{Max} for UltraScale+ Devices**

Device		Boot Time at ICAP F_{Max} (ms)	Initialization Time at ICAP F_{Max} (ms)
UltraScale+	XCKU3P	127	46
	XCKU5P	127	46
	XCKU9P	127	71
	XCKU11P	127	73
	XCKU13P	127	85
	XCKU15P	127	111
	XCVU3P(1)	127	84
	XCVU5P	204	137
	XCVU7P	204	137
	XCVU9P(1)	204	137
	XCVU11P	204	150
	XCVU13P	204	153
	XCVU27P	204	150
	XCVU29P	204	150
	XCVU31P	127	97
	XCVU33P	127	97
	XCVU35P	204	150
	XCVU37P	204	150
	XCVU45P	204	150
	XCVU47P	204	150
	XCZU1	127	16
	XCZU2	127	16
	XCZU3	127	16
	XCZU4	127	27
	XCZU5	127	27
	XCZU6	127	71
	XCZU7	127	66
	XCZU9	127	71
	XCZU11	127	73
	XCZU15	127	85
	XCZU17	127	111
	XCZU19	127	111

Table 2-6: **Maximum Start-up Latency at ICAP F_{Max} for UltraScale+ Devices *(Cont'd)***

Device		Boot Time at ICAP F_{Max} (ms)	Initialization Time at ICAP F_{Max} (ms)
UltraScale+ (Continued)	XCZU21DR	127	106
	XCZU25DR	127	106
	XCZU27DR	127	106
	XCZU28DR	127	106
	XCZU29DR	127	106
	XCZU46DR	127	106
	XCZU47DR	127	106
	XCZU48DR	127	106
	XCZU49DR	127	106
	U55N	204	150
	U55C	204	150
	XCVU57P	204	150
	XCVU19P	204	153
	XCVU15P	204	153
	XCVU23P	127	97
	XCK26	127	27
	XCUX35	127	97
	XCAU25P	127	46
	XCAU20P	127	46
	XCAU15P	127	46
	XCAU10P	127	46

Notes:

1. Measured in hardware. Data for other devices in this architecture are extrapolated based on this measurement.

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 2-5, Table 2-6, and Equation 2-1.

$$StartUpLatency_{ACTUAL} = StartUpLatency_{ICAP\_F_{Max}} \cdot \left[\frac{ICAP\_F_{Max}}{Frequency_{ACTUAL}}\right] \qquad \textit{Equation 2-1}$$

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 2-7 and Table 2-8 illustrate IP error detection time. These numbers are only applicable when the IP is in the mitigation mode (detect and correct), detect mode, and detect only state.

Table 2-7: **Maximum IP Error Detection Times at ICAP F_{Max} for UltraScale Devices**

Device		Detection Time at ICAP F_{Max} (ms)
UltraScale	XCKU035	22
	XCKU040[1]	22
	XCKU060	30
	XCKU085	30
	XCKU095	41
	XCKU115	30
	XCVU065	30
	XCVU080	41
	XCVU095	41
	XCVU125	30
	XCVU160	30
	XCVU190	30
	XCVU440	58

Notes:

1. Measured in hardware. Data for other devices in this architecture are extrapolated based on this measurement.

Table 2-8: **Maximum IP Error Detection Times at ICAP F_{Max} for UltraScale+ Devices**

Device		Detection Time at ICAP F_{Max} (ms)
UltraScale+	XCKU3P	19
	XCKU5P	19
	XCKU9P	28
	XCKU11P	29
	XCKU13P	33
	XCKU15P	42
	XCVU3P(1)	32
	XCVU5P	52
	XCVU7P	52
	XCVU9P(1)	52
	XCVU11P	57
	XCVU13P	57
	XCVU27P	57
	XCVU29P	57
	XCVU31P	37
	XCVU33P	37
	XCVU35P	57
	XCVU37P	57
	XCVU45P	57
	XCVU47P	57
	XCZU1	9
	XCZU2	9
	XCZU3	9
	XCZU4	13
	XCZU5	13
	XCZU6	28
	XCZU7	26
	XCZU9	28
	XCZU11	29
	XCZU15	33
	XCZU17	42
	XCZU19	42

Table 2-8: **Maximum IP Error Detection Times at ICAP F_{Max} for UltraScale+ Devices *(Cont'd)***

Device		Detection Time at ICAP F_{Max} (ms)
UltraScale+ (Continued)	XCZU21DR	40
	XCZU25DR	40
	XCZU27DR	40
	XCZU28DR	40
	XCZU29DR	40
	XCZU46DR	40
	XCZU47DR	40
	XCZU48DR	40
	XCZU49DR	40
	U55N	57
	U55C	57
	XCVU57P	57
	XCVU19P	57
	XCVU15P	57
	XCVU23P	37
	XCK26	13
	XCUX35	37
	XCAU25P	19
	XCAU20P	19
	XCAU15P	19
	XCAU10P	19

Notes:

1. Measured in hardware. Data for other devices in this architecture are extrapolated based on this measurement.

The IP error detection time for the target device, at the actual frequency of operation, can be estimated using data from Table 2-7, Table 2-8, and Equation 2-2.

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

The error detection latency can be bounded as follows:

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

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 is to multiply the number of single-bit error with the worst case detection time for a single-bit error.

The error detection latency when using the Diagnostic Scan feature is significantly larger than the mitigation mode, detect mode, or detect only scan and is dependent on the device, number of errors detected, and the actual frequency of operation.

For example, if a Diagnostic Scan is performed on a KU040 device that has three errors resident in its configuration memory using a 90 MHz clock, it takes 70 seconds to scan and report all errors.

Error Correction Latency

After detecting an error, the solution attempts correction. Errors are correctable depending on the selected correction mode and error type. Table 2-9 provides error correction latency for a configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-9: **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[2]
		Uncorrectable	21[2]
	Any	CRC-only (Uncorrectable)	9
UltraScale+ Monolithic	Repair	Correctable[1]	44[3]
		Uncorrectable	22[3]
	Any	CRC-only (Uncorrectable)	9
UltraScale+ SSI	Repair	Correctable[1]	99[4]
		Uncorrectable	38[4]
	Any	CRC-only (Uncorrectable)	15

Notes:

1. IP could correct up to four bits of error in frame depending on its physical adjacency.
2. Measured in hardware for XCKU040 device.
3. Measured in hardware for XCVU3P device.
4. Measured in hardware for XCVU9P device.

The error correction latency at the actual frequency of operation can be estimated using data from Table 2-9 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 2-10 provides error classification latency for a configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-10: **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(1)
	Uncorrectable	Any	5(1)
UltraScale+ Monolithic	Correctable	Enabled	154(2)
	Uncorrectable	Any	5(2)
UltraScale+ SSI	Correctable	Enabled	247(3)
	Uncorrectable	Any	30(3)

Notes:
1. Measured in hardware for XCKU040 device.
2. Measured in hardware for XCVU3P device.
3. Measured in hardware for XCVU9P device.

The error classification latency at the actual frequency of operation can be estimated using data from Table 2-10 and Equation 2-4.

$$ClassificationLatency_{ACTUAL} = ClassificationLatency_{ICAP\_F_{Max}} \cdot \left[\frac{ICAP\_F_{Max}}{Frequency_{ACTUAL}}\right] \qquad \textit{Equation 2-4}$$

Error Injection Latency

Table 2-11 provides error injection latency for a 1-bit configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-11: **Error Injection Latency When Using Linear Frame Addressing, No Throttling on Monitor Interface**

Device Family	Error Injection Latency at ICAP_F_{Max} (µs)
UltraScale KU040	50(1)
UltraScale+ VU3P	81(1)
UltraScale+ SSI VU9P	164(1)

Notes:
1. Measured in hardware.

The error injection latency value given in Table 2-11 changes based on the clock frequency and the number of the configuration frame in the device.

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.

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

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

Performing error injection using Linear Frame Addressing also adds additional latency because the controller needs to first translate this address to a Physical Frame Address. The latency for this translation is also dependent on where the error occurred—the larger the address the longer it takes to translate.

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.

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

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

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

$$MitigationLatency = 48.889ms + 0.091ms + 0.011ms = 48.991ms \qquad \text{Equation 2-9}$$

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 9,600 baud to 460,800 baud. The standard 8-N-1 protocol used requires 10-bit times on the serial link to transmit a 1-byte payload:

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

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 2-12: **Device Utilization – Monolithic Kintex and Virtex UltraScale and UltraScale+ Devices (Non-SSI)[(1)(2)]**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs	DSP48
UltraScale (All devices)	Complete solution in Mitigation and Testing mode with no optional features.	425	490	59	4 RAMB36	1

Table 2-12: **Device Utilization – Monolithic Kintex and Virtex UltraScale and UltraScale+ Devices (Non-SSI)[(1)(2)] *(Cont'd)***

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs	DSP48
UltraScale+ Monolithic (All devices)	Complete solution in Mitigation and Testing mode with no optional features.	430	530	64	4 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 Vivado® IDE option where the mode is set to Mitigation and Testing and error classification is disabled.
2. The Vivado Design Suite debug feature IPs delivered in the top-level example design is not included. Using the Vivado Design Suite debug feature increases LUTs/FFs, but decreases I/Os.

Table 2-13: **Device Utilization – Multi-SLR UltraScale and 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	8 RAMB36 and 2 RAMB18	2
VU9P	Complete solution in Mitigation and Testing mode with no optional features.	514	653	66	6 RAMB36	1

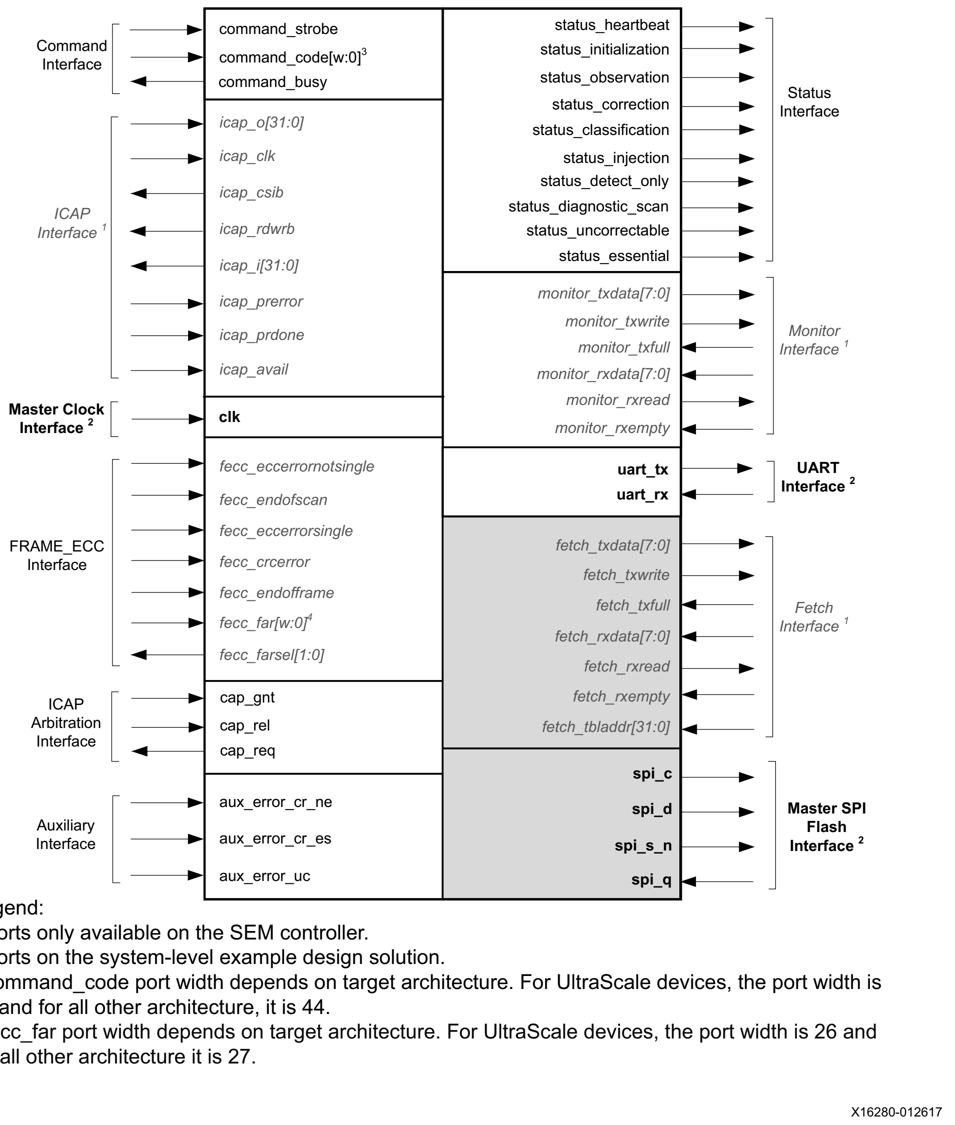
Notes:

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 Vivado IDE option where the mode is set to Mitigation and Testing and error classification is disabled.
2. The Vivado Design Suite debug feature IPs delivered in the top-level example design is not included. Using the Vivado Design Suite debug feature 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.

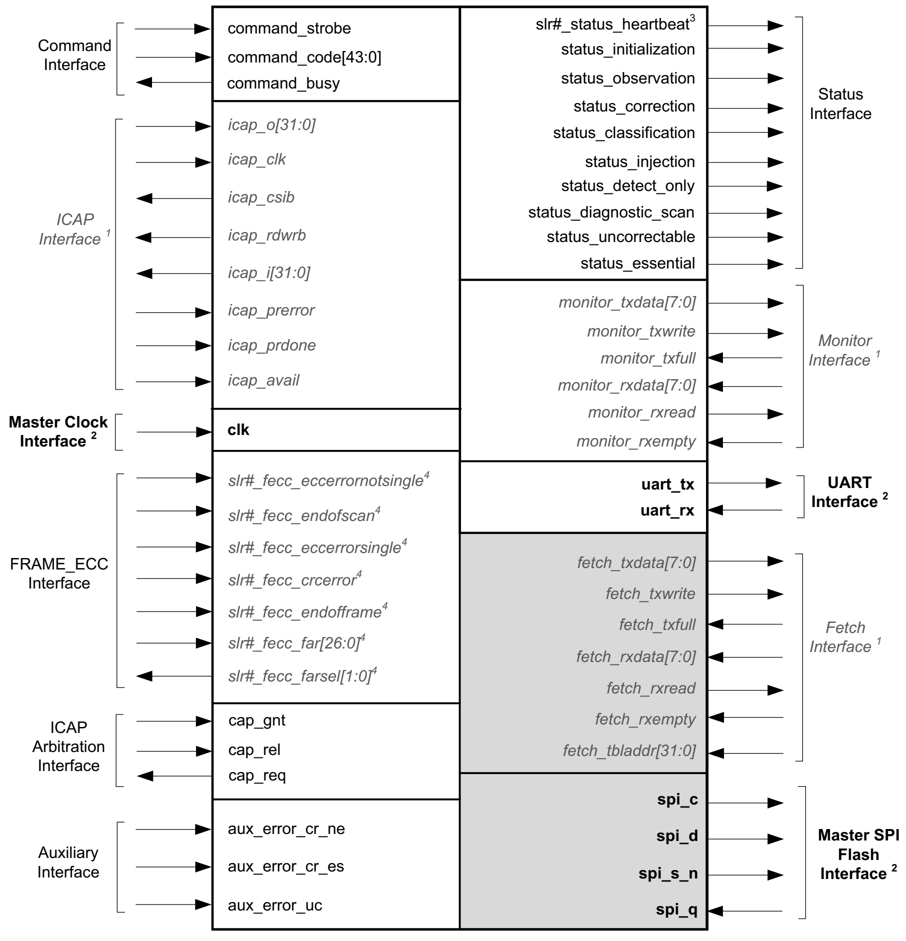
Figure 2-1 shows the SEM controller and the system-level example design ports for all UltraScale and monolithic UltraScale+ devices. Shading indicates port groups that only exist when error classification is enabled. Unless indicated, ports are available at the core and all levels of the example design hierarchy (see Figure 5-1).



Legend:
[1] Ports only available on the SEM controller.
[2] Ports on the system-level example design solution.
[3] command_code port width depends on target architecture. For UltraScale devices, the port width is 40 and for all other architecture, it is 44.
[4] fecc_far port width depends on target architecture. For UltraScale devices, the port width is 26 and for all other architecture it is 27.

Figure 2-1: **SEM Controller Ports for UltraScale and Monolithic UltraScale+ Devices**

Figure 2-2 shows the SEM controller and the system-level example design ports for UltraScale+ SSI devices. 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.
[3] There is a status_heartbeat port for each SLRs in a target device.
[4] SSI implementation has a FRAME_ECC interface for each SLR.

X18677-012617

Figure 2-2: **SEM Controller Ports for UltraScale+ SSI Devices**

The SSI implementation between UltraScale and UltraScale+ are unique. In UltraScale, the soft error mitigation function is executed separately in each SLR. There is one SEM controller per SLR, connected to the ICAP and FRAME_ECC in that SLR. In UltraScale+, the function is centralized to one controller. There is one SEM controller regardless of the number of SLRs in a device. The SEM controller is connected to one ICAP in the master SLR and a FRAME_ECC from each SLR. See Chapter 5, Example Design.

Table 2-14 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 2-14: **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. For UltraScale+ SSI devices, there is a FRAME_ECC interface for each SLR in the device.	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. For UltraScale+ SSI devices, there is a separate status_heartbeat signal for each SLR in the device.	SEM controller	SEM controller
Monitor	Interface provides a mechanism to interact with the controller that also provides comprehensive information about the controller behavior and current state. This interface is the preferred method for system interaction with the controller.	SEM controller	SEM controller

Table 2-14: **Interfaces with Hierarchy Level *(Cont'd)***

Interfaces	Description	Hierarchy Level	
		Configuration Primitive Location = Example Design	Configuration Primitive Location = Core
UART	Interface to the UART helper block that serializes and de-serializes the byte-stream ASCII codes used by Monitor Interface. This interface is the preferred method for system interaction with the controller.	Support level	Support wrapper level
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 set reset (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 2-15: **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.

Table 2-15: **ICAP Interface Signals *(Cont'd)***

Name	Sense	Direction	Description
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.
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 cell. This interface is available at the support wrapper level.

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

Table 2-16: **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 manage the SEM controller access to the ICAP or share the ICAP with other blocks and ensure safer hand-off of the ICAP controls. Ideally, this interface is used with a user-defined ICAP arbiter.

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

Table 2-17: **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. For UltraScale+ SSI devices, there are multiple FRAME_ECC Interfaces, one for each SLR in the device. The name of these FRAME_ECC Interface signals are pre-appended with "slr<#>_". The SLR number indicates which hardware SLR the FRAME_ECC primitive should be placed.

Table 2-18: **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. When targeting UltraScale devices, the variable far_width is equal to 26. For all other architecture, the variable far_width is equal to 27.

Status Interface

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

For UltraScale+ SSI devices, there is a `status_heartbeat` signal for each SLR in the device. The name of these signals are pre-appended with "slr<#>_".

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

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

Name	Sense	Direction	Description
status_heartbeat	High	Out	The heartbeat signal is active while status_observation, status_detect_only, or status_diagnostic_scan are asserted. This output issues a single cycle high pulse every time a single configuration frame is read. 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, status_detect_only, and status_diagnostic_scan is deasserted, the behavior of the heartbeat signal is unspecified. For UltraScale+ SSI solution, status_heartbeat for all SLRs are only active when in status_observation, status_detect_only, or status_diagnostic_scan are asserted, identical to the behavior of a monolithic solution.
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.
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_detect_only	High	Out	The detect only signal is active when the controller is executing a detect only scan. When the scan is interrupted due to a detected error, the controller transitions to Idle state and this signal returns inactive.

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

Name	Sense	Direction	Description
status_diagnostic_scan	High	Out	The diagnostic scan signal is active when the controller is executing a diagnostic scan. When the single-pass diagnostic scan of the entire device configuration memory completes, the controller transitions to the Idle state and this signal returns inactive. This feature should only be used as a diagnostic tool and should not be used for real-time soft error mitigation.
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. This signal is sticky and is only updated when the controller encounters another error and classifies it.
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. In IDLE, all `status_heartbeat` stops and in Observation, Diagnostic Scan, and Detect only, all `status_heartbeat` toggles.

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_detect_only`, `status_diagnostic_scan`, `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 seven controller state outputs. If all seven signals are Low, the controller is idle (inactive but ready to resume). If all seven 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 2-20: **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[width – 1:0]	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. The width of the command_code port is dependent on the target architecture. For UltraScale devices, the width is 40. For all other architecture, the width is 44.
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 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.

Note: The Monitor Interface is the comprehensive and preferred interface for user interaction with the IP. See Monitor 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 2-21: **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.
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 2-22: **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 2-23: **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 that connects to the Fetch Interface through a 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 Interface 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 2-24: **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.
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 monitoring of the configuration memory. For more information, see Auxiliary Interface in Chapter 3.

Table 2-25: **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 124.

For designs targeting stacked silicon interconnect (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, 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 51.*

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 alters and corrupts 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.

Integrate and Validate Early

The SEM IP core has a small programmable logic footprint, but activates the programmable logic configuration memory system. Integrate and validate your system with the SEM IP as early as possible and incrementally include as much functionality as practical that provides the most time for system evaluation under representative workloads. This recommendation complements the commonly used bottom-up design approach, facilitating design reuse and IP-based design.

For more information, see Integration and Validation, page 120.

Review the IP Checklist

For ease of use, a checklist is available that summarizes all the considerations you should evaluate when using the SEM IP. See Appendix F, IP Design Checklist.

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 Integrated Design Environment (IDE).

Note: For UltraScale implementations targeting SSI devices, only the configuration primitive included in the example design is supported.

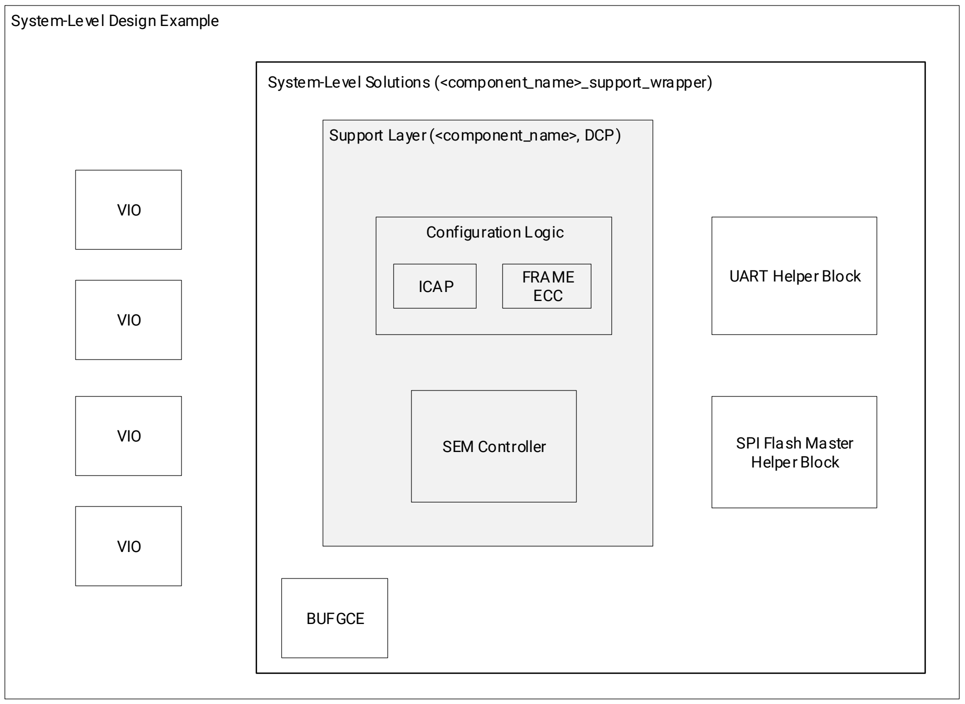

System-Level Design Example
System-Level Solutions (<component_name>_support_wrapper)
Support Layer (<component_name>, DCP)
Configuration Logic
ICAP
FRAME ECC
SEM Controller
UART Helper Block
SPI Flash Master Helper Block
VIO
VIO
VIO
VIO
BUFGCE
X25731-091321

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

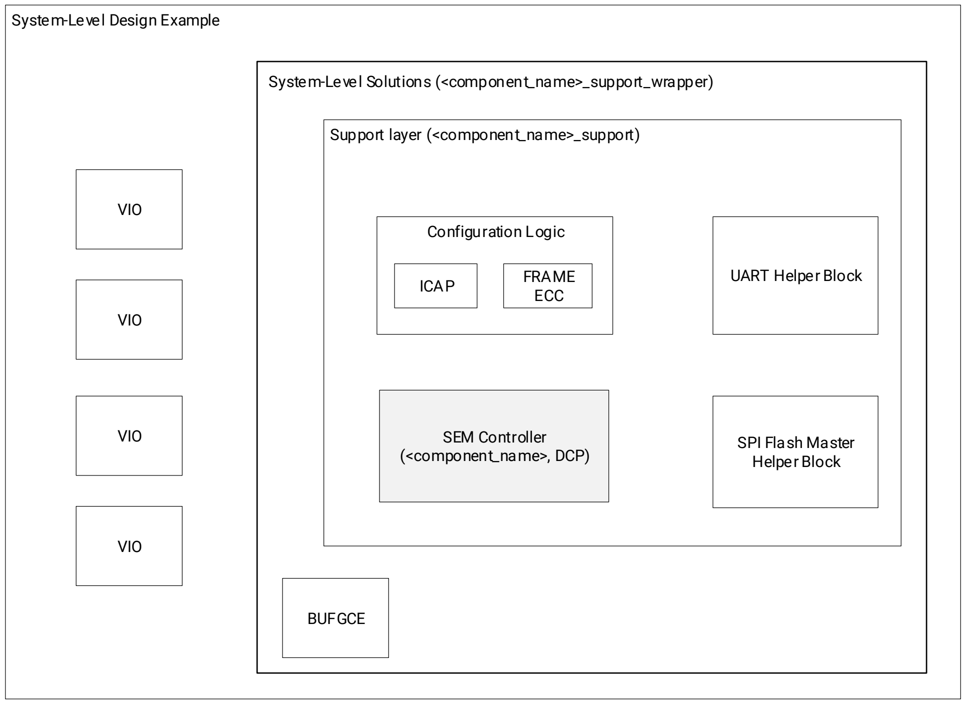

System-Level Design Example
System-Level Solutions (<component_name>_support_wrapper)
Support layer (<component_name>_support)
Configuration Logic
ICAP
FRAME ECC
UART Helper Block
SEM Controller (<component_name>, DCP)
SPI Flash Master Helper Block
VIO
VIO
VIO
VIO
BUFGCE
X25732-091321

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.

Note: This option is not available for designs targeting UltraScale SSI devices.

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.

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 (ICAP) 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 and the system-level design example is used:

- If the out-of-the-box classification feature using a dedicated SPI flash to store classification data is used:
  - The SPI bus timing budget must be evaluated to determine the maximum SPI bus clock frequency. For more information on a sample analysis, see Appendix G, SPI Bus Timing Budget.
  - The SPI bus clock is the input clock divided by two; therefore, the input clock cannot exceed twice the maximum SPI bus clock frequency.
- If the UART helper block and UART interface is used for communicating with the controller:
  - 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 is provided in Switching Behavior.

System-Level Requirements

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 delay-locked loop (DLL) or phase-locked loop (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.

For UltraScale+™ SSI devices, each SLR has its own FRAME_ECC interface and each of these interfaces must be connected to the FRAME_ECC primitive that is placed in that SLR.

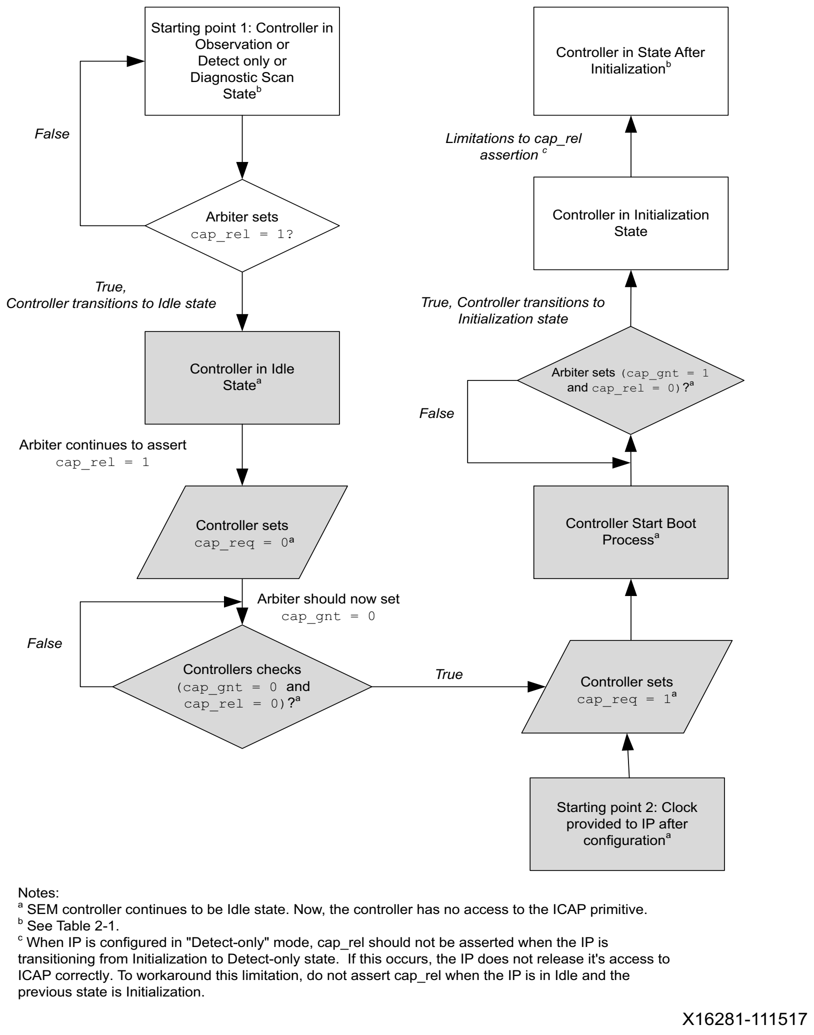
ICAP Arbitration Interface

This interface is provided as a convenience for use in systems where the ICAP primitive is shared between multiple functions or when there is a need to manage the SEM controller access to the ICAP. Ideally, this interface should be driven by a user-defined ICAP arbiter that controls when the different functions have ICAP control and when the SEM controller has access to the ICAP. For more information of the mechanism to sharing the ICAP with the SEM controller, contact Xilinx.

Figure 3-3 shows how the SEM controller behaves when the ICAP arbitration signals are manipulated. There are two potential starting points:

- Controller is in an Observation, Detect only, or Diagnostic Scan state (Starting point 1 in the diagram)
- Controller is being brought up the first time after configuration (Starting point 2 in the diagram)

The white blocks indicates states in which the IP expects to have sole access to the ICAP. The shaded blocks indicates the states in which the IP is not accessing the ICAP.



Starting point 1: Controller in Observation or Detect only or Diagnostic Scan State[b]
False
Arbiter sets cap_rel = 1?
True, Controller transitions to Idle state
Controller in Idle State[a]
Arbiter continues to assert cap_rel = 1
Controller sets cap_req = 0[a]
Arbiter should now set cap_gnt = 0
False
Controllers checks (cap_gnt = 0 and cap_rel = 0)?[a]
True
Controller sets cap_req = 1[a]
Starting point 2: Clock provided to IP after configuration[a]
Controller Start Boot Process[a]
False
Arbiter sets (cap_gnt = 1 and cap_rel = 0)?[a]
True, Controller transitions to Initialization state
Controller in Initialization State
Limitations to cap_rel assertion [c]
Controller in State After Initialization[b]
Notes:
[a] SEM controller continues to be Idle state. Now, the controller has no access to the ICAP primitive.
[b] See Table 2-1.
[c] When IP is configured in "Detect-only" mode, cap_rel should not be asserted when the IP is transitioning from Initialization to Detect-only state. If this occurs, the IP does not release it's access to ICAP correctly. To workaround this limitation, do not assert cap_rel when the IP is in Idle and the previous state is Initialization.
X16281-111517

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

When using the ICAP Arbitration Interface to relinquish the IP control of its ICAP access (by manipulating the `cap_rel` and `cap_gnt` signals), the IP reboots and reinitializes (identical to performing software reset) after the accessed is granted again. Because the IP has no knowledge of whether the configuration memory or settings have been manipulated while its access to ICAP was halted, this is the safest method to ensure that the controller functions correctly after ICAP access is granted back to the IP.

If this behavior is undesirable, you can alternatively perform the following manual steps to halt the IP access to the ICAP:

1. Command controller to Idle state.
2. Take over the ICAP access and perform the desired operation(s).
3. After you have completed the desired operation and relinquished access to ICAP, command the controller back to the state you desire (Observation, Detect only, etc.).

Note: Using these steps does not cause the IP to reboot and initialize but you are responsible to ensure that the configuration memory and setting have not been manipulated while performing your own operations on the ICAP. This is critical to ensure that the SEM controller continues to function correctly.

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 3-4. It starts with the SEM controller booting up. Then it 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 external 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.

While the SEM controller is in the Observation state, 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.

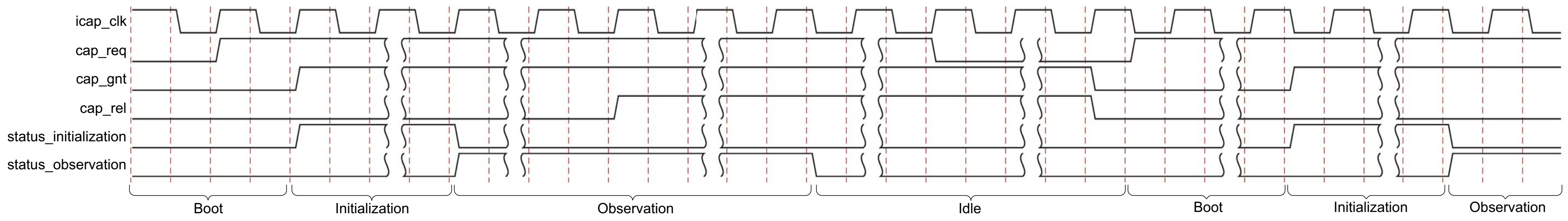


Figure 3-4: **ICAP Arbitration Switching Behavior**

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 configuration memory scanning is active.

For UltraScale SSI implementations of the system-level example design, there is a controller instance on each SLR and therefore an independent Status Interface per SLR.

For UltraScale+ SSI implementations of the system-level example design, there is only one controller instance that is connected to a single ICAP in the master SLR and FRAME_ECC primitives in each SLR. As a result, there is only one Status Interface but there are *n* number of heartbeat signals (where *n* is the number of SLR) since the configuration memory is scanned in each SLR in parallel.

Regardless of the implementation, in majority cases the 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 the 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 UltraScale 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.

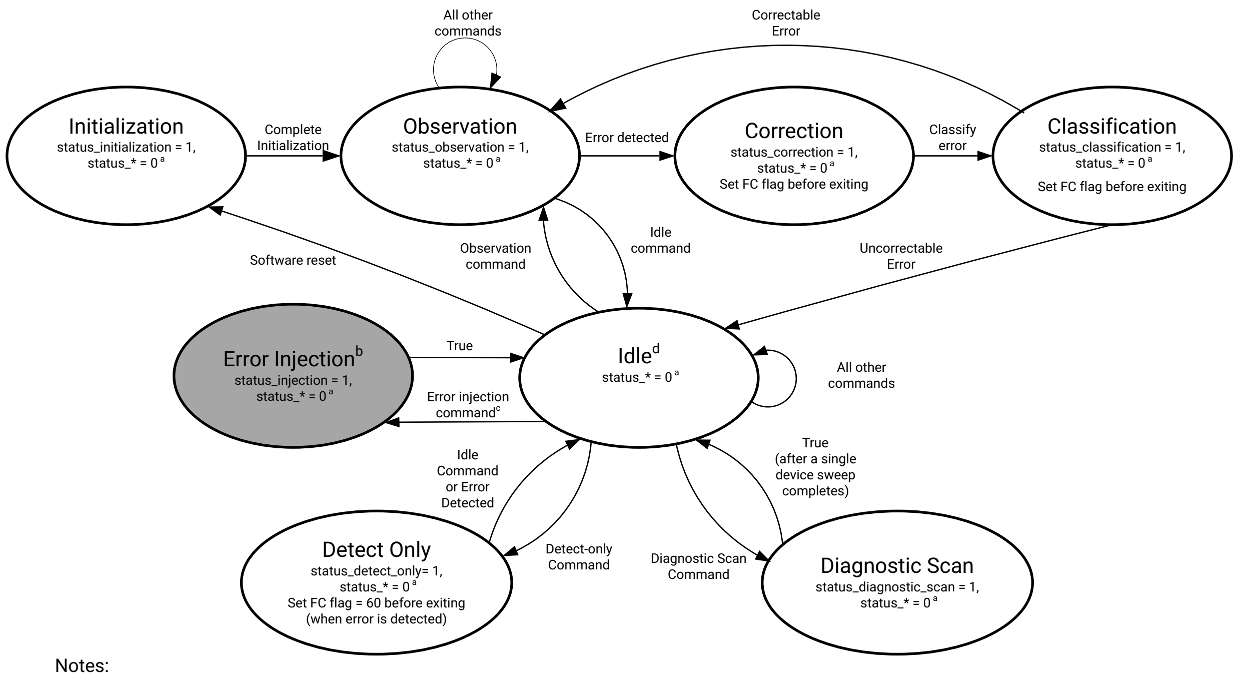
The signals in the Status Interface are generated by sequential logic processes in controllers using the clock supplied to the system-level design example. As a result, the pulse widths are always an integer number of clock cycles.

States

The SEM controller has nine valid states:

- Initialization
- Observation
- Correction
- Classification
- Idle
- Injection
- Detect only
- Diagnostic Scan
- Fatal Error

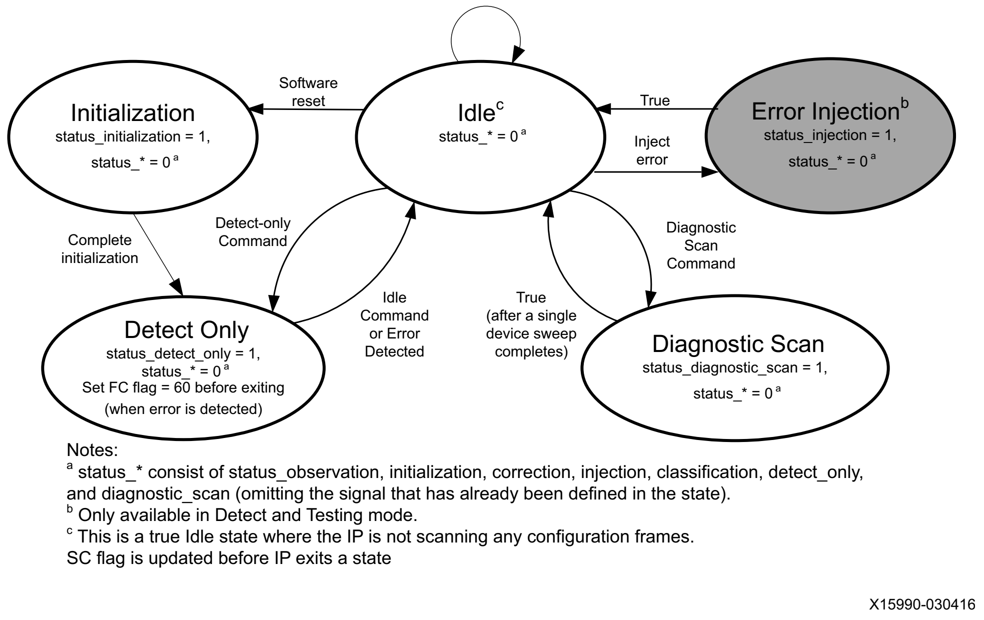
Figure 3-5 through Figure 3-7 illustrate the valid states and state transitions for each IP mode.



Notes:
[a] status_* consists of status_observation, initialization, correction, injection, classification, detect only, and diagnostic_scan (omitting the signal that has already been defined in the state).
[b] Only available in Mitigation and testing mode.
[c] Only SC flag is updated before IP exits a state.
[d] This is a true Idle state where the IP is not scanning any configuration frames.
Transition to Fatal Error state is not shown.

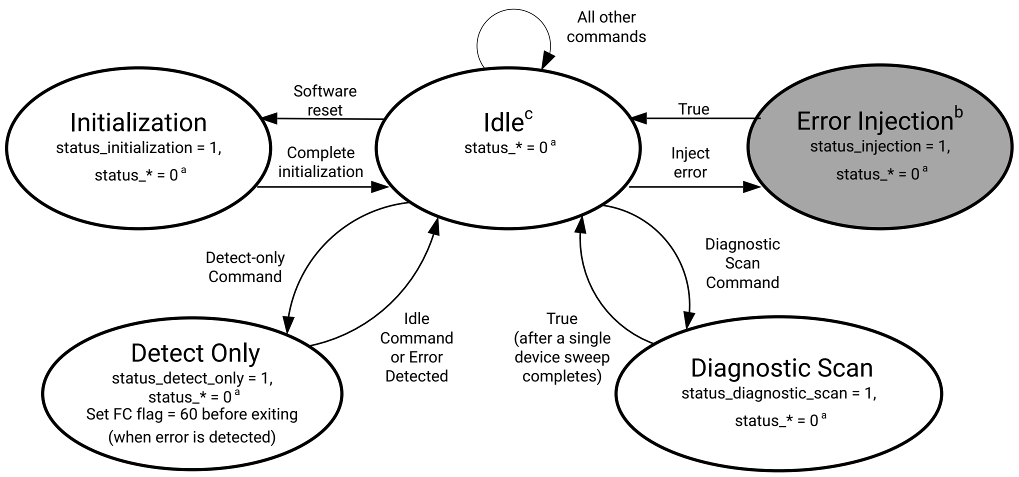
X25733-091321

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



Notes:
[a] status_* consist of status_observation, initialization, correction, injection, classification, detect_only, and diagnostic_scan (omitting the signal that has already been defined in the state).
[b] Only available in Detect and Testing mode.
[c] This is a true Idle state where the IP is not scanning any configuration frames.
SC flag is updated before IP exits a state

Figure 3-6: **Valid State Transition Diagram for Detect Modes**



Notes:
[a] status_* consist of status_observation, initialization, correction, injection, classification, detect_only, and diagnostic_scan (omitting the signal that has already been defined in the state).
[b] Only available in Emulation mode.
[c] This is a true Idle state where the IP is not scanning any configuration frames.
SC flag is updated before IP exits a state
X25734-091321

Figure 3-7: **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 seven 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 external 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, Detect only, or Idle state depending on the IP mode.

Zynq UltraScale+ MPSoC Considerations

Implementations that use the Zynq® UltraScale+ MPSoC devices must provide the SEM Controller access to the ICAP interface. This is performed by clearing the PCAP PR bit in the PS Configuration Security Unit (CSU) `pcap_ctrl` register.

During boot of the Zynq UltraScale+ MPSoC Processing System (PS), access to the configuration logic in the device is given to the PS through the Processor Configuration Access Port (PCAP). This provides a path for the PS bootloader to download a bitstream to the Zynq UltraScale+ MPSoC Programmable Logic (PL). When the PS bootloader is completed, the PS and PCAP remain in control of the configuration logic to support partial reconfiguration of the PL by the PS.

However, while the PS and PCAP are in control of the configuration logic, the PL and ICAP are locked out of the configuration logic. For the SEM controller to function, configuration logic access must be transferred to the ICAP. This is accomplished by clearing the `pcap_pr` (Bit[0]) in the PS CSU `pcap_ctrl` register (`pcap_ctrl`, address `0xFFCA3008`). To confirm that the SEM controller has access to the configuration logic, ICAP_AVAIL can be monitored. If ICAP_AVAIL = 0, a higher priority master such as MCAP, PCAP, or JTAG has control of the configuration logic. If ICAP_AVAIL = 1, the SEM controller has access to the configuration.

For more information regarding PCAP, see the *Zynq UltraScale+ MPSoC Technical Reference Manual* (UG1085) [Ref 3].

When software running on the PS has completed all necessary PCAP activity, it clears PCAP_PR and then sets the GPIO connected to the controller `icap_grant` input, allowing the controller to proceed with initialization. The signal applied to the `icap_grant` input must be properly synchronized to the `icap_clk` signal.

Although the software implementation of this behavior is outside the scope of this document, there are two application notes that can provide you with guidance on how to integrate the SEM controller with a PS. See *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices* (XAPP1298) [Ref 4] and *Integrating LogiCORE SEM IP with AXI in Zynq UltraScale+ Devices* (XAPP1303) [Ref 5].

For detailed information about software development for Baremetal and Linux environments, see the *Zynq UltraScale+ MPSoC Software Developer Guide* (UG1137) [Ref 7] and *OS and Libraries Document Collection* (UG643) [Ref 8].

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, as well as ECC-based ROM and auxiliary errors.

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`. This signal is sticky and does not update until the controller corrects and classifies the next error.*

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 seven 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
Enter detect only	This command is used to direct the controller to perform continuous monitoring of the configuration memory for errors. When an error is detected, it is reported through the monitor interface and the controller transitions to the Idle state.	Command and Monitor Interface
Enter diagnostic scan	This command is used to direct the controller to perform a single scan of the device configuration memory and reports all frame level ECC errors it detects through the monitor interface. After completing a single pass of the configuration memory, the controller automatically transitions to the Idle state.	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.	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 Linear Frame Address (LFA) addresses to Physical Frame Address (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.

Detect Only

The controller executes error detection monitoring in the Detect only state. In this state, the controller continuously monitors the device configuration memory for errors (frame level ECC and device level CRC), as well as ECC-based ROM and auxiliary errors. When an error is detected, the controller reports the error through the Monitor Interface. Then the controller transitions to the Idle state.

Only one command is supported in Detect only state, the Enter Idle command. 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.

The controller enters the Detect only state in two cases:

- **Case 1** – Controller is configured in Detect mode which means that it transitions into the Detect only state once it completes initialization.
- **Case 2** – Controller is commanded to enter the state from Idle state. The command can be given through the Command or Monitor Interface.

Diagnostic Scan

The controller executes a diagnostic scan in the Diagnostic Scan state. In this state, the controller performs a single scan of the device configuration memory and reports all frame level ECC errors it detects through the Monitor Interface. After completing a single pass of the configuration memory, the controller transitions to the Idle state.

The controller does not accept any commands while in the Diagnostic Scan state.

The controller enters the Diagnostic Scan state only when it is commanded to do so from the Idle state. The command can be given through the Command or Monitor Interface.

Injection (Mitigation and Testing, Detect and Testing, or Emulation Modes Only)

The controller performs error injections in the Injection state. When error injection is enabled, the controller only passes through the Injection state in response to a valid error injection command issued from the Idle state. If error injection is disabled or the error injection command is not valid, the controller does not transition to this state. 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 operations due to soft errors that affect the controller-related configuration memory or the controller design state elements. The Fatal Error state can be indicated by the assertion of all seven state bits on the Status Interface, and potentially with a fatal error report message (HLT). 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`, `status_injection`, `status_detect_only`, and `status_diagnostic_scan` are illustrated in Figure 3-8. In the figure, the `status_[state]` signal represents the seven state signals, as a group, which can be considered an indication of the controller state.

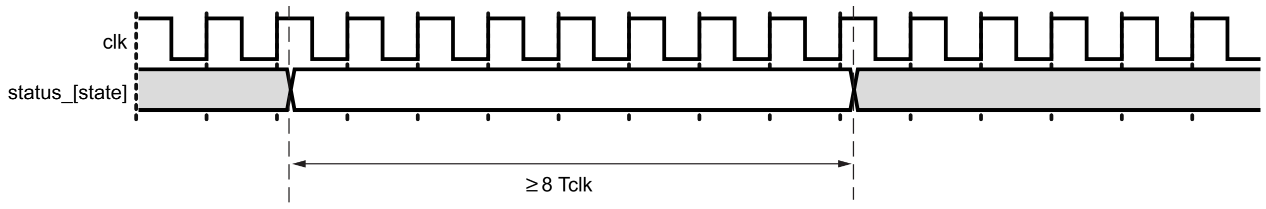


Figure 3-8: **Status Interface State Signals Switching Characteristics**

System-Level Requirements

Monitoring the status signals for completion of the Initialization state and transition to the Observation/Idle/Detect only 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 and are set when the controller is in the correction or classification states, respectively. In Detect modes, both these flags are asserted after an error is detected.

Switching Behavior

In mitigation modes, 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 3-9 and Figure 3-10. 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.

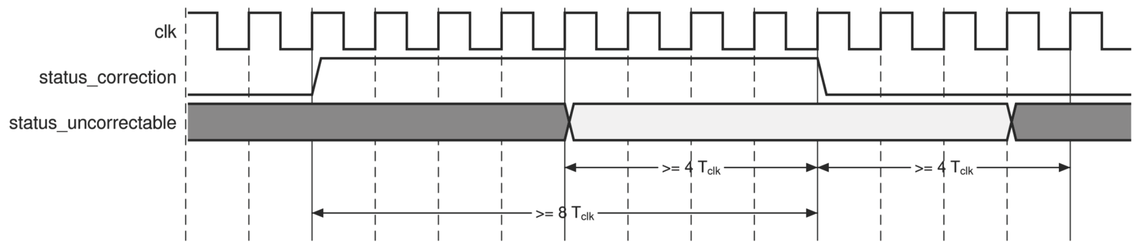


Figure 3-9: **Status Interface Uncorrectable Flag Switching Characteristics (Mitigation Modes)**

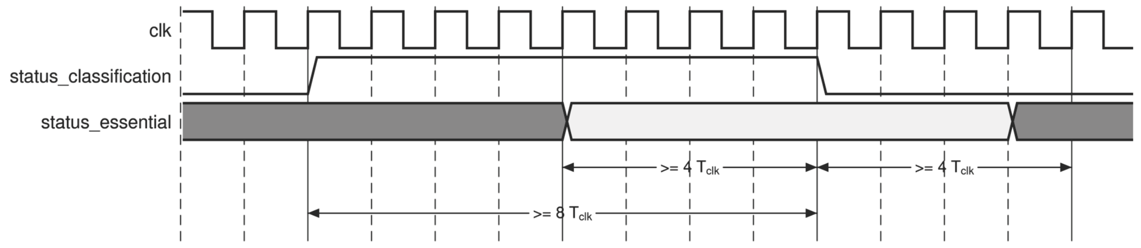


Figure 3-10: **Status Interface Essential Flag Switching Characteristics (Mitigation Modes)**

In detect modes, both these flags are asserted when an error is detected right before the controller transitions into the Idle state.

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 (when in its Correction state), it stops monitoring the configuration memory for SEUs and transitions into the Idle state.*

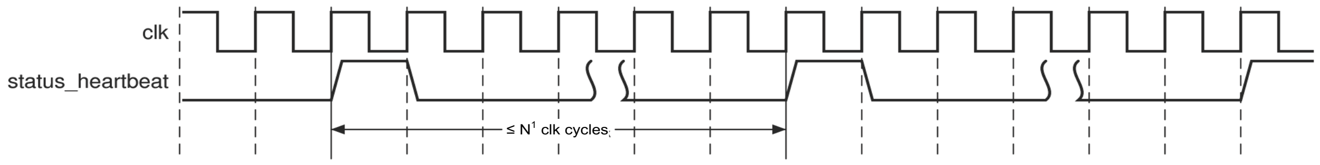
For other system-level recommendations to monitor the health and state of the SEM controller IP, see Systems.

Heartbeat

This signal is used to monitor the status of the configuration readback system (readback CRC) or any manual scanning of the configuration memory that is performed by the SEM controller. It is a direct output from the readback process and is active during the Observation, Detect only, and Diagnostic Scan states. After entering the Observation, Detect only, or Diagnostic Scan state, the heartbeat signal becomes active when the readback process is scanning for errors. The first heartbeat pulse observed during the Observation, Detect only, and Diagnostic Scan states 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 3-11.



1. N = 250K for UltraScale and UltraScale+ devices when controller is in Observation or Detect only state.

Figure 3-11: **Status Interface Heartbeat Switching Characteristics**

Due to the small pulse widths involved, approaches such as sampling the Status Interface signals through embedded processor general purpose I/O (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, Detect only, or Diagnostic Scan state is a good indicator of the overall health and function of the IP. Because of the varying gap sizes between the heartbeats due to device dependency, system clock, ICAP clock, and the current state of the IP, Xilinx recommends that an error is flagged if the heartbeat stops for more than one second. 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, Observation, Detect only, or Diagnostic Scan state, 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 enter Detect only state (resume normal scanning with no correction)
- Command to enter Diagnostic Scan state (perform diagnostic scan of all configuration memory in the device and report all frame level ECC errors)
- Command to perform a software reset (reboot and initialize)
- Inject error at a frame address

To execute the commands, the Status Interface must be monitored because some of the commands only execute if the IP is in Observation, Idle, or Detect only state.

Note that each command generated through the Command Interface is reflected and reported on the Monitor Interface.

RECOMMENDED: *The Monitor Interface is the comprehensive and preferred interface for user interaction with the IP. See Monitor Interface.*

For UltraScale 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. For UltraScale+ SSI implementations of the system-level example design, there is one controller instance for the entire device, identical to the monolithic implementation. 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 Design Suite debug feature) 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 queued. Do not present another command until the `command_busy` signal deasserts. Table 3-2 lists all the commands and how to generate them. The following sections describe the commands in detail.

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

Command	Command_code[n – 1:0] Format n = 40 for UltraScale and n = 44 for UltraScale+ Devices
Directed State Change to Idle	UltraScale = `1110 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` UltraScale+ = `1110 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Observation and Detect only states.
Error Injection Using LFA	UltraScale = `1100 0000 0ssL LLLL LLLL LLLL LLLL wwww wwwb bbbb` UltraScale+ = `1100 0000 0000 ssLL LLLL LLLL LLLL LLLL wwww wwwb bbbb` **Binary Value** | **Equals to** `ss` | Hardware slr number (2-bit) Valid range: 0..3 `LLLLLLLLLLLLLLLLL` or `LLLLLLLLLLLLLLLLLL` | Linear frame address (17-bit for UltraScale or 18-bit for UltraScale+) Valid range: 0..Max Frame – 2 `wwwwwww` | Word address (7-bit) UltraScale, valid range: 0..122 Other, valid range: 0..92 `bbbbb` | Bit address (5-bit) Valid range: 0..31 Valid when controller in Idle state. Valid for Mitigation and Testing, Detect and Testing, or Emulation modes only. For additional guidance on how to use error injection to test a design, see Appendix E, Error Injection Guidance.
Directed State Change to Observation	UltraScale = `1010 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` UltraScale+ = `1010 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Idle state. Valid for mitigation modes only.
Directed State Change to Detect only	UltraScale = `1111 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` UltraScale+ = `1111 xxxx xxxx xxxx xxxx xxxx xxxx  xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Idle state.

Table 3-2: **Command Format and Usage *(Cont'd)***

Command	Command_code[n – 1:0] Format n = 40 for UltraScale and n = 44 for UltraScale+ Devices
Directed State Change to Diagnostic Scan	UltraScale = `1101 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` UltraScale+ = `1101 xxxx xxxx xxxx xxxx xxxx xxxx  xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Idle state.
Software Reset	UltraScale = `1011 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` UltraScale+ = `1011 xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` `x` = Don't care Valid when controller in Idle state.

Directed State Change to Idle

This command is only effective if the IP is in the Observation or Detect only states. Monitor the `status_observation` and `status_detect_only` signals 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.

For additional guidance on how to use error injection to test a design, see Appendix E, Error Injection Guidance.

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.

Directed State Change to Detect Only

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 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 Diagnostic Scan

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 Directed State Change to Idle command to transition the IP to the Idle state. In the Idle state, this command can be executed.

Invalid Command Behavior

When a command is provided with a pulse of `command_strobe`, the `command_busy` is asserted and later deasserted indicating that the command is processed. However, if the command given is invalid, it does not execute; the status signals do not change nor do the invalid command be echoed on the Monitor or UART Interfaces.

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 3-12 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.

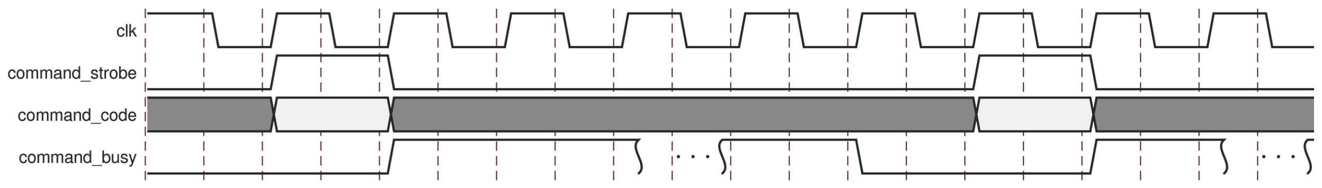


Figure 3-12: **Command Interface Switching Behavior**

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.

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 2-21 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.

For detect features like Detect only and Diagnostic Scan, any information about the errors that the IP encountered is only provided through the Monitor Interface. Hence if these features are used, the error information must be captured from this interface.

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.

For a detailed description of the Monitor Interface signaling, see Appendix H, Monitor Interface Signaling and Protocol.

There are three main use cases for this interface:

- Monitoring and debug
- Testing the controller and user design
- Interactive soft error mitigation

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 the intent of 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 × 8 in size.

If the Diagnostic Scan feature is used, the Monitor Interface provides not only the means to command the IP to perform these scans but also a means to record the type and location of errors if any errors are found. In this use case, the size of the FIFO needs to accommodate N additional error reports where N is the number of the error reports desired to be collected before taking any mitigation action.

If this information is received periodically, the size of the FIFO depends on how often data is retrieved.

In this use case, 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
```

See Figure 3-13 for a block diagram example of how to connect the Monitor TX Interface to a FIFO for monitoring and debugging.

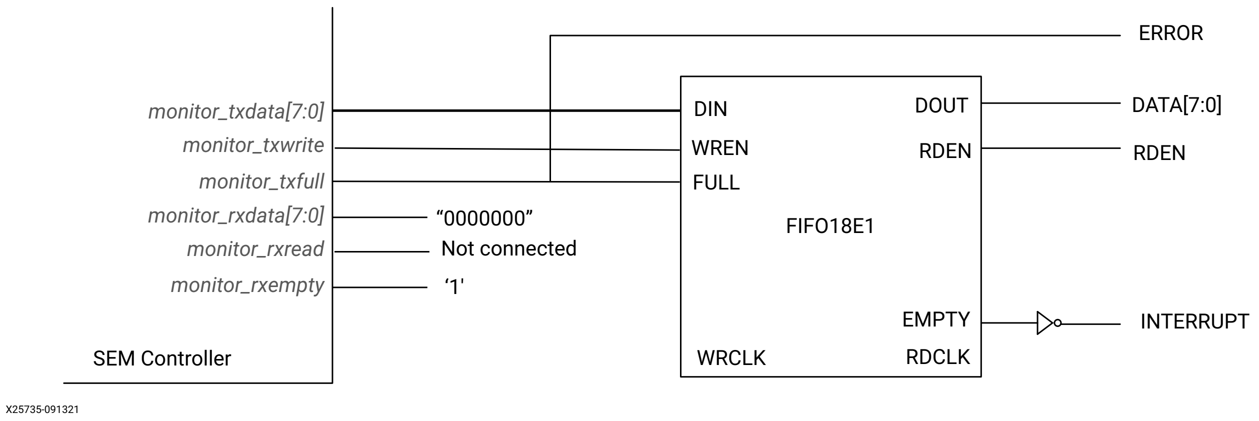


Figure 3-13: **Connecting Monitor TX Interface to a FIFO**

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. The UART Interface can be used 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.

Interactive Soft Error Mitigation

Beyond monitoring and testing, the Monitor Interface can also be used to implement an interactive soft error mitigation solution that is custom to the design needs. The Monitor Interface can be used to log any soft errors detected by the controller and take action based on the type of errors that occur rather than relying on the built-in IP feature (that is, Mitigation modes to perform correction).

For example, you can choose to only log errors as a soft error mitigation strategy but only perform device reconfiguration after approximately 10 errors have accumulated.

One method to implement this is by configuring the SEM controller in Emulation mode and issuing different types of error detection commands (Detect only or Diagnostic Scan) through the Monitor Interface. You can command the SEM controller to perform Detect only scan right after it completes initialization. After an error is detected and reported, the IP transitions to the Idle state. Now, you can periodically (for example, once a day) perform a Diagnostic Scan to track and log the number of accumulated errors. After it reaches 10 errors, the system can halt, reconfigure the device, and restart. You can also achieve the same strategy by configuring the SEM controller in Detect mode and issuing Diagnostic Scan commands after the first error is detected.

Note: Xilinx does not guarantee the functionality of the device if soft errors are left to accumulate and the device is not reconfigured to its original state.

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_V3_1              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, 20}             State transition to Idle, Observation, or Detect only
state
```

Command Prompt

The command prompt issued by the controller is one of three 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>`. If the controller is in the Detect only state, the prompt issued is `D>` (the default state after initialization for Detect and Testing or Detect only mode).

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 20	Detect only
SC 40	Diagnostic scan
SC 08	Classification
SC 10	Injection
SC 9F	Fatal Error

Entry into the Fatal Error state can occur at any time, even without an explicit state change report. Upon entering this state, the controller might or might not issue 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

The flag change report is not generated when the IP is in Diagnostic Scan state.

Error Detection Report – Mitigation Modes (Correction Enabled)

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]

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

Diagnosis: ECC-based error – Uncorrectable

```
RI XX                   Reserved information
SC 04                   State Transition to Correction state
ECC                     ECC Error Detected
TS {8-digit hex value}  Timestamp
PA {n-digit hex value}  PFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
LA {n-digit hex value}  LFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
```

The Detection Report PA includes both the SLR number and PFA address. The PA is formatted to match the PFA error injection command format without the Word and Bit fields. For more information, see "Error Injection Using PFA" in Table J-1.

```
UltraScale PA = 0sst trrr rrrc cccc cccc cmmm mmmm
UltraScale+ PA = 00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm
```

The Detection Report LA includes both the SLR and LFA address. The LA is formatted to match the LFA error injection command format without the Word and Bit fields and with the most significant nibble zeroed out. For more information, see "Error Injection Using LFA" in Table 3-5.

```
UltraScale = 0000 0000 0ssL LLLL LLLL LLLL LLLL
UltraScale+ = 0000 0000 0000 ssLL LLLL LLLL LLLL LLLL
```

Diagnosis: ECC-based error – Correctable

```
RI XX                   Reserved information
SC 04                   State Transition to Correction state
ECC                     ECC error detected
TS {8-digit hex value}  Timestamp
PA {n-digit hex value}  PFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
LA {n-digit hex value}  LFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
```

The Detection Report PA includes both the SLR number and PFA address. The PA is formatted to match the PFA error injection command format without the Word and Bit fields. For more information, see "Error Injection Using PFA" in Table J-1.

```
UltraScale PA = 0sst trrr rrrc cccc cccc cmmm mmmm
UltraScale+ PA = 00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm
```

The Detection Report LA includes both the SLR and LFA address. The LA is formatted to match the LFA error injection command format without the Word and Bit fields and with the most significant nibble zeroed out. For more information, see "Error Injection Using LFA" in Table 3-5.

```
UltraScale = 0000 0000 0ssL LLLL LLLL LLLL LLLL
UltraScale+ = 0000 0000 0000 ssLL LLLL LLLL LLLL LLLL
```

Diagnosis: Auxiliary-based error (error detected from auxiliary input). Auxiliary error is reported when the controller is Correction state.

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

Diagnosis: ECC-based ROM error – Correctable. As stated, the majority of the block RAMs used in the SEM controller are protected using ECC. If a correctable error (single-bit error) is detected by this logic, the following error message is reported. This error is reported when the controller is in the Correction state.

Note that when this error is detected, the actual data read from the block RAM does not contain the error because the block RAM automatically corrects the read output of a single-bit ECC error. To avoid accumulation of errors in that memory space, the corrected data is then re-written to the same address.

```
RI XX                   Reserved information
SC 04                   State Transition to Correction state
ROM
TS (8-digit hex value)  Timestamp
```

Error Detection Report – Detect Only

When the controller is in the Detect only state, it reports any detected error. After the error detection report completes, the controller transitions to the Idle state. The following scenarios exist:

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

```
RI XX                  Reserved information
CRC                    CRC error detected
TS (8-digit hex value) Timestamp
```

Diagnosis: ECC-based error – Uncorrectable

```
RI XX                  Reserved Information
ECC                    ECC error detected
TS {8-digit hex value} Timestamp
PA {n-digit hex value} PFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
LA {n-digit hex value} LFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
```

The Detection Report PA includes both the SLR number and PFA address. The PA is formatted to match the PFA error injection command format without the Word and Bit fields. For more information, see "Error Injection Using PFA" in Table J-1.

```
UltraScale PA = 0sst trrr rrrc cccc cccc cmmm mmmm
UltraScale+ PA = 00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm
```

The Detection Report LA includes both the SLR and LFA address. The LA is formatted to match the LFA error injection command format without the Word and Bit fields and with the most significant nibble zeroed out. For more information, see "Error Injection Using LFA" in Table 3-5.

```
UltraScale = 0000 0000 0ssL LLLL LLLL LLLL LLLL
UltraScale+ = 0000 0000 0000 ssLL LLLL LLLL LLLL LLLL
```

Diagnosis: ECC-based error – Correctable

```
RI XX                  Reserved Information
ECC                    ECC error detected
TS {8-digit hex value} Timestamp
PA {n-digit hex value} PFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
LA {n-digit hex value} LFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
WD {2-digit hex value} BT {2-digit hex value} Word and Bit of the Detected Error
```

The Detection Report PA includes both the SLR number and PFA address. The PA is formatted to match the PFA error injection command format without the Word and Bit fields. For more information, see "Error Injection Using PFA" in Table J-1.

```
UltraScale PA = 0sst trrr rrrc cccc cccc cmmm mmmm
UltraScale+ PA = 00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm
```

The Detection Report LA includes both the SLR and LFA address. The LA is formatted to match the LFA error injection command format without the Word and Bit fields and with the most significant nibble zeroed out. For more information, see "Error Injection Using LFA" in Table 3-5.

```
UltraScale = 0000 0000 0ssL LLLL LLLL LLLL LLLL
UltraScale+ = 0000 0000 0000 ssLL LLLL LLLL LLLL LLLL
```

For the case of correctable ECC errors, the controller reports all word and bit locations in which errors were detected. It reports a maximum of four errors within a single frame.

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

```
RI XX                   Reserved Information
AUX                     AUX error
TS {8-digit hex value}  Timestamp
```

Diagnosis: ECC-based ROM error – Correctable. As stated, the majority of the block RAMs used in the SEM controller are protected using ECC. If a correctable error (single-bit error) is detected by this logic, the following error message is reported.

Note that when this error is detected, the actual data read from the block RAM does not contain the error because the block RAM automatically corrects the read output of the single-bit ECC error. To avoid accumulation of errors in that memory space, the corrected data is then re-written into the same address space.

```
RI XX                   Reserved information
ROM
TS (8-digit hex value)  Timestamp
```

Error Detection Report – Diagnostic Scan

When the controller is in the Diagnostic Scan state, the controller reports errors immediately after detection of the error condition. After detecting an error, the controller interrupts the scan to report the error through the Monitor Interface. The controller does not resume the scan until all errors in the current frame have been reported. Hence, it is important that when this feature is executing, there is not any unnecessary back-pressure occurring on the Monitor Interface. The following scenarios exist:

Diagnosis: ECC-based error – Uncorrectable [cannot identify location or number of bits in error]

```
RI XX                   Reserved Information
ECC                     ECC error detected
TS {8-digit hex value}  Timestamp
PA {n-digit hex value}  PFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
LA {n-digit hex value}  LFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
```

The Detection Report PA includes both the SLR number and PFA address. The PA is formatted to match the PFA error injection command format without the Word and Bit fields. For more information, see "Error Injection Using PFA" in Table J-1.

```
UltraScale PA = 0sst trrr rrrc cccc cccc cmmm mmmm
UltraScale+ PA = 00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm
```

The Detection Report LA includes both the SLR and LFA address. The LA is formatted to match the LFA error injection command format without the Word and Bit fields and with the most significant nibble zeroed out. For more information, see "Error Injection Using LFA" in Table 3-5.

```
UltraScale = 0000 0000 0ssL LLLL LLLL LLLL LLLL
UltraScale+ = 0000 0000 0000 ssLL LLLL LLLL LLLL LLLL
```

Diagnosis: ECC-based error – Correctable

```
RI XX                  Reserved Information
ECC                    ECC error detected
TS {8-digit hex value} Timestamp
PA {n-digit hex value} PFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
LA {n-digit hex value} LFA of Detected Error. n = 7 for UltraScale and 8 for
UltraScale+.
WD {2-digit hex value} BT {2-digit hex value} Word and Bit of the Detected Error
```

The Detection Report PA includes both the SLR number and PFA address. The PA is formatted to match the PFA error injection command format without the Word and Bit fields. For more information, see "Error Injection Using PFA" in Table J-1.

```
UltraScale PA = 0sst trrr rrrc cccc cccc cmmm mmmm
UltraScale+ PA = 00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm
```

The Detection Report LA includes both the SLR and LFA address. The LA is formatted to match the LFA error injection command format without the Word and Bit fields and with the most significant nibble zeroed out. For more information, see "Error Injection Using LFA" in Table 3-5.

```
UltraScale = 0000 0000 0ssL LLLL LLLL LLLL LLLL
UltraScale+ = 0000 0000 0000 ssLL LLLL LLLL LLLL LLLL
```

For the case of correctable errors, the controller lists out all the word and bits (up to four for a single frame) of all the errors it detected.

Error Correction Report – Mitigation Modes Only

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:

```
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. 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 (Mitigation Modes Only)

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. 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, Detect only, 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 and Detect only states. 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 and Detect only states omits the Maximum Linear Frame Count (MF), Timestamp (TS), Table Base (TB), Classification Base (CB), and Classification Level (CL).

The Classification Level value is dependent on whether the classification feature is enabled. If the feature is enabled, the CL value is "02" or two levels of classification. If it is disabled, the CL value is "01" or one level of classification (all errors are classified as essential).

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 00                  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 01                  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 02                  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
```

For UltraScale+ SSI device status report, there is only one controller. Therefore, the State Change, Flag Change, and Classification Level information are identical across all SLRs.

UART Interface Commands

The UART 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.

Note: All UART interface commands must be entered in capital letters.

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

Command	UART Command Set
Directed State Changes	"O" = Enter Observation. Valid in Idle state and for mitigation modes only. "I" = Enter Idle. Valid in Observation and Detect only states. "D" = Enter Detect only. Valid in Idle state. "U" = Enter Diagnostic Scan. Valid in Idle state.
Status Report	"S" Valid in Idle and Observation states.

Error Injection Using LFA	"N {n-digit hex value}"
	n = 10 for UltraScale and n = 11 for UltraScale+.
	UltraScale = `1100 0000 0ssL LLLL LLLL LLLL LLLL wwww wwwb bbbb`
	UltraScale+ = `1100 0000 0000 ssLL LLLL LLLL LLLL LLLL wwww wwwb bbbb`

Binary Value	Equals to
`ss`	Hardware slr number (2-bit) Valid range: 0..3

`LLLLLLLLLLLLLLLLL`
or
`LLLLLLLLLLLLLLLLLL`
	Linear frame address (17-bit for UltraScale or 18-bit for UltraScale+)
	Valid range: 0..Max Frame – 2

`wwwwwww`	Word address (7-bit) UltraScale, valid range: 0..122 Other, valid range: 0..92
`bbbbb`	Bit address (5-bit) Valid range: 0..31

Command	UART Command Set
	Valid in Idle state. Valid for Mitigation and Testing, Detect and Testing, or Emulation modes only. For additional guidance on how to use error injection to test a design, see Appendix E, Error Injection Guidance.
Configuration Frame Reads (`Query` command)	"Q {n-digit hex value}" n-digit hex value can be either the PFA or LFA address format used in the error injection command. ***Note:*** The word and bit fields of the frame address are ignored. The command always returns the entire contents of a frame. n = 10 for UltraScale and n = 11 for UltraScale+. Valid in Idle state.

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

Command	UART Command Set
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 87`. Valid in Idle state.
External Memory Reads (`Xmem` command)	"X {8-digit hex value)" 8-digit hex 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 `xx`" `xx` = Don't care Valid in Idle state.
Frame Address Translation LFA <-> PFA (`Translate` command)	"T {n-digit hex value}" n-digit hex value can be either the PFA or LFA address format used in the error injection command. n = 10 for UltraScale and n = 11 for UltraScale+. Valid in Idle state.

Directed State Changes

The controller can be moved between Observation, Idle, Detect only, and Diagnostic Scan states by a directed state change; "O," "I," "D," and "U" commands respectively.

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 {n-digit hex value}
```

n = 10 for UltraScale and n = 11 for UltraScale+.

Note: The word and bit fields for the frame address are ignored. The command always returns the entire contents of a frame.

After execution of the UltraScale device Query command, the controller returns 123 lines (word 0 first; word 122 last) of hexadecimal data (31:0). The data might or might not be all zeros as shown:

```
I> Q C000000000
00000000
00000000
00000000
...
00000000
00000000
00000000
I>
```

After execution of the UltraScale+ device Query command, the controller returns 93 lines (word 0 first; word 92 last) of hexadecimal data (31:0). The data might or might not be all zeros as shown:

```
I> Q C0000000000
00000000
00000000
00000000
...
00000000
00000000
00000000
I>
```

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 reports its content. The format of the command is:

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

Table 3-6 lists all the registers from which the SEM controller supports reads.

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

Name	Address
CRC	00000
FAR	00001
CMD	00100
CTL0	00101

Table 3-6: **Configuration Registers Readable Through `Peek` Command *(Cont'd)***

Name	Address
MASK	00110
STAT	00111
COR0	01001
IDCODE	01100
AXSS	01101
COR1	01110
WBSTAR	10000
TIMER	10001
BOOTSTS	10110

After execution of the Peek command, the controller returns data in the hexadecimal format as shown:

```
I> P 00
20BD8EDE
I>
```

UltraScale and UltraScale+ devices share the same format for the command and the returned data.

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-digit hex value}
```

After execution of the External Memory read command, the controller returns data in the hexadecimal format as shown:

```
I> X 00000000
94
I>
```

UltraScale and UltraScale+ devices share the same format for the command and the returned data.

Error Injection

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

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

n = 10 for UltraScale and n = 11 for UltraScale+.

Issuing this command is analogous to presenting an error injection command on the Command Interface. The hex value supplied with this command is documented in Table 3-5. For additional guidance on how to use error injection to test a design, see Appendix E, Error Injection Guidance. Also, examples are provided in Appendix C, Utilizing an Evaluation Board to Demonstrate SEM Controller Behavior.

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 {n-digit hex value}
```

n = 10 for UltraScale and n = 11 for UltraScale+.

After execution of the UltraScale device Translate command, the controller returns data in the hexadecimal format as shown:

```
I> T C000000000
0000000000
I>
```

After execution of the UltraScale+ device Translate command, the controller returns data in the hexadecimal format as shown:

```
I> T C0000000000
00000000000
I>
```

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.

The controller transmits an initialization report if the command is successful. See the previous section.

Invalid Command Behavior

When an invalid command is given, it does not execute; the status signals do not change nor do the command be echoed back. Incomplete commands or partial commands that do not have complete or correct arguments are echoed back but not executed by the controller.

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 3-14 shows the switching behavior, and is representative of both transmit and receive.

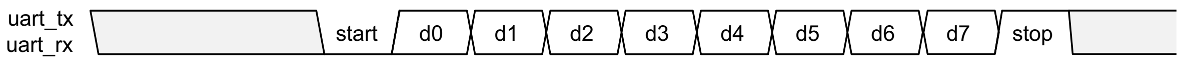


Figure 3-14: **UART Interface Switching Characteristics**

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 115,200 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: 9,600, 230,400, 460,800, and 921,600 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:

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

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 115,200 baud.

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

The actual bit rate that results is approximately 114583 baud, which deviates -0.54% from the nominal bit rate of 115,200 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.

TIP: *The UART helper block system-level example design module that is provided targets a baud rate of 115,200 baud. For most clock frequencies, the resulting actual bit rate does not deviate more than ±1%, but in lower clock frequencies, it might be larger than the given guidance. In this condition, a different baud rate and input clock frequency should be selected.*

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 2-23 for the port list and definition of this interface.

Per the description in Table 2-23, the `fetch_tbladdr` input is used to specify the starting address of the controller data table in the external memory source. See Figure 3-15 for a visual explanation.

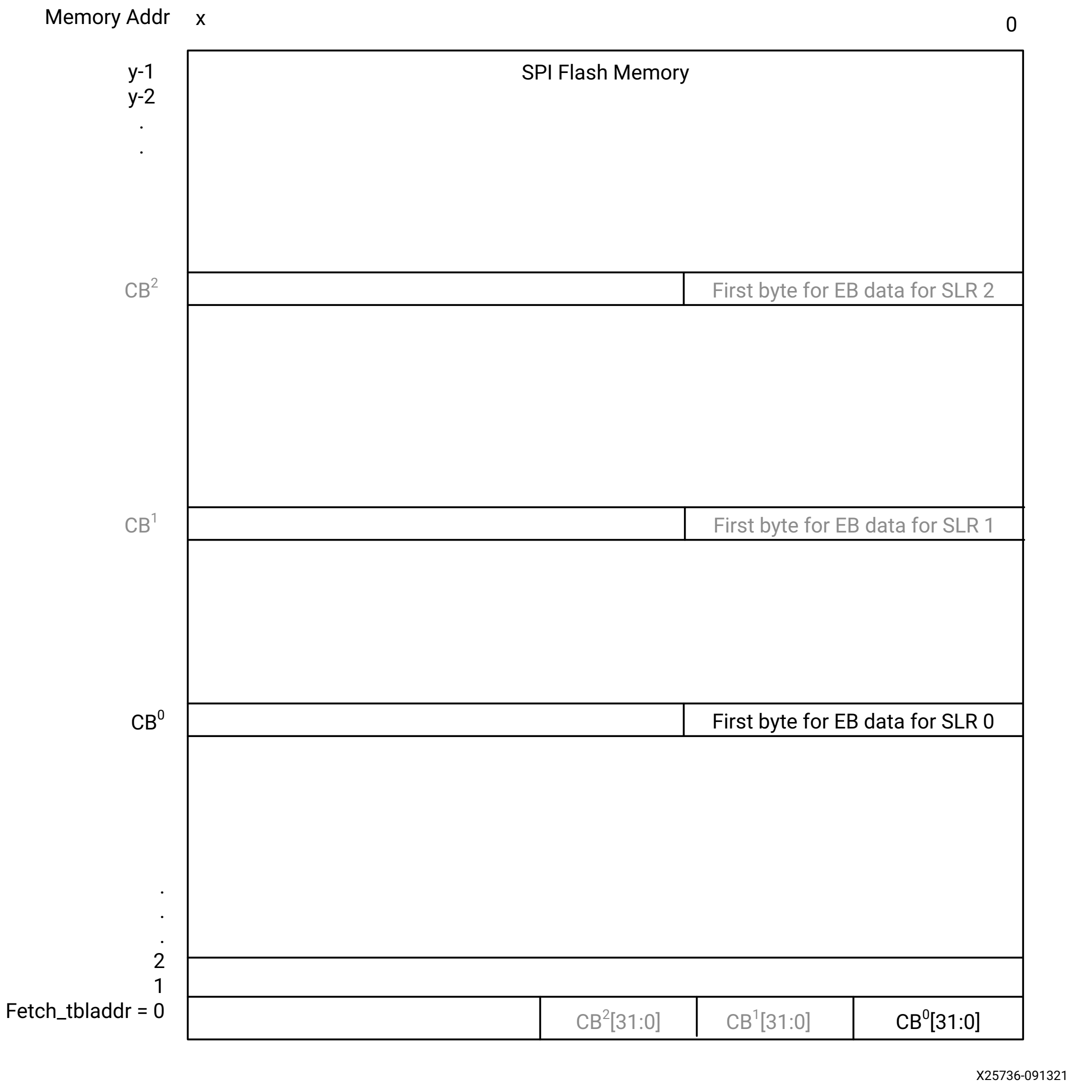


Figure 3-15: **fetch_tbladdr Input Diagram**

The controller expects that the lowest 31 bits of the `fetch_tbladdr` (TB) address to contain the classification base (CB^0) pointer to the address that contain the first byte of the essential bits data for SLR0. If the targeted device is a multi-SLR device, the following 32 bits of the `fetch_tbladdr` contain the classification base (CB^1) pointer to the address that contains the first byte of the essential bits data for SLR1, and so forth. If this is a monolithic device, only one classification base pointer is required.

The ability to define `fetch_tbladdr` (TB) and the classification base pointer (CB^n) enables you to use the SPI flash memory to store other data and organize the content of the memory as they need to.

To verify that correct TB and CB values are used by the controller, these values are echoed as part of the status report that can be performed through the Monitor Interface. See Status Report.

Note that the delivered `makedata.tcl` script (generates SPI flash programming file) assumes the `fetch_tbladdr` value is zero and needs to be modified if the TB and CB values needs to be customized. Ultimately, it is your responsibility to define and organize the data that the SEM controller requires (the classification base pointer and the essential bits data) to the appropriate address space if the TB and CB values are changed.

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.

If an interface other than SPI Interface is needed, the SPI flash master helper block can be replace with an alternate function. For example, the SPI flash master helper block could be replaced with a parallel flash memory controller or other scheme for inter-process communication. To enable this customization, a detailed description of the Fetch Interface signaling is available in Appendix I, Fetch Interface Signaling and Protocol.

SPI Interface

The SPI Interface consists of four signals implementing an 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 (Mb)
UltraScale	XCKU035	128
	XCKU040	128
	XCKU060	256
	XCKU085	512
	XCKU115	512
	XCVU065	256
	XCVU080	256
	XCVU095	256
	XCVU125	512
	XCVU160	1,024
	XCVU190	1,024
	XCVU440(1)	1,024

Table 3-7: **External Storage Requirements *(Cont'd)***

Device		Storage Requirements for Error Classification (Mb)
UltraScale+	XCKU3P	128
	XCKU5P	128
	XCKU9P	256
	XCKU11P	256
	XCKU13P	256
	XCKU15P	256
	XCVU3P	256
	XCVU5P	512
	XCVU7P	512
	XCVU9P	512
	XCVU11P	1024
	XCVU13P	1024
	XCVU27P	1024
	XCVU29P	1024
	XCVU31P	256
	XCVU33P	256
	XCVU35P	512
	XCVU37P	1024
	XCVU45P	512
	XCVU47P	1024
	XCZU1	32
	XCZU2	32
	XCZU3	32
	XCZU4	64
	XCZU5	64
	XCZU6	256
	XCZU7	256
	XCZU9	256
	XCZU11	256
	XCZU15	256
	XCZU17	256
	XCZU19	256

Table 3-7: **External Storage Requirements *(Cont'd)***

Device		Storage Requirements for Error Classification (Mb)
UltraScale+ (Continued)	XCZU21DR	256
	XCZU25DR	256
	XCZU27DR	256
	XCZU28DR	256
	XCZU29DR	256
	XCZU46DR	256
	XCZU47DR	256
	XCZU48DR	256
	XCZU49DR	256
	U55N	512
	U55C	1024
	XCVU57P	1024
	XCVU19P	1024
	XCVU15P	1024
	XCVU23P	256
	XCK26	64
	XCUX35	256
	XCAU25P	128
	XCAU20P	128
	XCAU15P	128
	XCAU10P	128

Notes:

1. XCVU440 requires the use of SPI flash memory devices (for example, the MT25Q devices) that can read seamlessly with a single command across any die boundaries.

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 and MT25Q 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 and MT25Q 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 4-byte addressing mode. Must be set to 1 for devices > 128 Mbits.

For storage requirements £ 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 and MT25Q 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 3-16 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.

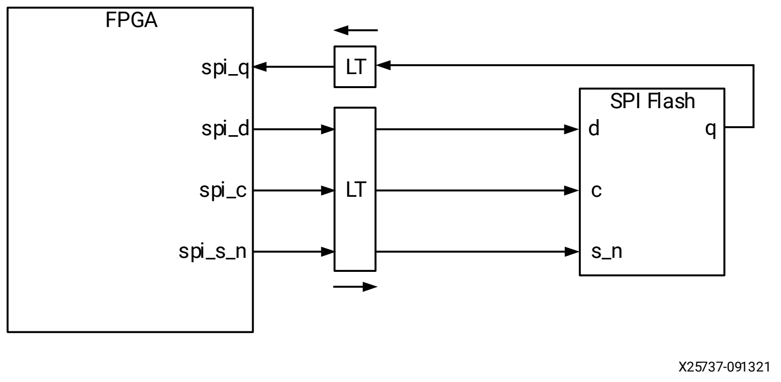


Figure 3-16: **SPI Flash Device Connection, Including Level Translators**

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 in Appendix G, SPI Bus Timing Budget.

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 incorrect classification values being reported and leading to wrong mitigation actions being taken. 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 the block RAM ECC function, the errors detected by this function can be inputs to this interface. Errors notified through this interface are detected when the IP is in the Observation and Detect only states 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 are illustrated in Figure 3-17. 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 when in the Observation state:

```
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 and Detect only states.

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. The 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 when in the Observation state:

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

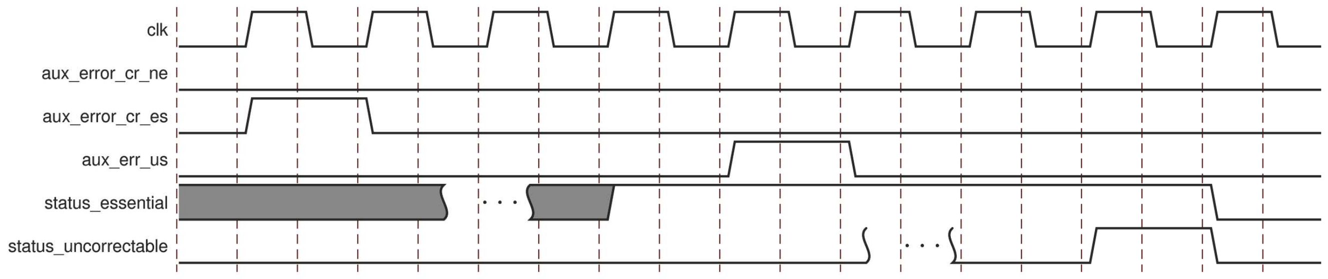


Figure 3-17: **Switching Behavior for Auxiliary Interface**

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 references below also include 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 in Mitigation modes, use the Status Interface `status_correction` and `status_uncorrectable` signals, the Status Interface `status_classification` and `status_essential` signals, or the UART Interface `uart_tx` signal for error detection, correction, and classification reports.

  To monitor the Soft Error Mitigation solution event reporting in Detect modes, use the Status Interface `status_uncorrectable`, or the UART Interface `uart_tx` signal for error detect 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 correct state, Observation, Detect only, or Idle state based on the selected modes.
    Xilinx specifies the Soft Error Mitigation solution will boot, initialize, and enter the designated state within the time specified through Table 2-5 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 correct state are usually design errors (versus soft error events) and include incorrect tie-offs of unused ports, 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 correct state, the system-level supervisory function can observe the Status Interface `status_initialization` and relevant `status_*` signals for assertion (see state diagrams Figure 3-5 through

Figure 3-7), or the UART Interface `uart_tx` signal for the expected initialization report.

Note that the CRC Indicator, `INIT_B`, can be ignored in this state.

- **Observation State (Mitigation Modes)** – 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 both UltraScale and UltraScale+ 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.

    See Chapter 3, Heartbeat.

  - **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 UltraScale and UltraScale+ SSI implementations, which have an internal CRC failure indicator per SLR, the indicators are wire-ORed to form the single `INIT_B` device pin. For UltraScale implementation, the Status Interface has a `status_correction` signal for each SLR.

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

  - **Controller Status Command and Report** – Using the UART Interface `uart_rx` and `uart_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 the 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 2-9/Equation 2-3 and Table 2-10/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.

- **Detect Only Mode or State** – The controller spends virtually all of its time in this state after it transitions into this mode after initialization or when it is commanded to do so. There are at least two 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 indicates the readback process is active. If, during the Detect only 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 UltraScale and UltraScale+ SSI implementations, which have a status_heartbeat output per SLR, it is necessary to monitor the heartbeat from all SLRs.

  See Chapter 3, Heartbeat.

- **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 Detect only state, `INIT_B` indicates an error and the controller does not respond with a state transition to Idle within one second, then the controller has experienced a fault. State transition can be determined using the Status Interface (to detect idle state) or the Monitor Interface state change report. This condition is an uncorrectable, essential error.

  In UltraScale and UltraScale+ 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 for each SLR must be monitored for an Idle state separately.

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

◦ **Diagnostic Scan State** – When commanded, the controller scans all the configuration memory in the device in this state and reports all ECC errors it encounters. Here is the recommended method for monitoring the controller in this state:

  - **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 Diagnostic Scan 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 UltraScale and UltraScale+ SSI implementations, which have a status_heartbeat output per SLR, it is necessary to monitor the heartbeat from all SLRs.

    See Chapter 3, Heartbeat.

- **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 previous section 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 mentioned previously 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.
- **Fatal Error 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 seven state indicators, and might be observable on the Monitor Interface as a HLT message. In UltraScale SSI implementations, where more than one controller instance exists, the solution is considered halted if one or more of the controller instances halts or transitions to idle as a result of an uncorrectable error event. 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, Idle, or Detect only (based on mode selected) state after device configuration as discussed in the Boot and Initialization. `INIT_B` signal should not be observed in the Boot and Initialization states.
2. Monitor `status_heartbeat` signal during Observation, Detect only, and Diagnostic Scan states to ensure that it is within the specification as discussed in the Heartbeat. An example of this monitoring logic is delivered in the example design. See the Functions in Chapter 5.
3. Ensure that IP has NOT halted or gone to Idle when it is deployed in any Mitigation and Detect modes. If either of these states occur, the IP has stopped any mitigation activity and can no longer detect or correct any SEU that might occur. This can be done by monitoring the `status_*` signals. An example logic to flag if the IP is halted is delivered in the example design. See the Functions in Chapter 5.
4. Monitor the `INIT_B` signal when the SEM controller is in the Observation and Detect only states. If `INIT_B` remains asserted for longer than one second and the controller has not transitioned to the Correction or Idle state respectively, 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_txdata[7:0]` 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 devices 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 you to design to implement data protection on these bits for purposes of soft error mitigation. Another approach is to modify your design to eliminate the use of features that introduce configuration memory masking. LUTROM functions will not be masked by SEM IP, you have to refer to LUT properties for finding the LUT usage details.

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 Vivado IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 13]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite. The SEM controller is device dependent and a device that supports SEM IP must be chosen. 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 Vivado 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.

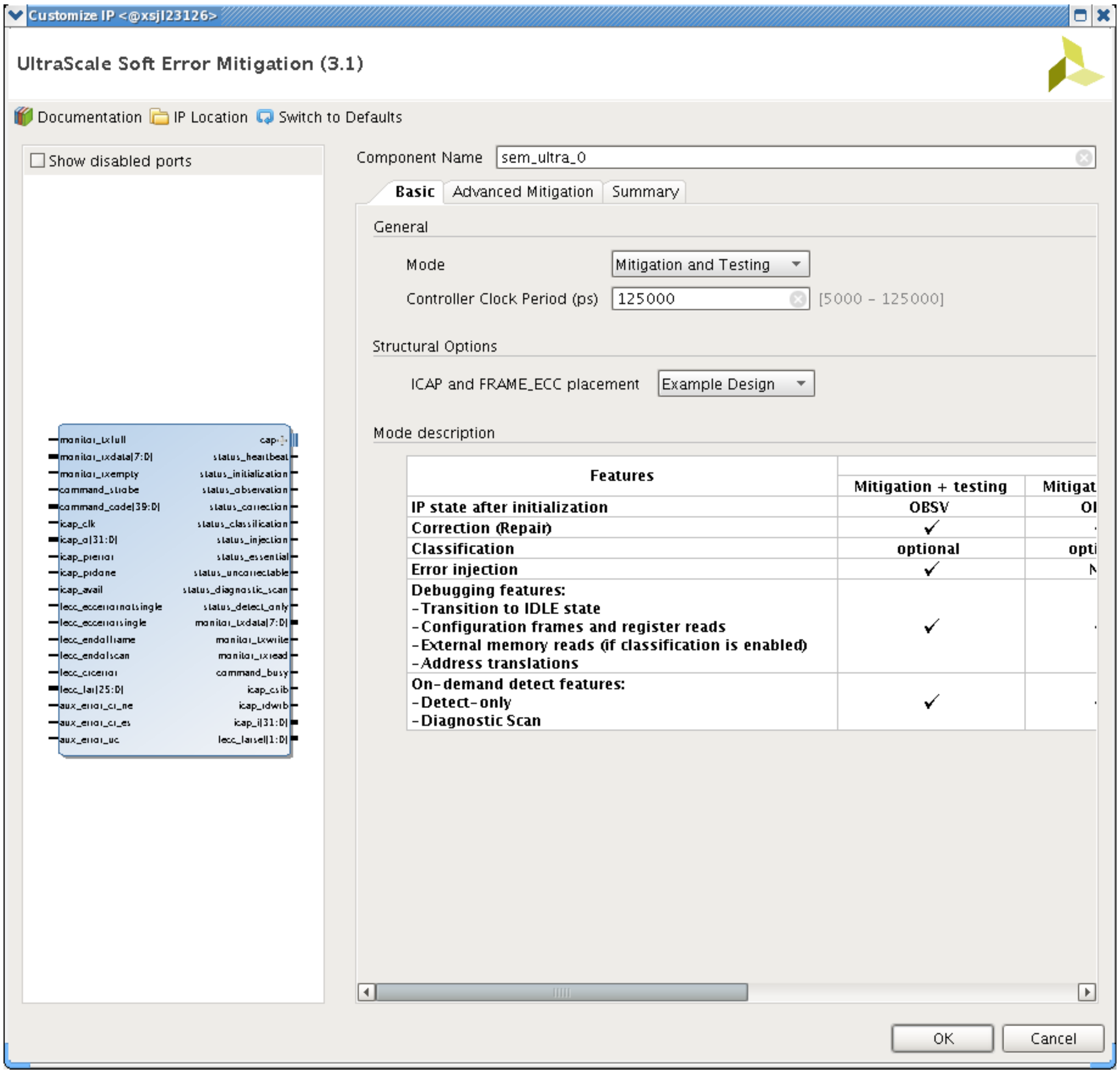
The SEM controller IP customization Vivado IDE is organized in three tabs:

- **Basic** – Provides customization options for the elementary SEU mitigation features including IP mode, 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.
- **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.

CAUTION! *Each SEM controller generated is specific to the targeted device. If a controller design checkpoint (DCP) is generated for one device and then used in a larger design targeted for a different device and hardware, the IP completes its boot and initialization process but it does not function correctly.*

Basic Tab



UltraScale Soft Error Mitigation (3.1)

Component Name: sem_ultra_0

General

Mode: Mitigation and Testing

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

Structural Options

ICAP and FRAME_ECC placement: Example Design

Mode description

Features	Mitigation + testing	Mitigat
IP state after initialization	OBSV	OI
Correction (Repair)	✓	
Classification	optional	opti
Error injection	✓	N
Debugging features: -Transition to IDLE state -Configuration frames and register reads -External memory reads (if classification is enabled) -Address translations	✓	
On-demand detect features: -Detect-only -Diagnostic Scan	✓	

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 six modes:

- Mitigation and Testing
- Mitigation only
- Detect and Testing
- Detect 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 Vivado IDE 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 Detect, Emulation, or Monitoring modes. In the Detect mode, the IP is in the Detect only state after initialization completes and it continuously scans the configuration memory for SEU errors. After an error is detected, it reports the error and transition to the Idle state without performing any corrections.

In Emulation and Monitoring modes, the IP is in the Idle state after initialization completes and you are not able to enable continuous scan of the configuration memory to detect and correct SEU errors (cannot transition to Observation state).

Independent of the modes, there are error detection and reporting features available from the Idle state that you can command the controller to perform: Detect only and Diagnostic Scan.

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 supervisory 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, Detect only, or Monitoring modes.

TIP: *When the Error Injection feature is disabled, the controller ignores the error injection command and does not transition to the error injection state.*

For a summary of features available in each mode, see Table 2-1, page 21.

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.

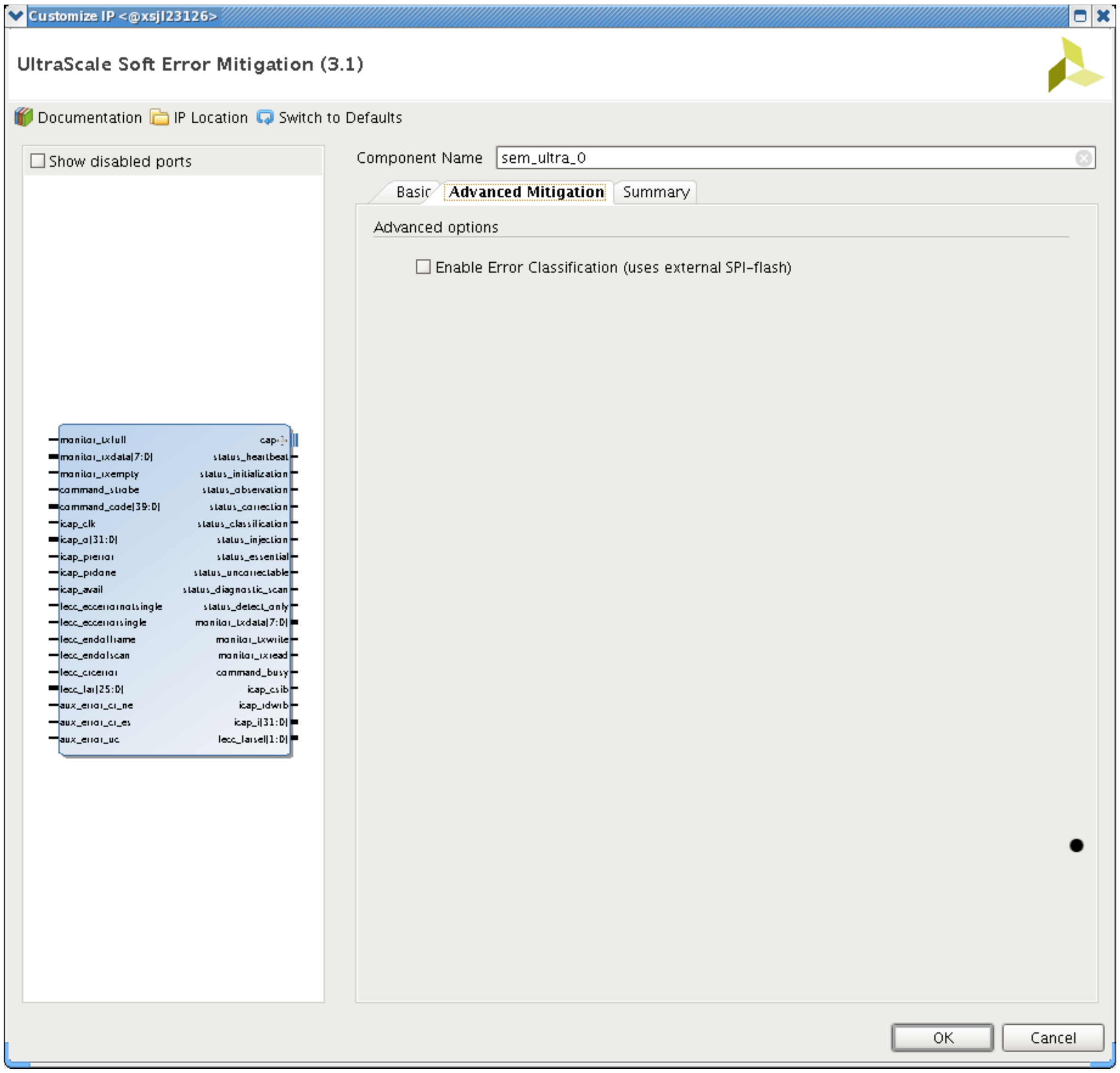
Note: When an evaluation board is targeted, the default Controller Clock Period is automatically selected to a specific clock frequency based pinout delivered in the example design.

Structural Options

The structural options are used to determine the organization of the required ICAP and FRAME_ECC primitives 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 out-of-context (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



UltraScale Soft Error Mitigation (3.1)

☐ Show disabled ports

Component Name sem_ultra_0

Advanced options

☐ Enable Error Classification (uses external SPI-flash)

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.

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. For the delivered example design, the Fetch Interface needs to be bridged to an external SPI flash. Additionally, a SPI flash master helper is included in the example design to enable this connection.

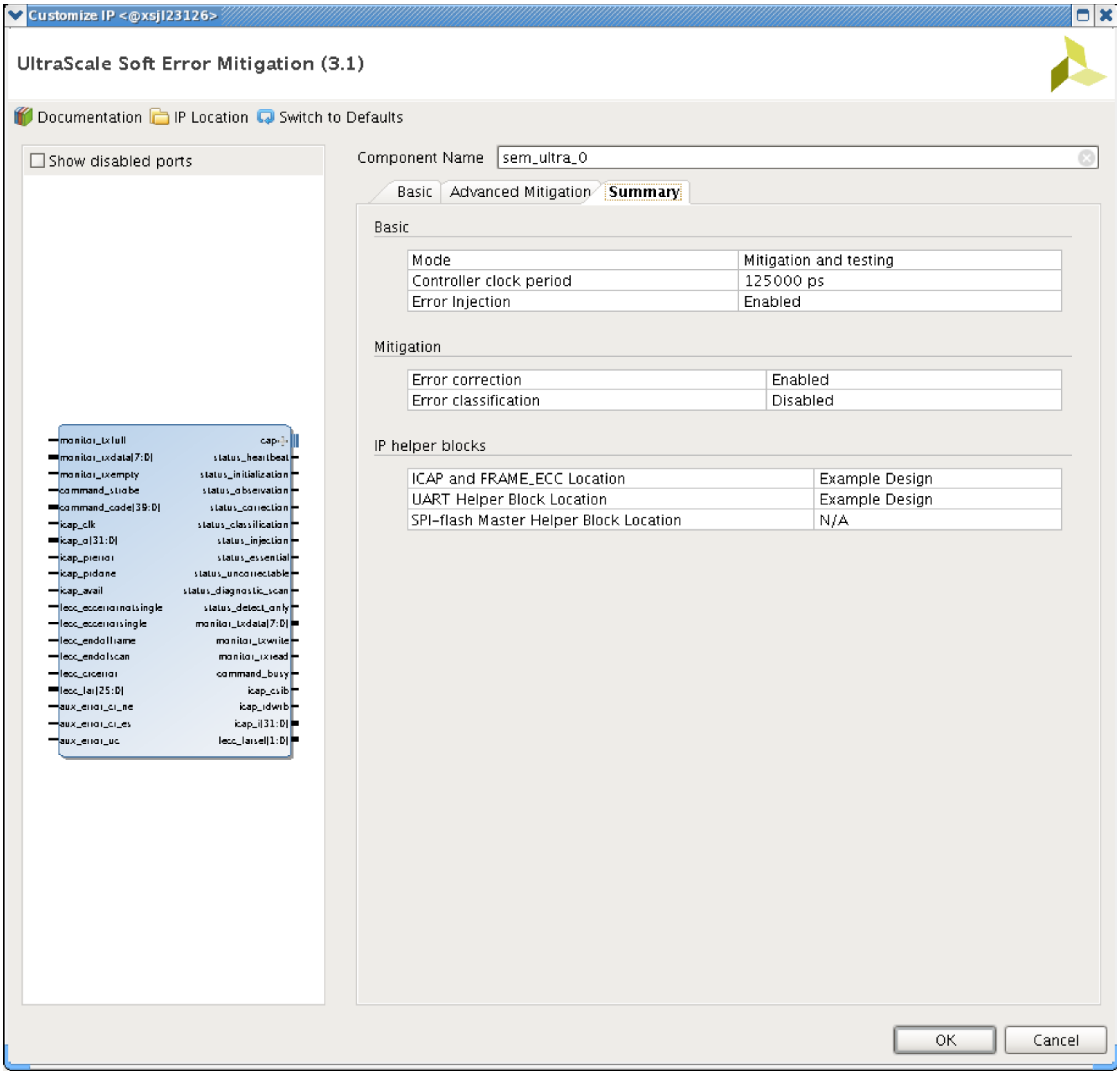
If error classification is enabled, and a detected error has been corrected, the controller looks up the error location. Depending on the information in the table, the controller either reports the error as essential or non-essential. If a detected error cannot be corrected, this is because the error 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.

Note: This option is not available when an evaluation board is targeted.

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.*

Summary Tab



UltraScale Soft Error Mitigation (3.1)

Component Name: sem_ultra_0

Basic

Mode	Mitigation and testing
Controller clock period	125000 ps
Error Injection	Enabled

Mitigation

Error correction	Enabled
Error classification	Disabled

IP helper blocks

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 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
Clock period	N/A	125,000 ps
Locate Config Prim	c_config_prim_loc	Example design
Enable Classification	c_feature_set	FALSE

Output Generation

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

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 or files.

For the UltraScale+™ example design, two XDC constraints are delivered; one containing timing (`*_synth.xdc`) and one containing placement constraints (`*_impl.xdc`). For the UltraScale™ example design, a single XDC constraint is delivered containing both the timing and placement constraints. Regardless the number of XDC files delivered, the type of constraints in the XDC file or files are the same.

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™ 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.

In general, the ICAP that the controller interfaces with needs to be placed in the top ICAP in a given SLR. For monolithic devices, no specific placement constraints are required because the tool places the ICAP and FRAME_ECC correctly without any directives.

For SSI devices, the tool requires a specific constraint to place the ICAP and FRAME_ECC in the correct SLR. See the `set_property` 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/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/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/controller_synchro_icap_avail/sync_a/D}]
```

The second set of constraints in this group is 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 is 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 actual 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 implemented FPGA. It varies based on the 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_X8Y14: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/sem_controller/*]
```

The following constraints are also included to force the FRAME_ECC and ICAP placement to the top ICAP but this is not required for monolithic devices (tools are able to place them correctly with no directives).

```
# Force FRAME_ECC to the site in this SLR.
set_property LOC CONFIG_SITE_X0Y0 [get_cells example_support_wrapper/
example_support/example_cfg/cfg_frame_ecce3]
# Force ICAP to the site in this SLR.
set_property LOC CONFIG_SITE_X0Y0 [get_cells example_support_wrapper/
example_support/example_cfg/cfg_icape3]
```

For SSI implementations, it is important that the FRAME_ECC and ICAP placement constraints are followed to ensure the functionality of the design.

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. See Appendix C, Utilizing an Evaluation Board to Demonstrate SEM Controller Behavior. 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]
```

When selecting I/O pins for the UART or SPI flash interface, it is necessary that these I/O pins are placed in an I/O bank closest the SEM controller to ensure timing closure.

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 UltraScale 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 number.

For the tool to place the ICAP and FRAME_ECC primitives in each SLR correctly, the following constraints need to be applied (this is an example for SLR2):

```
## Force FRAME_ECC to the site in SLR 2.
set_property LOC CONFIG_SITE_X0Y2 [get_cells example_support_wrapper/
example_support/slr2_example_cfg/cfg_frame_ecce3]
## Force ICAPs to the site in SLR 2.
set_property LOC CONFIG_SITE_X0Y2 [get_cells example_support_wrapper/
example_support/slr2_example_cfg/cfg_icape3]
```

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

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

When selecting I/O pins for the UART or SPI flash interface, Xilinx recommends that these I/O pins are placed in an I/O bank within the Master SLR (instance 0 of the controller) and closest to that instance.

Constraints for UltraScale+ SSI Devices

In the system-level design example, only one controller instance is generated but two to four FRAME_ECC instances are generated depending on the device. The FRAME_ECC instances are named to include an identification number ranging from 0 to 3. The controller instance numbering matches the hardware SLR number.

For the tool to place the ICAP and FRAME_ECC primitives correctly, the following constraints need to be applied (this is an example of two SLR devices):

```
# Force FRAME_ECC to the site in this SLR.
set_property LOC CONFIG_SITE_X0Y1 [get_cells example_support_wrapper/
example_support/example_cfg/slr1_cfg_frame_ecce4]
set_property LOC CONFIG_SITE_X0Y0 [get_cells example_support_wrapper/
example_support/example_cfg/slr0_cfg_frame_ecce4]
# Force ICAP to the site in this SLR.
set_property LOC CONFIG_SITE_X0Y0 [get_cells example_support_wrapper/
example_support/example_cfg/cfg_icape3]
```

An area constraint is applied to the controller to place it in the Master SLR in order to keep the controller located near the associated ICAP.

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

When selecting I/O pins for the UART or SPI flash interface, Xilinx recommends that these I/O pins are placed in an I/O bank within the Master SLR and closest to the controller.

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

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 9].

Design simulations that instantiate the controller are supported. In other words, including the controller in a larger project does not adversely affect the 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 details about synthesis and implementation using the Vivado Design Suite, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

Integration and Validation

In the development of a complex system, early integration and continual validation of major functional blocks and interfaces is an important best practice. Significant design changes near the end of the development cycle (late integration) or postponing system performance measurements (late validation) increase risk.

Include integration and validation milestones in your project planning and support them with a plan to confirm system functionality and performance throughout the development cycle. Starting as early as possible and incrementally including as much functionality as practical provides the most time for system evaluation under representative workloads. This recommendation complements the commonly used bottom-up design approach, facilitating design reuse and IP-based design.

Integration of the SEM IP Core

The SEM IP core has a small programmable logic footprint, but activates the programmable logic configuration memory system. The configuration memory system works much like a conventional SRAM, except that it is physically distributed throughout the programmable logic array. Just like all other digital switching activity in the device, activity in the configuration memory system generates power noise.

In all device families supported by the SEM IP core, the power noise contribution from the configuration memory system is minor, provided all implementation requirements and design guidance is observed.

RECOMMENDED: *Xilinx strongly recommends system designers integrating the SEM IP core to use the current version and to keep current with the SEM IP core design advisories and known issues through Xilinx Answer Records for Soft Error Mitigation IP Solutions.*

Integrate the SEM IP core as early as possible, ideally at the start of the project. Because the SEM IP core can automatically initialize, the first pass integration of this core can be as simple as instantiating it, connecting a clock, and adding tie-offs to other input ports. As part of this early infrastructure, Xilinx recommends implementing the SEM IP core provisioning controls to allow the system to enable/disable the SEM IP core clock and enable/disable the SEM IP core ICAP grant. System provisioning of the SEM IP core provides deployment flexibility and also facilitates debug of the SEM IP core integration.

At a later point, the integration can be expanded through definition and implementation of an interface for command/status exchange between the system and the SEM IP core. The preferred method for this uses ASCII communication over the SEM IP core Monitor Interface, either with the UART helper block for a serial connection, or without the UART helper block for a parallel connection using communication FIFOs.

Although the status exchange alone is adequate for the system to log and parse events reported by the SEM IP core, the command exchange is critical to support error injection. Without error injection, there is no practical way to completely test the integration of the SEM IP core and the system response to the SEM IP core event reports, outside of an accelerated particle test at a radiation effects facility. Error injection can also be useful for a minor aspect of continual validation.

Validation with the SEM IP Core

In a deployed system containing the SEM IP core, the configuration memory system activity is a mix of reads and writes during the SEM IP core initialization state. Afterwards, the activity is set to 100% reads during the Observation state to perpetually scan the configuration memory for single event upsets. Given the exceptionally low upset rate of the Xilinx configuration memory in a terrestrial environment (that is, mean time between upset is decades at sea level in NYC), the SEM IP core rarely transitions out of the Observation state into the Correction state during which writes can take place.

Continual validation of a system containing the SEM IP core should use a representative SEM IP core workload to ensure validation results will be representative of the system in deployment. Provided the system provisions the SEM IP core to complete the Initialization and enter the Observation state, the default workload of configuration memory scanning with no upsets is not only the easiest, but also representative for validation.

If desired, the SEM IP core error injection feature can be used to incorporate an occasional error detection and correction into the workload.

IMPORTANT: *Xilinx does not recommend "stress testing" with high rate error injection to generate a large number of error detection and correction events, as this stimulus is unnatural and can yield validation results that are irrelevant to reliable operation of the system.*

Continual validation of the system should extend beyond the research and development phase into production testing. With the SEM IP core, this requires little to no additional effort, as the system provisioning during production testing need to only enable the SEM IP core.

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 you 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.

Additionally, this example layer also provides logic to illustrate how you can monitor the Status Interface to ensure that the IP is behaving as expected. The following status error signals are provided:

- `heartbeat_timeout` – This signal asserts when the heartbeat stops toggling for more than 1 second when the SEM controller is in the expected states.
- `heartbeat_timeout_sticky` – This is a sticky version of the `heartbeat_timeout` signal. After assertion, it stays asserted the rest of the time until the design is reconfigured.
- `status_irregular_sticky` – This is a sticky signal that asserts if more than one status signal is asserted in the same clock cycle.
- `status_halt` – This is a signal that asserts when the IP is halted due to a fatal error (all status signals are asserted).

The monitoring of the status signals is described further in the Systems in Chapter 3.

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

- `<component_name>_vio_si14` – Displays SEM controller Status Interface and status error signals.
- `<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 defines input for the ICAP Arbitration and Auxiliary Interfaces.
- `<component_name>_vio_si1_so41` or `so45` – Displays output and defines input for the Command Interface.

For UltraScale 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_si16_so2` – Displays all outputs and defines inputs for the Status and ICAP Arbitration Interfaces for a single SLR.
- `<component_name>_vio_generic_so44` or `so76` – Drives shared SLR inputs for Auxiliary, Command, and Fetch Interfaces. Inputs for the Fetch Interface is only available when error classification is enabled and it increases the number of output probes from 44 to 76.

For UltraScale+™ SSI devices, there are four VIO IP cores instantiated:

- `<component_name>_vio_si17` or `si20` or `si23` – Displays SEM controller Status Interface and status error signals.
- `<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 defines input for the ICAP Arbitration and Auxiliary Interfaces.
- `<component_name>_sio_si1_so45` – Displays output and defines input for the Command Interface.

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.

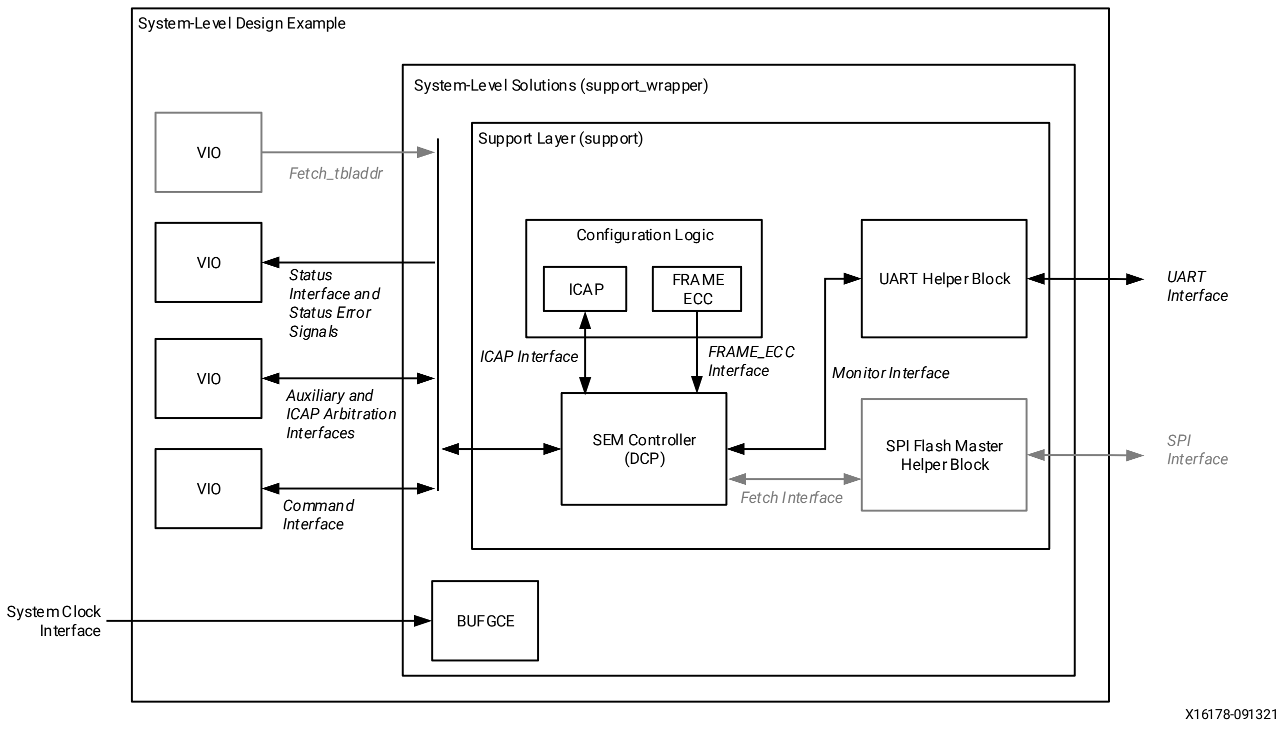


Figure 5-1: **Example Design Block Diagram for Non-SSI Devices**

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

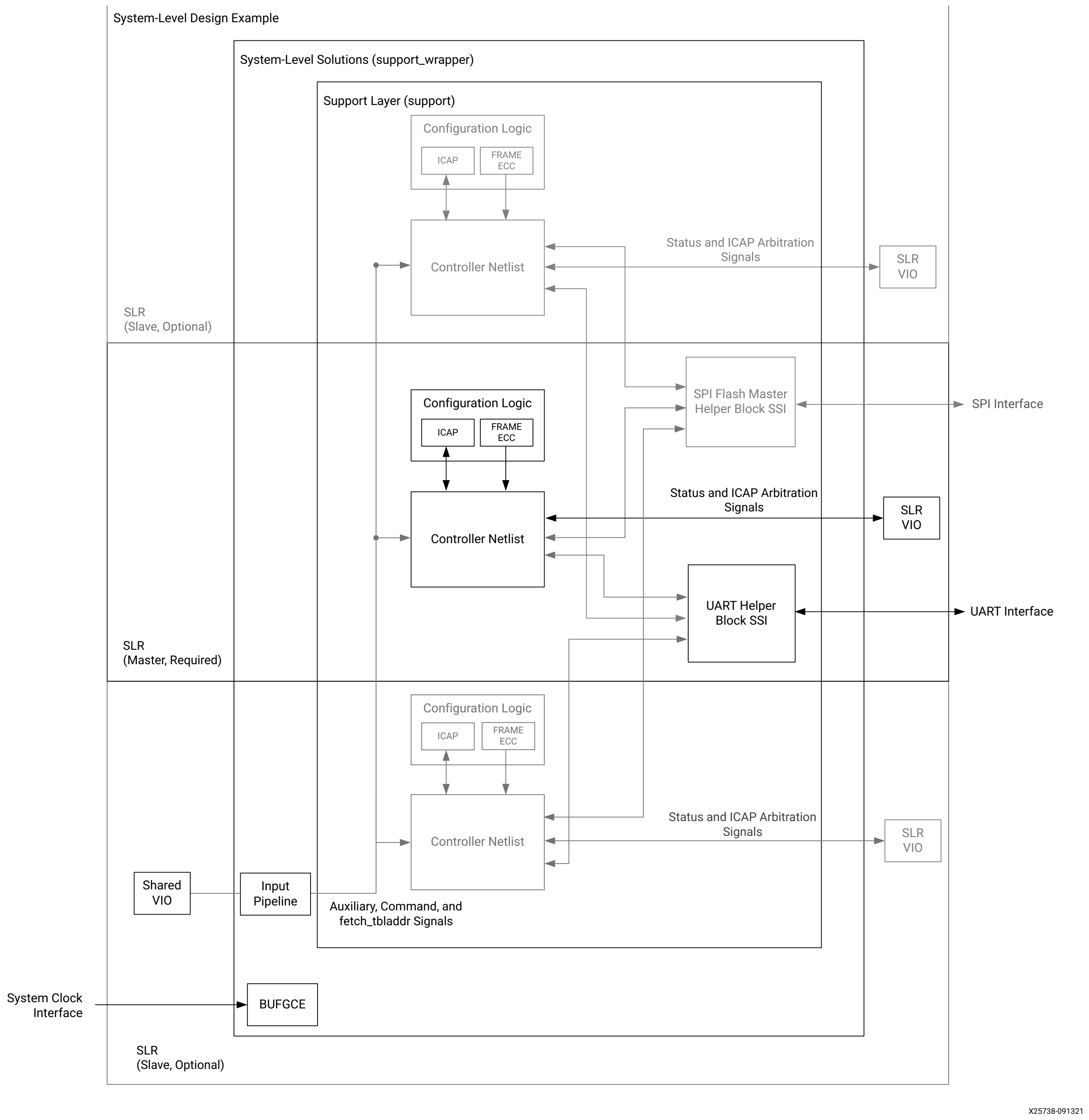


Figure 5-2: **Example Design Block Diagram for UltraScale SSI Devices**

Figure 5-3 shows a block diagram of the system-level design example for UltraScale+ SSI devices. The blocks drawn in gray only exist in certain configurations.

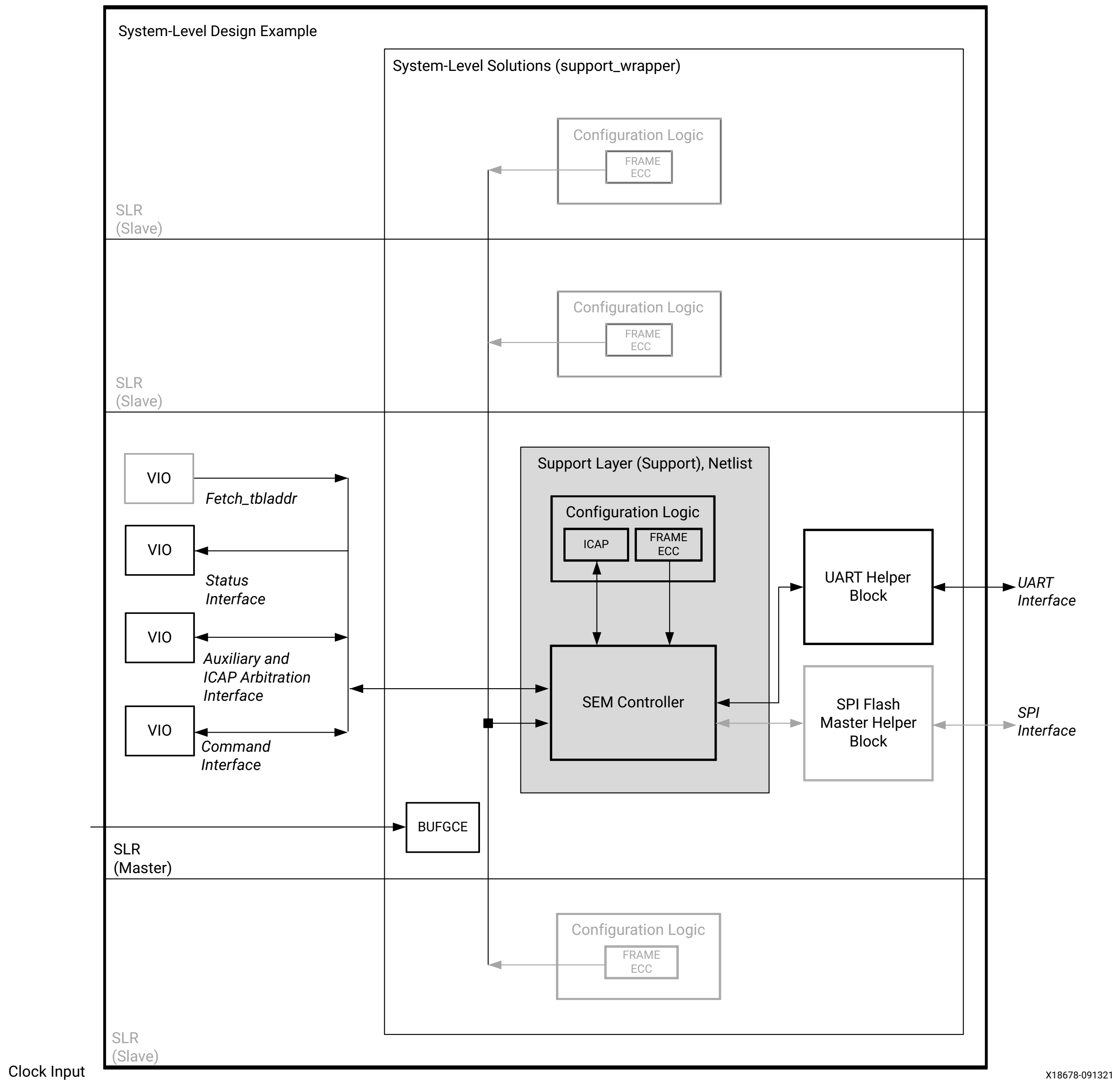


Figure 5-3: **Example Design Block Diagram for UltraScale+ SSI Devices**

Port Descriptions

Figure 5-4 shows the example design ports for all devices. The ports are clustered into three groups. The groups shaded in gray only exist when the error classification feature is enabled.

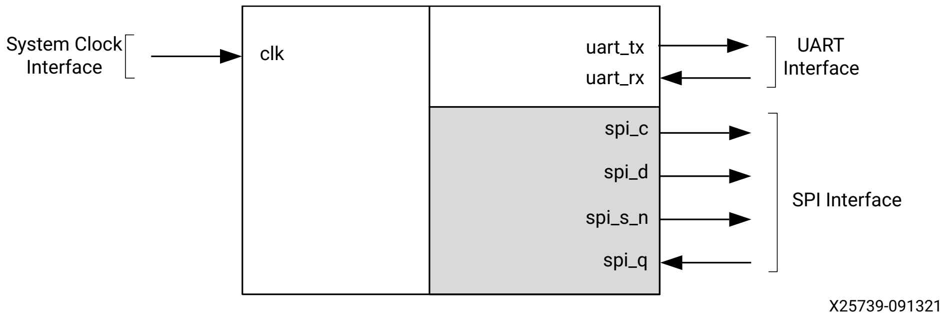


Figure 5-4: **Example Design Ports**

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.

For UltraScale devices, 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	Slave
	0	0	Master

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

Device	Software SLR Number	Hardware SLR Number	SLR Type
VU190	2	2	Slave
	1	0	Master
	0	1	Slave
VU440	2	2	Slave
	1	0	Master
	0	1	Slave
VU5P/VU7P/VU35P/ VU45P	1	1	Slave
	0	0	Master
VU9P	2	2	Slave
	1	0	Master
	0	1	Slave
VU11P/VU37P/ VU47P	2	2	Slave
	1	1	Slave
	0	0	Master
VU13P	3	3	Slave
	2	2	Slave
	1	0	Master
	0	1	Slave
VU27P/VU29P	3	3	Slave
	2	2	Slave
	1	0	Master
	0	1	Slave
U55N	1	1	Slave
	0	0	Master
U55C	2	2	Slave
	1	1	Slave
	0	0	Master
XCVU57P	2	2	Slave
	1	1	Slave
	0	0	Master
XCVU19P	3	3	Slave
	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
XCVU15P	3	3	Slave
	2	2	Slave
	1	0	Master
	0	1	Slave
XCVU23P	0	0	Master
XCK26	1	1	Slave
	0	0	Master

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 UltraScale 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

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 component_name>_ex/imports/makedata.tcl
source sem_ultra_0_ex/imports/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 <path to component_name>_ex/<component_name>_ex.runs/impl_1
cd sem_ultra_0_ex/sem_ultra_0_ex.runs/impl_1
```

Then execute the commands outlined in these steps. For monolithic devices, only one `*.ebd` file is generated by `write_bitstream` and this filename is passed as an argument. For UltraScale SSI devices, a separate `*.ebd` file is generated for each SLR in the device.

Note: In 2017.1, classification feature is not supported in UltraScale+ SSI devices.

To pack the essential bits data from all `*.ebd` files into one SPI flash data file, a space-separated list of `*.ebd` filenames are passed as arguments.

```
makedata -ebd <ebd filename(s)> datafile
```

Example for monolithic devices:

```
makedata -ebd sem_ultra_0_example_design.ebd datafile
```

Example for SSI devices with three SLR:

```
makedata -ebd sem_ultra_0_example_design_0.ebd sem_ultra_0_example_design_1.ebd
sem_ultra_0_example_design_2.ebd datafile
```

The command creates the VMF, BIN and MCS files.

RECOMMENDED: *There are two types of files delivered for the* `makedata.tcl` *file depending on the target device and required SPI flash device needed, see Table 3-7. Different SPI flash programming file needs to be generated and hence requiring different post-processing of the* `write_bitstream` *output files. Xilinx recommends that you use the* `makedata.tcl` *file generated for the example design of the targeted device and not reuse the* `makedata.tcl` *file from a different project.*

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 124 bytes are reserved, filled with ones

For SSI devices, the table format is:

- **Byte 0** – 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 0 (least significant byte)

- **Byte 1** – 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 1
- **Byte 2** – 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 2
- **Byte 3** – 32-bit pointer to start of hardware SLR0 (master) essential bit, byte 3 (most significant byte)
- **Byte 4** – 32-bit pointer to start of hardware SLR1 essential bit data, byte 0 (least significant byte)
- **Byte 5** – 32-bit pointer to start of hardware SLR1 essential bit data, byte 1
- **Byte 6** – 32-bit pointer to start of hardware SLR1 essential bit data, byte 2
- **Byte 7** – 32-bit pointer to start of hardware SLR1 essential bit, byte 3 (most significant byte)
- **Byte 8** – 32-bit pointer to start of hardware SLR2 essential bit data, byte 0 (least significant byte)
- **Byte 9** – 32-bit pointer to start of hardware SLR2 essential bit data, byte 1
- **Byte 10** – 32-bit pointer to start of hardware SLR2 essential bit data, byte 2
- **Byte 11** – 32-bit pointer to start of hardware SLR2 essential bit, byte 3 (most significant byte)
- **Byte 12** – 32-bit pointer to start of hardware SLR3 essential bit data, byte 0 (least significant byte)
- **Byte 13** – 32-bit pointer to start of hardware SLR3 essential bit data, byte 1
- **Byte 14** – 32-bit pointer to start of hardware SLR3 essential bit data, byte 2
- **Byte 15** – 32-bit pointer to start of hardware SLR3 essential bit, byte 3 (most significant byte)
- Remaining 112 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. The functional coverage compares design behavior against the expected behavior.
- Static checks through a checking tool suite:
    - Linting
    - Clock Domain Crossing

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

- Devices with 256 Mb read boundaries:
    - M25P128 (ST Microelectronics/Numonyx)
    - M25L25635E (Macronix)
    - N25Q512 (Micron)
    - N25Q00 (Micron)
- Devices with no read boundaries:
    - MT25QL01GB (Micron)
    - MT25QL02GC (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 takes 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.

Upgrading

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

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations between core versions.

In v3.0 of the core, there were several changes that make the core pin-incompatible with the previous version(s). These changes were required to support new error detection features (Detect only and Diagnostic Scan).

Changes from v2.0 to v3.0

Parameters Removed

In v3.0 of the core, the following unused IP parameters were removed:

- INTERFACE
- ENABLE_CONFIG_SCAN
- MEMORY_TYPE
- MEMORY_IO_TYPE
- LOCATE_HELPER_BLOCKS

Ports Added

Table B-1 shows two new output ports `status_detect_only` and `status_diagnostic_scan` which appear on the controller output port.

Table B-1: **Ports Added in v3.0**

I/O	Port Name	Description	What to do
O	status_detect_only	The Detect only status signal is active when the controller is executing a detect only scan.	This signal needs to be integrated into any user logic that monitors the status_* state signals to verify that the controller continues to function normally.
O	status_diagnostic_scan	The Diagnostic Scan status signal is active when the controller is executing a diagnostic scan.	This signal needs to be integrated into any user logic that monitors the status_* state signals to verify that the controller continues to function normally.

Other Changes

In v3.0 of the core, the error detection report has added one line of additional information in the beginning of the report:

```
RI XX                    Reserved Information
```

Migrating from UltraScale to UltraScale+ Devices

SEM IP is device dependent. When migrating a design from UltraScale™ to UltraScale+™ devices, Xilinx recommends that you regenerate the IP from scratch. There are differences in behavior and port widths which requires you to modify the surrounding logic and RTL accordingly.

Utilizing an Evaluation Board to Demonstrate SEM Controller Behavior

The SEM IP controller is device dependent. When generating the SEM controller IP, you can choose to target evaluation boards. When the following evaluation boards are targeted, the generated IP and system-level example designs are tailored for out-of-the-box hardware bring-up (the device is chosen by selecting the board):

- Kintex® UltraScale™ KCU105 Evaluation Platform
- Virtex® UltraScale™ VCU108 Evaluation Platform

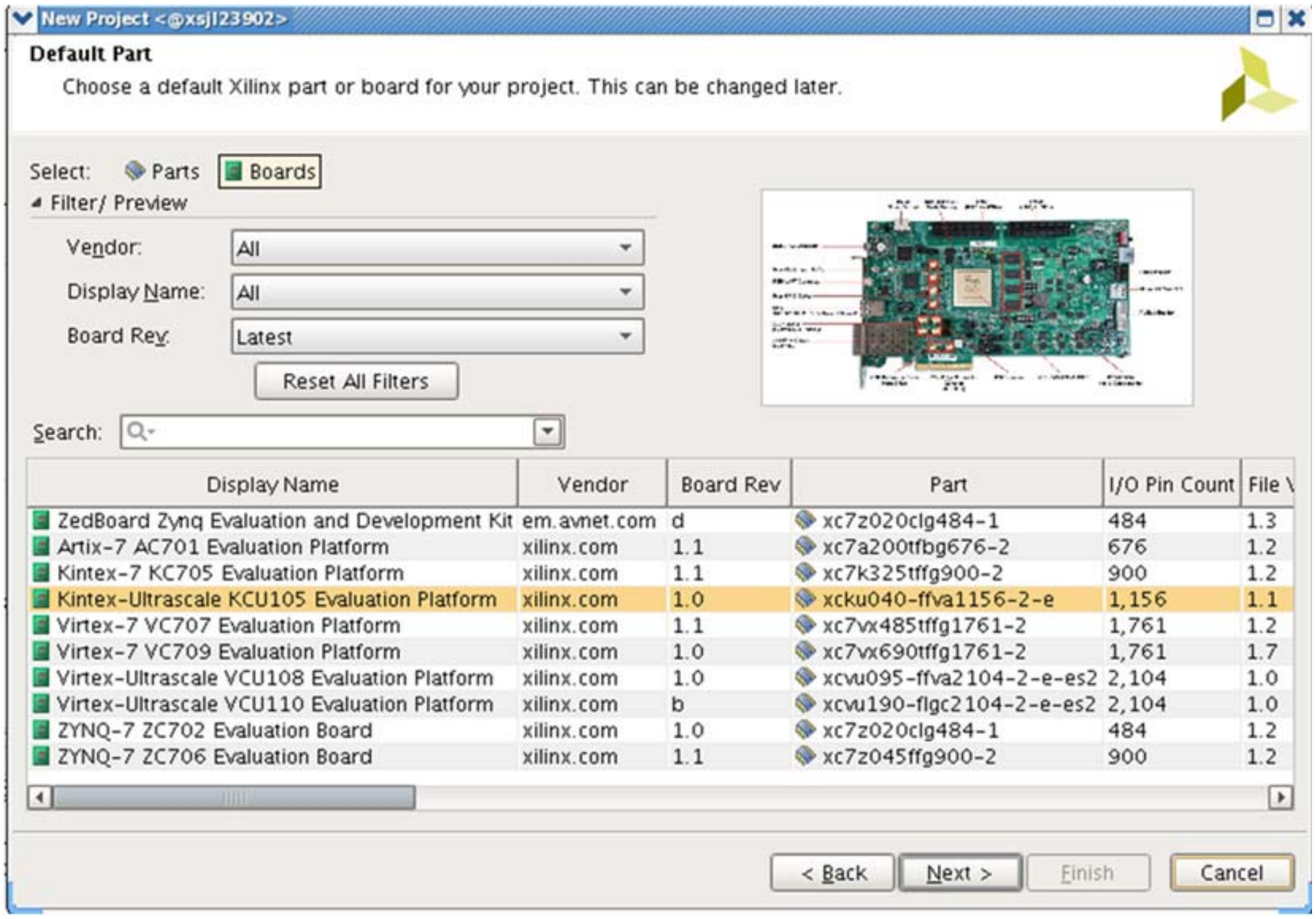
You can use these designs to build an understanding of the IP behavior and also demonstrate its characteristics.

For a demonstration of the SEM controller behavior on the Zynq® UltraScale+ MPSoC ZCU102 Evaluation Platform, see *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices* (XAPP1298) [Ref 4] and *Integrating LogiCORE SEM IP with AXI in Zynq UltraScale+ Devices* (XAPP1303) [Ref 5].

Generating Bitstream

When the supported evaluation boards are targeted, the IP configuration, generated system-level example design and constraints are configured by default to be compatible with the targeted device and platform. To generate a bitstream, follow these steps:

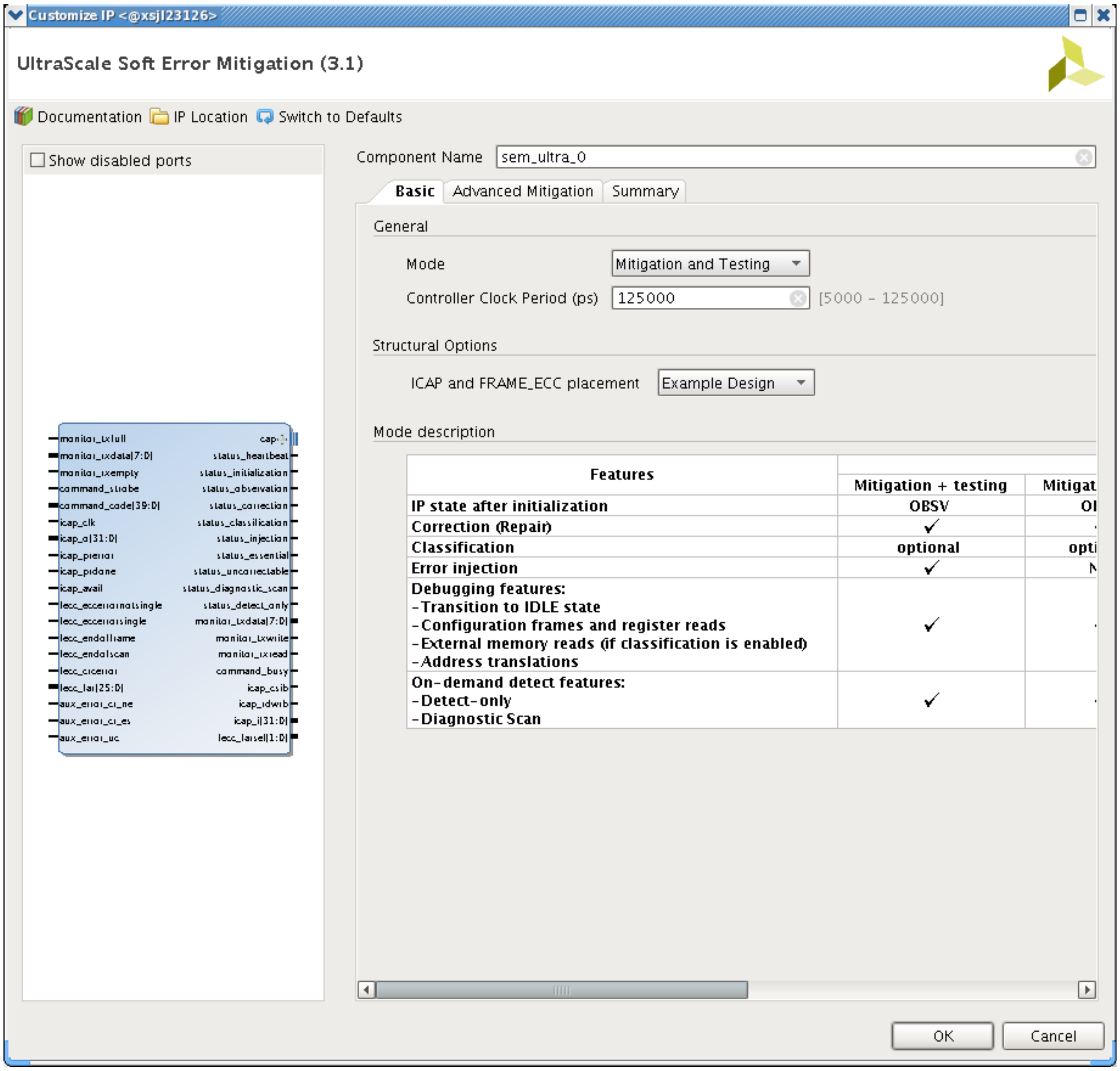
1. Create a new Vivado® RTL project and choose one of the supported boards (Figure C-1):



Display Name	Vendor	Board Rev	Part	I/O Pin Count	File V
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676	1.2
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.2
Kintex-Ultrascale KCU105 Evaluation Platform	xilinx.com	1.0	xcku040-ffva1156-2-e	1,156	1.1
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.2
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.7
Virtex-Ultrascale VCU108 Evaluation Platform	xilinx.com	1.0	xcvu095-ffva2104-2-e-es2	2,104	1.0
Virtex-Ultrascale VCU110 Evaluation Platform	xilinx.com	b	xcvu190-flgc2104-2-e-es2	2,104	1.0
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.2
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.2

Figure C-1: **Generating a Bitstream**

2. Find the **UltraScale Soft Error Mitigation** IP in the Vivado IP catalog and generate the core. You do not need to modify any of the parameters because the defaults are set to the targeted platform.



Component Name: sem_ultra_0

General

Mode: Mitigation and Testing

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

Structural Options

ICAP and FRAME_ECC placement: Example Design

Mode description

Features	Mitigation + testing	Mitigat
IP state after initialization	OBSV	OI
Correction (Repair)	✓	·
Classification	optional	opti
Error injection	✓	N
Debugging features: -Transition to IDLE state -Configuration frames and register reads -External memory reads (if classification is enabled) -Address translations	✓	·
On-demand detect features: -Detect-only -Diagnostic Scan	✓	·

Figure C-2: **SEM Controller Basic Tab**

3. Generation of the IP completes after the **Out-of-Context Module Runs** finishes. Select the generated IP and select **Open IP Example Design**.

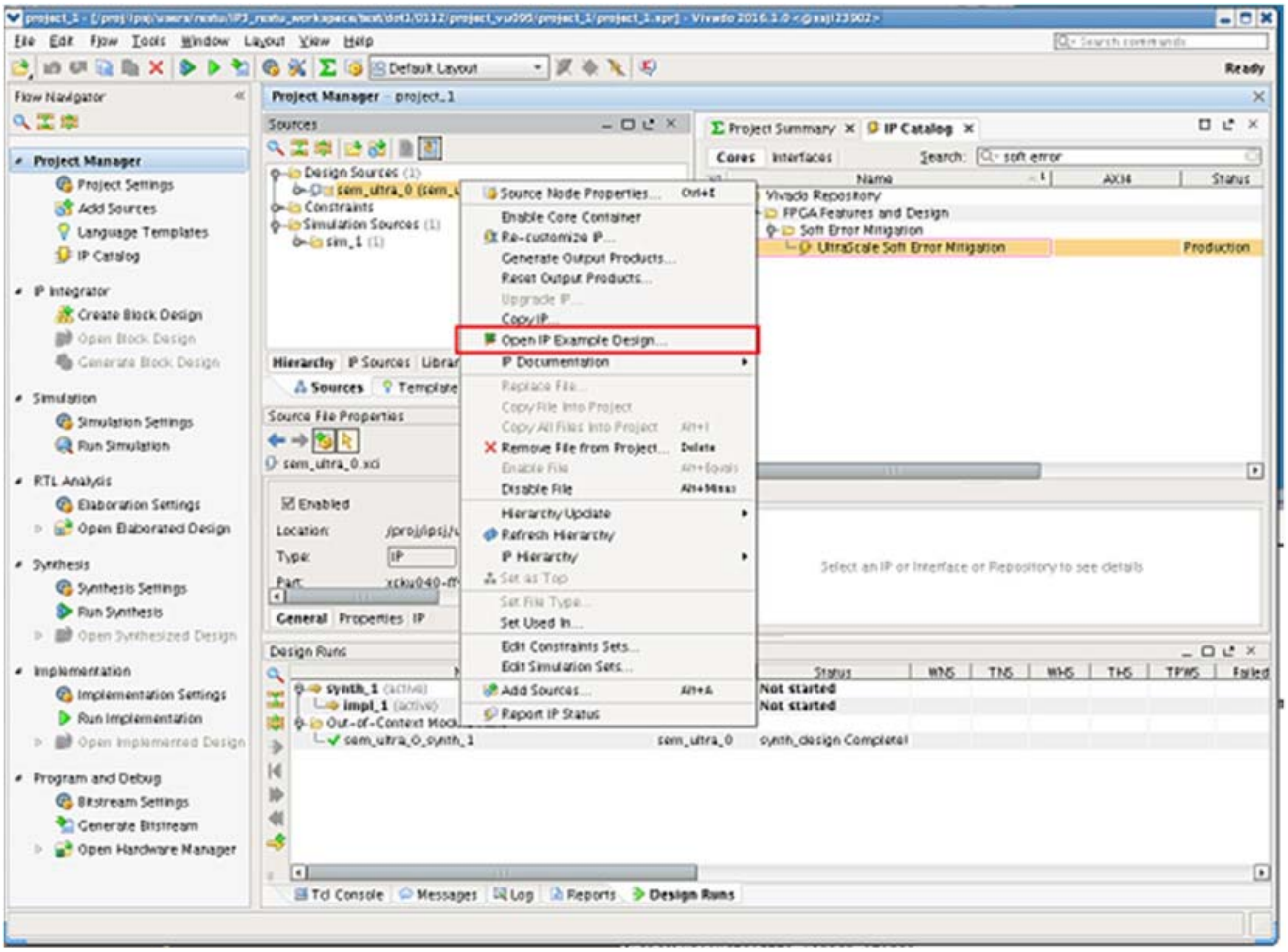
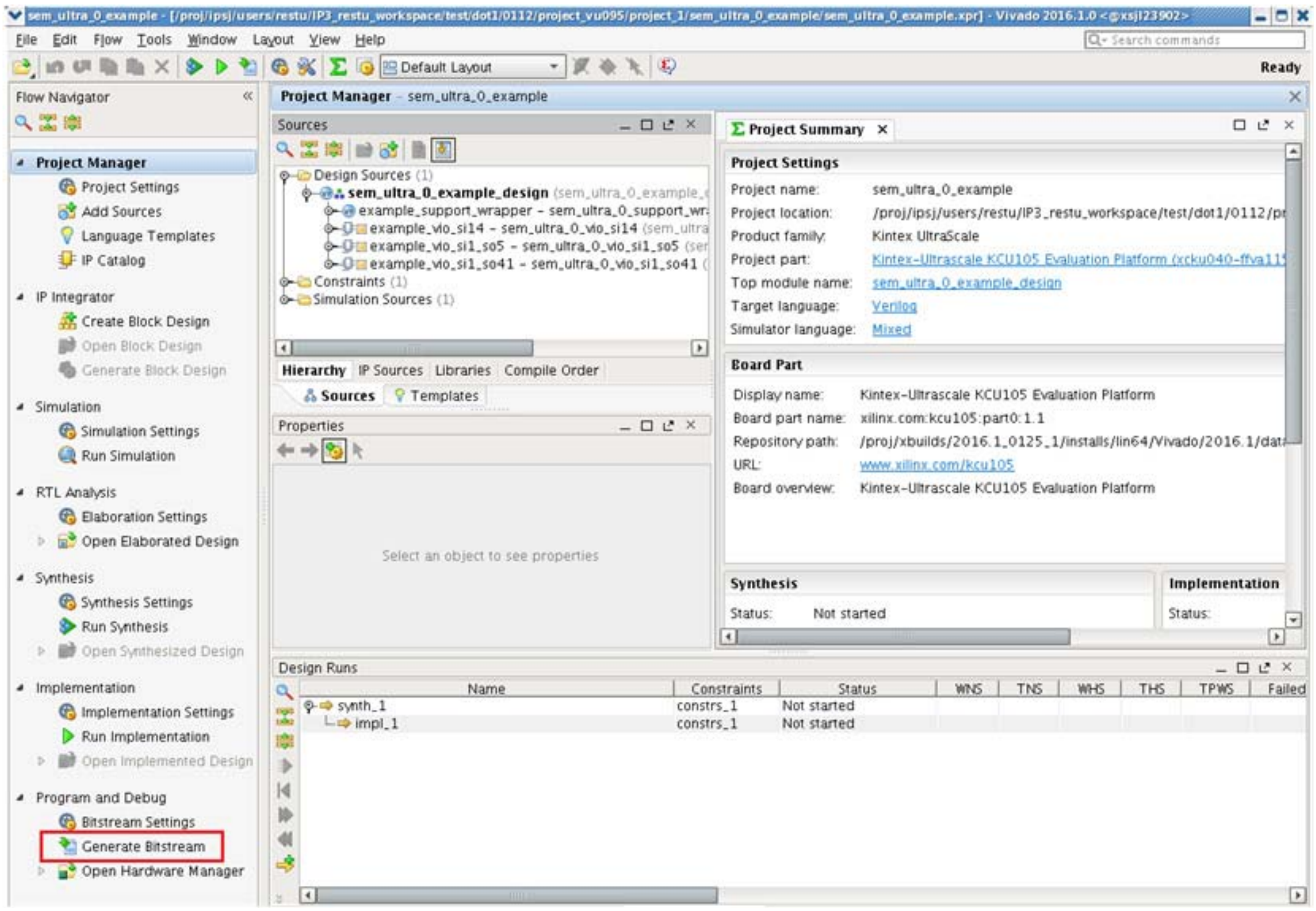


Figure C-3: **SEM Controller Open IP Example Design**

4. Next, synthesize, implement, and generate the bitstream for the example design. This bitstream is used to program the device on the evaluation platform.



Project Settings

Project name:	sem_ultra_0_example
Project location:	/proj/ipsj/users/restu/IP3_restu_workspace/test/dot1/0112/pr
Product family:	Kintex UltraScale
Project part:	Kintex-Ultrascale KCU105 Evaluation Platform (xcku040-ffva115
Top module name:	sem_ultra_0_example_design
Target language:	Verilog
Simulator language:	Mixed

Board Part

Display name:	Kintex-Ultrascale KCU105 Evaluation Platform
Board part name:	xilinx.com:kcu105:part0:1.1
Repository path:	/proj/xbuilds/2016.1_0125_1/installs/lin64/Vivado/2016.1/dat
URL:	www.xilinx.com/kcu105
Board overview:	Kintex-Ultrascale KCU105 Evaluation Platform

Synthesis Status: Not started

Implementation Status:

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed
synth_1	constrs_1	Not started						
impl_1	constrs_1	Not started						

Figure C-4: **SEM Controller Generate Bitstream**

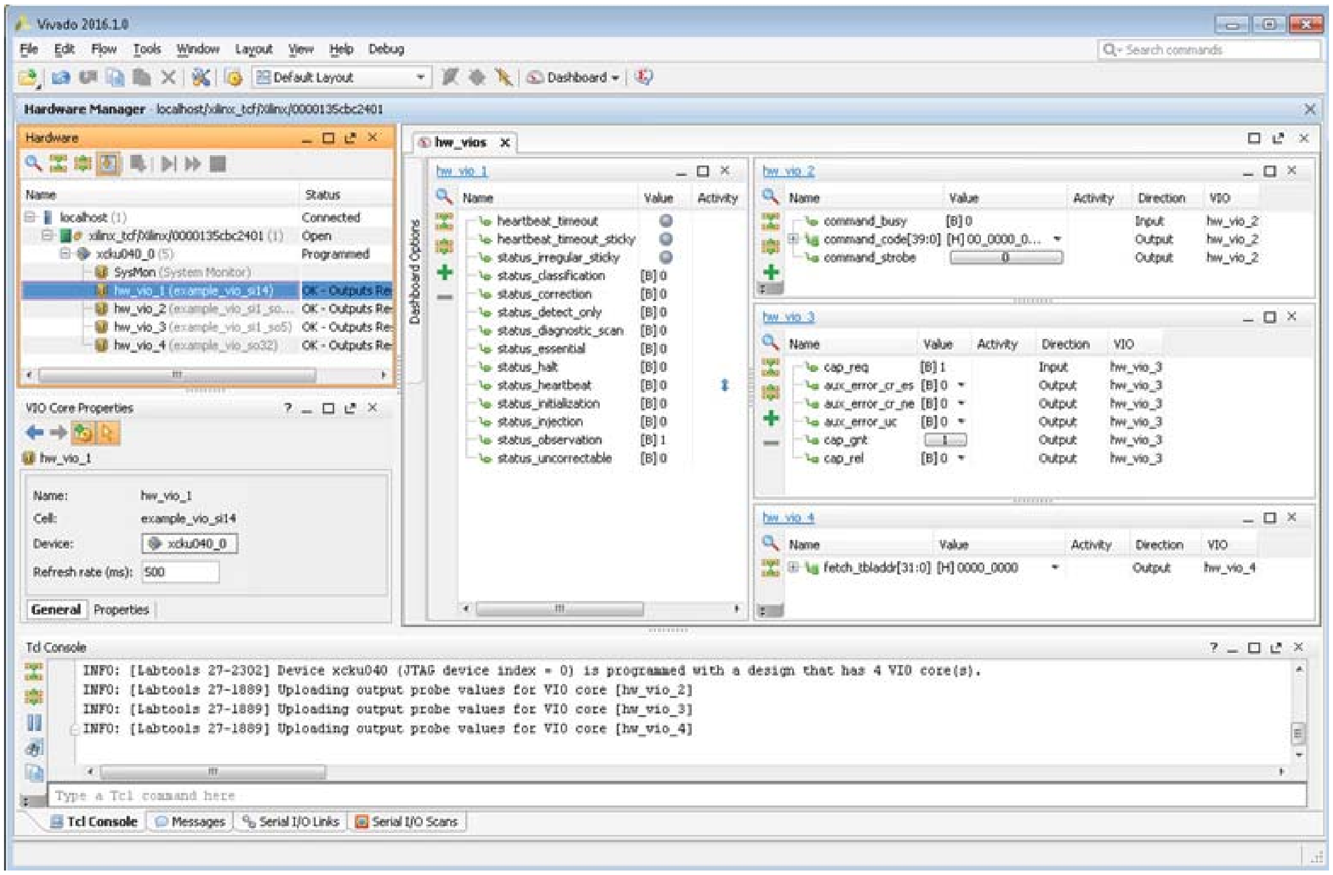
Hardware and Software Setup

After a bitstream is generated, connect the Platform USB (or Digilent®) cable and USB UART cable to the computer and board. Ensure that the cables are connected by opening up the **Devices and Printers** list in the **Control Panel** and confirm the following:

- Silicon Labs CP210x USB to UART Bridge
- Xilinx (this refers to the platform USB cable)

To issue commands and receive status information from UART Interface of the SEM controller, a terminal emulator program (for example, Teraterm) that supports serial port connection needs to be used and configured. For more information on how to configure the terminal emulator program, see Switching Behavior in Chapter 3.

After programming the bitstream and `debug_nets.ltx` file for the VIO cores, add the `hw_vio_*` windows into the **Hardware Manager** to view and also manipulate the SEM signals.


Vivado 2016.1.0
File Edit Flow Tools Window Layout View Help Debug
Default Layout
Dashboard
Search commands
Hardware Manager - localhost/xilinx_tcf/Xilinx/0000135cbc2401
Hardware
VIO Core Properties
Dashboard Options
hw_vios
General Properties
Tcl Console Messages Serial I/O Links Serial I/O Scans

Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/0000135cbc2401 (1)	Open
xcku040_0 (5)	Programmed
SysMon (System Monitor)	
hw_vio_1 (example_vio_si14)	OK - Outputs Re
hw_vio_2 (example_vio_si1_so...	OK - Outputs Re
hw_vio_3 (example_vio_si1_so5)	OK - Outputs Re
hw_vio_4 (example_vio_so32)	OK - Outputs Re

hw_vio_1

Name:	hw_vio_1
Cell:	example_vio_si14
Device:	xcku040_0
Refresh rate (ms):	500

hw_vio_1

Name	Value	Activity
heartbeat_timeout	●	
heartbeat_timeout_sticky	●	
status_irregular_sticky	●	
status_classification	[B] 0	
status_correction	[B] 0	
status_detect_only	[B] 0	
status_diagnostic_scan	[B] 0	
status_essential	[B] 0	
status_halt	[B] 0	
status_heartbeat	[B] 0	↕
status_initialization	[B] 0	
status_injection	[B] 0	
status_observation	[B] 1	
status_uncorrectable	[B] 0	

hw_vio_2

Name	Value	Activity	Direction	VIO
command_busy	[B] 0		Input	hw_vio_2
command_code[39:0]	[H] 00_0000_0...		Output	hw_vio_2
command_strobe	0		Output	hw_vio_2

hw_vio_3

Name	Value	Activity	Direction	VIO
cap_req	[B] 1		Input	hw_vio_3
aux_error_cr_es	[B] 0		Output	hw_vio_3
aux_error_cr_ne	[B] 0		Output	hw_vio_3
aux_error_uc	[B] 0		Output	hw_vio_3
cap_gnt	1		Output	hw_vio_3
cap_rel	[B] 0		Output	hw_vio_3

hw_vio_4

Name	Value	Activity	Direction	VIO
fetch_tbladdr[31:0]	[H] 0000_0000		Output	hw_vio_4

```
INFO: [Labtools 27-2302] Device xcku040 (JTAG device index = 0) is programmed with a design that has 4 VIO core(s).
INFO: [Labtools 27-1889] Uploading output probe values for VIO core [hw_vio_2]
INFO: [Labtools 27-1889] Uploading output probe values for VIO core [hw_vio_3]
INFO: [Labtools 27-1889] Uploading output probe values for VIO core [hw_vio_4]
```

```
Type a Tcl command here
```

Figure C-5: **SEM Controller Hardware Manager Window**

Demonstration on Communicating with the SEM Controller through the UART Interface

This section gives a step-by-setup instruction on how to perform a basic error injection and the expected status output from the SEM controller by using the terminal emulator to communicate with the SEM controller through the UART Interface. All commands are issued and status received using the terminal emulator. The instructions goes through the Initialization report, Status report, Idle versus Observation states, Error Injection, and Correction report. This example assumes that the IP is configured in the Mitigation and Testing mode with Classification disabled which is the default configuration of the Vivado Integrated Design Environment (IDE).

To understand how the IP transitions through each state during this example, see the state diagram in Figure 3-5, page 59. The list of commands (and its format) and reports used in this demonstration can be found in UART Interface Messages in Chapter 3 and UART Interface Commands in Chapter 3 of this document.

When the generated bitstream is programmed onto the device, an initialization report is received in the terminal emulator. The report looks like the following:

```
SEM_ULTRA_V3_1       Name and version
SC 01                State transition to Initialization state
FS 04                Core Configuration Information
AF 01                Additional Core Configuration Information
ICAP OK              ICAP Available
RDBK OK              Status: Readback Active
INIT OK              Status: Completed Setup
SC 02                State transition to Observation
O>                   Observation prompt
```

Now, the controller has finished initialization and is observing the FPGA configuration system for indication of error conditions. The `status_observation` signal is asserted in the VIO window.

To emulate an error occurring in the configuration memory, you have to transition the controller to the Idle state. You can do this by sending the command to the UART interface from the terminal emulator:

```
O> I                 Enter Idle Command
SC 00                State change into Idle state
I>                   Idle prompt
```

Now, all the status flags (all `status_*` ignoring `status_essential` and `status_uncorrectable`) are deasserted indicating that the controller is in Idle state.

When the controller enters Idle state, it disables the built-in configuration memory scan and checks. Available commands in this state are shown in Table 3-1, page 64.

If a Full status report is requested by sending an "S" through the UART interface, you receive the following output:

```
I> S                 Status request command ("S")
SN 00                SLR number
SC 00                Current State
FC 00                Current Flags
RI 00                Reserved Information
MF 00006643          Maximum Linear Frame Count
TS XXXXXXXX          Timestamp
TB XXXXXXXX          Table Base
CB XXXXXXXX          Classification Base
CL 001               Classification Level
I>                   Idle Prompt
```

After the Full status report is complete, the Idle prompt returns to receive the next command.

To emulate an error occurring in the configuration memory, you can perform an error injection. In general, Xilinx recommends performing an error injection in the ECC word of the frame. An error injection in the ECC word does not impact the configured design but enables you to observe what happens when a configuration error is found.

Xilinx also recommends that you query the content of the frame in which the error is injected into, before and after injecting errors. This enables you to confirm that the error injection is successful in altering the configuration memory and whether the controller will detect and correct the injected errors.

In this example, inject an error in Frame 0, Word 61, and Bit 0 which is a bit in the ECC word for an UltraScale device.

First, perform a `Query` command to read the content of frame 0.

```
I> Q C000000000    Query command using LFA
{61 lines of 00000000}
00000000           Location we will inject an error
{61 lines of 00000000}
I>
```

Now, perform the error injection at Frame 0, Word 61, and Bit 0.

```
I> N C0000007A0    Error injection command using LFA
SC 10              State transition to Error Injection
SC 00              State transition to Idle
I>                 Idle Prompt
```

When the IP performs error injection, it briefly enters the Error Injection state, asserting the `status_injection` signal during that time.

Finally, perform a `Query` command again to ensure an error has been introduced at the chosen location.

```
I> Q C000000000    Query command by LFA
{61 lines of 00000000}
00000001           Location we have injected an error
{61 lines of 00000000}
I>
```

Because the bit targeted by the error injection command has changed, it is confirmed that the error injection was successful.

To demonstrate error detection and correction capability of the controller, you need to command the controller to re-enter the Observation state and resume observing the FPGA configuration system for indication of error conditions.

To enter the Observation state, issue the following command:

```
I> O               Enter Observation Command
SC 02              State transition to Observation
O>                 Observation prompt
RI XX              Reserved information
```

```
SC 04               State transition to Correction
ECC                 ECC error detected
TS 0000245D         Timestamp
PA 0000000          PFA of detected error
LA 0000000          LFA of detected error
COR                 Error correction report begins
WD 3D BT 00         Corrected word and bit values
END                 Error correction report end
FC 00               Flag updated with correctability
SC 08               State transition to Classification
FC 40               Flag updated with essentialness (if classification is disabled, it
is always essential)
SC 02               State transition to Observation
O>                  Observation prompt
```

Observe that after transition to the Observation state, the controller subsequently detected and reported the error, indicated by the error detection and correction report generated in the Correction state.

After completing the error correction, the controller transitions to the Classification state before transitioning back into the Observation state and resumes observing the FPGA configuration system for further indication of error conditions.

Note: If the Classification feature is disabled, the flag always indicates that the error is essential.

The Flag Change update provides two pieces of information:

- Correctability (reflected on `status_uncorrectable`)
- Essentialness (reflected on `status_essential`).

Correctability gets updated after the correction report ends and essentialness gets updated before it transitions out of the Classification state. Until either of those stages are hit again, the Flag Change variable value continues to remain the same. See Table 3-4, page 77 to understand the different Flag Change Values and their corresponding meanings.

Note: Each State and Flag Change reported on the UART interface are reflected on the `status_*` signals in the VIO window.

For more information on emulating the occurrence of errors in the configuration memory, see Appendix E, Error Injection Guidance.

In this demonstration, all commands to the IP were given through the terminal emulator to the UART Interface. The same commands can be executed using the Command Interface through the VIO window. Each correctly given command through this interface is echoed on the UART Interface resulting in the same status and reports shown above. For more information on how to provide commands through the Command Interface, see Command Interface in Chapter 3.

Example UART Logs

The following sections provide example logs that are based on the UltraScale+™ FPGAs. These short examples from the SEM IP UART logs are a sample of the observable activity when the IP is configured in Mitigation and Testing mode with the Error Classification feature disabled, except where noted.

Initialization Report

```
SEM_ULTRA_V3_1
SC 01
FS 04
AF 01
ICAP OK
RDBK OK
INIT OK
SC 02
O>
```

Status Report

Non-SSI from Idle State

```
I> S
SN 00
SC 00
FC 00
RI 00
MF 0000BBB6
TS 00000877
TB XXXXXXXX
CB XXXXXXXX
CL 001
I>
```

Non-SSI from Observation State

```
O> S
SN 00
SC 02
FC 00
RI 00
O>
```

SSI from Idle State

```
I> S
SN 00
SC 00
FC 00
RI 00
MF 0000DD19
TS 00000020
TB XXXXXXXX
CB XXXXXXXX
CL 001
SN 01
SC 00
FC 00
RI 00
MF 0000DD19
TS 00000020
TB XXXXXXXX
CB XXXXXXXX
CL 001
SN 02
SC 00
FC 00
RI 00
MF 0000DD19
TS 00000020
TB XXXXXXXX
CB XXXXXXXX
CL 001
I>
```

SSI from Observation State

```
O> S
SN 00
SC 02
FC 40
RI 00
SN 01
SC 02
FC 40
RI 00
SN 02
SC 02
FC 40
RI 00
```

Injection Commands

Injection Is Enabled

```
I> N C000A098000
SC 10
SC 00
I>
```

Injection Is Disabled

```
I> N C000A098000
SC 00
I>
```

Enter Observation Command

```
I> O
SC 02
O>
```

Enter Idle Command

```
O> I
SC 00
I>
```

Enter Detect Only Command

```
I> D
SC 20
D>
```

Software Reset Command

```
I> R 04
SEM_ULTRA_V3_1
SC 01
FS 04
AF 01
ICAP OK
```

```
RDBK OK
INIT OK
SC 02
O>
```

Frame Address Translation Command

```
I> T C00051A090D
0008000090D
I>
```

Query Command

```
I> Q C00051A090D
(word  0 - 8-digit hex value)
(word  1 - 8-digit hex value)
...
(word 92 - 8-digit hex value)
I>
```

External Memory Reads Command (Xmem)

```
I> X 00000000
94
I>
```

Peek Command

```
I> P 01
00000001
I>
```

Observation Error Reports

1-bit Correctable ECC – Error Classification Disabled

```
O>
RI 00
SC 04
ECC
TS 00000971
```

```
PA 00180200
LA 0000A098
COR
WD 22 BT 10
END
FC 00
SC 08
FC 40
SC 02
O>
```

2-bit Correctable ECC With One Essential Bit – Error Classification Enabled

```
O>
RI 00
SC 04
ECC
TS 00000971
PA 00180200
LA 0000A098
COR
WD 00 BT 01
WD 00 BT 03
END
FC 00
SC 08
CLA
WD 00 BT 03 LV 01
END
FC 40
SC 02
O>
```

Uncorrectable ECC

```
O>
RI 00
SC 04
ECC
TS 00000971
PA 00180200
LA 0000A098
COR
END
FC 20
SC 08
FC 60
SC 00
I>
```

CRC Error

```
O>
RI 00
SC 04
CRC
TS 00000B4F
FC 20
SC 08
FC 60
SC 00
I>
```

AUX Error

```
O>
RI 00
SC 04
AUX
TS 00000B4F
FC 00
SC 08
FC 00
SC 02
O>
```

ROM Error

```
O>
RI 00
SC 04
ROM
TS 00000B4F
FC 40
SC 08
FC 00
SC 02
O>
```

Detect Only Error Reports

Detects the first error. then returns to idle.

Correctable ECC

In detect only mode, this error is not corrected.

```
D>
RI 00
ECC
TS 00001034
PA 00000000
LA 00000000
WD 00 BT 00
FC 60
SC 00
I>
```

Uncorrectable ECC

```
D>
RI 00
ECC
TS 0000328D
PA 00000000
LA 00000000
FC 60
SC 00
I>
```

CRC Error

```
D>
RI 00
CRC
TS 00001866
FC 60
SC 00
I>
```

ROM Error

```
D>
RI 00
ROM
TS 000002C6
FC 60
SC 00
I>
```

AUX Error

```
D>
RI 00
AUX
TS 00001BAC
FC 60
SC 00
I>
```

Diagnostic Scan

Scans device once for all errors. Reports all detected errors, then returns to idle.

Correctable 1-bit ECC

In a diagnostic scan, the reported errors are not corrected.

```
I> U
SC 40
RI 00
ECC
TS 0000108D
PA 00000000
LA 00000000
WD 00 BT 00
SC 00
I>
```

Uncorrectable ECC

```
I> U
SC 40
RI 00
ECC
TS 000032EB
PA 00000000
LA 00000000
SC 00
I>
```

Uncorrectable ECC followed by a Correctable ECC

```
I> U
SC 40
RI 00
ECC
TS 000032EB
PA 00000000
LA 00000000
RI 00
ECC
TS 00003E08
PA 018C0400
LA 0000D500
WD 00 BT 00
SC 00
I>
```

No Error Detected

```
I> U
SC 40
SC 00
I>
```

Error Injection Guidance

The SEM IP provides provision for error injection to support verification of the controller and evaluation of applications of the controller. Generally, the objective of the error injection exercise should drive the type of error injection that should be performed.

Objective: Basic Error Injection Testing

To perform this type of error injection, you are interested in achieving the following goals:

- Confirm that the controller detects and corrects errors
- Confirm that the system is logging errors that are detected and corrected as expected
- Not interested in testing system behavior when error impacts design function

Location of error injections:

- Inject errors in ECC word of a frame.
- For UltraScale™ architecture, the ECC word spans between word 60 and 61 of each frame. Xilinx recommends injecting errors in the lower byte of word 61.
- For UltraScale+™ FPGAs and MPSoCs, the ECC word spans between word 45 and 46 of each frame. Xilinx recommends injecting errors in the low byte of word 46. Many UltraScale+ devices contain configuration memory address ranges which are reserved for support of processing system integration (regardless of processing system presence or use). Error injected into these reserved configuration memory addresses will not be detected. Xilinx recommends using linear frame addresses larger than 2/3 × Max Linear Frame as general guidance in selection of an address which maps to physically implemented configuration memory.

By injecting errors in this location, the injected error does not interfere with the controller or design function.

If the objective of error injection does not fall into this scope, open a support case for guidance.

The following lists general guidance for injecting errors using the Monitor or Command interface from the controller:

- Perform error injection using Linear Frame Addresses (LFA). The valid range of address is from 0 to maximum LFA value –2. For example, the maximum frame for a XCKU040 device is 26,179. The valid range of addresses to inject errors are 0 to 26,177 (26,179 – 2). The maximum LFA value for a given device is recorded in Table 2-2 and this value is also reported by the controller in its status report (MF {8-digit hex value}).
- Always perform a `Query` command before and after error injection to verify that the error injection was successful and the bit is not masked from reads and writes.
- Inject one error at a time and confirm that the controller transitions to Injection state and returns to the Idle state again before performing the next injection. If the controller is not in Idle state, the error injection instruction can be dropped or lost.
- Xilinx recommends using the Monitor or UART Interface to perform and monitor error injection because this interface provides the most verbose information.
- When using the Command Interface, you must monitor the Status Interface to verify that the controller is in Idle before injecting any error and also to verify that controller transitions to error injection state after the command is given.
- If a single bit error is injected and is not detected by the controller, inject an error to the same bit again before proceeding to inject an error in a different bit.
- If injecting more than 1-bit at a time, you should back-out all the errors that are not corrected before performing another set of error injection. Alternatively, reprogram the device before resuming further error injection testing.

Special considerations when injecting errors:

- Due to masked and not implemented frames, there is a possibility that an error injected is not detected (hence not corrected). Performing a `Query` command before and after error injection gives insight on what to expect.
- Usually frames that are masked are related to dynamic memory (DRP, SRL, etc.).
- If an injected error caused the controller to report an uncorrectable error, reconfigure before doing anymore testing.

IP Design Checklist

The following checklist is provided with the SEM controller to assist your use and integration of the IP.

	Review the unsupported features (see Unsupported Features in Chapter 1) when using the SEM IP to ensure compatibility with the design.
	Decide on your SEU mitigation approach and make trade-offs of different features available in the IP. See Key Considerations for SEM IP Adoption. Most would 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 the Idle state. Others might opt to migrate to the Mitigation only mode during production to disable any error injection capabilities.
	Consider if your system would take different actions if the IP reports a correctable error, uncorrectable error, correctable error that is essential, etc. For example, if a correctable error that is not essential is detected, do nothing but if a correctable error that is essential is detected, reconfigure the device. At the minimum, you should log all errors that are detected using the Monitor or UART interface.
	Calculate a reliability estimation of a design (including SEM IP) using the pre-design (spreadsheet-based) SEU FIT estimation tool for your entire deployment to understand how frequently you can expect to log an error. The pre-design (spreadsheet-based) SEU FIT estimation tool can be downloaded from here: Xilinx Quality and Reliability Web Page
	Verify that the system clock provided to the controller follows the recommended guidelines. See System Clock Interface in Chapter 3. The SEM controller expects the clock provided to it and the configuration primitives to be stable (no glitching and does not violate F_{Max}). If the initial clock is unstable, employ the use of a BUFGCE to buffer the clock and only enable the buffer when the clock is stable.
	Consider setting initial cap_gnt input to 0 and adding a master control switch on the signal to control when the controller initializes. cap_gnt should only be set to 1 when ICAP is ready.
	Consider adding a master control switch on the clock provided to the controller. This determines when the clock is stable and provides it to the controller.
	Check that all unused input ports are tied off as described in the individual sections of the Interface, see Interfaces in Chapter 3. If the ports are not tied off correctly, the IP might not initialize.
	Review all the system-level considerations for each used interface as described in the individual sections of Interface, see Interfaces in Chapter 3.
	For future debugging purposes, at the minimum Xilinx recommends buffering the Monitor Interface output into a FIFO. This information is required to debug any future issues. See Monitor Interface in Chapter 3.
	For future debugging purposes, Xilinx recommends that the post-synthesis DCP of the design is stored to debug any future issues.

	When logging errors and location of the errored frame, use the Monitor or UART interface. See Monitor Interface in Chapter 3 and UART Interface in Chapter 3. Xilinx discourages monitoring the FRAME_ECC interface.
	If you are having issues with the UART interface (message is garbled, characters are being dropped, etc), check your V_ENABLETIME value settings are compatible to the baud rate and clock frequency that you selected. Also, that the external device or terminal emulator that the UART interface is connected to is configured correctly. See Switching Behavior in Chapter 3.
	Each SEM controller generated is specific to the device it is targeted for. Do not take a SEM controller generated for one device and use it for another device.
	Check that the controller is correctly constrained as described in Constraining the Core in Chapter 4. • Clock period constraints for the controller icap_clk • False path constraint on some asynchronous path between the controller and ICAP • ICAP and FRAME_ECC constraints might need to be applied for successful placement
	Note that it is not possible to observe controller behaviors in simulation but design simulations that instantiate the controller is supported. See Simulation in Chapter 4.
	Do not place any pipeline registers between the controller and the FRAME_ECC/ICAP primitives outside what is delivered in the example design.
	Review Systems in Chapter 3 to consider if a system-level supervisory function is needed to monitor the Soft Error Mitigation solution.
	Consider using block RAM ECC to protect your block RAMs from SEUs and other methods to further increase the design resiliency to soft errors if the estimated FIT of the design is not within the intended budget.
	Integrate the SEM IP into your design cycle early and validate your design in hardware with it enabled. See Integration and Validation in Chapter 4.
	When debugging issues, see Appendix J, Debugging for further guidance.
	For quick reference of how the IP should behave, see Figure 3-5 through Figure 3-7 that describe the valid state transition diagrams of the controller.
	When performing error injection, execute a Query command before and after to verify the effects of the error injection. See Appendix E, Error Injection Guidance.

SPI Bus Timing Budget

SPI Bus Clock Waveform and Timing Budget

The SPI flash device has requirements on the switching characteristics of its input clock. This analysis is for the clock signal generated for the SPI flash device by the system-level design example. Completion of this analysis requires board-level signal integrity simulation capability.

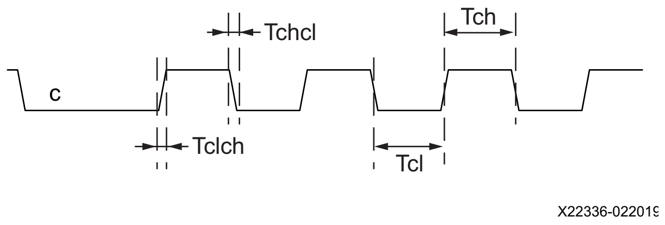


Figure G-1: **SPI Flash Device Input Clock Requirements**

The following parameters, shown in Figure G-1, are defined as requirements on the clock input to the SPI flash device:

- T_{clch} = SPI bus clock maximum rise time requirement
- T_{chcl} = SPI bus clock maximum fall time requirement
- T_{cl} = SPI bus clock minimum low time requirement
- T_{ch} = SPI bus clock minimum high time requirement

Based on the physical construction of the SPI bus, the I/O characteristics of the FPGA, and the I/O characteristics of any level translator used, the SPI bus clock signal originating at the FPGA exhibits maximum rise and fall times (T_{rise} and T_{fall}) at the SPI flash device. Satisfaction of T_{clch} and T_{chcl} requirements by T_{rise} and T_{fall} must be verified. Should T_{clch} and T_{chcl} requirements not be satisfied, avenues of correction include:

- Change I/O slew rate for the system-level design example SPI bus clock output.
- Change I/O drive strength for the system-level design example SPI bus clock output.
- Select an alternate level translator with more suitable I/O characteristics.

Generally, the T_{clch} and T_{chcl} requirements are easy to satisfy. They exist to prohibit exceptionally long rise and fall times that might occur on a true bus with many loads, rather than the point-to-point scheme used with the system-level design example.

The SPI bus clock generated by the system-level design example is the input clock divided by two. Therefore, the SPI bus clock high and low times are nominally equal to T_{clk}. However, considering actual T_{rise} and T_{fall}, also ensure satisfaction of the following:

- $T_{clk} \geq T_{rise} + T_{ch}$
- $T_{clk} \geq T_{fall} + T_{cl}$

Example:

- T_{clch} = 33 ns (from SPI flash data sheet)
- T_{chcl} = 33 ns (from SPI flash data sheet)
- T_{cl} = 3.375 ns (from SPI flash data sheet)
- T_{ch} = 3.375 ns (from SPI flash data sheet)
- T_{rise} = 2 ns (from PCB simulation)
- T_{fall} = 2 ns (from PCB simulation)

Given this data, perform the following:

1. Check: Is $T_{clch} \geq T_{rise}$? Is 33 ns ≥ 2 ns? Yes
2. Check: Is $T_{chcl} \geq T_{fall}$? Is 33 ns ≥ 2 ns? Yes
3. Calculate: $T_{clk} \geq T_{rise} + T_{ch}$ requires $T_{clk} \geq$ 2 ns + 3.375 ns, or $T_{clk} \geq$ 5.375 ns
4. Calculate: $T_{clk} \geq T_{fall}$ + Tcl requires $T_{clk} \geq$ 2 ns + 3.375 ns, or $T_{clk} \geq$ 5.375 ns

The rise time requirements are satisfied. These requirements on T_{clk} indicate that the SPI Bus Clock Waveform and Timing Budget restrict the system-level design example input clock cycle time to be 5.375 ns or larger.

SPI Bus Transmit Waveform and Timing Budget

The SPI flash device has requirements on the switching characteristics of its input data with respect to its input clock. This analysis is for data capture at the SPI flash device, when receiving data from the system-level design example.

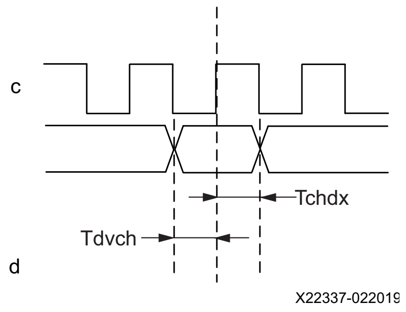


Figure G-2: **SPI Flash Device Input Data Capture Requirements**

The following parameters, shown in Figure G-2, are defined as requirements for successful data capture by the SPI flash device:

- T_{dvch} = SPI flash minimum data setup requirement with respect to clock
- T_{chdx} = SPI flash minimum data hold requirement with respect to clock

The analysis assumes minimum propagation delays are zero. This analysis also assumes the following skews are negligible:

- Skew on input clock distribution to FPGA output flip-flops.
- Skew on output signal paths from FPGA output flip-flops to FPGA pins.
- Skew in PCB level translator channel delays. The level translator on clock and datapaths must be matched for this to be true.
- Skew in PCB trace segment delays. The trace delay on clock and datapaths must be matched for this to be true.
- Duty cycle distortion.

The following parameters are defined as implementation parameters of the SPI flash master helper block and PCB:

- T_{clk} = input clock cycle time (`icap_clk`)
- T_{qfpga} = FPGA output delay with respect to `icap_clk`
- T_{w1} = FPGA to level translator PCB trace delay
- T_{w2} = Level translator to SPI flash PCB trace delay

- T_{dly} = Level translator channel delay

The memory system signaling generated by the SPI flash master helper block implementation is shown in Figure G-3.

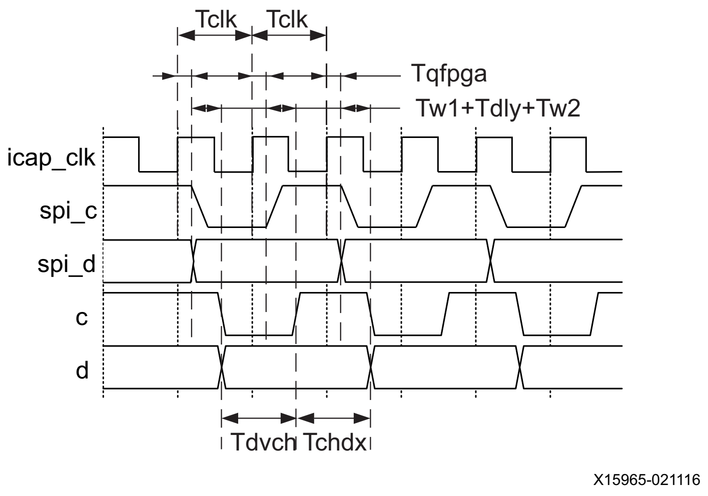


Figure G-3: **Input Data Capture Timing**

Given the stated assumptions, the delays on both the clock and datapaths are identical and track each other over process, voltage, and temperature variations. The following relationships exist:

- $T_{clk} \geq T_{dvch}$
- $T_{clk} \geq T_{chdx}$

Example:

- T_{dvch} = 1.75 ns (from SPI flash data sheet)
- T_{chdx} = 2 ns (from SPI flash data sheet)

1. Calculate: $T_{clk} \geq T_{dvch}$ requires $T_{clk} \geq 1.75$ ns
2. Calculate: $T_{clk} \geq T_{chdx}$ requires $T_{clk} \geq 2$ ns

These requirements on T_{clk} indicate that the SPI Transmit Waveform and Timing Budget restrict the system-level design example input clock cycle time to be 2 ns or larger.

SPI Bus Receive Waveform and Timing Budget

The SPI flash device exhibits certain output switching characteristics of its output data with respect to its input clock. This analysis is for data capture at the system-level design example, when receiving data from the SPI flash device.

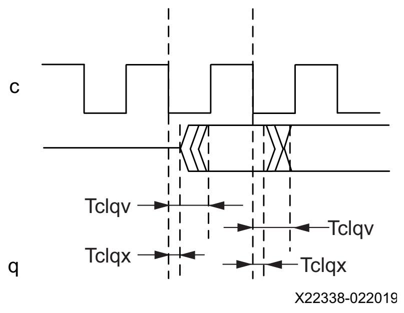


Figure G-4: **SPI Flash Device Output Data Switching Characteristics**

The following parameters, shown in Figure G-4, are defined as the output switching behavior of the SPI flash device:

- T_{clqv} = SPI flash maximum output valid with respect to clock
- T_{clqx} = SPI flash minimum output hold with respect to clock

The analysis assumes minimum propagation delays are zero. This analysis also assumes the following skews are negligible:

- Skew on input clock distribution to FPGA output and input flip-flops.
- Skew in PCB level translator channel delays. The level translator on clock and datapaths must be matched for this to be true.
- Duty cycle distortion.

The following parameters are defined as implementation parameters of the SPI flash master helper block and PCB:

- T_{clk} = input clock cycle time (`icap_clk`)
- T_{qfpga} = FPGA output delay with respect to `icap_clk`
- T_{sfpga} = FPGA input setup requirement with respect to `icap_clk`
- T_{hfpga} = FPGA input hold requirement with respect to `icap_clk`
- T_{w1} = FPGA to level translator PCB trace delay
- T_{w2} = Level translator to SPI flash PCB trace delay
- T_{w3} = SPI flash to level translator PCB trace delay

- T_{w4} = Level translator to FPGA PCB trace delay
- T_{dly} = Level translator channel delay

The timing path is a two cycle path for the SPI flash master helper block, but a single cycle path to the SPI flash device. For the timing analysis, the clock to out of the SPI flash device is modeled as a combinational delay. Both setup and hold requirements at the FPGA must be considered.

The memory system signaling generated by the SPI flash master helper block implementation is shown in Figure G-5 and Figure G-6.

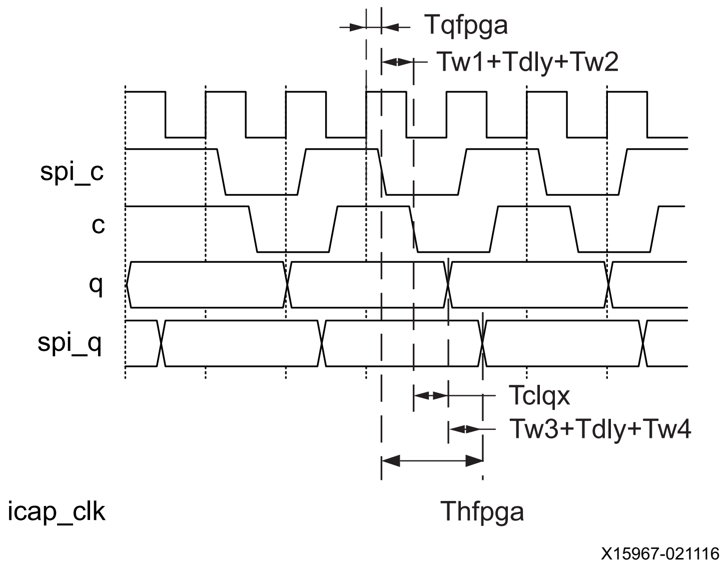


Figure G-5: **Output Data Capture Timing (Hold Analysis)**

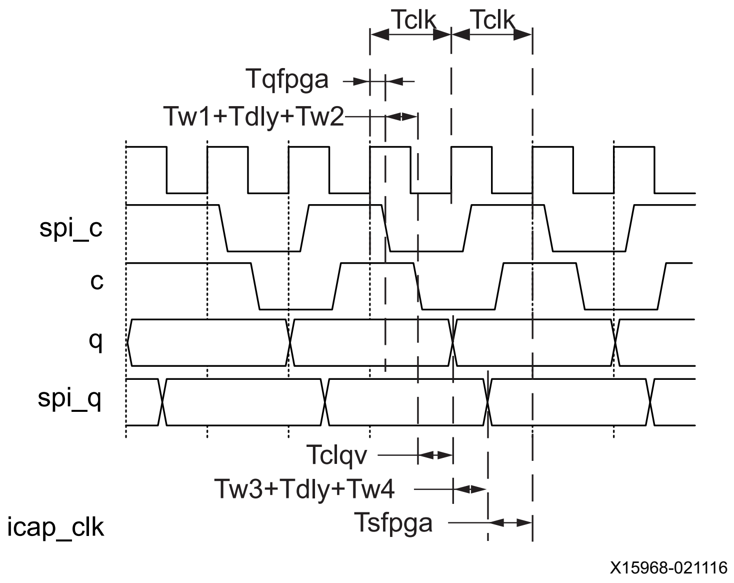


Figure G-6: **Output Data Capture Timing (Setup Analysis)**

The hold path analysis is a pass/fail test. The hold path analysis must be calculated using minimum delay values, for which the following relationship must be verified:

$$T_{hfpga} \leq T_{qfpga,min} + T_{w1} + T_{dly} + T_{w2} + T_{clqx} + T_{w3} + T_{dly} + T_{w4}$$

Substituting zero as a conservative minimum delay for T_{w1}, T_{w2}, T_{w3}, T_{w4}, and T_{dly} yields:

$$T_{hfpga} \leq T_{qfpga,min} + T_{clqx}$$

The setup path analysis must be calculated using maximum delay values:

$$T_{clk} \geq 0.5 \times (T_{qfpga,max} + T_{w1} + T_{dly} + T_{w2} + T_{clqv} + T_{w3} + T_{dly} + T_{w4} + T_{sfpga})$$

Example: Vivado Design Suite, Kintex UltraScale FPGA

- T_{clqv} = 6 ns (from SPI flash data sheet)
- T_{clqx} = 1 ns (from SPI flash data sheet)
- T_{dly} = 2.8 ns (from level translator data sheet)
- T_{w1} = 1 ns (from board simulation)
- T_{w2} = 1 ns (from board simulation)
- T_{w3} = 1 ns (from board simulation)
- T_{w4} = 1 ns (from board simulation)

The FPGA timing parameters must be obtained from the timing report from the implementation of the system-level design example in the FPGA targeted for use in the application. To generate the necessary report, use `report_timing_summary` to generate a report using the `min_max` option.

The examples that follow are excerpts from the timing report generated from a Kintex® UltraScale™ device implementation of the system-level example design. The purpose of the example is to illustrate where to find the required information. If the information is not easily located in the report, increase the maximum number of paths reported.

Locate T_{qfpga} by searching the timing report for flip-flop to pad path analysis at Max at Slow Process Corner, where the destination is identified as `spi_c`.

- T_{qfpga} = I/O Datapath Delay (`spi_c`)
- T_{qfpga} = 1.856 ns, maximum

```
Slack (MET):              15.037ns  (required time - arrival time)
  Source:                 example_support_wrapper/example_support/example_spi/
example_spi_byte/spi_c_ofd/C
                            (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@5.556ns period=11.111ns})
  Destination:            spi_c
                            (output port clocked by clk  {rise@0.000ns fall@5.556ns
period=11.111ns})
```

```
  Path Group:             clk
  Path Type:              Max at Slow Process Corner
  Requirement:            11.111ns  (clk rise@11.111ns - clk rise@0.000ns)
  Data Path Delay:        1.856ns  (logic 1.476ns (79.523%)  route 0.380ns (20.477%))
  Logic Levels:           1  (OBUF=1)
  Output Delay:           -11.111ns
  Clock Path Skew:        -5.294ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    0.000ns = ( 11.111 - 11.111 )
    Source Clock Delay      (SCD):    5.294ns
    Clock Pessimism Removal (CPR):    0.000ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
  Clock Net Delay (Source):      2.304ns (routing 0.335ns, distribution 1.969ns)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
    K20                                               0.000     0.000 r  clk (IN)
                         net (fo=0)                   0.000     0.000    example_ibuf/I
   K20                                                                r  example_ibuf/INBUF_INST/PAD
    K20                  INBUF (Prop_INBUF_HRIO_PAD_O)
                                               0.805     0.805 r  example_ibuf/INBUF_INST/O
                         net (fo=1, routed)           0.092     0.897    example_ibuf/OUT
   K20                                                               r  example_ibuf/IBUFCTRL_INST/I
    K20                  IBUFCTRL (Prop_IBUFCTRL_HRIO_I_O)
                                            0.043     0.940 r  example_ibuf/IBUFCTRL_INST/O
                     net (fo=1, routed)          1.967     2.907    example_support_wrapper/
clk
   BUFGCE_X1Y24                                                         r  example_support_wrapper/
example_bufg/I
    BUFGCE_X1Y24         BUFGCE (Prop_BUFCE_BUFGCE_I_O)
                                                0.083     2.990 r  example_support_wrapper/
example_bufg/O
    X2Y1 (CLOCK_ROOT)    net (fo=1742, routed)        2.304     5.294
example_support_wrapper/example_support/example_spi/example_spi_byte/clk
    BITSLICE_RX_TX_X0Y172
                       FDRE                                             r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_c_ofd/C
  -------------------------------------------------------------------
-------------------
    BITSLICE_RX_TX_X0Y172
                         FDRE (Prop_OUT_FF_BITSLICE_COMPONENT_RX_TX_C_Q)
                                                0.626     5.920 r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_c_ofd/Q
                         net (fo=1, routed)           0.380     6.300    spi_c_OBUF
    AC23                                                            r    spi_c_OBUF_inst/I
    AC23                 OBUF (Prop_OUTBUF_HPIOB_I_O)
                                                      0.850     7.150 r  spi_c_OBUF_inst/O
                         net (fo=0)                   0.000     7.150    spi_c
    AC23                                                            r    spi_c (OUT)
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)       11.111    11.111 r
                         clock pessimism              0.000    11.111
                         clock uncertainty            -0.035    11.076
                         output delay                 11.111    22.187
  -------------------------------------------------------------------
                         required time                          22.187
                         arrival time                           -7.150
  -------------------------------------------------------------------
                         slack                                  15.037
```

Locate T_{qfpga} by searching the timing report for flip-flop to pad path analysis at Min at Fast Process Corner, where the destination is identified as `spi_c`.

- T_{qfpga} = I/O Datapath Delay (`spi_c`)
- T_{qfpga} = 0.919 ns, minimum

```
Slack (MET) :              3.227ns  (arrival time - required time)
  Source:                  example_support_wrapper/example_support/example_spi/
example_spi_byte/spi_c_ofd/C
                            (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns 
fall@5.556ns period=11.111ns})
  Destination:             spi_c
                             (output port clocked by clk  {rise@0.000ns fall@5.556ns 
period=11.111ns})
  Path Group:              clk
  Path Type:               Min at Fast Process Corner
  Requirement:             0.000ns  (clk rise@0.000ns - clk rise@0.000ns)
  Data Path Delay:         0.919ns  (logic 0.752ns (81.829%)  route 0.167ns (18.171%))
  Logic Levels:            1  (OBUF=1)
  Output Delay:            0.000ns
  Clock Path Skew:         -2.308ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    0.000ns
    Source Clock Delay      (SCD):    2.308ns
    Clock Pessimism Removal (CPR):    -0.000ns
  Clock Net Delay (Source):      1.077ns (routing 0.127ns, distribution 0.950ns)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)         0.000     0.000 r
    K20                                                0.000     0.000 r  clk (IN)
                         net (fo=0)                    0.000     0.000    example_ibuf/I
   K20                                                        r  example_ibuf/INBUF_INST/PAD
    K20                  INBUF (Prop_INBUF_HRIO_PAD_O)
                                                0.418     0.418 r  example_ibuf/INBUF_INST/O
                         net (fo=1, routed)            0.025     0.443    example_ibuf/OUT
   K20                                                       r  example_ibuf/IBUFCTRL_INST/I
    K20                  IBUFCTRL (Prop_IBUFCTRL_HRIO_I_O)
                                             0.015     0.458 r  example_ibuf/IBUFCTRL_INST/O
                     net (fo=1, routed)          0.746     1.204    example_support_wrapper/
clk
   BUFGCE_X1Y24                                                  r  example_support_wrapper/
example_bufg/I
    BUFGCE_X1Y24         BUFGCE (Prop_BUFCE_BUFGCE_I_O)
                                                0.027     1.231 r  example_support_wrapper/
example_bufg/O
    X2Y1 (CLOCK_ROOT)    net (fo=1742, routed)         1.077     2.308
example_support_wrapper/example_support/example_spi/example_spi_byte/clk
```

```
    BITSLICE_RX_TX_X0Y172
                       FDRE                                    r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_c_ofd/C
  -------------------------------------------------------------------
-------------------
    BITSLICE_RX_TX_X0Y172
                         FDRE (Prop_OUT_FF_BITSLICE_COMPONENT_RX_TX_C_Q)
                                                  0.268     2.576 r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_c_ofd/Q
                         net (fo=1, routed)             0.167     2.743    spi_c_OBUF
    AC23                                                              r  spi_c_OBUF_inst/I
    AC23                 OBUF (Prop_OUTBUF_HPIOB_I_O)
                                                        0.484     3.227 r  spi_c_OBUF_inst/O
                         net (fo=0)                     0.000     3.227    spi_c
    AC23                                                              r  spi_c (OUT)
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)          0.000     0.000 r
                         clock pessimism                0.000     0.000
                         output delay                  -0.000     0.000
  -------------------------------------------------------------------
                         required time                           -0.000
                         arrival time                             3.227
  -------------------------------------------------------------------
                         slack                                    3.227
```

Locate T_{sfpga} by searching the timing report for pad to flip-flop path analysis at Setup (Max at Fast Process Corner), where the source pad is identified as `spi_q`.

- Tsfpga = I/O Datapath Delay (`spi_q`)
- Tsfpga = 0.658 ns, maximum

```
Slack (MET):              23.825ns  (required time - arrival time)
  Source:                 spi_q
                            (input port clocked by clk  {rise@0.000ns fall@5.556ns
period=11.111ns})
  Destination:            example_support_wrapper/example_support/example_spi/
example_spi_byte/spi_q_ifd/D
                           (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@5.556ns period=11.111ns})
  Path Group:             clk
  Path Type:              Setup (Max at Fast Process Corner)
  Requirement:            11.111ns  (clk rise@11.111ns - clk rise@0.000ns)
  Data Path Delay:        0.658ns  (logic 0.495ns (75.210%)  route 0.163ns (24.790%))
  Logic Levels:           2  (IBUFCTRL=1 INBUF=1)
  Input Delay:            -11.111ns
  Clock Path Skew:        2.318ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.318ns = ( 13.429 - 11.111 )
    Source Clock Delay      (SCD):    0.000ns
    Clock Pessimism Removal (CPR):    0.000ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
  Clock Net Delay (Destination): 1.087ns (routing 0.127ns, distribution 0.960ns)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
                         input delay                -11.111   -11.111
    U21                                               0.000   -11.111 r  spi_q (IN)
                         net (fo=0)                   0.000   -11.111    spi_q_IBUF_inst/I
   U21                                                    r  spi_q_IBUF_inst/INBUF_INST/PAD
    U21                  INBUF (Prop_INBUF_HPIOB_PAD_O)
                                                   0.495   -10.616 r  spi_q_IBUF_inst/INBUF_INST/O
                       net (fo=1, routed)           0.047   -10.569    spi_q_IBUF_inst/OUT
    U21                                                           r  spi_q_IBUF_inst/
IBUFCTRL_INST/I
    U21                  IBUFCTRL (Prop_IBUFCTRL_HPIOB_I_O)
                                                      0.000   -10.569 r  spi_q_IBUF_inst/
IBUFCTRL_INST/O
                    net (fo=1, routed)          0.116   -10.453    example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q
    BITSLICE_RX_TX_X0Y186
                      FDRE                                        r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q_ifd/D
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)       11.111    11.111 r
    K20                                               0.000    11.111 r  clk (IN)
                         net (fo=0)                   0.000    11.111    example_ibuf/I
   K20                INBUF                                   r  example_ibuf/INBUF_INST/PAD
    K20                  INBUF (Prop_INBUF_HRIO_PAD_O)
                                                0.418    11.529 r  example_ibuf/INBUF_INST/O
                         net (fo=1, routed)           0.025    11.554    example_ibuf/OUT
   K20                IBUFCTRL                               r  example_ibuf/IBUFCTRL_INST/I
    K20                  IBUFCTRL (Prop_IBUFCTRL_HRIO_I_O)
                                                   0.015    11.569 r  example_ibuf/IBUFCTRL_INST/O
                    net (fo=1, routed)          0.746    12.315    example_support_wrapper/
clk
   BUFGCE_X1Y24         BUFGCE                                    r  example_support_wrapper/
example_bufg/I
    BUFGCE_X1Y24         BUFGCE (Prop_BUFCE_BUFGCE_I_O)
                                                 0.027    12.342 r  example_support_wrapper/
example_bufg/O
    X2Y1 (CLOCK_ROOT)    net (fo=1742, routed)        1.087    13.429
example_support_wrapper/example_support/example_spi/example_spi_byte/clk
    BITSLICE_RX_TX_X0Y186
                      FDRE                                        r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q_ifd/C
                         clock pessimism              0.000    13.429
                         clock uncertainty           -0.035    13.394
    BITSLICE_RX_TX_X0Y186
                         FDRE (Setup_IN_FF_BITSLICE_COMPONENT_RX_TX_C_D)
                                                -0.022    13.372    example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q_ifd
  -------------------------------------------------------------------
                         required time                         13.372
                         arrival time                          10.453
  -------------------------------------------------------------------
                         slack                                 23.825
```

Locate T_{hfpga} by searching the timing report for pad to flip-flop path analysis at Hold (Min at Slow Process Corner), where the source pad is identified as `spi_q`.

- T_{hfpga} = I/O Datapath Delay (`spi_q`)
- T_{hfpga} = 0.468 ns, minimum

```
Slack (MET):              17.334ns  (arrival time - required time)
  Source:                 spi_q
                            (input port clocked by clk  {rise@0.000ns fall@5.556ns
period=11.111ns})
  Destination:            example_support_wrapper/example_support/example_spi/
example_spi_byte/spi_q_ifd/D
                           (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@5.556ns period=11.111ns})
  Path Group:             clk
  Path Type:              Hold (Min at Slow Process Corner)
  Requirement:            0.000ns  (clk rise@0.000ns - clk rise@0.000ns)
  Data Path Delay:        0.468ns  (logic 0.242ns (51.750%)  route 0.226ns (48.250%))
  Logic Levels:           2  (IBUFCTRL=1 INBUF=1)
  Input Delay:            22.222ns
  Clock Path Skew:        5.302ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    5.302ns
    Source Clock Delay      (SCD):    0.000ns
    Clock Pessimism Removal (CPR):    -0.000ns
  Clock Net Delay (Destination): 2.312ns (routing 0.335ns, distribution 1.977ns)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
                         input delay                 22.222    22.222
    U21                                               0.000    22.222 r  spi_q (IN)
                         net (fo=0)                   0.000    22.222    spi_q_IBUF_inst/I
   U21                                                       r  spi_q_IBUF_inst/INBUF_INST/PAD
    U21                  INBUF (Prop_INBUF_HPIOB_PAD_O)
                                             0.242    22.464 r  spi_q_IBUF_inst/INBUF_INST/O
                        net (fo=1, routed)          0.050    22.514    spi_q_IBUF_inst/OUT
    U21                                                                r  spi_q_IBUF_inst/
IBUFCTRL_INST/I
    U21                  IBUFCTRL (Prop_IBUFCTRL_HPIOB_I_O)
                                                      0.000    22.514 r  spi_q_IBUF_inst/
IBUFCTRL_INST/O
                     net (fo=1, routed)          0.176    22.690    example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q
    BITSLICE_RX_TX_X0Y186
                      FDRE                                        r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q_ifd/D
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)        0.000     0.000 r
    K20                                               0.000     0.000 r  clk (IN)
                         net (fo=0)                   0.000     0.000    example_ibuf/I
   K20               INBUF                                    r  example_ibuf/INBUF_INST/PAD
    K20                  INBUF (Prop_INBUF_HRIO_PAD_O)
```

```
                                               0.805     0.805 r  example_ibuf/INBUF_INST/O
                         net (fo=1, routed)           0.092     0.897    example_ibuf/OUT
    K20               IBUFCTRL                                      r  example_ibuf/IBUFCTRL_INST/I
    K20                  IBUFCTRL (Prop_IBUFCTRL_HRIO_I_O)
                                             0.043     0.940 r  example_ibuf/IBUFCTRL_INST/O
                      net (fo=1, routed)          1.967     2.907    example_support_wrapper/
clk
    BUFGCE_X1Y24        BUFGCE                                         r  example_support_wrapper/
example_bufg/I
    BUFGCE_X1Y24         BUFGCE (Prop_BUFCE_BUFGCE_I_O)
                                                0.083     2.990 r  example_support_wrapper/
example_bufg/O
    X2Y1 (CLOCK_ROOT)    net (fo=1742, routed)        2.312     5.302
example_support_wrapper/example_support/example_spi/example_spi_byte/clk
    BITSLICE_RX_TX_X0Y186
                       FDRE                                            r  example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q_ifd/C
                         clock pessimism              0.000     5.302
    BITSLICE_RX_TX_X0Y186
                         FDRE (Hold_IN_FF_BITSLICE_COMPONENT_RX_TX_C_D)
                                                0.054     5.356    example_support_wrapper/
example_support/example_spi/example_spi_byte/spi_q_ifd
  -------------------------------------------------------------------
                         required time                         -5.356
                         arrival time                          22.690
  -------------------------------------------------------------------
                         slack                                 17.334
```

Check:

- Is $T_{hfpga} \leq T_{qfpga,min} + T_{clqx}$?
- Is 0.468 ns ≤ 0.919 ns + 1 ns?
- Is 0.468 ns ≤ 1.919 ns? Yes

Calculate:

$T_{clk} \geq 0.5 \times (T_{qfpga,max} + T_{w1} + T_{dly} + T_{w2} + T_{clqv} + T_{w3} + T_{dly} + T_{w4} + T_{sfpga})$

requires

$T_{clk} \geq 0.5 \times (1.856 \text{ ns} + 1 \text{ ns} + 2.8 \text{ ns} + 1 \text{ ns} + 6 \text{ ns} + 1 \text{ ns} + 2.8 \text{ ns} + 1 \text{ ns} + 0.658 \text{ ns})$

or

$T_{clk} \geq 9.057 \text{ ns}$

The hold requirement is satisfied and the requirement on T_{clk} indicates that the SPI Receive Waveform and Timing Budget restrict the system-level design example input clock cycle time to be 9.057 ns or larger.

SPI Bus Timing Budget Conclusions

When the SPI flash master helper block and external memory system are present, the SPI bus timing budget must be analyzed to ensure a robust implementation. The result of the analysis helps to confirm that the external memory system is functional, and reveals any constraints it might pose on the maximum frequency of the system-level design example input clock.

Using the example data from the Vivado® Design Suite timing report for a Kintex UltraScale SEM IP implementation, the memory interface is functional. The most stringent requirement on T_{clk} is that $T_{clk} \geq 9.057$ ns, as the memory interface only works when the input clock frequency is 110.412 MHz or lower. Other input clock frequency limits, such as the ICAP maximum clock frequency and the system-level example maximum clock frequency, must also be considered.

Monitor Interface Signaling and Protocol

This appendix describes the Monitor Interface signaling and protocol. It provides helpful information for you to replace the UART helper block with an alternate function to parse the SEM controller messages for specific information like error location.

Monolithic and UltraScale+ SSI Devices

Figure H-1 illustrates the protocol used on the transmit portion of the Monitor Interface. When the controller wants to transmit a byte of data, it first samples the `monitor_txfull` signal to determine if the transmit buffer is capable of accepting a byte of data. Provided that `monitor_txfull` is Low, the controller transmits the byte of data by applying the data to `monitor_txdata[7:0]` and asserting `monitor_txwrite` for a single clock cycle.

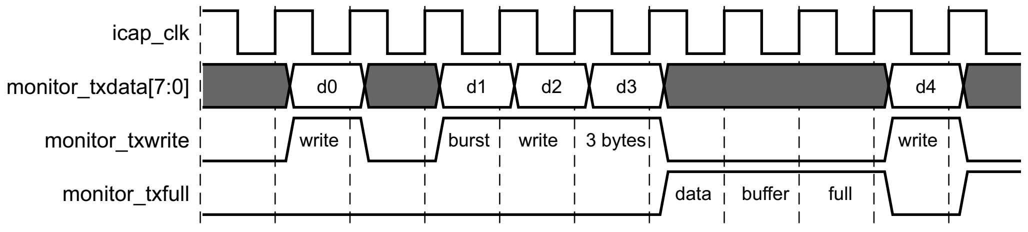


Figure H-1: **Monitor Interface Transmit Protocol**

The controller can perform burst writes by applying a data sequence to `monitor_txdata[7:0]` and asserting `monitor_txwrite` for multiple cycles. However, the controller observes `monitor_txfull` so that it never over-runs the transmit buffer.

The peripheral receiving the data is responsible for tracking its correct buffer availability and reporting it through `monitor_txfull`. In the cycle after the peripheral samples `monitor_txwrite` High, it must assert `monitor_txfull` if the data written completely fills the buffer.

Further, the peripheral must only assert `monitor_txfull` in response to `monitor_txwrite` from the controller. Under no circumstances can the peripheral assert `monitor_txfull` based on events internal to the peripheral. This requirement exists because the controller can sample `monitor_txfull` several cycles in advance of performing a write.

While monitor_txfull is asserted by the peripheral, the controller stalls if it is waiting to transmit additional data. This can have negative side effects on the error mitigation performance of the controller. For example, if a correction takes place, the controller successfully corrects (or handles) the error and then send a status report. If the entire message cannot be accepted by the peripheral, the controller stalls, preventing it from returning to the Observation state. Therefore, a custom peripheral must have an adequate balance of buffer depth and data consumption rate.

Figure H-2 illustrates the protocol used on the receive portion of the Monitor Interface. When the controller wants to receive a byte of data, it first samples the monitor_rxempty signal to determine if the receive buffer is providing a byte of data. Provided that monitor_rxempty is Low, the controller receives the byte of data from monitor_rxdata[7:0] and acknowledges reception by asserting monitor_rxread for a single clock cycle.

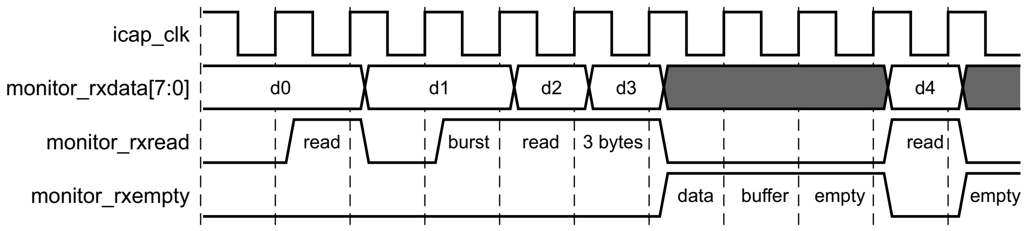


Figure H-2: **Monitor Interface Receive Protocol**

The controller can perform burst reads by obtaining a data sequence from monitor_rxdata[7:0] and asserting monitor_rxread for multiple cycles. However, the controller observes monitor_rxempty so that it never under-runs the receive buffer.

The peripheral providing the data is responsible for tracking its buffer status and reporting it through monitor_rxempty. In the cycle after the peripheral samples monitor_rxread High, it must assert monitor_rxempty if the buffer is empty.

Further, the peripheral must only assert monitor_rxempty in response to monitor_rxread from the controller. Under no circumstances can the peripheral assert monitor_rxempty based on events internal to the peripheral. This requirement exists because the controller can sample monitor_rxempty several cycles in advance of performing a read.

The format of the data sent and received on the Monitor Interface is byte-stream ASCII codes. Each ASCII code is represented by one byte of data. All transactions are one byte at a time.

SEM core reads on the monitor interface when rxempty is high. Two pulses are seen on the monitor_rxread due to two times reset sequence (Two reads are done, to cover corner case of one data showing up between flow control read, test, and issuance of reset read.) Customer can either mask the errors or can just feed the FIFO with two bytes that do not do anything.

For example to send the error injection command on `N C0000007A4`, the peripheral indicates data is ready by setting `monitor_rxempty` to Low and presenting the `N` ASCII code (`x4E`) first, then a " " or space (space = `x20`), and then C (`x43`) and so on, with 4 (`x34`) being the last ASCII code sent on the monitor RX interface. Figure H-3 illustrates this transaction.

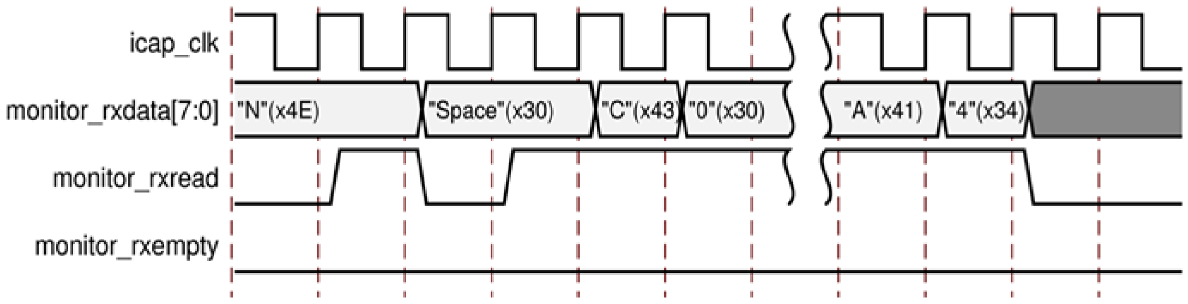


Figure H-3: **Peripheral Sending Error Injection Command Through Monitor RX Interface**

After the controller receives the command, it echoes (sends back) the command to the peripheral through the monitor TX interface. Echoing the command back signals to the peripheral that the controller has accepted the command and is ready to accept the next command. This is also indicated by setting `monitor_txfull` to Low. Figure H-4 illustrates this transaction.

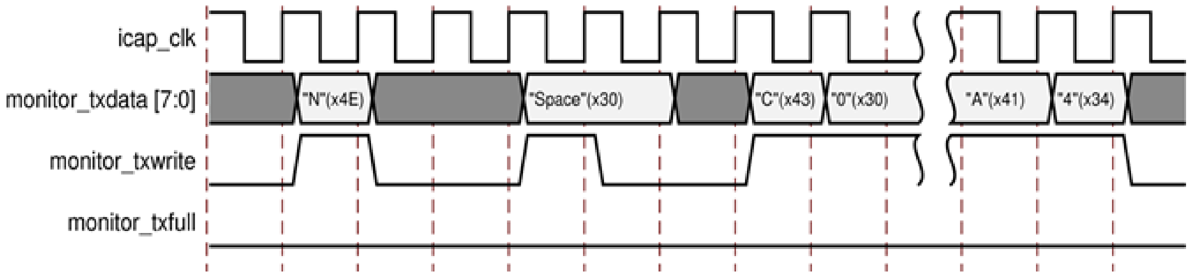


Figure H-4: **SEM Controller Receiving Error Injection Command from Peripheral Successfully**

Note: If the controller receives a data or command from the peripheral that is incorrect or invalid, it does not echo back the command to the peripheral.

UltraScale SSI Devices

For UltraScale™ SSI devices, the example design delivered contains additional logic that combines the individual Monitor Interfaces for each controller in each SLR into a single UART interface.

RECOMMENDED: *If you choose not to use the delivered example design, you should manage the Monitor Interface for each controller independently. In this use case, the behavior of the interface is identical to what is described in the previous section.*

Fetch Interface Signaling and Protocol

This appendix describes the Fetch Interface which is used to provide the SEM controller a frame of essential bits data to use for classifying errors. It is only needed when the classification feature of the SEM controller is enabled. It also provides helpful information for you to replace the SPI flash master helper block with an alternate function like a parallel flash memory controller.

For UltraScale™ SSI devices, the example design SPI flash master demonstrates how to manage a single SPI flash with the Fetch interfaces of multiple SLR. For all others, the example SPI flash master manages a single SPI flash with a single fetch interface.

Figure I-1 illustrates the protocol used on the transmit portion of the Fetch Interface. When the controller wants to transmit a byte of data, it first samples the `fetch_txfull` signal to determine if the transmit buffer is capable of accepting a byte of data. Provided that `fetch_txfull` is Low, the controller transmits the byte of data by applying the data to `fetch_txdata[7:0]` and asserting `fetch_txwrite` for a single clock cycle.

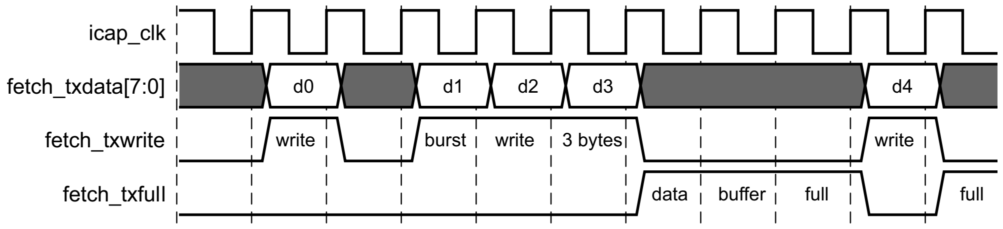


Figure I-1: **Fetch Interface Transmit Protocol**

The controller can perform burst writes by applying a data sequence to `fetch_txdata[7:0]` and asserting `fetch_txwrite` for multiple cycles. However, the controller observes `fetch_txfull` so that it never over-runs the transmit buffer.

The peripheral receiving the data is responsible for tracking its correct buffer availability and reporting it through `fetch_txfull`. In the cycle after the peripheral samples `fetch_txwrite` High, it must assert `fetch_txfull` if the data written completely fills the buffer.

Further, the peripheral must only assert `fetch_txfull` in response to `fetch_txwrite` from the controller. Under no circumstances can the peripheral assert `fetch_txfull` based on events internal to the peripheral. This requirement exists because the controller can sample `fetch_txfull` several cycles in advance of performing a write.

While `fetch_txfull` is asserted by the peripheral, the controller stalls if it is waiting to transmit additional data. This can have negative side effects on the error mitigation performance of the controller.

Figure I-2 illustrates the protocol used on the receive portion of the Fetch Interface. When the controller wants to receive a byte of data, it first samples the `fetch_rxempty` signal to determine if the receive buffer is providing a byte of data. Provided that `fetch_rxempty` is Low, the controller receives the byte of data from `fetch_rxdata[7:0]` and acknowledges reception by asserting `fetch_rxread` for a single clock cycle.

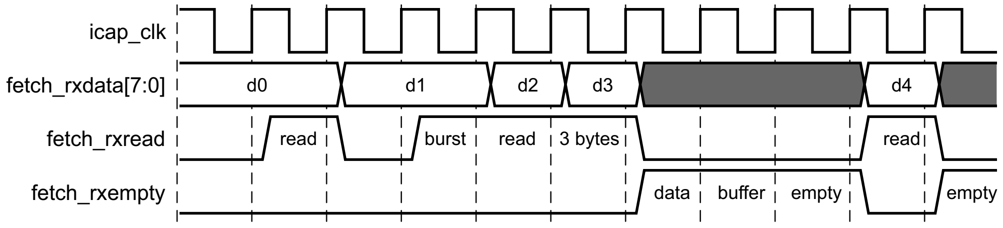


Figure I-2: **Fetch Interface Receive Protocol**

The controller can perform burst reads by obtaining a data sequence from `fetch_rxdata[7:0]` and asserting `fetch_rxread` for multiple cycles. However, the controller observes `fetch_rxempty` so that it never under-runs the receive buffer.

The peripheral providing the data is responsible for tracking its correct buffer status and reporting it through `fetch_rxempty`. In the cycle after the peripheral samples `fetch_rxread` High, it must assert `fetch_rxempty` if the data buffer is empty.

Further, the peripheral must only assert `fetch_rxempty` in response to `fetch_rxread` from the controller. Under no circumstances can the peripheral assert `fetch_rxempty` based on events internal to the peripheral. This requirement exists because the controller can sample `fetch_rxempty` several cycles in advance of performing a read.

The data exchanged is binary and represents a command-response pair. The data format is little endian. For `data[7:0]`, 0 is the LSB and 7 is the MSB. For a 32-bit word Byte 0 is the least significant byte, and Byte 3 is the most significant byte. The controller uses the Fetch Interface to send commands and to read the classification data from external memory during the classification state (`status_classification` = 1). All addresses are byte addresses, and all transactions are one byte at a time.

The controller transmits a 6-byte command sequence to initiate a data fetch using `fetch_tx` interface. The command sequence is:

- Byte 1: ADD[31:24]
- Byte 2: ADD[23:16]
- Byte 3: ADD[15:8]
- Byte 4: ADD[7:0]
- Byte 5: LEN[15:8]
- Byte 6: LEN[7:0]

The ADD field is the address, and LEN field represents the number of bytes of data the controller expects to receive from the peripheral. In general, the command type is "read LEN bytes starting at ADD address." Note that the ADD field value that the controller provides is dependent on the classification base pointer that is defined by the user. See Fetch Interface, page 91.

Figure I-3 illustrates this command sequence.

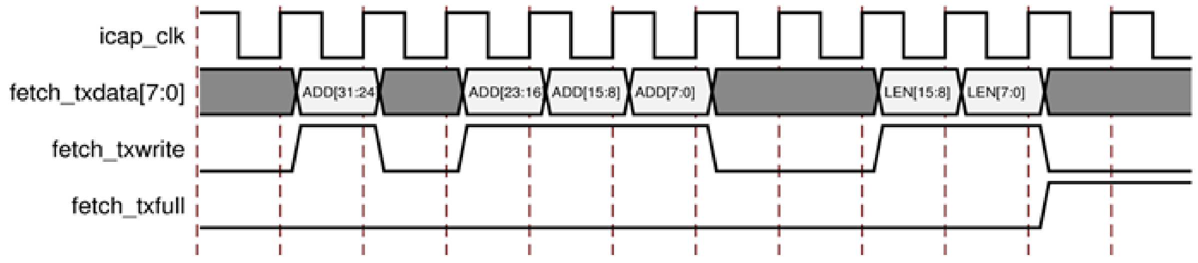


Figure I-3: **SEM Controller Sending Commands to Peripheral**

In response, the peripheral must fetch LEN[15:0] bytes of data starting from address ADD[31:0] and return this data to the controller. After the controller has issued a command, the peripheral must fulfill the command with the exact number of requested bytes on the `fetch_rx` interface. Figure I-4 illustrates transaction with the LEN field equals 4.

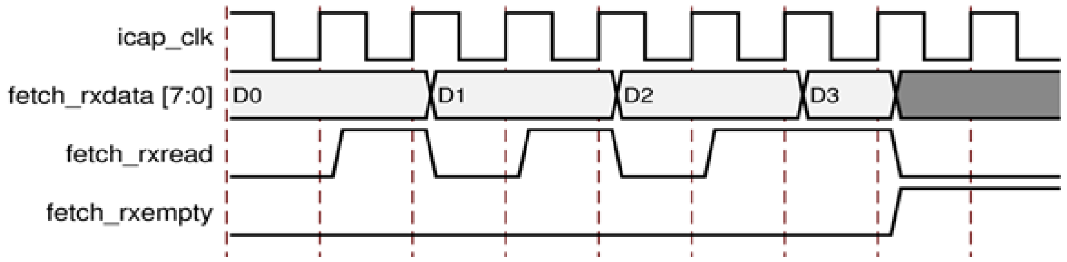


Figure I-4: **Peripheral Returning Data to SEM Controller**

Note: When the SEM controller is in boot or initialization state, it performs two special sequence of reads on the `fetch_rxread` port to reset this interface. The controller does not expect any data from this read sequence and it can be ignored.

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 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 or by using the Xilinx® Documentation Navigator.

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

Answer Records

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

Answer Records for this 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

Technical Support

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

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

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

Debug Tools

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

Vivado Design Suite Debug Feature

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

The Vivado logic analyzer is used with the logic debug 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 12].

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. The Vivado Design Suite debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Design Suite debug feature 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.

RECOMMENDED: *Xilinx recommends integrating the SEM IP core as early as possible, ideally at the start of the project. For more information, see Integration and Validation, page 120.*

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_V3_1
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 Switching Behavior and UART 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, yields 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_V3_1
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 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 3-5 through Figure 3-7.

Additional Error Injection Options

In addition to Linear Frame Address (LFA), error injections can also be performed using Physical Frame Address (PFA). This alternate format can be used for the error injection commands executed from both the Monitor (or UART) and Command Interfaces.

Table J-1: **Additional UART Commands for Error Injection**

Command	UART Command Set
Error Injection Using PFA	"N {10-digit hex value}" UltraScale = Binary value = `0sst trrr rrrc cccc cccc cmmm mmmm wwww wwwb bbbb` UltraScale+ = Binary value = `00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm wwww wwwb bbbb`

Binary Value	Equals to
`ss`	Hardware slr number (2-bit)

`tt`
or
`ttt`
	Block type (2-bit for UltraScale or 3-bit for UltraScale+)

`rrrrrr`	Row address (6-bit)
`cccccccccc`	Column address (10-bit)
`mmmmmmm`	Minor address (7-bit for UltraScale or 8-bit for UltraScale+)
`wwwwwww`	Word address (7-bit)
`bbbbb`	Bit address (5-bit)

Valid in Idle state. Valid for Mitigation and Testing, Detect and Testing, or Emulation modes only.

Table J-2: **Additional Command Format for Error Injection**

Command	Command_code[n – 1:0] Format n = 40 for UltraScale and n = 44 for UltraScale+ Devices
Error Injection Using PFA	UltraScale = Binary value = `0sst trrr rrrc cccc cccc cmmm mmmm wwww wwwb bbbb` UltraScale+ = Binary value = `00ss 0ttt rrrr rrcc cccc cccc mmmm mmmm wwww wwwb bbbb`

Binary Value	Equals to
`ss`	Hardware slr number (2-bit)

`tt`
or
`ttt`
	Block type (2-bit for UltraScale or 3-bit for UltraScale+)

`rrrrrr`	Row address (6-bit)
`cccccccccc`	Column address (10-bit)
`mmmmmmmm`	Minor address (7-bit for UltraScale or 8-bit for UltraScale+)
`wwwwwww`	Word address (7-bit)
`bbbbb`	Bit address (5-bit)

Valid when controller is in Idle state. Valid for Mitigation and Testing, Detect and Testing, or Emulation modes only.

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 mixed-mode clock manager (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

Device Dependency

Each SEM controller generated is specific to the device it is targeted for. If a controller design checkpoint (DCP) is generated for one device and then used in a larger design targeted for a different device and hardware, the IP completes its boot and initialization process but it does not function correctly.

Design Properties and Constraints

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.

Documentation Navigator and Design Hubs

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

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

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

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

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

References

These documents provide supplemental material useful with this user guide:

1. *Xilinx Device Reliability Report (*UG116)
2. *UltraScale Architecture Configuration User Guide* (UG570)
3. *Zynq UltraScale+ MPSoC Technical Reference Manual* (UG1085)
4. *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices* (XAPP1298)
5. *Integrating LogiCORE SEM IP with AXI in Zynq UltraScale+ Devices* (XAPP1303)
6. *Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolithic Devices* (XAPP1261)
7. *Zynq UltraScale+ MPSoC Software Developer Guide* (UG1137)
8. *OS and Libraries Document Collection* (UG643)
9. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
10. *Vivado Design Suite User Guide: Implementation* (UG904)
11. *Vivado Design Suite User Guide: Designing with IP* (UG896)
12. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
13. *Vivado Design Suite User Guide: Getting Started* (UG910)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/05/2022	3.1	Updated Table 2-2.
10/27/2021	3.1	• Added XCZU1, XCAU20P, XCAU15P, and XCAU10P in Table 2-2, Table 2-6, Table 2-8, and Table 3-7. • Added LUTROM functions in Configuration Memory Masking.
08/06/2021	3.1	Added XCUX35 and XCAU25P in Table 2-2, Table 2-6, Table 2-8, and Table 3-7.
02/04/2021	3.1	Updated Table 2-2, Table 2-6, Table 2-8, Table 3-7, and Table 5-1.
08/27/2020	3.1	• Added XCVU45P, XCVU47P, XCZU46DR, XCZU47DR, XCZU48DR, and XCZU49DR in Table 2-2, Table 2-6, Table 2-8, and Table 3-7. • Added VU45P and VU47P in Table 5-1.
05/22/2019	3.1	• Added XCVU27P and XCVU29P in Table 2-2, Table 2-6, and Table 2-8. • Added detection report description in Error Detection Report – Mitigation Modes (Correction Enabled), Error Detection Report – Detect Only, and Error Detection Report – Diagnostic Scan. • Added UART note in UART Interface Commands. • Added XCVU27P and XCVU29P in Table 3-7. • Added VU27P/VU29P in Table 5-1. • Removed headings in Fetch Interface Signaling and Protocol.
11/14/2018	3.1	• Added Partial Reconfiguration Support. • Added space radiation description to Unsupported Features.
04/04/2018	3.1	• Updated Unsupported Features section in Overview chapter. • Updated Solution Reliability section in Product Specification chapter. • Updated System Clock Interface section in Designing with the Core chapter. • Updated first paragraph in Fetch Interface Signaling and Protocol appendix. • Added note for boot state in Fetch Interface Signaling and Protocol appendix.

Date	Version	Revision
12/20/2017	3.1	• Updated Overview description. • Added Key Considerations for SEM IP Adoption in Overview. • Updated Encryption and Authentication Support section.l • Added footnote to XCVU9P in Maximum Start-up Latency at ICAP FMax for UltraScale+ Devices and Maximum IP Error Detection Times at ICAP FMax for UltraScale+ Devices tables. • Added footnote to UltraScale+ Monolithic and UltraScale+ SSI in Error Correction Latency section. • Added footnote to UltraScale+ Monolithic and UltraScale+ SSI in Error Classification Latency section. • Added footnote to UltraScale+ VU3P and UltraScale+ SSI VU9P in Error Injection Latency section. • Updated Status Interface Heartbeat Switching Characteristics for KU040 figure. • Added description to Monitor Interface appendix in Monitor Interface section. • Updated SC code in Initialization Report of UART Interface Messages section. • Changed Fatal Error to 9F in State Change Report Decoding table. • Added Detect only description in Status Report section. • Added description to Fetch Interface appendix in Fetch Interface section. • Added Monitor and Fetch Interface appendices.
10/04/2017	3.1	• Updated IP Facts chapter. • Removed UltraScale+ Device Support sections. • Added XCVU31P to XCVU37P and XCZU21DR to XCZU29DR device information in Product Specification chapter. • Added UltraScale+ SSI description to Error Classification Latency, No Throttling on Monitor Interface table. • Added UltraScale+ SSI VU3P description to Error Injection Latency When Using Linear Frame Addressing, No Throttling on Monitor Interface table. • Added VU9P description to Device Utilization – Multi-SLR UltraScale and UltraScale+ Devices (SSI) table. • Added description to status_essential in Status Interface Signals table. • Added description to the status_essential Tip in Classification (Mitigation Modes Only) section. • Added description in Fetch Interface section. • Updated UltraScale+ description to External Storage Requirements table. • Added description in Required Constraints section in Designing with the Core chapter. • Added SSI device descriptions in External Memory Programming File section. • Added SSI device description in Status Report appendix.

Date	Version	Revision
04/05/2017	3.1	• Added new supported devices in IP Facts note. • Updated UltraScale+ Device Support sections. • Added golden/fallback description in Unsupported Features section. • Added devices in Maximum Number of Configuration Frames to Error Correction Latency, No Throttling on Monitor Interface tables. • Updated description in Resource Utilization section. • Updated description in Port Descriptions section. • Added description to FRAME_ECC and Status Interface sections. • Updated description in General Design Guidelines section. • Updated description in Systems section. • Added note in Structural Options section. • Added description to FRAME_ECC and Status Interface sections in Designing chapter. • Added Zynq UltraScale+ MPSoC Considerations section. • Added description to Command Interface section. • Added note in Configuration Primitive Included in Core section. • Added UltraScale+ in External Storage Requirements table. • Updated description in Customizing and Generating the Core section. • Added description in Placement Constraints and Pin Constraints sections. • Added description in Constraints for UltraScale/UltraScale+ SSI Devices sections. • Added description in Functions section. • Updated description in Port Descriptions section. • Updated code in Creating the External Memory Programming File section.
		• Added Zynq UltraScale+ and Virtex UltraScale+ in Utilizing an Evaluation Board appendix. • Added UltraScale+ description in Error Injection Guidance appendix. • Added description in IP Design Checklist appendix. • Updated code in Creating the External Memory Programming File section. • Added Zynq UltraScale+ and Virtex UltraScale+ in Utilizing an Evaluation Board appendix. • Added Zynq UltraScale+ and UltraScale+ description in Error Injection Guidance appendix. • Added description in IP Design Checklist appendix.

Date	Version	Revision
10/05/2016	3.1	• Updated IP Facts table note. • Added Encryption and Authentication Support in Overview chapter. • Updated description in Unsupported Features section. • Updated Features section and added UltraScale+ devices in Maximum Number of Configuration Frames table. • Updated UltraScale+ in Maximum Estimated FIT Rate table. • Added UltraScale+ devices in Maximum Start-up Latency at ICAP FMax table. • Added UltraScale+ devices in Maximum IP Error Detection Times at ICAP FMax table. • Added UltraScale+ devices in Error Correction Latency, No Throttling on Monitor Interface, Error Classification Latency, No Throttling on Monitor Interface, and Error Injection Latency When Using Linear Frame Addressing, No Throttling on Monitor Interface tables. • Updated SEM Controller Ports figure. • Added description in Initialization section in Designing with the Core chapter. • Added description on UltraScale+ SSI devices in Designing with the Core, Design Flow Steps, and Example Design chapters. • Changed monitor_rx/tx to uart_rx/tx. • Added Example UART Logs appendix.
04/06/2016	3.1	• Added support for UltraScale+ families. • Added Detect and Testing and Detect only modes in document. • Updated Fig. 2-1: SEM Controller Ports. • Updated Table 3-2: Command Format and Usage. • Updated Error Detection Report – Mitigation Modes (Correction Enabled) section. • Updated Error Detection Report – Detect Only section. • Updated Error Detection Report – Diagnostic Scan section. • Updated Table 3-5: UART Commands and Usage. • Updated Design Flow Steps chapter. • Updated Creating the External Memory Programming File section. • Updated Verification section in Verification appendix. • Added UltraScale+ description to Migrating and Upgrading appendix. • Added SPI Bus Timing and Utilizing Evaluation Board appendices. • Added Post-Synthesis DCP in IP Design Checklist appendix.

Date	Version	Revision
09/30/2015	3.0	• Added Detect only and Diagnostic Scan states throughout document. • Updated IP Facts section. • Added description to Features Summary section. • Added detect feature description in Features section. • Added Table 2-2: Maximum Number of Configuration Frames. • Updated Table 2-5: Maximum Start-up Latency at ICAP FMax. • Updated Table 2-6: Maximum IP Error Detection Times at ICAP FMax. • Updated Table 2-9: Error Injection Latency When Using Physical Frame Addressing, No Throttling on Monitor Interface. • Added description in Sources of Additional Latency section. • Updated Resource Utilization section. • Updated Fig. 2-1: SEM Controller Ports. • Updated description in Status Interface section. • Added Integrate and Validate Early description in General Design Guidelines section. • Updated ICAP Arbitration Interface section. • Updated Figs. 3-4 to 3-6. • Added Directed State Change for Detect only and Diagnostic states. • Updated description and figure in Heartbeat section. • Updated UART Interface section. • Updated Systems section. • Updated GUIs in Figs. 4-1 to 4-3. • Updated User Parameter section. • Updated Functions section. • Added descriptions in Controller Constraints, Example Design Constraints, and Constraints for SSI Devices sections. • Added Integration and Validation section in Design Flow Steps chapter. • Updated Table 5-1: Device Number SLR. • Updated Migrating and Upgrading. • Added Error Injection Guidance and IP Design Checklist appendices. • Added Additional Error Injection Options, Device Dependency and Other Incompatibilities to Debugging.
04/01/2015	2.0	Initial Xilinx release.

Please Read: Important Legal Notices