Soft Error Mitigation Controller v4.1

LogiCORE IP Product Guide

Vivado Design Suite

PG036 April 4, 2018

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The LogiCORE™ IP 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 25 ms in many devices.
- Integration of built-in silicon primitives to fully leverage and improve upon the inherent error detection capability of the FPGA.
- Optional error correction, using selectable method: repair, enhanced repair, or replace.
  - Correction by repair method is ECC algorithm based.
  - Correction by enhanced repair method is ECC and CRC algorithm based.
  - Correction by replace method is data re-load based.
- 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 real effect on design operation.
  - Reduces effective failures-in-time (FIT).
- Optional error injection to support evaluation of SEM Controller applications.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq®-7000 All Programmable SoC, 7 Series
Supported User Interfaces	RS-232, SPI
Resources	See Table 2-9 through Table 2-13
Provided with Core	
Design Files	Encrypted RTL
Example Design	VHDL and Verilog
Test Bench	VHDL and Verilog(2)
Constraints File	XDC
Simulation Model	Verilog and/or VHDL Behavioral model or source HDL(2)
Supported S/W Driver	N/A
Tested Design Flows(3)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.(2)
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Functional and timing simulation of designs that include the SEM Controller is supported. However, it is not possible to observe the SEM Controller behaviors in simulation. Hardware-based evaluation is required.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

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

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

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

Memory Types

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

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

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

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

Mitigation Approaches

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

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

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

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

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

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

If the features offered by the SEM Controller are not required, the integrated soft error detection and correction capability in the silicon should be sufficient for SEU mitigation. See the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] for information on how to

use the built-in error detection and correction capability for 7 series FPGAs and Zynq®-7000 SoCs.

Reliability Estimation

As a starting point, your specification for system reliability should highlight critical sections of the system design and provide a value for the required reliability of each sub-section. 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 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 *Device Reliability Report* (UG116) [Ref 2]. The data reveals the overall infrequency of soft errors.

TIP: *The failure rates involved are so small that most designs do not 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 implements five main functions: initialization, error injection, error detection, error correction, and error classification. All functions, except initialization and detection, are optional; desired functions are selected during the IP core configuration and generation process.

The SEM Controller initializes by bringing the integrated soft error detection capability of the FPGA into a known state after the FPGA enters user mode. After this initialization, the SEM Controller observes the integrated soft error detection status. 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 optionally corrects the soft error by repairing it or by replacing the affected bits. The repair methods use active partial reconfiguration to perform a localized correction of Configuration Memory using a read-modify-write scheme. These methods use algorithms to identify the error in need of correction. The replace method also uses active partial reconfiguration with the same goal, but this method uses a write-only scheme to replace Configuration Memory with original data. This data is provided by the implementation tools and stored outside the SEM Controller.

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

When the SEM Controller is idle, it optionally accepts input from the user to inject errors into Configuration Memory. This feature is useful for testing the integration of the SEM Controller into a larger 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.

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's reliability requirements and system-level supervisory functions to make informed decisions.

Is an error mitigation solution even required? Is the solution built into the target device sufficient for the application requirements, or is the SEM Controller 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 example components 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 97*.

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.

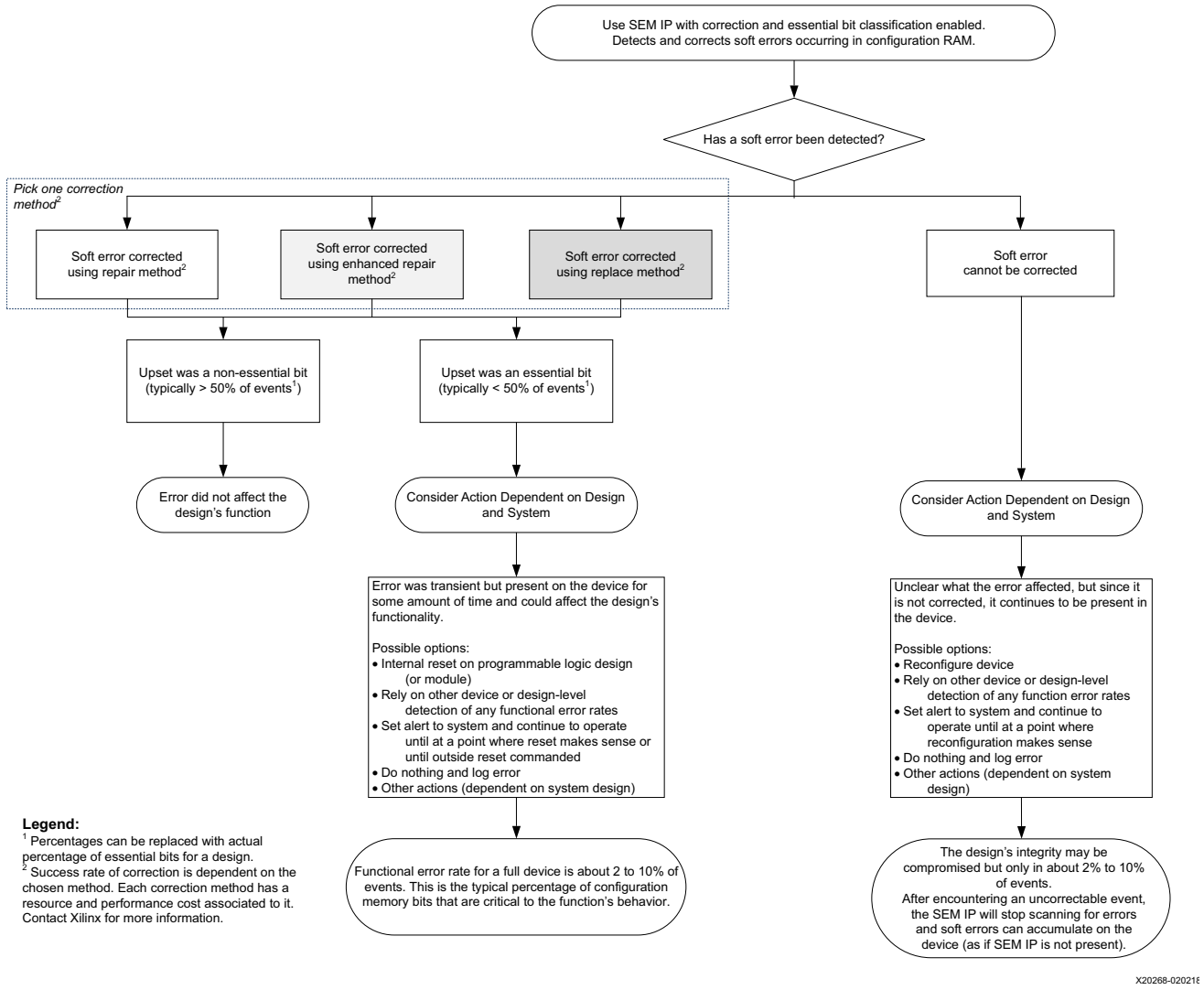

Figure 1-1: **Decision Tree When a Soft Error is Detected**

Encryption and Authentication Support

In 7 series devices, private key encryption and authentication of the bitstream is supported with the SEM IP. Bitstream encryption using AES-256 along with SHA2-256 HMAC for authentication has been verified to be compatible with SEM IP in hardware.

In Zynq-7000 AP SoC devices, the same private key encryption and authentication of the bitstream is also supported with SEM IP.

Public key authentication using RSA-2048 has not been verified, and as a result, is not supported by 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.

The SEM Controller does not operate on soft errors in state elements contained within embedded processor systems, such as those found in Xilinx Zynq-7000 devices. Soft error mitigation in these memory resources must be addressed through software running on the processor system.

Unsupported features and specific limitations include functional, implementation, and use considerations. For more details, see Additional Considerations in Chapter 3.

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.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

This chapter contains the specification of the LogiCORE IP Soft Error Mitigation (SEM) Controller. This configurable controller for mitigation of soft errors in configuration memory also comes with a system-level example design showing use of the controller in a system.

Features

The SEM controller includes:

- Integration of silicon features to leverage built-in error detection capability.
- Implementation of error correction capability to support correction of soft errors. The error correction method can be defined as:
  - **Repair**: ECC algorithm-based correction. This method supports correction of configuration memory frames with single-bit errors. This covers correction of all single-bit upset events. It also covers correction of multi-bit upset events when errors are distributed one per frame as a result of configuration memory interleaving.
  - **Enhanced Repair**: ECC and CRC algorithm-based correction. This method supports correction of configuration memory frames with single-bit errors or double-bit adjacent errors. This covers correction of all single-bit upset events and all double-bit adjacent upset events. This also covers correction of multi-bit upset events when errors are distributed one or two adjacent per frame as a result of configuration memory interleaving.
  - **Replace**: Data reload based correction. This method supports correction of configuration memory frames with arbitrary errors. This covers correction of any upset event that can be resolved to specific configuration memory frames, even if the exact bit locations in the frames cannot be determined.
- 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.

The example design includes:

- Instantiation of the user-configured controller.
- An interface between the controller and external storage. This is required when the controller is configured to perform error classification or error correction by replace.
- An interface between the controller and an external processor for ease of use when the controller is configured to perform error injection.

Standards Compliance

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.

Solution Reliability

Table 2-1 captures the estimated FIT for the SEM controller. This estimation includes the contribution of configuration RAM and block RAM used by the SEM controller and its system-level example design. Because this estimation is for a design that has all features enabled and all signals brought to I/O pins, it presents an upper bound value.

Table 2-1: **Estimated FIT Rate**

Device	FIT
Monolithic solution	~7
SSI solution(1)	~26

Notes:
1. Estimated for XC7VX1140T with four SLRs and four SEM controllers.

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 internal configuration access port (ICAP) F_{Max} specified in the relevant FPGA data sheet as configuration interface AC timing parameter Frbcck. Table 2-2 provides a summary of ICAP F_{Max} values.

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

Device	ICAP F_{Max}
Zynq-7000	100 MHz
Zynq-7000A	100 MHz
Zynq-7000Q	100 MHz
Kintex-7	100 MHz
Kintex-7 Low Voltage	70 MHz
Kintex-7Q	100 MHz
Kintex-7Q Low Voltage	70 MHz
Virtex-7 (Non-SSI)	100 MHz
Virtex-7Q (Non-SSI)	100 MHz
Virtex-7 (SSI)	70 MHz
Artix-7	100 MHz
Artix-7 Low Voltage	70 MHz
Artix-7Q	100 MHz
Artix-7A	100 MHz
Spartan-7	100 MHz

Other maximum frequency limitations might apply. For more details on determining the maximum frequency of operation for the SEM Controller, see Interfaces in Chapter 3.

Solution Latency

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

Estimation Data

The solution behaviors are based on processing of FPGA configuration memory frames. Single-bit errors always reside in a single frame. Generally, an *N*-bit error can present in several ways, ranging from one frame containing all bit errors, to *N* frames each containing one 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. It is also a function of the selected correction mode.

The start-up latency is incurred only once. It is not part of the mitigation process. Table 2-3 illustrates start-up latency, decomposed into sub-steps of boot and initialization. The boot time is independent of the selected correction mode, while the initialization time is dependent on the selected correction mode.

Table 2-3: **Maximum Start-Up Latency at ICAP F_{Max}**

Device	Boot Time at ICAP F_{Max}	Initialization Time at ICAP F_{Max} (Repair/Replace)	Initialization Time at ICAP F_{Max} (Enhanced Repair)
XC7A12T	110 ms	8.5 ms	1.0 s
XC7A15T	110 ms	13.8 ms	1.7 s
XC7A25T	110 ms	8.5 ms	1.0 s
XC7A35T	110 ms	13.8 ms	1.7 s
XC7A50T	110 ms	13.8 ms	1.7 s
XC7A75T	110 ms	24.1 ms	2.9 s
XC7A100T	110 ms	24.1 ms	2.9 s
XC7A200T	110 ms	55.4 ms	6.6 s
XC7K70T	110 ms	17.8 ms	2.1 s
XC7K160T	110 ms	38.8 ms	4.6 s
XC7K325T	110 ms	71.0 ms	9.3 s
XC7K355T	110 ms	79.9 ms	11.9 s
XC7K410T	110 ms	91.5 ms	15.4 s
XC7K420T	110 ms	106.6 ms	21.1 s
XC7K480T	110 ms	106.6 ms	21.1 s
XC7VX330T	110 ms	77.3 ms	11.1 s
XC7VX415T	110 ms	98.1 ms	17.7 s
XC7VX485T	110 ms	115.7 ms	24.5 s
XC7VX550T	110 ms	163.5 ms	48.7 s
XC7VH580T (SSI)	110 ms	74.3 ms	10.2 s
XC7V585T	110 ms	124.4 ms	28.3 s
XC7VX690T	110 ms	163.5 ms	48.7 s
XC7VH870T (SSI)	110 ms	74.3 ms	10.2 s
XC7VX980T	110 ms	213.3 ms	82.6 s
XC7VX1140T (SSI)	110 ms	74.3 ms	10.2 s

Table 2-3: **Maximum Start-Up Latency at ICAP F_{Max} *(Cont'd)***

Device	Boot Time at ICAP F_{Max}	Initialization Time at ICAP F_{Max} (Repair/Replace)	Initialization Time at ICAP F_{Max} (Enhanced Repair)
XC7V2000T (SSI)	110 ms	95.4 ms	16.7 s
XC7Z007S	110 ms	12.2 ms	1.5 s
XC7Z012S	110 ms	21.7 ms	2.6 s
XC7Z014S	110 ms	24.2 ms	2.9 s
XC7Z010	110 ms	12.2 ms	1.5 s
XC7Z015	110 ms	21.7 ms	2.6 s
XC7Z020	110 ms	24.2 ms	2.9 s
XC7Z030	110 ms	34.1 ms	4.1 s
XC7Z035	110 ms	82.0 ms	11.8 s
XC7Z045	110 ms	82.0 ms	11.8 s
XC7Z100	110 ms	103.4 ms	19.6 s
XC7S6	110 ms	3.8 ms	1.0 s
XC7S15	110 ms	3.8 ms	1.0 s
XC7S25	110 ms	8.5 ms	1.0 s
XC7S50	110 ms	13.8 ms	1.7 s
XC7S75	110 ms	23.8 ms	2.9 s
XC7S100	110 ms	23.8 ms	2.9 s

The start-up latency is the sum of the boot and initialization latency, using the correct column of initialization latency data for the selected correction mode. The start-up latency at the actual frequency of operation can be estimated using data from Table 2-3 and Equation 2-1.

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

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 2-4 illustrates full device scan times.

Table 2-4: **Maximum Device Scan Times at ICAP F_{Max}**

Device	Scan Time at ICAP F_{Max}
XC7A12T	2.6 ms
XC7A15T	4.6 ms
XC7A25T	2.6 ms

Table 2-4: **Maximum Device Scan Times at ICAP F_{Max} *(Cont'd)***

Device	Scan Time at ICAP F_{Max}
XC7A35T	4.6 ms
XC7A50T	4.6 ms
XC7A75T	8.0 ms
XC7A100T	8.0 ms
XC7A200T	18.3 ms
XC7K70T	5.9 ms
XC7K160T	12.9 ms
XC7K325T	23.5 ms
XC7K355T	26.5 ms
XC7K410T	30.3 ms
XC7K420T	35.3 ms
XC7K480T	35.3 ms
XC7VX330T	25.6 ms
XC7VX415T	32.5 ms
XC7VX485T	38.3 ms
XC7VX550T	54.1 ms
XC7VH580T (SSI)	24.6 ms
XC7V585T	41.2 ms
XC7VX690T	54.1 ms
XC7VH870T (SSI)	24.6 ms
XC7VX980T	70.7 ms
XC7VX1140T (SSI)	24.6 ms
XC7V2000T (SSI)	31.6 ms
XC7Z007S	4.0 ms
XC7Z012S	7.2 ms
XC7Z014S	8.0 ms
XC7Z010	4.0 ms
XC7Z015	7.2 ms
XC7Z020	8.0 ms
XC7Z030	11.3 ms
XC7Z035	27.2 ms
XC7Z045	27.2 ms
XC7Z100	34.3 ms
XC7S6	1.3 ms
XC7S15	1.3 ms

Table 2-4: **Maximum Device Scan Times at ICAP F_{Max} *(Cont'd)***

Device	Scan Time at ICAP F_{Max}
XC7S25	2.6 ms
XC7S50	4.6 ms
XC7S75	8.0 ms
XC7S100	8.0 ms

The device scan time for the target device, at the actual frequency of operation, can be estimated using data from Table 2-4 and Equation 2-2.

$$ScanTime_{ACTUAL} = ScanTime_{ICAP\_F_{Max}} \cdot \left[\frac{ICAP\_F_{Max}}{Frequency_{ACTUAL}}\right] \quad \text{Equation 2-2}$$

The error detection latency can be bounded as follows:

- Absolute minimum error detection latency is effectively zero.
- Average error detection latency for detection by ECC is 0.5 × Scan $Time_{ACTUAL}$
- Maximum error detection latency for detection by ECC is Scan $Time_{ACTUAL}$
- Absolute maximum error detection latency for detection by CRC alone is 2.0 × Scan $Time_{ACTUAL}$

The frame-based ECC method used always detects single, double, triple, and all odd-count bit errors in a frame. The remaining error types are usually detected by the frame-based ECC method as well. It is rare to encounter an error that defeats the ECC and is detected by CRC alone.

Error Correction Latency

After detecting an error, the solution attempts correction. Errors are correctable depending on the selected correction mode and error type. Table 2-5 and Table 2-6 provide error correction latency for a configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-5: **Non-SSI: Max Error Correction Latency (100 MHz) No Throttling on Monitor Interface**

Correction Mode	Errors in Frame (Correctability)	Error Correction State at ICAP_F_{Max}
Repair	1-bit (Correctable)	610 µs
	2-bit (Uncorrectable)	25 µs
Enhanced Repair	1-bit (Correctable)	610 µs
	2-bit (Correctable)	18,790 µs
	2-bit (Uncorrectable)	9,110 µs
	BFR(1)-only (Uncorrectable)	10 µs
Replace	Any (Correctable)	830 µs
Any	CRC-only (Uncorrectable)	10 µs

Notes:
1. BFR is an error condition due to a multi-bit upset in an enhanced repair checksum stored in block RAM.

Table 2-6: **SSI: Max Error Correction Latency (70 MHz) No Throttling on Monitor Interface**

Correction Mode	Errors in Frame (Correctability)	Error Correction State at ICAP_F_{Max}
Repair	1-bit (Correctable)	915 µs
	2-bit (Uncorrectable)	70 µs
Enhanced Repair	1-bit (Correctable)	910 µs
	2-bit (Correctable)	26,900 µs
	2-bit (Uncorrectable)	13,010 µs
	BFR(1)-only (Uncorrectable)	10 µs
Replace	Any (Correctable)	1,220 µs
Any	CRC-only (Uncorrectable)	10 µs

Notes:
1. BFR is an error condition due to a multi-bit upset in an enhanced repair checksum stored in block RAM.

The error correction latency at the actual frequency of operation can be estimated using data from Table 2-5 and Equation 2-3.

$$CorrectionLatency_{ACTUAL} = CorrectionLatency_{ICAP\_F_{Max}} \cdot \left[\frac{ICAP\_F_{Max}}{Frequency_{ACTUAL}}\right] \quad \text{Equation 2-3}$$

Error Classification Latency

After attempting correction of an error, the solution classifies the error. The classification result depends on the correction mode, error type, error location, and selected classification mode. Table 2-7 and Table 2-8 provide error classification latency for a configuration frame upset, assuming no throttling on the Monitor Interface.

Table 2-7: **Non-SSI: Max Error Classification Latency (100 MHz) No Throttling on Monitor Interface**

Correction Mode	Errors in Frame (Correctability)	Classification Mode	Error Classification State at ICAP_F_{Max}
Any	Correctable	Enabled	750 µs
Any	Uncorrectable	Disabled	10 µs
Any	Uncorrectable	Any	10 µs

Table 2-8: **SSI: Max Error Classification Latency (70 MHz) No Throttling on Monitor Interface**

Correction Mode	Errors in Frame (Correctability)	Classification Mode	Error Classification State at ICAP_F_{Max}
Any	Correctable	Enabled	1,090 µs
Any	Uncorrectable	Disabled	10 µs
Any	Uncorrectable	Any	10 µs

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

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

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 correctable status flag is updated), the controller issues a detection and correction report through the Monitor Interface. If the MON Shim transmit FIFO becomes full during this report generation, the controller dwells in this state until it has written the entire report into the MON Shim 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 essential status flag is updated), the controller issues a classification report through the Monitor Interface. If the MON Shim transmit FIFO becomes full during this report generation, the controller dwells in this state until it has written the entire report into the MON Shim transmit FIFO. When this happens, the error classification latency increases.

The approaches to completely eliminate the potential bottleneck are to remove the MON Shim and leave the Monitor Interface unused, or use the Monitor Interface with a peripheral that never signals a buffer full condition. In the event the Monitor Interface is unused, the Status Interface remains available for monitoring activity.

For 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.

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

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

Sample Latency Estimation

The first sample estimation illustrates the calculation of error mitigation latency for a single-bit error by the solution implemented in an XC7K325T device with a 66 MHz clock. The solution is configured for error correction by repair, with error classification disabled. The initial assumption is that no throttling occurs on the Monitor Interface.

$$DetectionLatency = 0.5 \cdot ScanTime_{ACTUAL} = 0.5 \cdot 23.5ms \cdot \left[\frac{100MHz}{66MHz}\right] = 17.803ms \quad \text{Equation 2-6}$$

$$CorrectionLatency = 610\mu s \cdot \left[\frac{100MHz}{66MHz}\right] = 0.924ms \quad \text{Equation 2-7}$$

$$ClassificationLatency = 10\mu s \cdot \left[\frac{100MHz}{66MHz}\right] = 0.015ms \quad \text{Equation 2-8}$$

$$MitigationLatency = 17.803ms + 0.924ms + 0.015ms = 18.742ms \quad \text{Equation 2-9}$$

The second sample estimation illustrates the calculation of error mitigation latency for a two-bit error by the solution implemented in an XC7K325T device with a 66 MHz clock. The solution is configured for error correction by replace, with error classification enabled. Again, it is assumed that no throttling occurs on the Monitor Interface.

$$DetectionLatency = 0.5 \cdot ScanTime_{ACTUAL} = 0.5 \cdot 23.5ms \cdot \left[\frac{100MHz}{66MHz}\right] = 17.803ms \quad \text{Equation 2-10}$$

$$CorrectionLatency = 830\mu s \cdot \left[\frac{100MHz}{66MHz}\right] = 1.258ms \quad \text{Equation 2-11}$$

$$ClassificationLatency = 750\mu s \cdot \left[\frac{100MHz}{66MHz}\right] = 1.136ms \quad \text{Equation 2-12}$$

$$MitigationLatency = 17.803ms + 1.258ms + 1.136ms = 20.197ms \quad \textit{Equation 2-13}$$

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 MON Shim is 32 bytes. Further, the MON Shim has been modified to raise the bit rate from 9600 baud to 460800 baud. The standard 8-N-1 protocol used requires 10 bit times on the serial link to transmit a 1-byte payload:

$$AdditionalLatency = \frac{80bytes - 32bytes}{\left[\frac{460800bittimes}{s} \cdot \frac{byte}{10bittimes} \cdot \frac{s}{1000ms}\right]} = 1.042ms \quad \textit{Equation 2-14}$$

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

Throughput

The throughput metrics of the SEM Controller are not specified.

Power

The power metrics of the SEM Controller are not specified.

Resource Utilization

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

Table 2-9: **Resource Utilization for Zynq-7000 Devices**[(1)(2)]

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Zynq-7000 All Devices	Complete solution with no optional features	509	361	11	3 RAMB18
Zynq-7000 XC7Z007S	Complete solution with all optional features	877	673	56	3 RAMB18, 2 RAMB36
Zynq-7000 XC7Z012S	Complete solution with all optional features	912	674	56	3 RAMB18, 3 RAMB36
Zynq-7000 XC7Z014S	Complete solution with all optional features	912	674	56	3 RAMB18, 3 RAMB36
Zynq-7000 XC7Z010	Complete solution with all optional features	863	681	56	3 RAMB18, 2 RAMB36

Table 2-9: **Resource Utilization for Zynq-7000 Devices(1)(2) *(Cont'd)***

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Zynq-7000 XC7Z015	Complete solution with all optional features	834	682	58	3 RAMB18, 3 RAMB36
Zynq-7000 XC7Z020	Complete solution with all optional features	838	682	56	3 RAMB18, 3 RAMB36
Zynq-7000 XC7Z030	Complete solution with all optional features	887	683	56	3 RAMB18, 5 RAMB36
Zynq-7000 XC7Z035	Complete solution with all optional features	994	696	56	3 RAMB18, 10 RAMB36
Zynq-7000 XC7Z045	Complete solution with all optional features	994	696	56	3 RAMB18, 10 RAMB36
Zynq-7000 XC7Z100	Complete solution with all optional features	1,065	696	56	3 RAMB18, 13 RAMB36

Notes:

1. The complete solution is the SEM Controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of logic debug IP (VIO) increases LUTs/FFs but decreases I/Os.

Table 2-10: **Resource Utilization for Kintex-7 Devices(1)(2)**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Kintex-7 All Devices	Complete solution with no optional features	511	361	11	3 RAMB18
Kintex-7 XC7K70T	Complete solution with all optional features	849	682	56	3 RAMB18, 3 RAMB36
Kintex-7 XC7K160T	Complete solution with all optional features	887	683	56	3 RAMB18, 5 RAMB36
Kintex-7 XC7K325T	Complete solution with all optional features	991	686	56	3 RAMB18, 9 RAMB36
Kintex-7 XC7K355T	Complete solution with all optional features	994	696	56	3 RAMB18, 10 RAMB36
Kintex-7 XC7K410T	Complete solution with all optional features	1,028	696	56	3 RAMB18, 11 RAMB36
Kintex-7 XC7K420T	Complete solution with all optional features	1,065	696	56	3 RAMB18, 13 RAMB36
Kintex-7 XC7K480T	Complete solution with all optional features	1,065	696	56	3 RAMB18, 13 RAMB36

Table 2-10: **Resource Utilization for Kintex-7 Devices(1)(2) *(Cont'd)***

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Kintex-7 Low Voltage, Kintex-7Q, Kintex-7Q Low Voltage, All Devices	Same as Kintex-7				

Notes:

1. The complete solution is the SEM Controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of logic debug IP (VIO) increases LUTs/FFs but decreases I/Os.

Table 2-11: **Resource Utilization for Virtex-7 Devices (Non-SSI)(1)(2)**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Virtex-7 All Devices	Complete solution with no optional features	510	361	11	3 RAMB18
Virtex-7 XC7VX330T	Complete solution with all optional features	996	696	56	3 RAMB18, 10 RAMB36
Virtex-7 XC7VX415T	Complete solution with all optional features	1,065	696	56	3 RAMB18, 12 RAMB36
Virtex-7 XC7VX485T	Complete solution with all optional features	1,101	697	56	3 RAMB18, 14 RAMB36
Virtex-7 XC7VX550T	Complete solution with all optional features	1,218	691	56	3 RAMB18, 20 RAMB36
Virtex-7 XC7V585T	Complete solution with all optional features	1,073	698	56	3 RAMB18, 15 RAMB36
Virtex-7 XC7VX690T	Complete solution with all optional features	1,218	691	56	3 RAMB18, 20 RAMB36
Virtex-7 XC7VX980T	Complete solution with all optional features	1,366	692	56	3 RAMB18, 26 RAMB36
Virtex-7Q All Devices	Same as Virtex-7				

Notes:

1. The complete solution is the SEM Controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of logic debug IP (VIO) increases LUTs/FFs but decreases I/Os.

Table 2-12: **Resource Utilization for Virtex-7 Devices (SSI)(1)(2)**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Virtex-7 XC7VH580T	Complete solution with no optional features	1,394	980	19	7 RAMB18
Virtex-7 XC7VH580T	Complete solution with all optional features	2,410	1,692	64	7 RAMB18, 18 RAMB36

Table 2-12: **Resource Utilization for Virtex-7 Devices (SSI)[(1)(2)] *(Cont'd)***

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Virtex-7 XC7VH870T	Complete solution with no optional features	1,853	1,289	27	10 RAMB18
Virtex-7 XC7VH870T	Complete solution with all optional features	3,245	2,230	72	10 RAMB18, 27 RAMB36
Virtex-7 XC7VX1140T	Complete solution with no optional features	2,245	1,598	35	13 RAMB18
Virtex-7 XC7VX1140T	Complete solution with all optional features	4,032	2,768	80	13 RAMB18, 36 RAMB36
Virtex-7 XC7V2000T	Complete solution with no optional features	2,256	1,598	35	13 RAMB18
Virtex-7 XC7V2000T	Complete solution with all optional features	4,342	2,808	80	13 RAMB18, 48 RAMB36
Virtex-7Q, All Devices	Same as Virtex-7				

Notes:

1. The complete solution is the SEM Controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of logic debug IP (VIO) increases LUTs/FFs but decreases I/Os.

Table 2-13: **Resource Utilization for Artix-7 Devices[(1)(2)]**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Artix-7 All Devices	Complete solution with no optional features	512	361	11	3 RAMB18
Artix-7 XC7A12T	Complete solution with all optional features	877	673	56	3 RAMB18, 2 RAMB36
Artix-7 XC7A15T	Complete solution with all optional features	800	681	56	3 RAMB18, 2 RAMB36
Artix-7 XC7A25T	Complete solution with all optional features	877	673	56	3 RAMB18, 2 RAMB36
Artix-7 XC7A35T	Complete solution with all optional features	802	681	56	3 RAMB18, 2 RAMB36
Artix-7 XC7A50T	Complete solution with all optional features	800	681	56	3 RAMB18, 2 RAMB36
Artix-7 XC7A75T	Complete solution with all optional features	837	682	56	3 RAMB18, 3 RAMB36
Artix-7 XC7A100T	Complete solution with all optional features	837	682	56	3 RAMB18, 3 RAMB36
Artix-7 XC7A200T	Complete solution with all optional features	919	681	56	3 RAMB18, 7 RAMB36

Table 2-13: **Resource Utilization for Artix-7 Devices(1)(2) *(Cont'd)***

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Artix-7 Low Voltage, Artix-7A, Artix-7Q, All Devices	Same as Artix-7				

Notes:

1. The complete solution is the SEM Controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of logic debug IP (VIO) increases LUTs/FFs but decreases I/Os.

Table 2-14: **Resource Utilization for Spartan-7 Devices(1)(2)**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Spartan-7 All Devices	Complete solution with no optional features	498	406	11	3 RAMB18
Spartan-7 XC7S6	Complete solution with all optional features	842	673	56	3 RAMB18, 1 RAMB36
Spartan-7 XC7S15	Complete solution with all optional features	842	673	56	3 RAMB18, 1 RAMB36
Spartan-7 XC725	Complete solution with no optional features	877	673	56	3 RAMB18, 2 RAMB36
Spartan-7 XC7S50	Complete solution with all optional features	877	673	56	3 RAMB18, 2 RAMB36
Spartan-7 XC7S75	Complete solution with all optional features	912	674	56	3 RAMB18, 3 RAMB36
Spartan-7 XC7S100	Complete solution with all optional features	912	674	56	3 RAMB18, 3 RAMB36

Notes:

1. The complete solution is the SEM Controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of logic debug IP (VIO) increases LUTs/FFs but decreases I/Os.

Port Descriptions

The SEM Controller is the kernel of the soft error mitigation solution. Figure 2-1 shows the SEM Controller ports. The ports are clustered into six groups. Shading indicates port groups that only exist in certain configurations.

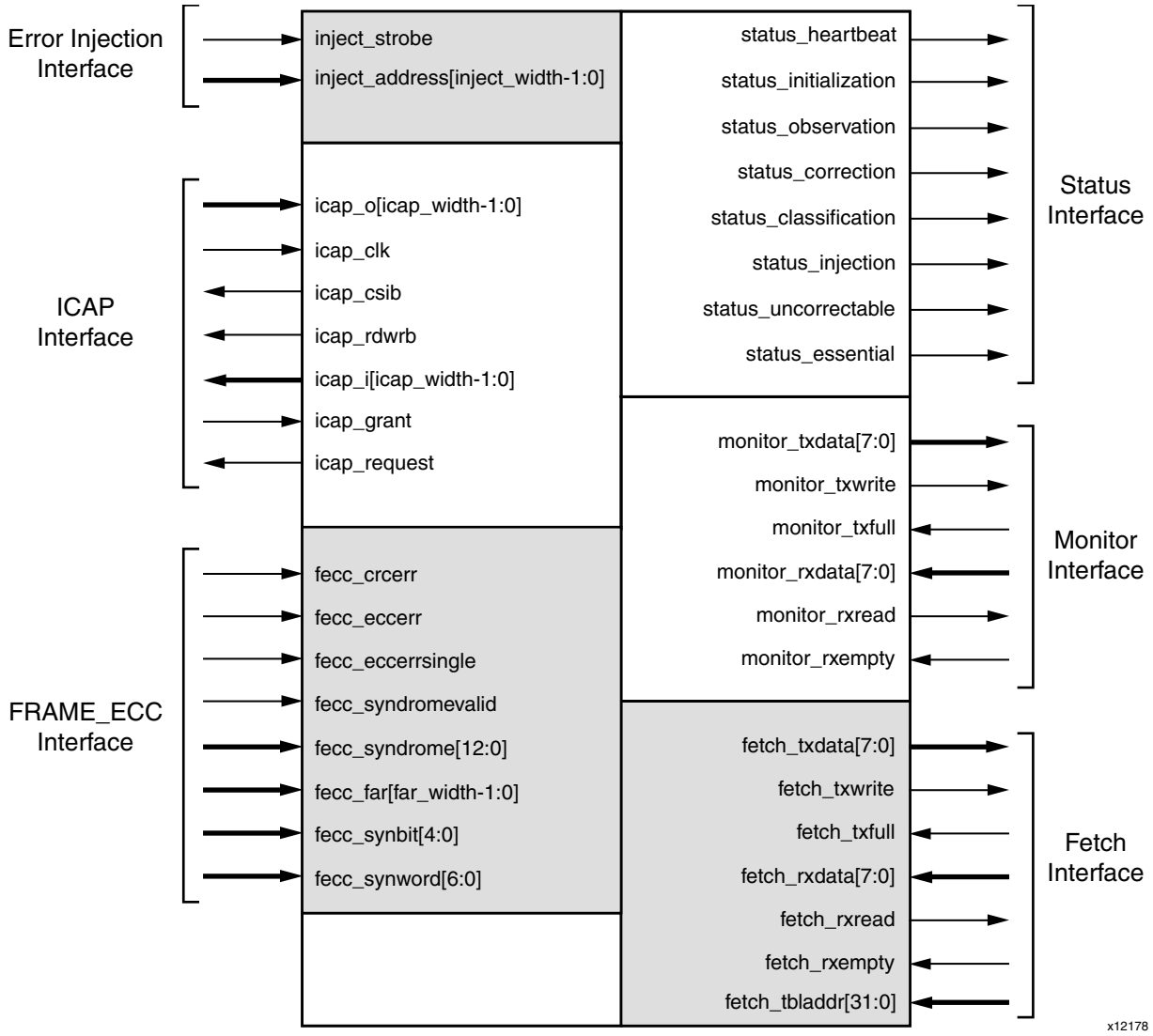

Figure 2-1: **SEM Controller Ports**

TIP: *The system-level example design encapsulates the SEM Controller and various shims that serve to interface the Controller to other devices, as shown in Figure 5-1. The example design is not a reference design, but is an integral part of the total soft error mitigation solution. The SEM Controller ports shown in Figure 2-1 connect to the shims delivered in the example design. See Chapter 5, Example Design for more information on the delivered example design. See Port Descriptions in Chapter 5 for the system-level ports for the solution.*

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

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

ICAP Interface

The ICAP Interface is a point-to-point connection between the SEM Controller and the ICAP primitive. The ICAP primitive enables read and write access to the registers inside the FPGA configuration system. The ICAP primitive and the behavior of the signals on this interface are described in the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1].

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

Name	Sense	I/O	Description
icap_o[icap_width-1:0]	HIGH	I	Receives O output of ICAP. The variable icap_width is equal to 32.
icap_csib	LOW	O	Drives CSIB input of ICAP.
icap_rdwrb	LOW	O	Drives RDWRB input of ICAP.
icap_i[icap_width-1:0]	HIGH	O	Drives I input of ICAP. The variable icap_width is equal to 32.
icap_clk	EDGE	I	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_request	HIGH	O	This signal is reserved for future use. Leave this port OPEN.
icap_grant	HIGH	I	Tie this port to VCC. Receives an ICAP initialization grant signal from the user. icap_grant can be used to hold off the controller initialization state.

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 is an output-only primitive that provides a window into the soft error detection function in the FPGA configuration system. The FRAME_ECC primitive and the behavior of the signals on this interface are described in Answer Record 54350.

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

Name	Sense	I/O	Description
fecc_crcerr	HIGH	I	Receives CRCERROR output of FRAME_ECC.
fecc_eccerr	HIGH	I	Receives ECCERROR output of FRAME_ECC.
fecc_eccerrsingle	HIGH	I	Receives ECCERRORSINGLE output of FRAME_ECC.
fecc_syndromevalid	HIGH	I	Receives SYNDROMEVALID output of FRAME_ECC.
fecc_syndrome[12:0]	HIGH	I	Receives SYNDROME output of FRAME_ECC.
fecc_far[25:0]	HIGH	I	Receives FAR output of FRAME_ECC.

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

Name	Sense	I/O	Description
fecc_synbit[4:0]	HIGH	I	Receives SYNBIT output of FRAME_ECC.
fecc_synword[6:0]	HIGH	I	Receives SYNWORD output of FRAME_ECC.

Status Interface

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

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

Name	Sense	I/O	Description
status_heartbeat	HIGH	O	The heartbeat signal is active while status_observation is asserted. This output issues a single-cycle high pulse at least once every 150 clock cycles. This signal can be used to implement an external watchdog timer to detect "controller stop" scenarios that can occur if the controller or clock distribution is disabled by soft errors. When status_observation is deasserted, the behavior of the heartbeat signal is unspecified.
status_initialization	HIGH	O	The initialization signal is active during controller initialization, which occurs one time after the design begins operation.
status_observation	HIGH	O	The observation signal is active during controller observation of error detection signals. This signal remains active after an error detection while the controller queries the hardware for information.
status_correction	HIGH	O	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	O	The classification signal is active during controller classification of an error or during transition through this controller state if classification is disabled.
status_injection	HIGH	O	The injection signal is active during controller injection of an error. When an error injection is complete, and the controller is ready to inject another error or return to observation, this signal returns inactive.
status_essential	HIGH	O	The essential signal is an error classification status signal. Prior to exiting the classification state, the controller sets this signal to reflect whether the error occurred on an essential bit(s). Then, the controller exits classification state.
status_uncorrectable	HIGH	O	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, the user can take remedial action.

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

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

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

Error Injection Interface

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

Table 2-18: **Error Injection Interface Signals**

Name	Sense	I/O	Description
inject_strobe	HIGH	I	The error injection control is used to indicate an error injection request. The inject_strobe signal should be pulsed high for one cycle, synchronous to icap_clk, concurrent with the application of a valid address to the inject_address input. The error injection control must only be used when the controller is idle.
inject_address[39:0]	HIGH	I	The error injection address bus is used to specify the parameters for an error injection. The value on this bus is captured at the same time inject_strobe is sampled active.

The user provides an error injection address and command on `inject_address` and asserts `inject_strobe` to indicate an error injection request.

In response, the controller injects a bit error. The controller confirms receipt of the error injection command by asserting `status_injection`. When the injection command has completed, the controller deasserts `status_injection`. To inject errors affecting multiple bits, a sequence of error injections can be performed.

For more information on error injection commands, see Error Injection Interface in Chapter 3.

Monitor Interface

The Monitor Interface provides a mechanism for the user to interact with the controller.

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 Error Injection Interface. The Monitor Interface is intended for use in processor based systems.

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

Name	Sense	I/O	Description
monitor_txdata[7:0]	HIGH	O	Parallel transmit data from controller.
monitor_txwrite	HIGH	O	Write strobe, qualifies validity of parallel transmit data.
monitor_txfull	HIGH	I	This signal implements flow control on the transmit channel, from the shim (peripheral) to the controller.
monitor_rxdata[7:0]	HIGH	I	Parallel receive data from the shim (peripheral).
monitor_rxread	HIGH	O	Read strobe, acknowledges receipt of parallel receive data.
monitor_rxempty	HIGH	I	This signal implements flow control on the receive channel, from the shim (peripheral) to the controller.

The Monitor Interface connects to the MON shim in the system-level example design. The MON shim implements an RS-232 protocol-compatible, full duplex serial port for the exchange of commands and status. See Monitor Interface in Chapter 3 for more details on the MON shim functionality.

Fetch Interface

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

During error correction and error classification, the controller may need to fetch a frame of configuration data or 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.

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

Name	Sense	I/O	Description
fetch_txdata[7:0]	HIGH	O	Parallel transmit data from controller.
fetch_txwrite	HIGH	O	Write strobe, qualifies validity of parallel transmit data.
fetch_txfull	HIGH	I	This signal implements flow control on the transmit channel, from the shim (peripheral) to the controller.
fetch_rxdata[7:0]	HIGH	I	Parallel receive data from the shim (peripheral).

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

Name	Sense	I/O	Description
fetch_rxread	HIGH	O	Read strobe, acknowledges receipt of parallel receive data.
fetch_rxempty	HIGH	I	This signal implements flow control on the receive channel, from the shim (peripheral) to the controller.
fetch_tbladdr[31:0]	HIGH	I	Used to specify the starting address of the controller data table in the external source.

Designing with the Core

This chapter provides details on how to apply the core from three different levels, using a bottom-up approach. This chapter includes the following sections:

- Interfaces describes how to connect to the solution.
- Behaviors describes how to interact with the solution through its interfaces.
- Systems describes integrating the solution into a larger system.

Interfaces

The system-level design example exposes four to six interfaces, depending on the options selected when it is generated. Each interface is described separately. The interface-level descriptions are intended to convey how to connect each interface.

Clock Interface

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

- Duty Cycle: 45% minimum, 55% maximum

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

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

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

- For the EXT shim and memory interface:
  - The SPI bus timing budget must be evaluated to determine the maximum SPI bus clock frequency; a sample analysis is presented in External Interface, page 40.
  - The SPI bus clock is the input clock divided by two; therefore, the input clock cannot exceed twice the maximum SPI bus clock frequency.
- For the MON shim and serial interface:
  - The input clock and the serial interface baud rate are related by an integer multiple of sixteen. 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 presented in Monitor Interface, page 38.

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

Status Interface

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

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

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

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

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

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

The collective switching behavior of the state signals `status_initialization`, `status_observation`, `status_correction`, `status_classification`, and `status_injection` is illustrated in Figure 3-1. In the figure, the `status_[`*`state`*`]`

signal represents the five state signals, as a group, which can be considered an indication of the controller state.

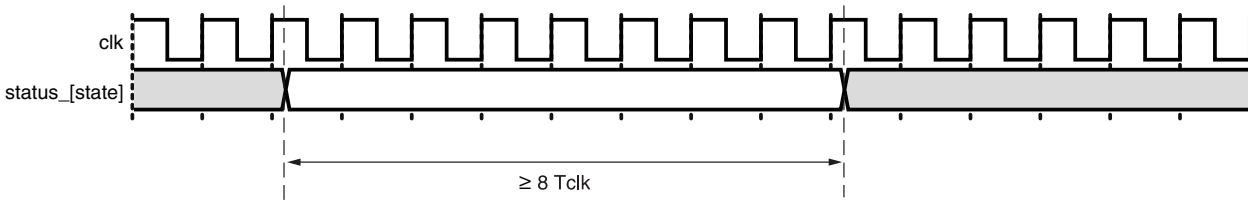

Figure 3-1: **Status Interface State Signals Switching Characteristics**

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 3-2 and Figure 3-3. 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.

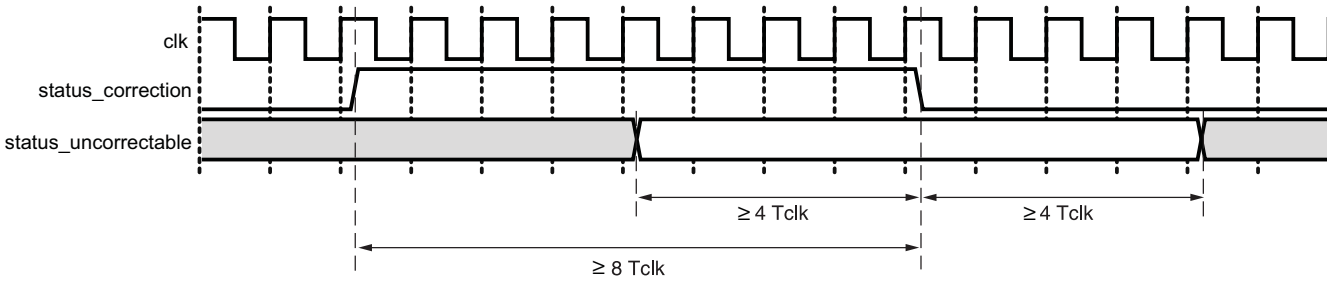

Figure 3-2: **Status Interface Uncorrectable Flag Switching Characteristics**

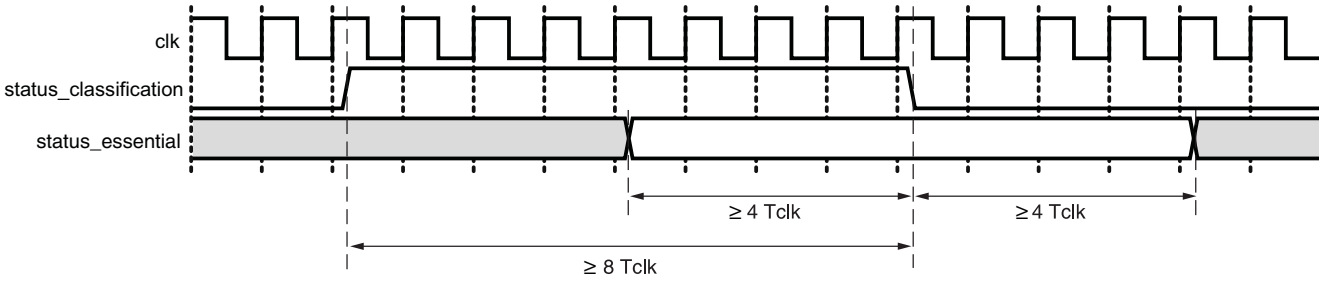

Figure 3-3: **Status Interface Essential Flag Switching Characteristics**

The switching behavior of the heartbeat signal `status_heartbeat` is illustrated in Figure 3-4. This signal is a direct output from the readback process, and is active during the observation state. Upon entering the observation state, the heartbeat signal becomes active when the readback process is scanning for errors. The first heartbeat pulse observed during the observation state must be used to arm any circuit that monitors for loss of heartbeat. In all other states, the heartbeat signal behavior is unspecified.

The first heartbeat after entering the observation state should occur within three readback scan times. A readback scan time depends on the device and clock frequency. See Table 2-4 to determine how long a readback scan takes for the device being used.

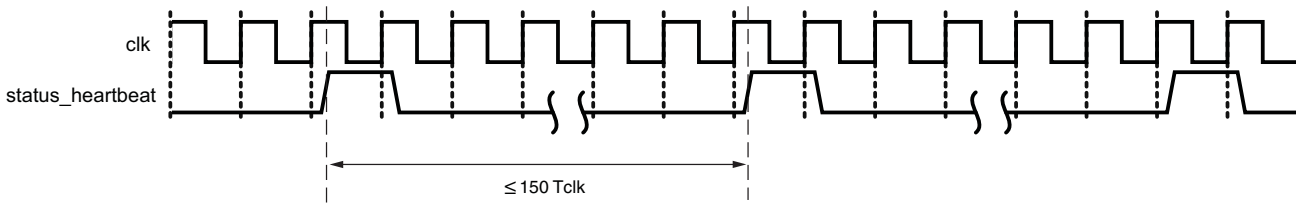

Figure 3-4: **Status Interface Heartbeat Switching Characteristics**

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

Monitor Interface

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: 9600
- 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 Monitor Interface must support this configuration. Figure 3-5 shows the switching behavior, and is representative of both transmit and receive.

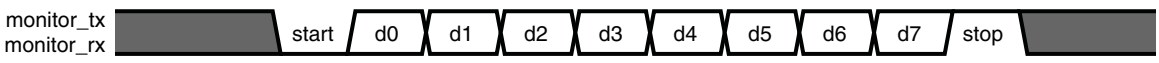

Figure 3-5: **Monitor Interface Switching Characteristics**

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 9600 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 disadvantage of such a low bit rate is communication performance (both data rate and latency). This performance can throttle the controller. For this reason, changing to a higher bit rate is strongly encouraged. A wide variety of other bit rates are possible, including standard bit rates: 115200, 230400, 460800, and 921600 baud.

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

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

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

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

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

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

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

Electrically, the I/O pins used by the Monitor 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.

TIP: *For details about customizing the MON shim that attaches to the Monitor Interface, see MON Shim Customizations.*

External Interface

The External Interface consists of four signals implementing a SPI bus protocol compatible, full duplex serial port. This interface is only present when one or both of the following controller options are enabled:

- Error Correction by Replacement
- Error Classification

The implementations of these functions require external storage. The system-level design example provides a fixed-function SPI bus master in the EXT shim to fetch data from a single external SPI flash device. Table 3-1 provides the SPI flash density requirement for each supported FPGA.

Table 3-1: **External Storage Requirements**

Device	Error Classification Only	Error Correction by Replacement Only	Error Classification with Error Correction by Replacement
XC7A12T	16 Mbit	16 Mbit	32 Mbit
XC7A15T	16 Mbit	16 Mbit	32 Mbit
XC7A25T	16 Mbit	16 Mbit	32 Mbit
XC7A35T	16 Mbit	16 Mbit	32 Mbit
XC7A50T	16 Mbit	16 Mbit	32 Mbit
XC7A75T	32 Mbit	32 Mbit	64 Mbit
XC7A100T	32 Mbit	32 Mbit	64 Mbit
XC7A200T	64 Mbit	64 Mbit	128 Mbit
XC7K70T	32 Mbit	32 Mbit	64 Mbit
XC7K160T	64 Mbit	64 Mbit	128 Mbit
XC7K325T	128 Mbit	128 Mbit	256 Mbit
XC7K355T	128 Mbit	128 Mbit	256 Mbit
XC7K410T	128 Mbit	128 Mbit	256 Mbit
XC7K420T	128 Mbit	128 Mbit	256 Mbit
XC7K480T	128 Mbit	128 Mbit	256 Mbit
XC7VX330T	128 Mbit	128 Mbit	256 Mbit
XC7VX415T	128 Mbit	128 Mbit	256 Mbit
XC7VX485T	128 Mbit	128 Mbit	256 Mbit
XC7VX550T	256 Mbit	256 Mbit	512 Mbit
XC7VH580T (SSI)	256 Mbit	256 Mbit	512 Mbit
XC7V585T	128 Mbit	128 Mbit	256 Mbit
XC7VX690T	256 Mbit	256 Mbit	512 Mbit

Table 3-1: **External Storage Requirements *(Cont'd)***

Device	Error Classification Only	Error Correction by Replacement Only	Error Classification with Error Correction by Replacement
XC7VH870T (SSI)	256 Mbit	256 Mbit	512 Mbit
XC7VX980T	256 Mbit	256 Mbit	512 Mbit
XC7VX1140T (SSI)	512 Mbit	512 Mbit	1024 Mbit
XC7V2000T (SSI)	512 Mbit	512 Mbit	1024 Mbit
XC7Z007S	16 Mbit	16 Mbit	32 Mbit
XC7Z012S	32 Mbit	32 Mbit	64 Mbit
XC7Z014S	32 Mbit	32 Mbit	64 Mbit
XC7Z010	16 Mbit	16 Mbit	32 Mbit
XC7Z015	32 Mbit	32 Mbit	64 Mbit
XC7Z020	32 Mbit	32 Mbit	64 Mbit
XC7Z030	64 Mbit	64 Mbit	128 Mbit
XC7Z035	128 Mbit	128 Mbit	256 Mbit
XC7Z045	128 Mbit	128 Mbit	256 Mbit
XC7Z100	128 Mbit	128 Mbit	256 Mbit
XC7S6	8 Mbit	8 Mbit	16 Mbit
XC7S15	8 Mbit	8 Mbit	16 Mbit
XC7S25	16 Mbit	16 Mbit	32 Mbit
XC7S50	16 Mbit	16 Mbit	32 Mbit
XC7S75	32 Mbit	32 Mbit	64 Mbit
XC7S100	32 Mbit	32 Mbit	64 Mbit

The EXT shim 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 3-1. In the EXT shim system-level example design module, there are three parameters that control the command sequence sent to the SPI flash device.

- B_ISSUE_WREN
  - Indicates if a write enable command (0x06) must be issued prior to any other commands that modify the device behavior.
  - Must be set to "1" for N25Q devices, but generally set to "0" for other devices.
- B_ISSUE_WVCR
  - Indicates if a write volatile configuration register command (0x81) must be issued to explicitly set the fast read dummy cycle count to eight cycles. The state machine

in the EXT shim is byte-oriented and expects the fast read dummy cycle count to be eight. The volatile configuration register data is overwritten (0x8b).

  - Must be set to "1" for N25Q devices, but generally set to "0" for other devices.

- B_ISSUE_EN4B
  - Indicates if an enable four-byte addressing command (0xb7) must be issued to explicitly enter the four-byte addressing mode.
  - Must be set to "1" for devices greater than 128 Mbits.

For storage requirements ≤ 128 Mbits, the EXT shim supports M25P devices by default (B_ISSUE_WREN = 0, B_ISSUE_WVCR = 0, B_ISSUE_EN4B = 0). These devices are not capable of four-byte addressing mode.

For storage requirements > 128 Mbits, the EXT shim supports higher-density N25Q devices by default (B_ISSUE_WREN = 1, B_ISSUE_WVCR = 1, B_ISSUE_EN4B = 1). These devices are capable of four-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 greater than 128 Mbits (B_ISSUE_WREN = 0, B_ISSUE_WVCR = 0, B_ISSUE_EN4B = 1).

Figure 3-6 shows the connectivity between an FPGA and a 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.

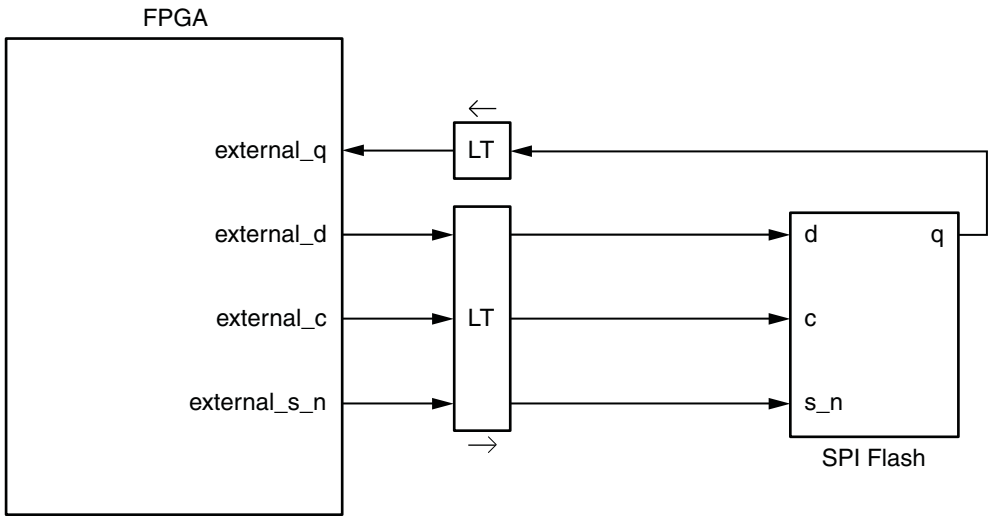

Figure 3-6: **SPI flash Device Connection, Including Level Translators**

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.

The following sections illustrate how to analyze the SPI bus timing budgets. This is a critical analysis which must be performed to ensure reliable data transfer over the SPI bus. Every implementation should be considered unique and be carefully evaluated to ensure the timing budgets posed in the example are satisfied.

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.

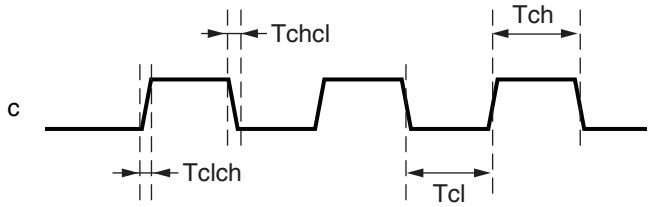

Figure 3-7: **SPI Flash Device Input Clock Requirements**

The following parameters, shown in Figure 3-7, 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} = 9 ns (from SPI flash data sheet)
- T_{ch} = 9 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 + 9 ns, or $T_{clk} \geq$ 11 ns
4. Calculate: $T_{clk} \geq T_{fall}$ + Tcl requires $T_{clk} \geq$ 2 ns + 9 ns, or $T_{clk} \geq$ 11 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 11 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.

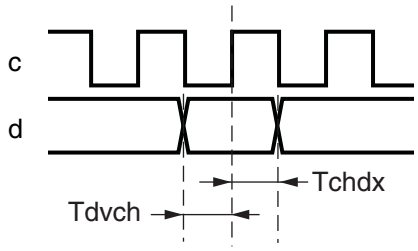

Figure 3-8: **SPI Flash Device Input Data Capture Requirements**

The following parameters, shown in Figure 3-8, 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 EXT shim 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 EXT shim implementation is shown in Figure 3-9.

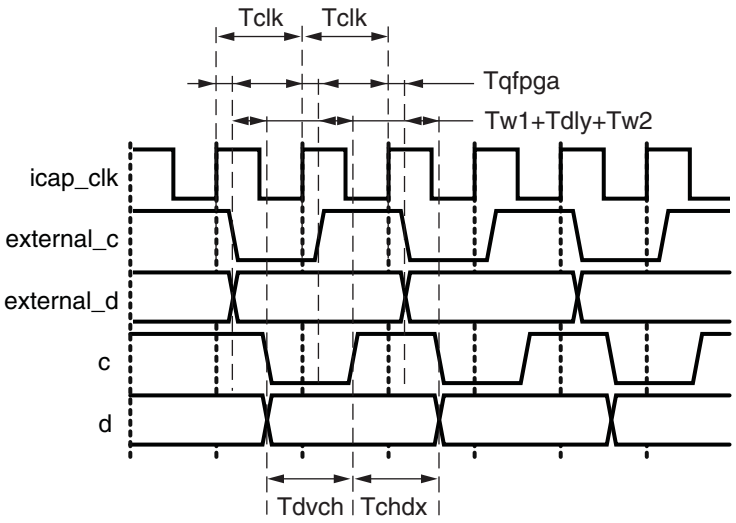

Figure 3-9: **Input Data Capture Timing**

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} = 2 ns (from SPI flash data sheet)
- T_{chdx} = 5 ns (from SPI flash data sheet)

1. Calculate: $T_{clk} \geq T_{dvch}$ requires $T_{clk} \geq 2$ ns
2. Calculate: $T_{clk} \geq T_{chdx}$ requires $T_{clk} \geq 5$ 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 5 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.

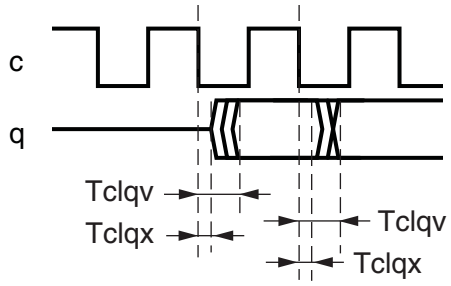

Figure 3-10: **SPI Flash Device Output Data Switching Characteristics**

The following parameters, shown in Figure 3-10, 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 EXT shim 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 EXT shim, 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 EXT shim implementation is shown in Figure 3-11 and Figure 3-12.

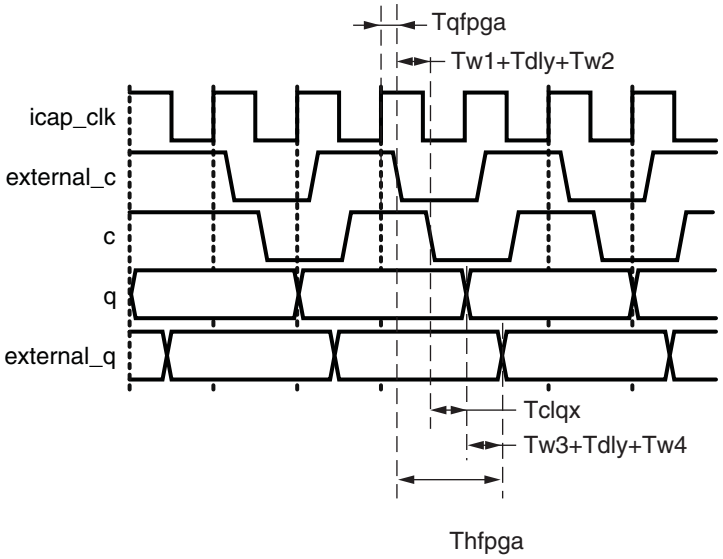

Figure 3-11: **Output Data Capture Timing (Hold Analysis)**

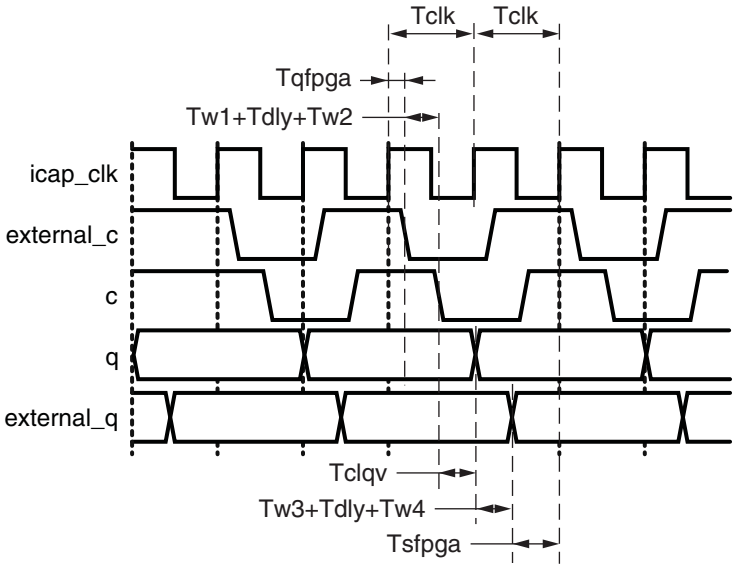

Figure 3-12: **Output Data Capture Timing (Setup Analysis)**

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-7 FPGA

- T_{clqv} = 8 ns (from SPI flash data sheet)
- T_{clqx} = 0 ns (from SPI flash data sheet)
- T_{dly} = 3 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-7 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 clearly 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 "external_c".

- T_{qfpga} = I/O Datapath Delay (external_c)
- T_{qfpga} = 3.211 ns, maximum

```
Slack (MET) :             20.670ns
  Source:                 example_ext/example_ext_byte/ext_c_ofd/C
                            (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@7.576ns period=15.151ns})
  Destination:            external_c
                            (output port clocked by clk  {rise@0.000ns fall@7.576ns
period=15.151ns})
  Path Group:             clk
  Path Type:              Max at Slow Process Corner
  Requirement:            15.151ns
  Data Path Delay:        3.211ns  (logic 3.211ns (100.000%)  route 0.000ns (0.000%))
  Logic Levels:           1  (OBUF=1)
  Output Delay:           -15.151ns
```

```
  Clock Path Skew:        -6.385ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    0.000ns
    Source Clock Delay      (SCD):    6.385ns
    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

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
    R24                                               0.000     0.000 r  clk
                         net (fo=0)                   0.000     0.000    clk
    R24                                                             r  example_ibuf/I
    R24                  IBUF (Prop_ibuf_I_O)         1.176     1.176 r  example_ibuf/O
                         net (fo=1, routed)           3.130     4.305    clk_ibufg
    BUFGCTRL_X0Y0                                                   r  example_bufg/I
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)         0.093     4.398 r  example_bufg/O
                         net (fo=477, routed)         1.987     6.385    example_ext/
example_ext_byte/icap_clk
    OLOGIC_X0Y37                                                    r  example_ext/
example_ext_byte/ext_c_ofd/C
  -------------------------------------------------------------------
-------------------
    OLOGIC_X0Y37         FDRE (Prop_fdre_C_Q)         0.366     6.751 r  example_ext/
example_ext_byte/ext_c_ofd/Q
                         net (fo=1, routed)           0.000     6.751    n_96_example_ext
    AB20                                                          r  external_c_OBUF_inst/I
    AB20               OBUF (Prop_obuf_I_O)        2.845     9.596 r  external_c_OBUF_inst/O
                         net (fo=0)                   0.000     9.596    external_c
    AB20                                                            r  external_c
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)       15.151    15.151 r
                         clock pessimism              0.000    15.151
                         clock uncertainty           -0.035    15.116
                         output delay                15.151    30.267
  -------------------------------------------------------------------
                         required time                         30.267
                         arrival time                          -9.596
  -------------------------------------------------------------------
                         slack                                 20.670
```

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 "external_c".

- T_{qfpga} = I/O Datapath Delay (external_c)
- T_{qfpga} = 1.379 ns, minimum

```
Slack (MET) :             4.460ns
  Source:                 example_ext/example_ext_byte/ext_c_ofd/C
```

```
                          (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@7.576ns period=15.151ns})
  Destination:            external_c
                            (output port clocked by clk  {rise@0.000ns fall@7.576ns
period=15.151ns})
  Path Group:             clk
  Path Type:              Min at Fast Process Corner
  Requirement:            0.000ns
  Data Path Delay:        1.379ns  (logic 1.379ns (100.000%)  route 0.000ns (0.000%))
  Logic Levels:           1  (OBUF=1)
  Output Delay:           0.000ns
  Clock Path Skew:        -3.117ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    0.000ns
    Source Clock Delay      (SCD):    3.117ns
    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

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
    R24                                               0.000     0.000 r  clk
                         net (fo=0)                   0.000     0.000    clk
    R24                                                             r  example_ibuf/I
    R24                  IBUF (Prop_ibuf_I_O)         0.616     0.616 r  example_ibuf/O
                         net (fo=1, routed)           1.675     2.291    clk_ibufg
    BUFGCTRL_X0Y0                                                   r  example_bufg/I
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)         0.026     2.317 r  example_bufg/O
                         net (fo=477, routed)         0.800     3.117    example_ext/
example_ext_byte/icap_clk
    OLOGIC_X0Y37                                                    r  example_ext/
example_ext_byte/ext_c_ofd/C
  -------------------------------------------------------------------
-------------------
    OLOGIC_X0Y37         FDRE (Prop_fdre_C_Q)         0.192     3.309 r  example_ext/
example_ext_byte/ext_c_ofd/Q
                         net (fo=1, routed)           0.000     3.309    n_96_example_ext
    AB20                                                             r  external_c_OBUF_inst/I
    AB20                 OBUF (Prop_obuf_I_O)         1.187     4.495 r  external_c_OBUF_inst/O
                         net (fo=0)                   0.000     4.495    external_c
    AB20                                                            r  external_c
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)        0.000     0.000 r
                         clock pessimism              0.000     0.000
                         clock uncertainty            0.035     0.035
                         output delay                -0.000     0.035
  -------------------------------------------------------------------
                         required time                         -0.035
                         arrival time                           4.495
  -------------------------------------------------------------------
                         slack                                  4.460
```

Locate T_{sfpga} by searching the timing report for pad to flip-flop path analysis at Max at Slow Process Corner, where the source pad is identified as "external_q".

- Tsfpga = I/O Datapath Delay (external_q)
- Tsfpga = 7.813 ns, maximum

```
Slack (MET) :             28.041ns
  Source:                 external_q
                            (input port clocked by clk  {rise@0.000ns fall@7.576ns
period=15.151ns})
  Destination:            example_ext/example_ext_byte/ext_q_ifd/D
                           (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@7.576ns period=15.151ns})
  Path Group:             clk
  Path Type:              Max at Slow Process Corner
  Requirement:            15.151ns
  Data Path Delay:        7.813ns  (logic 7.813ns (100.000%)  route 0.000ns (0.000%))
  Logic Levels:           2  (IBUF=1 ZHOLD_DELAY=1)
  Input Delay:            -15.151ns
  Clock Path Skew:        5.588ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    5.588ns
    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

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
                         input delay                -15.151   -15.151
    AD21                                              0.000   -15.151 r  external_q
                         net (fo=0)                   0.000   -15.151    external_q
    AD21                                                        r  external_q_IBUF_inst/I
    AD21                IBUF (Prop_ibuf_I_O)        1.161   -13.990 r  external_q_IBUF_inst/O
                         net (fo=1, routed)           0.000   -13.990    example_ext/
example_ext_byte/external_q_IBUF
    ILOGIC_X0Y30                                                    r  example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIN
    ILOGIC_X0Y30         ZHOLD_DELAY (Prop_zhold_delay_DLYIN_DLYIFF)
                                                     6.797    -7.193 r  example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIFF
                         net (fo=1, routed)           0.000    -7.193    example_ext/
example_ext_byte/OPT_ZHD_N_ext_q_ifd
    ILOGIC_X0Y30                                                    r  example_ext/
example_ext_byte/ext_q_ifd/D
    ILOGIC_X0Y30         FDRE (Setup_fdre_C_D)       -0.145    -7.338    example_ext/
example_ext_byte/ext_q_ifd
  -------------------------------------------------------------------
-------------------

                         (clock clk rise edge)       15.151    15.151 r
    R24                                              0.000    15.151 r  clk
                         net (fo=0)                  0.000    15.151    clk
```

```
    R24                                                              r  example_ibuf/I
    R24                  IBUF (Prop_ibuf_I_O)         1.113    16.264 r  example_ibuf/O
                         net (fo=1, routed)           2.604    18.868    clk_ibufg
    BUFGCTRL_X0Y0                                                    r  example_bufg/I
    BUFGCTRL_X0Y0        BUFG (Prop_bufg_I_O)         0.083    18.951 r  example_bufg/O
                         net (fo=477, routed)         1.788    20.739    example_ext/
example_ext_byte/icap_clk
    ILOGIC_X0Y30                                                     r  example_ext/
example_ext_byte/ext_q_ifd/C
                         clock pessimism              0.000    20.739
                         clock uncertainty           -0.035    20.703
  -------------------------------------------------------------------
                         required time                         20.703
                         arrival time                           7.338
  -------------------------------------------------------------------
                         slack                                 28.041
```

Locate T_{hfpga} by searching the timing report for pad to flip-flop path analysis at Min at Fast Process Corner, where the source pad is identified as "external_q".

- T_{hfpga} = I/O Datapath Delay (external_q)
- T_{hfpga} = -3.386 ns, minimum

```
Slack (MET) :             29.797ns
  Source:                 external_q
                            (input port clocked by clk  {rise@0.000ns fall@7.576ns
period=15.151ns})
  Destination:            example_ext/example_ext_byte/ext_q_ifd/D
                           (rising edge-triggered cell FDRE clocked by clk  {rise@0.000ns
fall@7.576ns period=15.151ns})
  Path Group:             clk
  Path Type:              Min at Fast Process Corner
  Requirement:            0.000ns
  Data Path Delay:        3.386ns  (logic 3.386ns (100.000%)  route 0.000ns (0.000%))
  Logic Levels:           2  (IBUF=1 ZHOLD_DELAY=1)
  Input Delay:            30.302ns
  Clock Path Skew:        3.856ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    3.856ns
    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

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------
-------------------
                         (clock clk rise edge)        0.000     0.000 r
                         input delay                 30.302    30.302
    AD21                                              0.000    30.302 r  external_q
                         net (fo=0)                   0.000    30.302    external_q
    AD21                                                        r  external_q_IBUF_inst/I
    AD21                IBUF (Prop_ibuf_I_O)        0.601    30.903 r  external_q_IBUF_inst/O
```

```
                          net (fo=1, routed)           0.000    30.903    example_ext/
example_ext_byte/external_q_IBUF
    ILOGIC_X0Y30                                                      r  example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIN
    ILOGIC_X0Y30          ZHOLD_DELAY (Prop_zhold_delay_DLYIN_DLYIFF)
                                                       2.939    33.842 r  example_ext/
example_ext_byte/ext_q_ifd_OPT_INSERTED/DLYIFF
                          net (fo=1, routed)           0.000    33.842    example_ext/
example_ext_byte/OPT_ZHD_N_ext_q_ifd
    ILOGIC_X0Y30                                                      r  example_ext/
example_ext_byte/ext_q_ifd/D
    ILOGIC_X0Y30          FDRE (Hold_fdre_C_D)        -0.154    33.688    example_ext/
example_ext_byte/ext_q_ifd
  -------------------------------------------------------------------
-------------------

                          (clock clk rise edge)        0.000     0.000 r
    R24                                                0.000     0.000 r  clk
                          net (fo=0)                   0.000     0.000    clk
    R24                                                                r  example_ibuf/I
    R24                   IBUF (Prop_ibuf_I_O)         0.782     0.782 r  example_ibuf/O
                          net (fo=1, routed)           1.984     2.765    clk_ibufg
    BUFGCTRL_X0Y0                                                      r  example_bufg/I
    BUFGCTRL_X0Y0         BUFG (Prop_bufg_I_O)         0.030     2.795 r  example_bufg/O
                          net (fo=477, routed)         1.061     3.856    example_ext/
example_ext_byte/icap_clk
    ILOGIC_X0Y30                                                       r  example_ext/
example_ext_byte/ext_q_ifd/C
                          clock pessimism              0.000     3.856
                          clock uncertainty            0.035     3.892
  -------------------------------------------------------------------
                          required time                        -3.892
                          arrival time                         33.688
  -------------------------------------------------------------------
                          slack                                29.797
```

Check:

- Is $T_{hfpga} \leq T_{qfpga,min} + T_{clqx}$?
- Is -3.386 ns ≤ 1.379 ns + 0 ns?
- Is -3.386 ns ≤ 1.379 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

$T_{clk} \geq 0.5 \times (3.211 \text{ ns} + 1 \text{ ns} + 3 \text{ ns} + 1 \text{ ns} + 8 \text{ ns} + 1 \text{ ns} + 3 \text{ ns} + 1 \text{ ns} + 7.813 \text{ ns})$

or

$T_{clk} \geq 14.512 \text{ ns}$

The hold requirement is satisfied, and the requirement on Tclk indicates that the SPI Receive Waveform and Timing Budget restrict the system-level design example input clock cycle time to be 14.512 ns or larger.

SPI Bus Timing Budget Conclusions

When the EXT shim and external memory system are present, the SPI bus timing budget must be analyzed to ensure a robust implementation. The result of the analysis confirms that the external memory system is functional, and reveal any constraints it might pose on the maximum frequency of the system-level design example input clock.

Example Conclusion

Using the example data from the Vivado Design Suite timing report for a Kintex-7 SEM IP implementation, the memory interface is functional. The most stringent requirement on Tclk is that Tclk ≥ 14.512 ns, as the memory interface only works when the input clock frequency is 68.908 MHz or lower. Other input clock frequency limits, such as the internal configuration access port (ICAP) maximum clock frequency and the system-level example maximum clock frequency, must also be considered.

Error Injection Interface

The Error Injection Interface consists of an input bus and input strobe, implementing a simple parallel input port. This interface is only present when Error Injection is enabled and I/O Pins are selected. Direct, logic-signal-based error injection is possible from the Error Injection Interface. The use of this interface is entirely optional. This interface accepts five types of commands:

- Command to enter Idle state
- Command to enter Observation state
- Command to perform a software reset (reboot and initialize)
- Inject error at physical frame address
- Inject error at linear frame address

TIP: *For more details about Error Injection Interface commands, see Error Injection Interface Commands.*

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

In one configuration, the system-level design example brings all of the signals to I/O pins. It is possible to generate and drive the error injection signals inside the FPGA; this is what

is done when the Vivado Design Suite debug feature interface is selected rather than I/O pins.

Externally, the error injection signals can be connected to another device for control. To properly capture supplied commands, the timing requirements of the Error Injection Interface must be accounted for when interfacing to another device.

The signals in the Error Injection Interface are received by a sequential logic process in controllers using the strobe to enable an input register.

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.

Using Vivado Design Suite Debug Feature

The status signals received by the synchronous input port should be represented with virtual LEDs. The `inject_address` output value should be represented as HEX for data entry. The `inject_strobe` control output must be represented as a synchronous 1-cycle pulse output for proper operation. See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 6] for more information on using the VIO core.

Behaviors

The system-level design example exhibits certain high-level functional behaviors during operation, based on the controller design. This section is intended to convey expected behaviors and how to interact with the system-level design example. In SSI implementations of the system-level design example, the multiple controller instances exhibit the same behaviors as the single controller in a non-SSI implementation.

Controller Activity

Initialization

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

The controller polls its `icap_grant` input during boot to determine if it has been granted permission to enter the initialization state and begin using ICAP. In implementations for most devices, `icap_grant` should be tied High. Implementations in Zynq-7000 devices, however, require special handling of the `icap_grant` signal.

During boot of the Zynq-7000 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-7000 Programmable Logic (PL). When the PS bootloader completes, 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. In order for the controller to function, configuration logic access must be transferred to the ICAP. This is accomplished by clearing PCAP_PR (bit 27) in the PS device configuration control register (DEVCFG CTRL, address 0xF8007000). See the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) for more information regarding PCAP [Ref 3]. Figure 3-13 illustrates how PCAP, ICAP, and PCAP_PR interact.

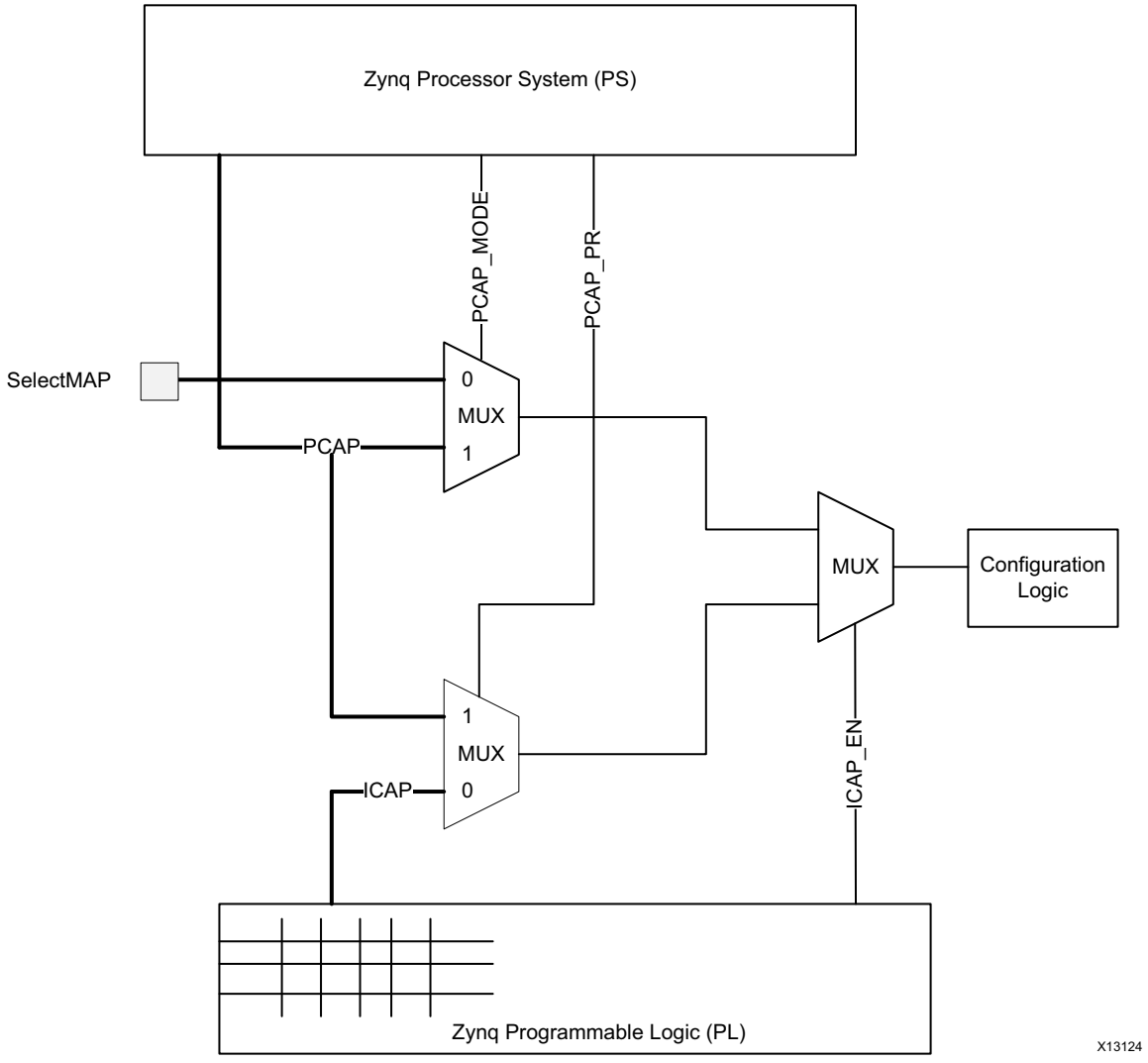

Figure 3-13: **Configuration Logic Access in Zynq-7000**

The controller has no simple method to sense the ICAP is locked out. During the initialization state, the controller polls the ICAP, attempting to read the configuration logic IDCODE register, until it observes the expected vendor identification value. However, if the PS clears PCAP_PR during a controller attempt to access the ICAP, the configuration logic might receive a malformed ICAP transaction. This results in unpredictable behavior of the configuration logic.

To eliminate this possibility, it is necessary for the PS to drive the controller `icap_grant` input through the GPIO and prevent the controller from entering the initialization state until after PCAP_PR has been cleared. The GPIO used might either be an EMIO from the PS or a GPIO in the PL, but it must be initialized so that the controller observes `icap_grant` deasserted immediately upon completion of PL configuration.

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. The software implementation of this behavior is outside the scope of this document. See *Zynq-7000 EPP Software Developers Guide* (UG821) [Ref 4] and *Xilinx OS and Libraries Document Collection* (UG643) [Ref 5], for detailed information about software development for Baremetal and Linux environments.

During the initialization state, status_initialization is TRUE. Initialization includes some internal housekeeping, as well as directly observable events such as the generation of a status report on the Monitor Interface. The specific activities are:

- A first readback cycle during which the frame-level ECC checksums are computed
- A second readback cycle during which the device-level CRC checksum is computed
- An additional readback cycle during which an additional checksum is computed
- An additional readback cycle during which the frame-level CRC checksums are computed and stored in block RAM (only executed if correction by enhanced repair is used)

At the completion of initialization, the controller transitions to the observation state.

Observation

The controller spends virtually all of its time in the observation state. During the observation state, `status_observation` is TRUE and the controller observes the FPGA configuration system for indication of error conditions. If no error exists, and the controller receives a command (from either the Error Injection 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.

The “enter idle” command can be applied through either the Error Injection Interface or the Monitor Interface, and 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 “status report” 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 hardware in preparation for a correction attempt. After the controller has gathered the available information, it transitions to the correction state.

Correction

The controller attempts to correct errors in the correction state. The controller always passes through the correction state, even if correction is disabled. During the correction state, `status_correction` is TRUE.

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 the behavior of the controller depends on how it has been configured to correct errors.

If the controller is configured for correction by replace, it generates a replacement data request on the Fetch Interface. In the system-level design example, the EXT shim translates this request into a read of the external memory. The return data is provided to the controller by the EXT shim. The controller then performs active partial reconfiguration to re-write the frame with the correct contents. The controller clears `status_uncorrectable` and generates a report on the Monitor Interface. It then transitions to the classification state.

If the controller is configured for correction by repair or correction by enhanced repair, it attempts to correct the error using algorithmic methods. If the error is correctable, the controller performs active partial reconfiguration to re-write the frame with the corrected contents and clears `status_uncorrectable`. Otherwise, the controller sets `status_uncorrectable`. In either case, the controller generates a report and then transitions to the classification state.

Classification

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

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. And, if this is the case, the controller cannot look up the error to determine whether it is essential. In these cases, the controller sets `status_essential`, generates a report, 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.

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 a classification data request on the Fetch Interface. In the system-level design example, the EXT shim translates this request into a read of the external memory. The return data is provided to the controller by the EXT shim. With this data, the controller then determines whether it is essential. In all cases, the controller generates a report, changes `status_essential` as appropriate, and then transitions to the observation state to resume looking for errors.

Idle

The idle state is similar to the observation state, except that the controller does not observe the FPGA configuration system for indication of error conditions. The idle state is indicated by the deassertion of all five state bits on the Status Interface. If the controller receives a command (from either the Error Injection Interface or the Monitor Interface), then the controller processes the received command. The error injection and software reset commands are only supported in the idle state.

The “enter observation” command can be applied through either the Error Injection Interface or the Monitor Interface, and is used to return the controller to the observation state so that errors can be detected.

The “status report” 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.

Any desired set of “error injection” commands can be applied through either the Error Injection Interface or the Monitor Interface. These commands direct the controller to perform error injections. The primary reason the idle state exists is to halt actions taken in response to error detections so that multi-bit errors can be constructed.

The "software reset" command can also be applied through either the Error Injection Interface or the Monitor Interface. This command directs the controller to perform a software reset.

Injection

The controller performs error injections in the injection state. The controller always passes through the injection state in response to a valid error injection command issued from the idle state, even if error injection is disabled; this can occur if error injection commands are presented on the Monitor Interface, as the Monitor Interface exists even when error injection is disabled. During the injection state, `status_injection` is TRUE.

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 injections commands, each resulting in a transition through the injection state. At the end of error injection, the controller must be moved from the idle state back into the observation state.

Fatal Error

The controller enters the fatal error state when it detects an internal inconsistency. Although very unlikely, it is possible for the controller to halt due to soft errors that affect the controller-related configuration memory or the controller design state elements.

The fatal error state is indicated by the assertion of all five state bits on the Status Interface, along with a fatal error report message. This condition is non-recoverable, and the FPGA must be reconfigured.

Error Injection Interface Commands

The Error Injection Interface commands define what a user can send to the controller through the Error Injection Interface. This command set offers basic capability to inject errors.

Commands are presented by applying a value to the `inject_address` bus, and then asserting the `inject_strobe` signal. After a command is presented, do not present another command until the Status Interface or Monitor Interface has confirmed completion of the previous command.

Directed State Changes

The controller can be moved between observation and idle states by a directed state change. The command format is shown in Figure 3-14 and Figure 3-15, with "X" representing a "don't care."

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 3-14: **Enter Idle State Command**

39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 3-15: **Enter Observation State Command**

Completion of the command is indicated on the Status Interface by a change in the state outputs indicating the controller has entered the requested state. Completion of the command is indicated on the Monitor Interface by a change in the command prompt indicating the controller has entered the requested state.

Error Injection

There are two addressing schemes to specify the frame address for an error injection. These are linear frame addressing and physical frame addressing. Linear frame addressing is simple, but the addresses do not provide information about type and physical location of the frame. The error injection command format for linear frame address is shown in Figure 3-16, with borders marking nibble boundaries and shading marking separate bit fields in the command.

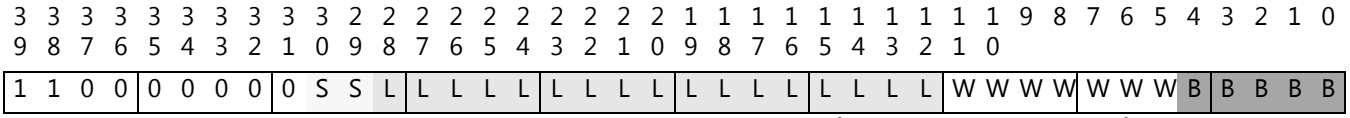

Figure 3-16: **Error Injection Command (Linear Frame Address)**

Where:

- SS = Hardware SLR number for SSI (2-bit) and set to 00 for non-SSI
- LLLLLLLLLLLLLLLL = linear frame address (17-bit) [0..MF]
- WWWWWWW = word address (7-bit) [0..100]
- BBBBB = bit address (5-bit) [0..31]

An error injection command using physical frame address has the form shown in Figure 3-17.

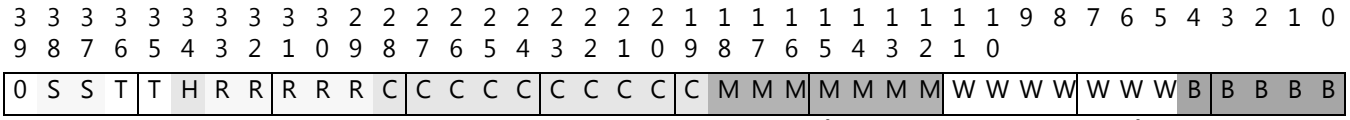

Figure 3-17: **Error Injection Command (Physical Frame Address)**

Where:

- SS = Hardware SLR number for SSI (2-bit) and set to 00 for non-SSI
- TT = block type (2-bit)
- H = half address (1-bit)
- RRRRR = row address (5-bit)
- CCCCCCCCCC = column address (10-bit)

- MMMMMMM = minor address (7-bit)
- WWWWWWW = word address (7-bit) [0..100]
- BBBBB = bit address (5-bit) [0..31]

Completion of the command is indicated on the Status Interface by the deassertion of the `status_injection` output indicating the controller has exited the error injection state. Completion of the command is indicated on the Monitor Interface by a return of the command prompt indicating the controller has exited the error injection state.

Software Reset

The controller can be moved from the idle state, through the initialization state, to the observation state by a software reset using the software reset command. The command format is shown in Figure 3-18, with "X" representing a "don't care."

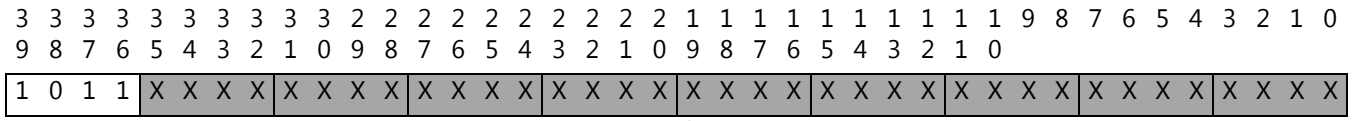

Figure 3-18: **Software Reset Command**

Completion of the command is indicated on the Status Interface by a change in the state outputs indicating the controller has entered the observation state. Completion of the command is indicated on the Monitor Interface by an initialization report followed by a command prompt indicating the controller has entered the observation state.

Monitor Interface Commands

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

Directed State Changes

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

Status Report

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

Error Injection

“N” command is used to perform an error injection. The controller only accepts this command when in the idle state. The format of the command is:

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

Issuing this command is analogous to presenting an error injection command on the Error Injection Interface. The hex value supplied with this command represents the same value that would be applied to the Error Injection Interface.

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}
```

Issuing this command is analogous to presenting a software reset command on the Error Injection Interface. The hex value supplied with this command is a "don't care".

Monitor Interface Messages

The Monitor 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. This report is generated when the controller first starts and at subsequent software resets.

```
X7_SEM_V4_1             Name and Version
SC 01                   State Change, Initialization
FS {2-digit hex value}  Core Configuration Information
ICAP OK                 Status: ICAP Available
RDBK OK                 Status: Readback Active
INIT OK                 Status: Completed Setup
SC 02                   State Change, Observation
```

Command Prompt

The command prompt issued by the controller is one of two characters, depending on the controller state. If the controller is in observation state (the default state after initialization completes) the prompt issued is `O>`. If the controller is in idle state, the prompt issued is `I>`.

State Change Report

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

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

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

Table 3-2: **State Change Report Decoding**

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

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

```
HLT
```

Flag Change Report

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

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

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

Table 3-3: **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

Status Report

A status report provides more information about the controller state. It is a multiple-line report that is generated in response to the `S` command, provided the controller is in the observation or idle states. The non-SSI device status report has the following format:

```
MF {8-digit hex value}   Maximum Frame (linear count)
SN {2-digit hex value}   Hardware SLR Number
SC {2-digit hex value}   Current State
FC {2-digit hex value}   Current Flags
FS {2-digit hex value}   Feature Set
```

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 four SLRs generates a report in this format:

```
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
MF {8-digit hex value} Maximum Frame (linear count)
SN {2-digit hex value} Hardware SLR Number
SC {2-digit hex value} Current State for this Hardware SLR
FC {2-digit hex value} Current Flags for this Hardware SLR
FS {2-digit hex value} Feature Set
```

Error Detection Report

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

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

```
SC 04
CRC
```

Diagnosis: Enhanced repair checksum buffer uncorrectable error

```
SC 04
BFR
```

Diagnosis: 1-bit ECC error, SYNDROME is valid. Controller reports physical frame address, linear frame address, word in frame, and bit in word.

```
SC 04
SED OK
PA {8-digit hex value}
LA {8-digit hex value}
WD {2-digit hex value} BT {2-digit hex value}
```

Diagnosis: 1-bit ECC error, SYNDROME is invalid, unexpectedly indicating an out-of-range word address. Controller reports physical frame address and linear frame address. If this diagnosis code occurs, the FPGA must be reconfigured.

```
SC 04
SED NG
PA {8-digit hex value}
LA {8-digit hex value}
```

Diagnosis: 2-bit ECC error. Controller reports physical frame address and linear frame address.

```
SC 04
DED
PA {8-digit hex value}
LA {8-digit hex value}
```

Error Correction Report

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

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

```
COR
END
```

Followed by:

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

or

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

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

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

Followed by:

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

or

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

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

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

Error Classification Report

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

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

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

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

```
SC 08
CLA
{classification list}
END
FC 40          Bit 6, essential is set
```

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

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

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

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

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

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

Systems

Although the soft error mitigation solution can operate autonomously, many applications are in conjunction 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.

The following points illustrate methods by which a system-level supervisory function can monitor the soft error mitigation solution.

TIP: *None of these are required.*

- Monitor the soft error mitigation solution to determine if additional system-level actions are necessary in response to a soft error event. This can be as generic as a soft error event logging function, or can be a substantially complex and application specific response, based on the type of error and its classification (for example, reset logic, reconfigure device, reboot system, etc.).

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

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

  - **Boot and Initialization States**: Monitor the soft error mitigation solution to confirm it boots, initializes, and enters the observation state. Xilinx specifies the soft error mitigation solution boots, initializes, and enters the observation state within the time specified through Table 2-3 and Equation 2-1, provided that the `icap_grant` signal is asserted, the FPGA configuration logic is available to the soft error mitigation solution through the ICAP primitive, and there is no throttling on the Monitor Interface.

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

To confirm the solution initializes and enters the observation state, the system-level supervisory function can observe the Status Interface `status_initialization` and `status_observation` signals for assertion, or the Monitor Interface `monitor_tx` signal for the expected initialization report.

- **Observation State**: The controller spends virtually all of its time in this state. There are at least three methods for monitoring the controller in this state, each provides slightly different information about the health of the controller:
  - Controller heartbeat, `status_heartbeat`: This signal is a direct output from the soft error mitigation solution. This signal exhibits pulses, specified in the "Status Interface" section, which indicate the readback process is active. If, during the observation state, these pulses become out-of-specification, the system-level supervisory function should conclude that the readback process has experienced a fault. This condition is an uncorrectable, essential error. In SSI implementations, which have a status_heartbeat output per SLR, it is necessary to monitor the heartbeat from all SLRs.

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

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

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

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

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

Status command and report processing by the controller can be an undesirable source of additional latency. For example, a status command transmission period of 60 seconds might be a reasonable trade-off to guard against rare "controller unresponsive" conditions while not adding significant additional latency to general operation. As a counter example, one second would be a poor choice. In this counter example, the status reports could keep the Monitor Shim 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 Monitor Shim 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 2-5/Equation 2-3 and Table 2-7/Equation 2-4, provided there is no throttling on the Monitor Interface. Due to the infrequency of soft errors, the controller spends very little time in these states and normally transitions back to the observation state, or less frequently, the idle state. If the controller dwells continuously in either 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. Several rare issues might generate this symptom, ranging from soft errors in the controller to hard errors in the device itself.

- **Idle State and Injection State**: The controller only enters the idle state as a result of an uncorrectable error, or if specifically directed. In the event of an uncorrectable error, see the section above about monitoring event reporting. Directed entry to the idle state is generally for the purpose of issuing other commands for error injection or ICAP sharing. It is inadvisable to implement the “Observation State” point above for status command and report monitoring during the idle state as it might conflict with commands issued by other processes at the application level. Instead, the application-level processes should test that any issued command completes and generates a response within one second. Otherwise, an uncorrectable, essential error has occurred and the application should report this to the system.
- **Halt State**: The controller only enters this state when it has detected inconsistent internal state. This condition is observable on the Status Interface as the assertion of all five state indicators, and on the Monitor Interface as a HLT message.

In SSI implementations, where more than one controller instance exists, the solution is considered halted if one or more of the controller instances halts. This is an uncorrectable, essential error.

Customizations

The system-level design example encapsulates the SEM Controller and various shims that serve to interface the controller to other devices. These shims can include I/O Pins, Vivado Design Suite debug feature interface, I/O Interfaces, Memory Controllers, or application-specific system management interfaces.

As delivered, the system-level design example is not a reference design, but an integral part of the total solution and fully verified by Xilinx. While designers do have the flexibility to modify the system-level design example, the recommended approach is to use it as delivered. However, if modifications are desired, this chapter provides additional detail required for success.

This chapter does not describe customization of the application that exists in the system-level design example. This portion of the system-level design example is for demonstration purposes only and not functionally involved in soft error mitigation. The only anticipated customization of the application is to remove it.

HID Shim Customizations

The HID shim is a bridge between the controller and an interface device. The resulting interface can be used to exchange commands and status with the controller. The HID shim is only present in certain controller configurations. When present, it exports access to the Status Interface and Error Injection Interface through the use of Vivado logic analyzer. When absent, the Status Interface and Error Injection Interface are only accessible through I/O pins.

If desired, the Status Interface and Error Injection Interface can be connected in other ways. These interfaces are easy to disconnect from the HID shim or I/O Pins and reconnect to other logic, such as register files or finite state machines. See the Status Interface, page 36 and Error Injection Interface, page 55.

MON Shim Customizations

The MON shim is a bridge between the controller and a standard RS-232 port. The resulting interface can be used to exchange commands and status with the controller. This interface is designed for connection to processors.

Increase Bit Rate

If the RS-232 interface is needed, changing to the highest feasible bit rate is strongly encouraged. This reduces the potential for throttling of the controller due to status report transmission. See Monitor Interface, page 38 for more information.

Increase Buffer Depth

Increasing the MON shim bit rate is the easiest method to reduce the potential for throttling of the controller due to status report transmission. Another method is to increase the buffer depth. The MON shim contains two FIFOs, a transmit buffer and a receive buffer.

There is no need for the buffer depths to be symmetric and little advantage is gained from increasing the depth of the receive buffer. However, increasing the depth of the transmit buffer reduces the potential for throttling of the controller due to status report transmission.

The Xilinx LogiCORE IP FIFO Generator can be used to create replacements for the MON shim FIFOs. The FIFO configuration must be for a common clock (that is, fully synchronous to a single clock) with first word fall through enabled. The data width must be eight, with the depth as great as desired.

TIP: *When making a FIFO replacement, note that an "empty" flag is the logical inverse of a "data present" flag.*

Replace with Alternate Function (Non-SSI Devices)

If an interface other than RS-232 is desired, the MON shim can be replaced with an alternate function. For example, the MON shim could be replaced with a custom processor interface or other scheme for inter-process communication. When replacing the MON shim with an alternate function, it becomes critical to understand the behavior of the controller monitor interface.

For an overview of the signals, see Monitor Interface, page 38. The data exchanged over this interface is ASCII. For a summary of the status and command formats, see Monitor Interface Commands, page 63 and Monitor Interface Messages, page 64.

Figure 3-19 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

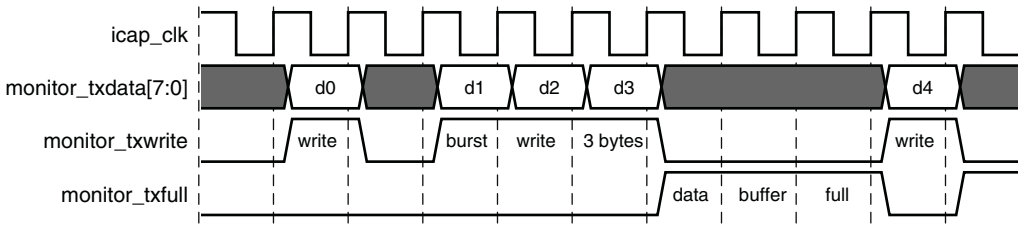

Figure 3-19: **Monitor Interface Transmit Protocol**

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.

It is the responsibility of the peripheral receiving the data to correctly track its buffer availability and report 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 3-20 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.

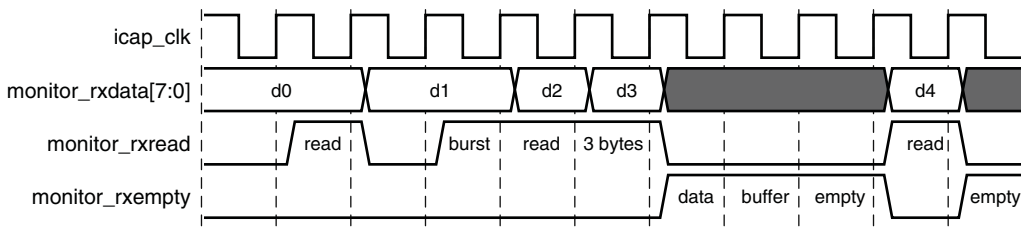

Figure 3-20: **Monitor Interface Receive Protocol**

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.

It is the responsibility of the peripheral providing the data to correctly track its buffer status and report it through `monitor_rxempty`. In the cycle after the peripheral samples `monitor_rxread` High, it must assert `monitor_rxempty` if the data written completely fills the buffer.

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.

During the initialization state, the controller purges the receive buffer by reading and discarding data until it is empty. The controller assumes the transmit buffer is ready immediately and does not wait for the transmit buffer to empty, as it has no way to observe this condition.

Replace with Alternate Function (SSI Devices)

The MON shim delivered with the system-level example design for implementation in SSI devices implements additional functions such as arbitration and report collation for multiple controller instances.

Xilinx strongly discourages replacing the MON shim in SSI devices.

Removal

Although it is possible to remove the MON shim, Xilinx strongly discourages this customization. If removing the MON shim, Xilinx recommends preserving the two I/O pins used by the MON shim and making those accessible for probing at test points. Even though the MON shim might not be necessary in certain applications, it offers a critical debugging capability that is required when obtaining assistance from Xilinx technical support teams.

To eliminate the MON shim, disconnect and remove it, including any related signals and ports used to bring the RS-232 signals to I/O pins at the design top level. Then, address the exposed monitor interface on controllers:

- Leave all controller monitor interface outputs “open” or “unconnected.”
- Connect all `monitor_txfull` inputs to logic zero.
- Connect all `monitor_rxempty` inputs to logic one.
- Connect all `monitor_rxdata[7:0]` inputs to logic zero.

Elimination of the MON shim reduces the size of the solution and also prevents throttling of the controller due to status report transmission.

EXT Shim Customizations

The EXT shim is a bridge between the controller and a standard SPI bus. The resulting interface can be used to fetch data by the controller. This shim is only present in certain controller configurations and is designed for connection to standard SPI flash.

Alter Command Sequence

If the SPI bus interface is needed, it might be necessary to alter the command sequence sent to the SPI flash device. The EXT shim can be configured to support one of several different families of SPI flash. See External Interface for more information.

Replace with Alternate Function (Non-SSI Devices)

If an interface other than SPI bus is needed, the EXT shim can be replaced with an alternate function. For example, the EXT shim could be replaced with a parallel flash memory controller or other scheme for inter-process communication. When replacing the EXT shim with an alternate function, it becomes critical to understand the behavior of the Controller Fetch Interface.

For an overview of the signals, Fetch Interface in Chapter 2. The byte transfer protocols for the Fetch Interface are identical to those of the Monitor Interface.

Figure 3-21 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.

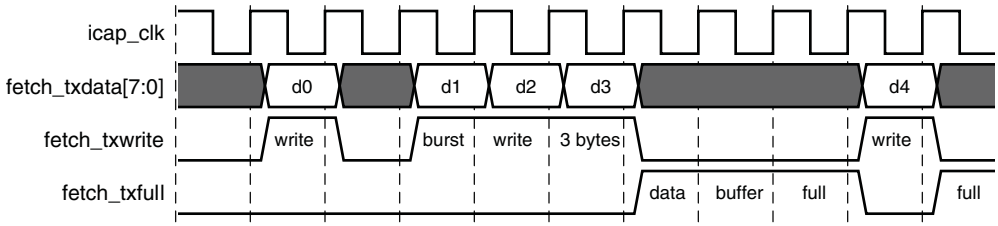

Figure 3-21: **Fetch Interface Transmit Protocol**

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.

It is the responsibility of the peripheral receiving the data to correctly track its buffer availability and report 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 3-22 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.

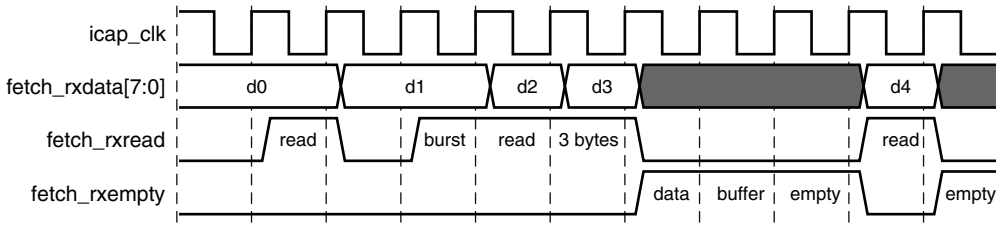

Figure 3-22: **Fetch Interface Receive Protocol**

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.

It is the responsibility of the peripheral providing the data to correctly track its buffer status and report it through `fetch_rxempty`. In the cycle after the peripheral samples `fetch_rxread` High, it must assert `fetch_rxempty` if the data written completely fills the buffer.

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.

During the initialization state, the controller purges the receive buffer by reading and discarding data until it is empty. It then performs two more reads. Reads with an empty buffer condition are decoded as a reset signal. The controller assumes the reset signal initializes the transmit buffer and therefore does not wait for the transmit buffer to empty, as it has no way to observe this condition.

The data exchanged is binary and represents a command-response pair. The controller transmits a 6-byte command sequence to initiate a data fetch. 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]

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. For additional information about the organization of the data in the address space, see External Memory Programming File in Chapter 5.

Replace with Alternate Function (SSI Devices)

The EXT shim delivered with the system-level example design for implementation in SSI devices is complex, as it implements additional functions such as arbitration for multiple controller instances. Xilinx strongly discourages replacing the EXT shim in SSI devices.

Data Consistency

When using optional features such as error correction by replacement and error classification, 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. The files are related.

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

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

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, selects if masking is enabled or disabled. The controller always enables masking.

7 series FPGAs and Zynq-7000 SoCs implement fine grain masking at a resource level. This means individual resources, when configured for dynamic operation, have their configuration memory bits masked. Only the required memory bits are masked, without impacting unrelated memory bits. The masked bits are no longer monitored by the controller.

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

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

Clocking

The master 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 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 the FPGA configuration system and the maximum clock frequency of the system-level design example and controller are not transiently violated. For example, the clock output of a DLL or PLL might be "out of specification" while those functions lock. One method of handling this is to use a global clock buffer with enable. Only enable the global clock buffer after lock is achieved.

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.

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.

Additional Considerations

Design limitations of the SEM core include the following:

- EasyPath™ devices are not compatible with the error correction by replace method.
- The SEM Controller initializes and manages the FPGA integrated silicon features for soft error mitigation. When the controller is included in a design, do not include any design constraints or options that would enable the built-in detection functions. Enabling the necessary functions to provide detection is performed autonomously by the SEM Controller. 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 core.
- Software computed ECC and CRC values are not supported.
- Simulation of designs that instantiate the controller is supported. However, it is not possible to observe the controller behaviors in simulation. Simulation of a design 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.

- When the controller requires storage of configuration data for correction by replace, this data must be available to the controller through the Fetch Interface, typically through the EXT shim. This decouples the controller from the FPGA configuration method and allows customers flexibility in selection of configuration method, configuration data storage, and soft error mitigation solution data storage.
- The EXT shim implementation supports only one SPI flash read command (fast read) in SPI Mode 0 (CPOL = 0, CPHA = 0) to a single SPI flash device.
- Due to potential I/O voltage incompatibility between the FPGA and standard SPI flash devices, level translation might be required in the design of the SPI memory system.
- ICAP Arbitration and ICAP Switchover are not supported. Only a single ICAP instance is supported, and it must reside at the primary/top physical location.
- Use of design capture, including the use of the capture primitive and related functionality, is not supported by the controller.
- Controller implementations for 7 series FPGAs and Zynq-7000 SoCs operate on soft errors in Type 0, Type 2, and Type 3 configuration frames. See the *Xilinx 7 Series FPGA Configuration User Guide*, for information on these configuration frame types [Ref 1].
- 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.

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 IP Subsystems Using IP Integrator* (UG994) [Ref 7]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 9]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10]

Customizing and Generating the Core

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

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

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

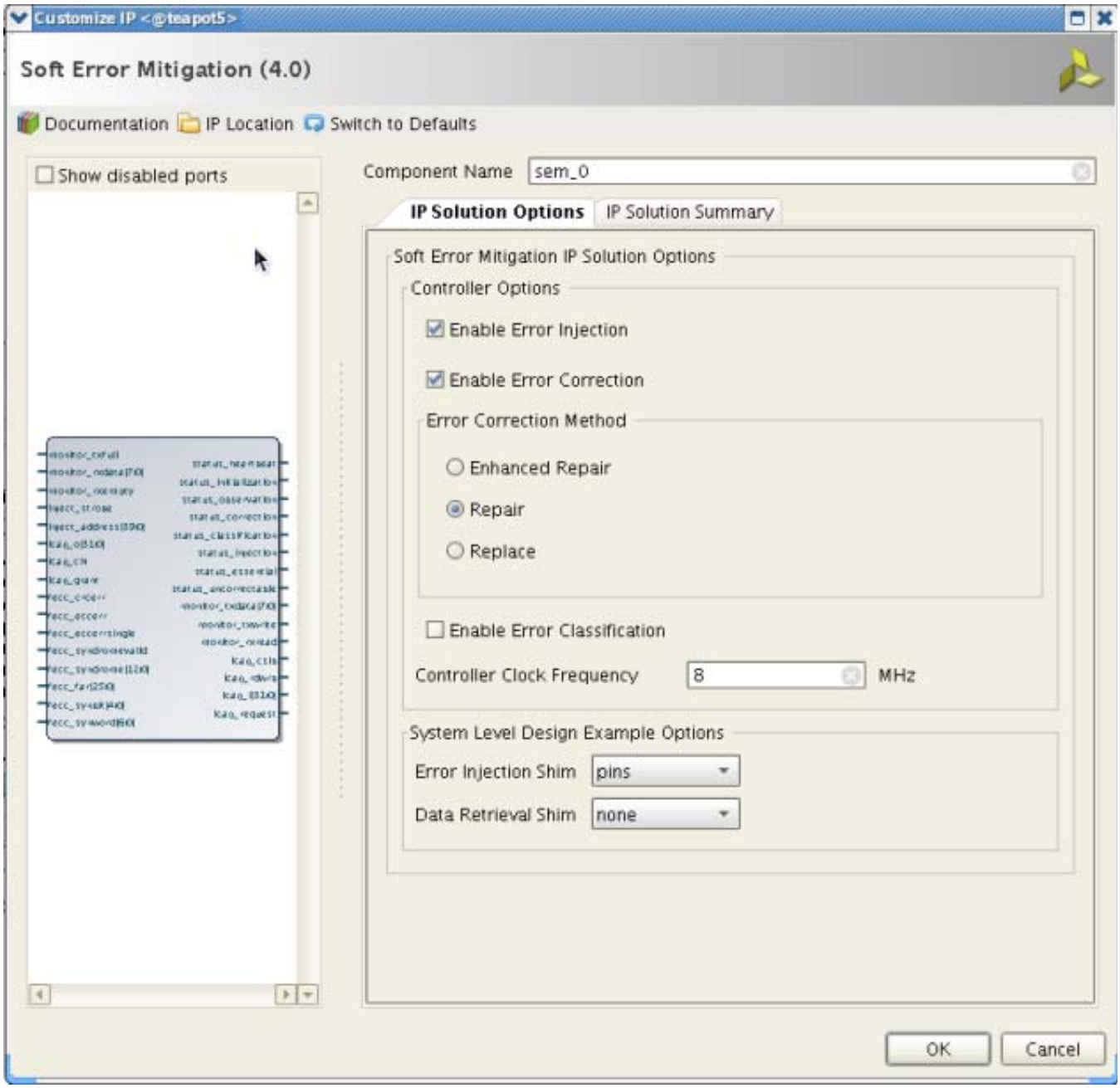

Figure 4-1: **Solution Customization Dialog Box Tab 1**

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 is integrated. The following sub-sections discuss the options in detail to serve as a guide.

Component Name and Symbol

The name of the generated component is set by the Component Name field. The name "sem_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.

Controller Options: Enable Error Injection

The Enable Error Injection check box is used to enable or disable the error injection feature. Error injection is a design verification function that provides a mechanism for users 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.

If error injection is enabled, the Error Injection Interface is generated (as indicated by the Component Symbol) and the controller performs error injections in response to your commands. These commands can be applied to the Error Injection Interface or to the Monitor Interface.

If error injection is disabled, the Error Injection Interface is removed (as indicated by the Component Symbol) and the controller does not perform any error injections.

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

Controller Options: Enable Error Correction

The Enable Error Correction check box is used to enable or disable the error correction feature. No matter what setting is used, the controller monitors the error detection circuits and reports error conditions.

If error correction is enabled, the controller attempts to correct errors that are detected. The method by which corrections are performed is selectable. Most errors are correctable, and upon 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.

If error correction is disabled, the controller does not attempt to correct errors. At the first error event, the controller stops scanning after reporting the error condition. Further, when error correction is disabled, the error classification feature is also disabled.

Controller Options: Error Correction Method

With error correction enabled, the error correction method is selectable. The available methods are correction by repair, correction by enhanced repair, and correction by replace.

The controller corrects errors using the method you selected. The correction possibilities are:

Correction by Repair

- Correction by repair for one-bit errors. The ECC syndrome is used 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.

Correction by Enhanced Repair

- Correction by repair for one-bit and two-bit adjacent errors. For one-bit errors, the behavior is identical to correction by repair. For two-bit errors, an enhanced CRC-based algorithm capable of correcting two-bit adjacent errors is used. The frame containing the error is read, the relevant bits inverted, and the frame is written back. This is signaled as correctable.

Correction by Replace

- Correction by replace for one-bit and multi-bit errors. The frame containing the error is read. The controller requests replacement data for the entire frame from the Fetch Interface. When the replacement data is available, the controller compares the damaged frame with the replacement frame to identify the location of the errors. Then, the replacement data is written back. This is signaled as correctable.

The difference between repair, enhanced repair, and replace methods becomes clear when considering what happens in the uncommon case of errors involving more bits than can be corrected by the repair or enhanced repair methods. In these cases, the repair or enhanced repair methods does not yield a successful correction. However, if such errors are corrected by the replace method, the error is correctable regardless of how many bits were affected.

The fundamental trade-off between the methods is error correction success rate versus the cost of adding or increasing data storage requirements. Using correction by repair as the baseline for comparison:

- Correction by enhanced repair provides superior correction capability through use of additional block RAM to store frame-level CRCs.
- Correction by replace provides ultimate correction capability through the addition of external SPI flash to store “golden” frame replacement data.

EasyPath™ devices are not compatible with the error correction by replace method.

Controller Options: Enable Error Classification

The Enable Error Classification check box is used to enable or disable the error classification feature. Error classification is automatically disabled if error correction is disabled.

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 2 out of every 10 soft errors (on average) actually merits a system-level mitigation response. The error classification feature is a table lookup to determine if a soft error event has affected essential Configuration Memory locations. Use of this feature reduces the effective FIT of the design. The cost of enabling this feature is the external storage required to hold the lookup table. When error classification is enabled, the Fetch Interface is generated (as indicated by the Component Symbol) so that the controller has an interface through which it can retrieve external data.

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

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

TIP: *Error classification need not be performed by the controller. It is possible to disable error classification by the controller, and implement it elsewhere in the system using the essential bit data provided by the implementation tools and the error report messages issued by the controller through the Monitor Interface.*

Controller Options: Controller Clock Frequency

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

For designs that require a data retrieval interface to fetch external data for error classification or error correction by replace, 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 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.

Example Design Options: Error Injection Shim

For customizations that include error injection, the example design provides two options for a shim to external control of the Error Injection Interface:

- Direct control through physical pins
- Indirect control through JTAG using Vivado Design Suite debug feature

When selecting Vivado Lab Edition to control the Error Injection Interface, the VIO LogiCORE IP is generated and included in the SEM example design.

Example Design Options: Data Retrieval Shim

For customizations that require a data retrieval interface to fetch external data for error classification or error correction by replace, the controller Fetch Interface must be bridged to an external storage device. The example design provides a shim to an external SPI flash device as the only option. If the data retrieval interface is not required, no shim is generated.

Reviewing the Customizations

Proceed to the second tab of the solution customization dialog box. This tab is shown in Figure 4-2.

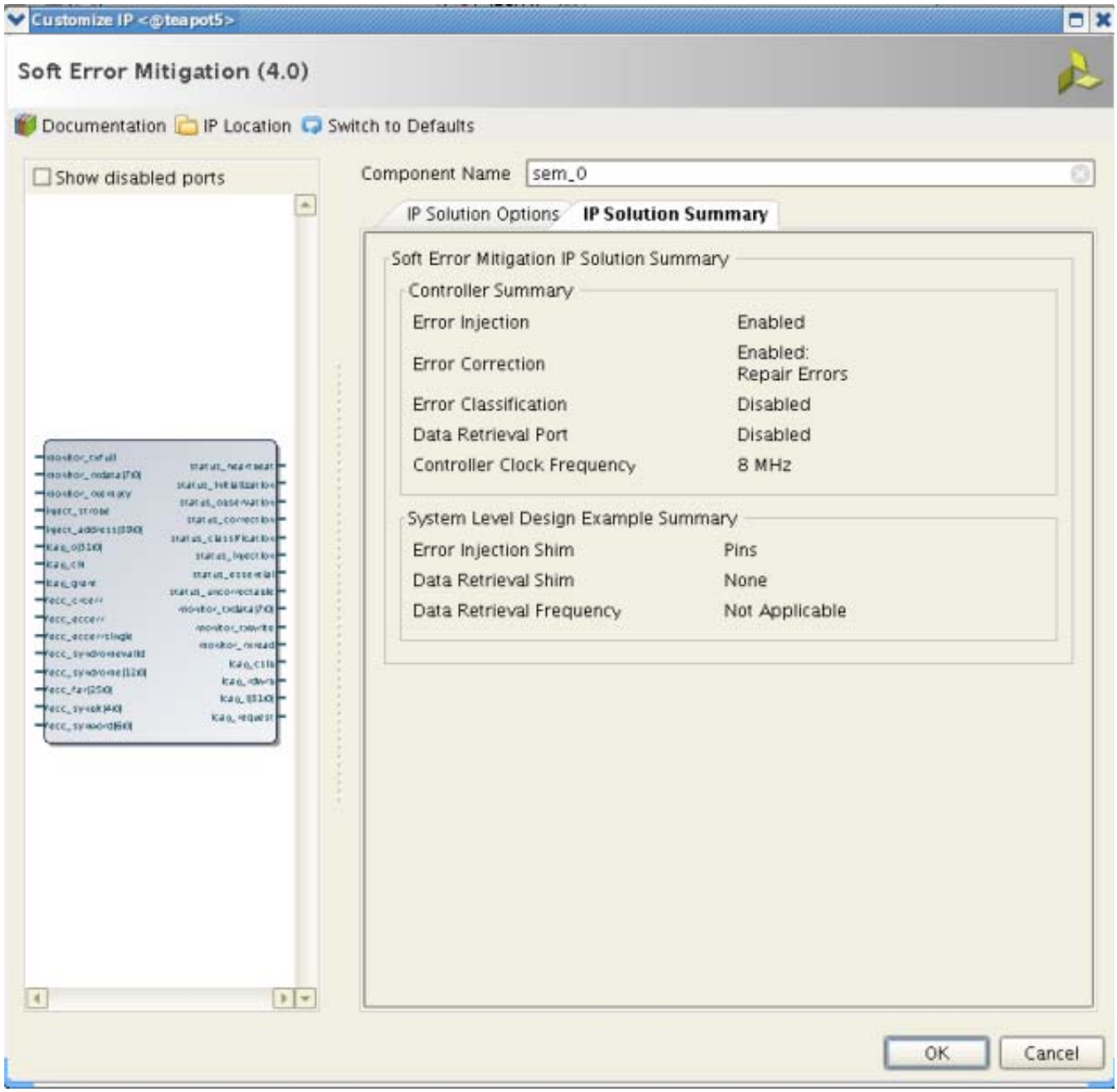

Figure 4-2: **Solution Customization Dialog Box Tab 2**

Review the summary 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
clock_freq	c_clock_per	8 MHz
enable_injection	c_feature_set	TRUE
injection_shim	c_feature_set, c_hardware_cfg	pins
enable_correction	c_feature_set	TRUE
correction_method	c_feature_set	repair
enable_classification	c_feature_set	FALSE
retrieval_shim	N/A	none

Output Generation

The SEM IP solution delivers files into several file groups. The file groups generated can be seen in the IP Sources tab of the Sources window where they are listed for each IP in the project. The file groups available for the SEM core are:

- **Examples**: Includes all source required to be able to open and implement the IP example design project, (Example design HDL and the example design XDC file).
- **Synthesis**: Includes all synthesis sources required by the core. For the SEM core, this is a mix of both encrypted and unencrypted source. Only the unencrypted sources are visible.
- **Instantiation Template**: Example instantiation template.

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. The XDC file, `<component name>_sem_example.xdc`, can be found in the IP Sources tab of the Sources window in the Examples file group.

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™-7 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 master 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 internal configuration access port (ICAP) primitive and the controller are analyzed as part of a clock period constraint on the master clock, because the ICAP clock pin is required to be connected to the same master clock signal.

However, the situation is different for the FRAME_ECC primitive, as it does not have a clock pin. Based on the specific use of the FRAME_ECC primitive with the ICAP primitive, it is known that FRAME_ECC primitive pins are synchronous to the master clock signal. Therefore, additional constraints with values derived from the clock period constraint are added:

```
set_max_delay 12.151 -from [get_pins example_cfg/example_frame_ecc/*] -quiet -datapath_only
set_max_delay 30.302 -from [get_pins example_cfg/example_frame_ecc/*] -to [all_outputs]
-quiet -datapath_only
```

Example Design Constraints

The example design constraints are organized by interface, rather than constraint type. The first group is for the master clock input to the entire design. It applies an I/O standard and a period constraint. The period constraint value is based on options set at generation:

```
create_clock -name clk -period 15.151 [get_ports clk]
```

```
set_property IOSTANDARD LVCMOS25 [get_ports clk]
```

The second group is for the Status Interface, applying an I/O standard and an output timing constraint to the interface. The output timing constraint is set at two times the period constraint.

```
set_property DRIVE 8 [get_ports status_initialization]
set_property SLEW FAST [get_ports status_initialization]
set_property IOSTANDARD LVCMOS25 [get_ports status_initialization]

set_property DRIVE 8 [get_ports status_observation]
set_property SLEW FAST [get_ports status_observation]
set_property IOSTANDARD LVCMOS25 [get_ports status_observation]

set_property DRIVE 8 [get_ports status_correction]
set_property SLEW FAST [get_ports status_correction]
set_property IOSTANDARD LVCMOS25 [get_ports status_correction]

set_property DRIVE 8 [get_ports status_classification]
set_property SLEW FAST [get_ports status_classification]
set_property IOSTANDARD LVCMOS25 [get_ports status_classification]

set_property DRIVE 8 [get_ports status_injection]
set_property SLEW FAST [get_ports status_injection]
set_property IOSTANDARD LVCMOS25 [get_ports status_injection]

set_property DRIVE 8 [get_ports status_uncorrectable]
set_property SLEW FAST [get_ports status_uncorrectable]
set_property IOSTANDARD LVCMOS25 [get_ports status_uncorrectable]

set_property DRIVE 8 [get_ports status_essential]
set_property SLEW FAST [get_ports status_essential]
set_property IOSTANDARD LVCMOS25 [get_ports status_essential]

set_property DRIVE 8 [get_ports status_heartbeat]
set_property SLEW FAST [get_ports status_heartbeat]
set_property IOSTANDARD LVCMOS25 [get_ports status_heartbeat]

set_output_delay -clock clk -15.151 [get_ports [list status_observation
status_correction status_classification status_injection status_uncorrectable
status_essential status_heartbeat status_initialization]] -max
set_output_delay -clock clk 0 [get_ports [list status_observation status_correction
status_classification status_injection status_uncorrectable status_essential
status_heartbeat status_initialization]] -min
```

The third group is for the MON shim, applying an I/O standard and input/output timing constraints to the interface. The input and output timing constraints are set at two times the period constraint.

```
set_property DRIVE 8 [get_ports monitor_tx]
set_property SLEW FAST [get_ports monitor_tx]
set_property IOSTANDARD LVCMOS25 [get_ports monitor_tx]
set_property IOSTANDARD LVCMOS25 [get_ports monitor_rx]

set_input_delay -clock clk -max -15.151 [get_ports monitor_rx]
set_input_delay -clock clk -min 30.302 [get_ports monitor_rx]
set_output_delay -clock clk -15.151 [get_ports monitor_tx] -max
```

```
set_output_delay -clock clk 0 [get_ports monitor_tx] -min
```

The following group is for the EXT shim, and is only present when the EXT shim is generated based on options set at generation. It applies an I/O standard and 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 the input and output timing of the FPGA implementation. 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_property IOB TRUE [get_cells example_ext/example_ext_byte/ext_c_ofd]
set_property IOB TRUE [get_cells example_ext/example_ext_byte/ext_d_ofd]
set_property IOB TRUE [get_cells example_ext/example_ext_byte/ext_q_ifd]
set_property IOB TRUE [get_cells example_ext/ext_s_ofd]

set_property DRIVE 8 [get_ports external_c]
set_property SLEW FAST [get_ports external_c]
set_property IOSTANDARD LVCMOS25 [get_ports external_c]

set_property DRIVE 8 [get_ports external_d]
set_property SLEW FAST [get_ports external_d]
set_property IOSTANDARD LVCMOS25 [get_ports external_d]

set_property DRIVE 8 [get_ports external_s_n]
set_property SLEW FAST [get_ports external_s_n]
set_property IOSTANDARD LVCMOS25 [get_ports external_s_n]

set_property IOSTANDARD LVCMOS25 [get_ports external_q]

set_input_delay -clock clk -max -15.151 [get_ports external_q]
set_input_delay -clock clk -min 30.302 [get_ports external_q]
set_output_delay -clock clk -15.151 [get_ports [list external_d external_s_n
external_c]] -max
set_output_delay -clock clk 0 [get_ports [list external_d external_s_n external_c]]
-min
```

The following group is for the HID shim, and is only present when the HID shim is I/O Pins. It applies an I/O standard and input timing constraint to the interface. The input timing constraint is set at two times the period constraint.

```
set_property IOSTANDARD LVCMOS25 [get_ports inject_strobe]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[0]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[1]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[2]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[3]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[4]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[5]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[6]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[7]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[8]}]
```

```
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[9]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[10]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[11]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[12]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[13]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[14]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[15]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[16]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[17]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[18]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[19]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[20]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[21]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[22]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[23]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[24]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[25]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[26]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[27]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[28]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[29]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[30]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[31]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[32]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[33]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[34]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[35]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[36]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[37]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[38]}]
set_property IOSTANDARD LVCMOS25 [get_ports {inject_address[39]}]

set_input_delay -clock clk -max -14.285 [get_ports [list {inject_address[0]}
{inject_address[1]} {inject_address[2]} {inject_address[3]} {inject_address[4]}
{inject_address[5]} {inject_address[6]} {inject_address[7]} {inject_address[8]}
{inject_address[9]} {inject_address[10]} {inject_address[11]} {inject_address[12]}
{inject_address[13]} {inject_address[14]} {inject_address[15]} {inject_address[16]}
{inject_address[17]} {inject_address[18]} {inject_address[19]} {inject_address[20]}
{inject_address[21]} {inject_address[22]} {inject_address[23]} {inject_address[24]}
{inject_address[25]} {inject_address[26]} {inject_address[27]} {inject_address[28]}
{inject_address[29]} {inject_address[30]} {inject_address[31]} {inject_address[32]}
{inject_address[33]} {inject_address[34]} {inject_address[35]} {inject_address[36]}
{inject_address[37]} {inject_address[38]} {inject_address[39]} inject_strobe]]
set_input_delay -clock clk -min 28.57 [get_ports [list {inject_address[0]}
{inject_address[1]} {inject_address[2]} {inject_address[3]} {inject_address[4]}
{inject_address[5]} {inject_address[6]} {inject_address[7]} {inject_address[8]}
{inject_address[9]} {inject_address[10]} {inject_address[11]} {inject_address[12]}
{inject_address[13]} {inject_address[14]} {inject_address[15]} {inject_address[16]}
{inject_address[17]} {inject_address[18]} {inject_address[19]} {inject_address[20]}
{inject_address[21]} {inject_address[22]} {inject_address[23]} {inject_address[24]}
{inject_address[25]} {inject_address[26]} {inject_address[27]} {inject_address[28]}
{inject_address[29]} {inject_address[30]} {inject_address[31]} {inject_address[32]}
{inject_address[33]} {inject_address[34]} {inject_address[35]} {inject_address[36]}
{inject_address[37]} {inject_address[38]} {inject_address[39]} inject_strobe]]
```

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, minimizing configuration memory used by the design hence minimizing the FIT contribution of the system-level design example.

```
create_pblock SEM_CONTROLLER
resize_pblock -pblock SEM_CONTROLLER -add RAMB18_X2Y50:RAMB18_X4Y59
resize_pblock -pblock SEM_CONTROLLER -add RAMB36_X2Y25:RAMB36_X4Y29
resize_pblock -pblock SEM_CONTROLLER -add SLICE_X36Y125:SLICE_X47Y149
add_cells_to_pblock -pblock SEM_CONTROLLER -cells [get_cells example_controller/*]
add_cells_to_pblock -pblock SEM_CONTROLLER -cells [get_cells example_mon/*]
add_cells_to_pblock -pblock SEM_CONTROLLER -cells [get_cells example_ext/*]

set_property LOC FRAME_ECC_X0Y0 [get_cells example_cfg/example_frame_ecc]
set_property LOC ICAP_X0Y1 [get_cells example_cfg/example_icap]
```

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 necessarily board-specific and therefore cannot be automatically generated. To apply these constraints, uncomment them and fill in valid I/O pin locations for the target board.

```
## set_property LOC <pin loc> [get_ports clk]

## set_property LOC <pin loc> [get_ports status_initialization]
## set_property LOC <pin loc> [get_ports status_observation]
## set_property LOC <pin loc> [get_ports status_correction]
## set_property LOC <pin loc> [get_ports status_classification]
## set_property LOC <pin loc> [get_ports status_injection]
## set_property LOC <pin loc> [get_ports status_uncorrectable]
## set_property LOC <pin loc> [get_ports status_essential]
## set_property LOC <pin loc> [get_ports status_heartbeat]

## set_property LOC <pin loc> [get_ports monitor_tx]
## set_property LOC <pin loc> [get_ports monitor_rx]

## set_property LOC <pin loc> [get_ports external_c]
## set_property LOC <pin loc> [get_ports external_d]
## set_property LOC <pin loc> [get_ports external_q]
## set_property LOC <pin loc> [get_ports external_s_n]

## set_property LOC <pin loc> [get_ports inject_strobe]
## set_property LOC <pin loc> [get_ports {inject_address[0]}]
## set_property LOC <pin loc> [get_ports {inject_address[1]}]
## set_property LOC <pin loc> [get_ports {inject_address[2]}]
## set_property LOC <pin loc> [get_ports {inject_address[3]}]
## set_property LOC <pin loc> [get_ports {inject_address[4]}]
## set_property LOC <pin loc> [get_ports {inject_address[5]}]
## set_property LOC <pin loc> [get_ports {inject_address[6]}]
## set_property LOC <pin loc> [get_ports {inject_address[7]}]
## set_property LOC <pin loc> [get_ports {inject_address[8]}]
## set_property LOC <pin loc> [get_ports {inject_address[9]}]
## set_property LOC <pin loc> [get_ports {inject_address[10]}]
```

```
## set_property LOC <pin loc> [get_ports {inject_address[11]}]
## set_property LOC <pin loc> [get_ports {inject_address[12]}]
## set_property LOC <pin loc> [get_ports {inject_address[13]}]
## set_property LOC <pin loc> [get_ports {inject_address[14]}]
## set_property LOC <pin loc> [get_ports {inject_address[15]}]
## set_property LOC <pin loc> [get_ports {inject_address[16]}]
## set_property LOC <pin loc> [get_ports {inject_address[17]}]
## set_property LOC <pin loc> [get_ports {inject_address[18]}]
## set_property LOC <pin loc> [get_ports {inject_address[19]}]
## set_property LOC <pin loc> [get_ports {inject_address[20]}]
## set_property LOC <pin loc> [get_ports {inject_address[21]}]
## set_property LOC <pin loc> [get_ports {inject_address[22]}]
## set_property LOC <pin loc> [get_ports {inject_address[23]}]
## set_property LOC <pin loc> [get_ports {inject_address[24]}]
## set_property LOC <pin loc> [get_ports {inject_address[25]}]
## set_property LOC <pin loc> [get_ports {inject_address[26]}]
## set_property LOC <pin loc> [get_ports {inject_address[27]}]
## set_property LOC <pin loc> [get_ports {inject_address[28]}]
## set_property LOC <pin loc> [get_ports {inject_address[29]}]
## set_property LOC <pin loc> [get_ports {inject_address[30]}]
## set_property LOC <pin loc> [get_ports {inject_address[31]}]
## set_property LOC <pin loc> [get_ports {inject_address[32]}]
## set_property LOC <pin loc> [get_ports {inject_address[33]}]
## set_property LOC <pin loc> [get_ports {inject_address[34]}]
## set_property LOC <pin loc> [get_ports {inject_address[35]}]
## set_property LOC <pin loc> [get_ports {inject_address[36]}]
## set_property LOC <pin loc> [get_ports {inject_address[37]}]
## set_property LOC <pin loc> [get_ports {inject_address[38]}]
## set_property LOC <pin loc> [get_ports {inject_address[39]}]
```

Constraints for SSI Devices

In the system-level design example, two to four controller instances are generated depending on the device. The controller instances are named to include an identification number ranging from 0 to 3. The controller instance numbering matches the hardware super logic region (SLR) numbering.

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

The shared blocks such as the HID Shim, MON Shim, and EXT Shim also have area constraints to locate them in the Master SLR, which is centrally located in the device and also contains the BSCAN primitives used by the HID Shim.

Enabling Essential Bits Generation in Vivado

If Error Classification or Correction by Replace options are enabled, a bitstream property must be specified prior to bitstream generation to create supplemental files. This property is set in the XDC file or in the Vivado Tcl Console.

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

Device, Package, and Speed Grade

There are no additional device, package or speed grade constraints.

Clock Frequency

There are no additional clock frequency constraints.

Clock Management

There are no additional clock management constraints.

Clock Placement

There are no additional clock placement constraints.

I/O Pins

This section contains details about I/O pins constraints.

I/O Standard

There are no additional I/O standard constraints.

I/O Banking

There are no additional I/O banking constraints.

I/O Placement

There are no additional I/O placement constraints.

Simulation

Simulation of designs that instantiate the controller is supported. In other words, including the controller in a larger project does not adversely affect ability to run simulations of functionality unrelated to the controller. However, it is not possible to observe the controller behaviors in simulation. Simulation of a design including the controller compiles, but the controller does not exit the initialization state.

Hardware-based evaluation of the controller behaviors is required.

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 10].

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

Synthesis and Implementation

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, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

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. Afterward the activity is 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 SEM Controller system-level example design and the interfaces it exposes. The system-level example design encapsulates the controller and various shims that serve to interface the controller to other devices. These shims can include I/O Pins, VIO LogiCORE IP, I/O Interfaces, Memory Controllers, or application-specific system management interfaces.

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

Functions

Also serving as an instantiation vehicle for the controller itself, the system-level design example incorporates five main functions:

- The FPGA configuration system primitives and their connection to the controller.
- The MON shim, 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 EXT shim, a bridge between controllers and a standard SPI bus. The resulting interface can be used to fetch data by controllers. This shim is only present in certain controller configurations and is designed for connection to standard SPI flash.
- The HID shim, a bridge between controllers and an interface device. The resulting interface can be used to exchange commands and status with controllers. This shim is only present in certain controller configurations.

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

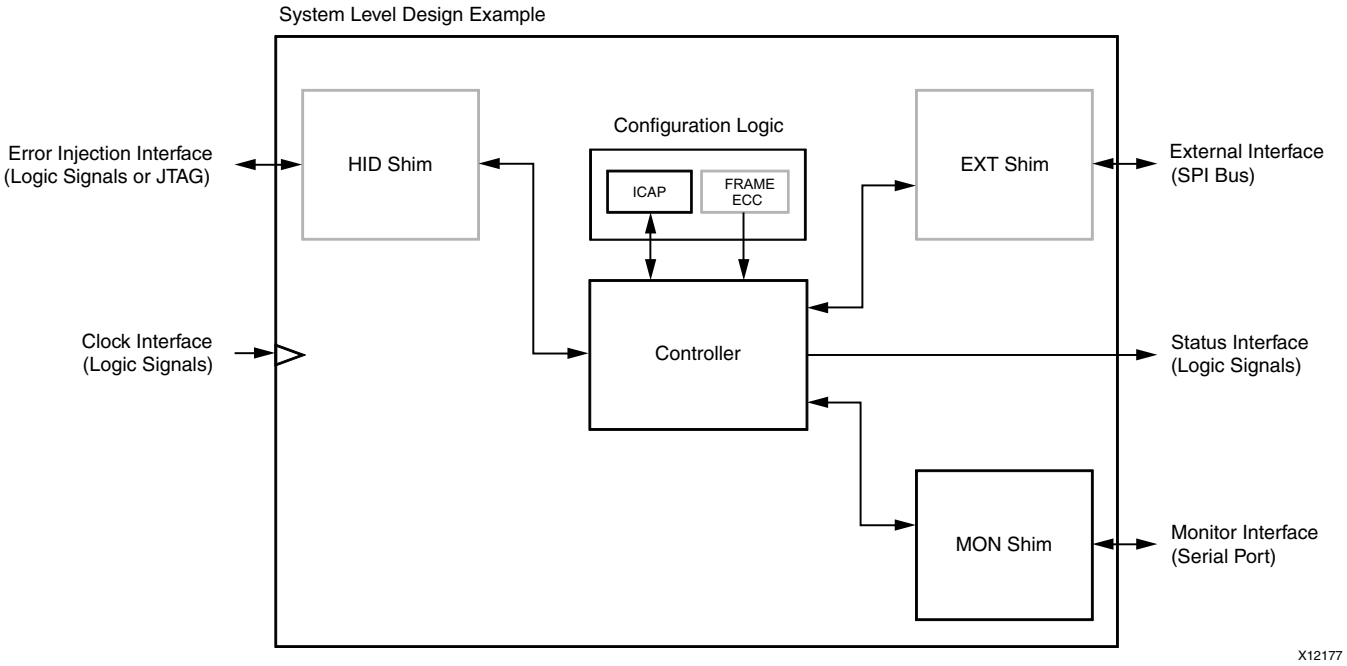

Figure 5-1: **Example Design Block Diagram**

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

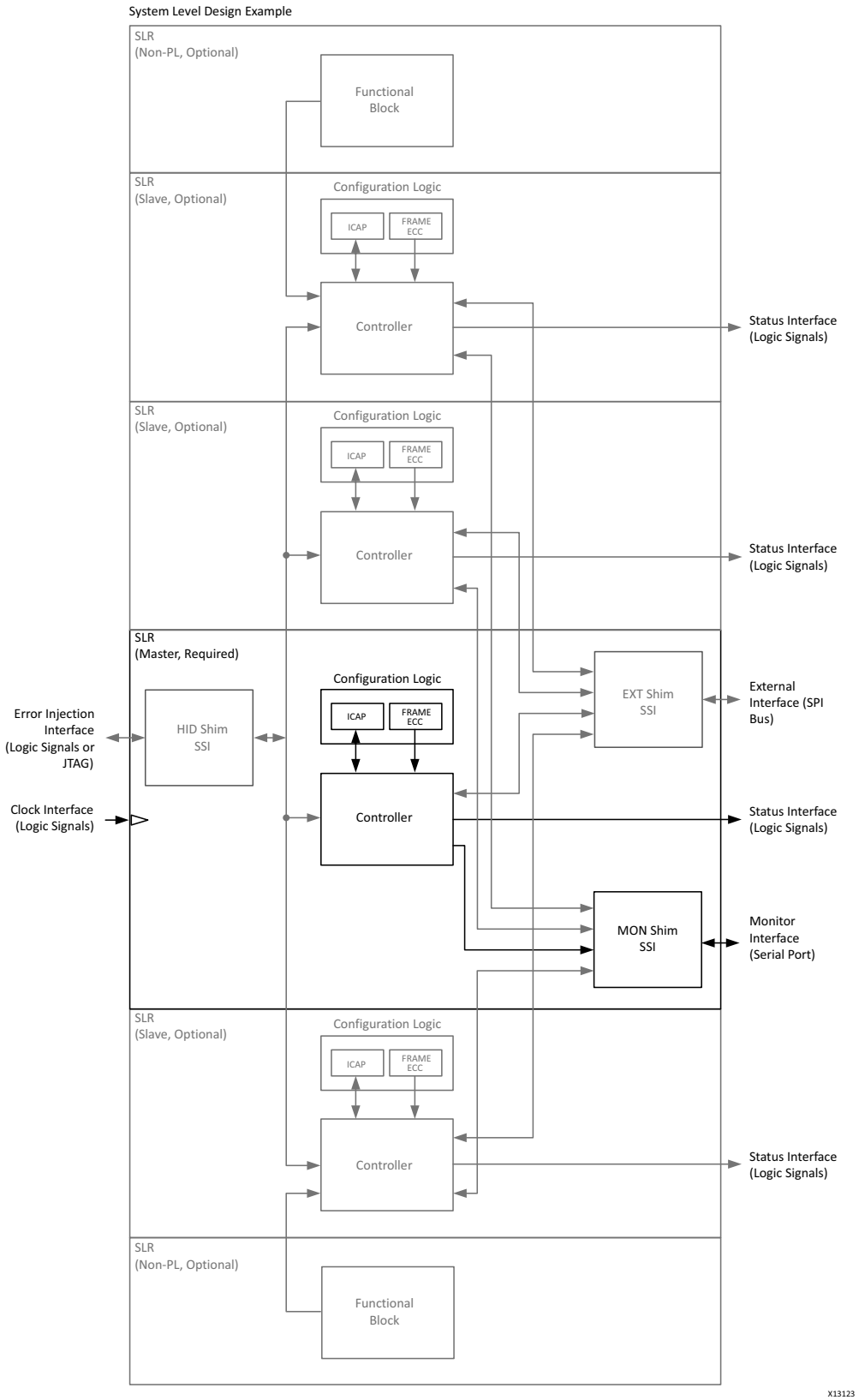

Figure 5-2: **Example Design Block Diagram for SSI Devices**

The system-level design example is provided to allow flexibility in system-level interfacing. To support this goal, the system-level design example is provided as RTL source code, unlike the controller itself.

Port Descriptions

Figure 5-3 shows the example design ports for non-SSI devices. The ports are clustered into six groups. The groups shaded in gray only exist in certain configurations.

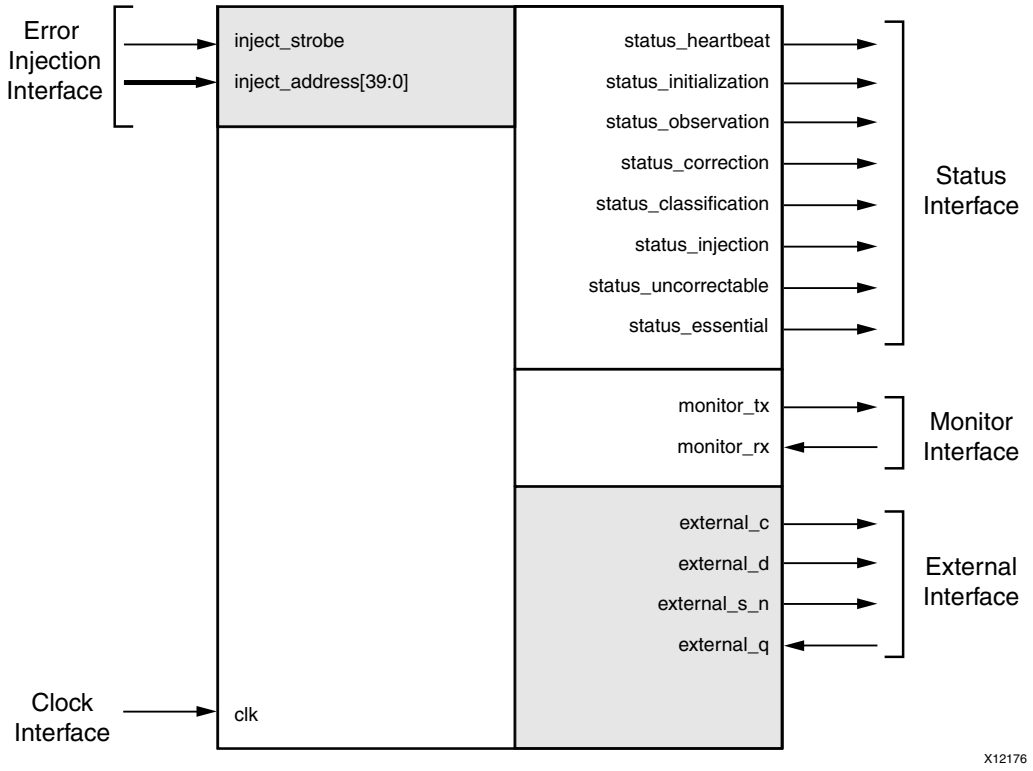

Figure 5-3: **Example Design Ports**

In an SSI device, each super logic region (SLR) is numbered. There are two numbering methods: hardware SLR numbering and software SLR numbering.

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

A controller instance located in an SLR determines the hardware SLR number at runtime by reading the IDCODE register through the internal configuration access port (ICAP) on that 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 SLR Numbers**

Device	Software SLR Number	Hardware SLR Number	SLR Type
XC7VH580T	2	X	GTZ
	1	1	SLAVE
	0	0	MASTER
XC7VH870T	4	X	GTZ
	3	2	SLAVE
	2	0	MASTER
	1	1	SLAVE
	0	X	GTZ
XC7VX1140T	3	3	SLAVE
	2	2	SLAVE
	1	0	MASTER
	0	1	SLAVE
XC7V2000T	3	3	SLAVE
	2	2	SLAVE
	1	0	MASTER
	0	1	SLAVE

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.*

Figure 5-4 shows the example design ports for SSI devices. The ports are clustered into six groups. The groups shaded in gray only exist in certain configurations. In SSI devices, the Status Interface ports become buses, where the bus width is determined by the number of SLRs in the SSI device.

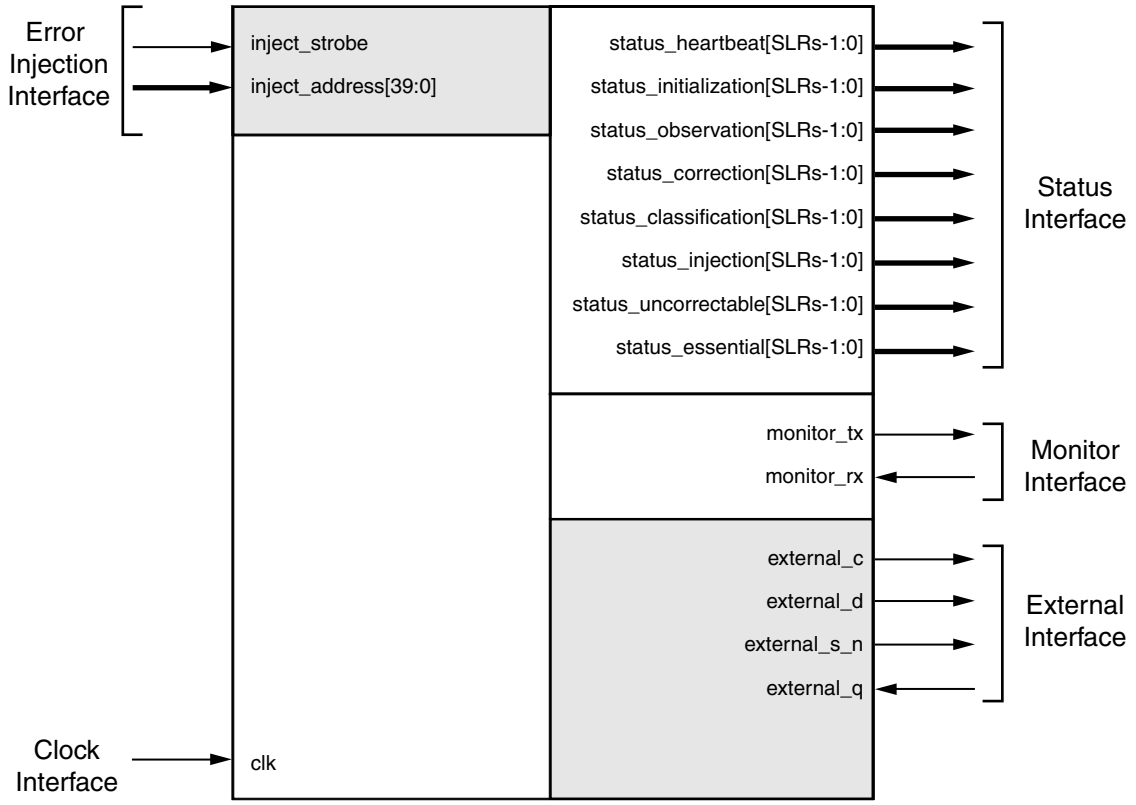

Figure 5-4: **Example Design Ports for SSI Devices**

The system-level design example has no reset input or output. The controller automatically initializes itself. The controller then initializes the shims, 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.

Status Interface

The Status Interface is a direct pass-through from controllers. See Chapter 2, Status Interface for a description of this interface.

Clock Interface

The Clock Interface is used to provide a clock to the system-level design example. Internally, the clock signal is distributed on a global clock buffer to all synchronous logic cells.

Table 5-2: **Clock Interface Details**

Name	Sense	Direction	Description
clk	EDGE	IN	Receives the master clock for the system-level design example.

Monitor Interface

The Monitor Interface is always present. The MON shim in the system-level design example is a UART. This shim serializes status information generated by controllers (a byte stream of ASCII codes) for serial transmission. Similarly, the shim de-serializes command information presented to controllers (a bitstream of ASCII codes) for parallel presentation to controllers.

The shim uses a standard serial communication protocol. The shim contains synchronization and over sampling logic to support asynchronous serial devices that operate at the same nominal baud rate. See Chapter 3, Designing with the Core for additional information.

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 5-3: **Monitor Interface Details**

Name	Sense	Direction	Description
monitor_tx	LOW	OUT	Serial transmit data from system-level design example.
monitor_rx	LOW	IN	Serial receive data to system-level design example.

External Interface

The External Interface is present when controllers require access to external data. When present, the EXT shim in the system-level design example is a fixed-function SPI bus master. This shim accepts commands from controllers that consist of an address and a byte count. The shim generates SPI bus transactions to fetch the requested data from an external SPI flash. The shim formats the returned data for controllers to pick up.

The shim 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 master clock for the system-level design example. See Chapter 3, Designing with the Core for information on external timing budgets.

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 5-4: **External Interface Details**

Name	Sense	Direction	Description
external_c	EDGE	OUT	SPI bus clock for an external SPI flash.
external_d	HIGH	OUT	SPI bus “master out, slave in” signal for an external SPI flash.
external_s_n	LOW	OUT	SPI bus select signal for an external SPI flash.
external_q	HIGH	IN	SPI bus “master in, slave out” signal for an external SPI flash.

Error Injection Interface

The Error Injection Interface is present when controllers support error injection and the HID shim is set to I/O Pins. This interface is bused to all controllers. See Chapter 2, Error Injection Interface for a description of this interface.

When controllers support error injection and the HID shim is set to Vivado Design Suite debug feature, or when error injection is disabled, this interface is absent.

Demonstration Test Bench

A simulation test harness is provided with the example design. This enables functional and timing simulation of designs that include the 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.

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 users 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 or optional correction by replace functions, a Tcl property must be set prior to bitstream generation to enable essential bits.

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

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

TIP: *Xilinx recommends the use of a 64-bit operating system with 16 Gb or more of system RAM.*

Creating the External Memory Programming File for Non-SSI Devices

If the solution requires external data storage to support error classification or error correction by replace, 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
```

If classification and correction by replace are enabled (used in the example outlined in these steps):

```
makedata -ebc <ebc filename> -ebd <ebd filename> datafile
makedata -ebc sem_0_sem_ex.ebc -ebd sem_0_sem_ex.ebd datafile
```

If correction by replace is enabled:

```
makedata -ebc <ebc filename> datafile
makedata -ebc sem_0_sem_ex.ebc datafile
```

If classification is enabled:

```
makedata -ebd <ebd filename> datafile
makedata -ebd sem_0_sem_ex.ebd datafile
```

The command creates the VMF, BIN and MCS files.

Creating the External Memory Programming File for SSI Devices

If the solution requires external data storage to support error classification or error correction by replace, an additional Tcl script is called to post-process special write_bitstream output files into a SPI flash programming file. This `makedata.tcl` script is generated in the example project. After bitstream generation is complete, source this script in the Tcl Console.

```
source <path to component_name>_ex/imports/makedata.tcl
source sem_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 <component_name>_ex/<component_name>_ex.runs/impl_1
cd sem_0_ex/sem_0_ex.runs/impl_1
```

The essential bits files, EBC and EBD files, are created during bitstream generation. A separate EBC and EBD file is generated for each SLR in the target device. Depending on which options are enabled, the following command generates the necessary files. These examples show the general path and the specific instances used in this example flow. The number of files specified after the **`-ebc`** or **`-ebd`** switch corresponds to the number of SLRs in the target device.

If classification and correction by replace are enabled (used in the example outlined in these steps):

```
makedata -ebc <file0> <file1> <file2> <file3> -ebd <file0> <file1> <file2> <file3>
datafile
makedata -ebc sem_0_sem_example_0.ebc sem_0_sem_example_1.ebc
sem_0_sem_example_2.ebc sem_0_sem_example_3.ebc -ebd
sem_0_sem_example_0.ebd sem_0_sem_example_1.ebd
sem_0_sem_example_2.ebd sem_0_sem_example_3.ebd datafile
```

If correction by replace is enabled:

```
makedata -ebc <file0> <file1> <file2> <file3> datafile
makedata -ebc sem_0_sem_example_0.ebc sem_0_sem_example_1.ebc
sem_0_sem_example_2.ebc sem_0_sem_example_3.ebc datafile
```

If classification is enabled:

```
makedata -ebd <file0> <file1> <file2> <file3> datafile
makedata -ebd sem_0_sem_example_0.ebd sem_0_sem_example_1.ebd
sem_0_sem_example_2.ebd sem_0_sem_example_3.ebd datafile
```

The command creates the VMF, BIN and MCS files.

External Memory Programming File

When error correction by replace is enabled, an image of the configuration data is required. When error classification is enabled, an image of the essential bit lookup data is required. As a result, one or both of these data sets might be required. The data sets are identical in size, with their size 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(s) 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 replacement data, byte 0 (least significant byte)
- Byte 1: 32-bit pointer to start of replacement data, byte 1
- Byte 2: 32-bit pointer to start of replacement data, byte 2
- Byte 3: 32-bit pointer to start of replacement data, byte 3 (most significant byte)
- Byte 4: 32-bit pointer to start of essential bit data, byte 0 (least significant byte)
- Byte 5: 32-bit pointer to start of essential bit data, byte 1
- Byte 6: 32-bit pointer to start of essential bit data, byte 2
- Byte 7: 32-bit pointer to start of essential bit data, byte 3 (most significant byte)
- Remaining bytes are reserved, filled with ones

For SSI devices, the table format is:

- Byte 0: 32-bit pointer to start of hardware SLR0 (master) replacement data, byte 0 (least significant byte)
- Byte 1: 32-bit pointer to start of hardware SLR0 (master) replacement data, byte 1
- Byte 2: 32-bit pointer to start of hardware SLR0 (master) replacement data, byte 2
- Byte 3: 32-bit pointer to start of hardware SLR0 (master) replacement data, byte 3 (most significant byte)
- Byte 4: 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 0 (least significant byte)
- Byte 5: 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 1
- Byte 6: 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 2

- Byte 7: 32-bit pointer to start of hardware SLR0 (master) essential bit data, byte 3 (most significant byte)
- Byte 8: 32-bit pointer to start of hardware SLR1 (optional slave) replacement data, byte 0 (least significant byte)
- Byte 9: 32-bit pointer to start of hardware SLR1 (optional slave) replacement data, byte 1
- Byte 10: 32-bit pointer to start of hardware SLR1 (optional slave) replacement data, byte 2
- Byte 11: 32-bit pointer to start of hardware SLR1 (optional slave) replacement data, byte 3 (most significant byte)
- Byte 12: 32-bit pointer to start of hardware SLR1 (optional slave) essential bit data, byte 0 (least significant byte)
- Byte 13: 32-bit pointer to start of hardware SLR1 (optional slave) essential bit data, byte 1
- Byte 14: 32-bit pointer to start of hardware SLR1 (optional slave) essential bit data, byte 2
- Byte 15: 32-bit pointer to start of hardware SLR1 (optional slave) essential bit data, byte 3 (most significant byte)
- Byte 16: 32-bit pointer to start of hardware SLR2 (optional slave) replacement data, byte 0 (least significant byte)
- Byte 17: 32-bit pointer to start of hardware SLR2 (optional slave) replacement data, byte 1
- Byte 18: 32-bit pointer to start of hardware SLR2 (optional slave) replacement data, byte 2
- Byte 19: 32-bit pointer to start of hardware SLR2 (optional slave) replacement data, byte 3 (most significant byte)
- Byte 20: 32-bit pointer to start of hardware SLR2 (optional slave) essential bit data, byte 0 (least significant byte)
- Byte 21: 32-bit pointer to start of hardware SLR2 (optional slave) essential bit data, byte 1
- Byte 22: 32-bit pointer to start of hardware SLR2 (optional slave) essential bit data, byte 2
- Byte 23: 32-bit pointer to start of hardware SLR2 (optional slave) essential bit data, byte 3 (most significant byte)
- Byte 24: 32-bit pointer to start of hardware SLR3 (optional slave) replacement data, byte 0 (least significant byte)

- Byte 25: 32-bit pointer to start of hardware SLR3 (optional slave) replacement data, byte 1
- Byte 26: 32-bit pointer to start of hardware SLR3 (optional slave) replacement data, byte 2
- Byte 27: 32-bit pointer to start of hardware SLR3 (optional slave) replacement data, byte 3 (most significant byte)
- Byte 28: 32-bit pointer to start of hardware SLR3 (optional slave) essential bit data, byte 0 (least significant byte)
- Byte 29: 32-bit pointer to start of hardware SLR3 (optional slave) essential bit data, byte 1
- Byte 30: 32-bit pointer to start of hardware SLR3 (optional slave) essential bit data, byte 2
- Byte 31: 32-bit pointer to start of hardware SLR3 (optional slave) essential bit data, byte 3 (most significant byte)
- Remaining bytes are reserved, filled with ones

A pointer value of 0xFFFFFFFF is used if a particular block of data is not present. The essential bit data and replacement 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

Verification, Compliance, and Interoperability

The controller and example design are verified together, using several methods including an automated hardware test bench and hardware co-simulation tools. The controller and example design are 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 with a methodology that uses a hybrid approach. Testing included an emphasis on hardware verification and was complemented by a co-simulation test bench. The techniques and tools used were:

- Dynamic checks, through a hardware test bench
  - Functional Coverage: Compares design behavior against expected behavior
- Dynamic checks, through a co-simulation test bench
  - Functional Coverage: Compares design behavior against expected behavior
  - Code Coverage: Records execution trace of controller FSM for analysis
- Static checks, through a checking tool suite
  - Linting
  - Clock Domain Crossing

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

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

Validation

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

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

Conformance Testing

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

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

Upgrading

This appendix contains information about upgrading to a more recent version of the IP core.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 12].

Upgrading in the Vivado Design Suite

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

Parameter Changes

The injection shim 'chipscope' option has been replaced by the 'vivado lab tools' option. If ‘Vivado lab tools’ is selected as the injection shim, a VIO LogiCORE IP will be generated and included with the SEM v4.1 IP example design.

Port Changes

There are no port changes for this version.

Debugging

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

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the SEM core, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support Web Case.

Documentation

This product guide is the main document associated with the SEM core. 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 SEM Core

AR: 54642

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 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 6].

Reference Boards

Various Xilinx development boards support the SEM core. These boards can be used to prototype designs and establish that the core can communicate with the system.

- Xilinx evaluation boards:
  - AC701
  - KC705
  - VC707
  - ZC702
  - ZC706

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.

Review Additional Considerations in Chapter 3 to ensure that no unsupported settings or features have been used.

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 97.*

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 MON shim in the system-level design example is a UART that can be connected to a standard RS232 port, or to USB through a USB-to-UART bridge.

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:

```
X7_SEM_V4_1
SC 01
FS 02
ICAP OK
RDBK OK
INIT OK
SC 02
O>
```

The first line lists the device and core version. The second 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 MON shim is used, and the initialization report appears scrambled or garbage characters appear, verify that the terminal program communication settings match those listed in Monitor Interface in Chapter 3. Also verify that the frequency of the actual clock provided to the SEM controller, coupled with the MON shim V_ENABLETIME parameter value, yield a standard baud rate and that the terminal program communication settings match the bit rate. This is described in Equation 3-1 and Equation 3-2.

If the SEM controller cannot achieve communication with the FPGA configuration logic through the internal configuration access port (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 looks like this:

```
X7_SEM_V4_1
SC 01
FS 02
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.
  - Use the example design to get the correct instantiation.
  - For implementations in Zynq-7000 devices, review the hardware and software control of the SEM controller `icap_grant` input.
- 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 method documented in XAPP517 is used. Never add pipelining between the SEM controller and the ICAP.

As documented in Additional Considerations in Chapter 3, the SEM controller is not compatible 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.

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 DCM is very small, it is best to strive for the highest reliability possible. However, if a PLL or DCM 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/DCM 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/DCM lock is achieved.

Appendix D

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. *7 Series FPGAs Configuration User Guide (*UG470)
2. *Xilinx Device Reliability Report (*UG116)
3. *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585)
4. *Zynq-7000 EPP Software Developers Guide* (UG821)
5. *Xilinx OS and Libraries Document Collection* (UG643)
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
7. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Vivado Design Suite User Guide: Getting Started* (UG910)
10. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
11. *Vivado Design Suite User Guide: Implementation* (UG904)
12. *ISE to Vivado Design Suite Migration Methodology Guide* (UG911)
13. *Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits* (WP286)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/04/2018	4.1	• Added Key Considerations for SEM IP Adoption section in Overview chapter. • Updated to remove obsolete method of calculating FIT in Solution Reliability section in Product Specification chapter.
04/05/2017	4.1	• Added Encryption and Authentication Support section. • Added Spartan-7 data to ICAP Maximum Frequencies to Maximum Device Scan Times at ICAP FMax tables. • Added Zynq data to Resource Utilization for Zynq-7000 Devices table. • Added Artix-7 data to Resource Utilization for Artix-7 Devices table. • Added Resource Utilization for Spartan-7 Devices table. • Added Artix, Spartan, and Zynq data for External Storage Requirements table. • Added AR to Additional Considerations section. • Updated code examples for Creating the External Memory Programming File for Non-SSI Devices and Creating the External Memory Programming File for SSI Devices sections. • Updated description in Interface Debug section.
09/30/2015	4.1	• Updated Maximum Start-Up Latency at ICAP Fmax table. • Added User Parameter and Integration and Validation sections in Design Flow Steps chapter.
11/19/2014	4.1	• Added resource utilization and performance metrics for new Artix-7 (XC7A15T) and Zynq-7000 (XC7Z035) devices. • Updated text to indicate bitstream encryption is supported. • Corrected status_heartbeat specification to indicate a maximum of 150 cycles between heartbeat pulses.
04/02/2014	4.1	• Updated core to v4.1. • Replaced legacy ChipScope support with Vivado Lab Tools support. • Added resource utilization and performance metrics for new Artix-7 and Zynq-7000 devices. • Updated instructions to execute makedata.tcl script. • Added Synthesis and Implementation section.

Date	Version	Revision
06/19/2013	4.0	• Updated instructions to execute makedata.tcl script in Implementation in Chapter 5. • Updated resource utilization for Virtex-7 SSI devices in Resource Utilization in Chapter 2. • Added solution latencies for Zynq-7000 7Z100 devices in Solution Latency in Chapter 2. • Added more information about possible system-level supervisory functions in Chapter 3, Designing with the Core.
03/20/2013	4.0	• Updated core to v4.0. • Removed support for ISE Design Suite. • Increased the maximum clock frequency to 100 MHz for select 7 series devices.
12/18/2012	3.0	• Updated core to v3.4, ISE Design Suite to v14.4 and Vivado Design Suite to v2012.4. • Added pre-production support for Zynq-7000, Artix-7, and Virtex-7 SSI devices. • Redesigned the Spartan-6 solution based on new guidance for configuration readback.
07/25/2012	2.0	Added support for Vivado Design Suite.
04/24/2012	1.0	Initial Xilinx release. Replaces DS796, *LogiCORE IP Soft Error Mitigation Controller Data Sheet,* and UG764, *LogiCORE IP Soft Error Mitigation Controller User Guide*.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.