

# LogiCORE IP Soft Error Mitigation Controller v3.2

## *Product Guide*

PG036 April 24, 2012

# Table of Contents

---

## Chapter 1: Overview

Memory Types .....	6
Mitigation Approaches .....	7
Reliability Estimation.....	8
Feature Summary.....	9
Applications .....	9
Unsupported Features.....	10
Licensing.....	10

---

## Chapter 2: Product Specification

Features .....	11
Standards Compliance.....	12
Resource Utilization.....	12
Performance.....	15
Port Descriptions .....	30

---

## Chapter 3: Customizing and Generating the Core

Creating a Project in ISE Design Suite.....	36
Customizing and Generating the Core in ISE Design Suite .....	37
Output Generation.....	42
Generating and Using ChipScope Tool Files.....	45

---

## Chapter 4: Example Design

Functions .....	48
Port Descriptions .....	50
Simulation .....	52
Demonstration Test Bench .....	52
Implementation in ISE Design Suite .....	53

External Memory Programming File.....	54
Directory and File Contents.....	55

---

## Chapter 5: Designing with the Core

Interfaces.....	56
Behaviors.....	75
Systems.....	87
Customizations.....	88
Data Consistency.....	94
Configuration Memory Masking.....	94
Clocking.....	96
Resets.....	96
Additional Considerations.....	97

---

## Chapter 6: Constraining the Core

Required Constraints.....	99
Contents of the User Constraints File.....	99
Device, Package, and Speed Grade.....	104
Clock Frequency.....	104
Clock Management.....	104
Clock Placement.....	104
I/O Pins.....	104

---

## Appendix A: Verification, Compliance, and Interoperability

Verification.....	105
Validation.....	106
Conformance Testing.....	106

---

## Appendix B: Migrating

Customization and Generation Changes.....	107
Port Changes.....	107
Functionality Changes.....	107

---

## Appendix C: Debugging

Release Notes.....	108
Monitor Interface.....	108
Clocking.....	109

---

## Appendix D: Additional Resources

Xilinx Resources.....	110
Solution Centers.....	110
References.....	110
Technical Support.....	111
Ordering Information.....	111
Revision History.....	111
Notice of Disclaimer.....	112

## 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 (7 series devices only).
  - Correction by replace method is data re-load based (Virtex®-6 and 7 series devices only).
- 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	
<b>Core Specifics</b>	
Supported Device Family <sup>(1)</sup>	Kintex-7, Virtex-7 <sup>(2)</sup> , Virtex-6, Spartan-6
Supported User Interfaces	RS-232, SPI
Resources	See <a href="#">Table 2-1</a> through <a href="#">Table 2-4</a>
<b>Provided with Core</b>	
Design Files	NGC
Example Design	VHDL and Verilog
Test Bench	Not Applicable <sup>(3)</sup>
Constraints File	UCF
Simulation Model	Not Applicable <sup>(3)</sup>
<b>Tested Design Tools</b>	
Design Entry Tools	CORE Generator™ tool v14.1
Simulation <sup>(4)</sup>	Not Applicable <sup>(3)</sup>
Synthesis Tools <sup>(4)</sup>	Synopsys Synplify Pro Xilinx XST
<b>Support</b>	
Provided by Xilinx @ <a href="http://www.xilinx.com/support">www.xilinx.com/support</a>	

### Notes:

1. For a complete listing of supported devices, see the [release notes](#) for this core.
2. Excluding SSI devices.
3. 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.
4. For the supported versions of the tools, see the [ISE Design Suite 14: 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 may 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 relevant FPGA Configuration User Guide for information on how to use the built-in error detection and correction capability.

---

## Reliability Estimation

As a starting point, a designer's 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 UG116, *Device Reliability Report*. The data reveals the overall infrequency of soft errors. It is important to note that the failure rates involved are so small that most designs need 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/Mbit 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 in the Xilinx CORE Generator™.

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.

---

## 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.

Unsupported features and specific limitations include functional, implementation, and use considerations. For more details, see [Additional Considerations in Chapter 5](#).

---

## Licensing

This Xilinx LogiCORE IP module is provided at no additional cost with the ISE Design Suite software under the terms of the [Xilinx End User License](#). The core may be accessed through the ISE CORE Generator IP catalog.

Information about this and other Xilinx LogiCORE IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx LogiCORE modules and software, please contact your [local Xilinx sales representative](#).

# 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 (7 series devices only).
  - 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 (Virtex-6 and 7 series devices only).
- 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.

## Resource Utilization

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

*Table 2-1: Resource Utilization for Kintex-7 Devices<sup>ab</sup>*

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Kintex-7 All Devices	Complete solution with no optional features	433	322	11	3 RAMB18
Kintex-7 XC7K70T	Complete solution with all optional features	835	506	56	3 RAMB18, 3 RAMB36
Kintex-7 XC7K160T	Complete solution with all optional features	869	508	56	3 RAMB18, 5 RAMB36
Kintex-7 XC7K325T	Complete solution with all optional features	974	510	56	3 RAMB18, 9 RAMB36
Kintex-7 XC7K355T	Complete solution with all optional features	975	510	56	3 RAMB18, 10 RAMB36
Kintex-7 XC7K410T	Complete solution with all optional features	1005	510	56	3 RAMB18, 11 RAMB36
Kintex-7 XC7K420T	Complete solution with all optional features	1005	510	56	3 RAMB18, 12 RAMB36

**Table 2-1: Resource Utilization for Kintex-7 Devices<sup>ab</sup> (Cont'd)**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Kintex-7 XC7K480T	Complete solution with all optional features	1040	510	56	3 RAMB18, 13 RAMB36
Kintex-7 -2L, Kintex-7Q, Kintex-7Q -2L, All Devices	Same as Kintex-7				

a. The complete solution is the SEM Controller and the example design, which are intended to be used together.

b. The Error Injection Interface is connected to I/Os; use of ChipScope increases LUTs/FFs but decreases I/Os.

**Table 2-2: Resource Utilization for Virtex-7 Devices<sup>ab</sup>**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Virtex-7 All Devices	Complete solution with no optional features	433	322	11	3 RAMB18
Virtex-7 XC7VX330T	Complete solution with all optional features	975	510	56	3 RAMB18, 10 RAMB36
Virtex-7 XC7VX415T	Complete solution with all optional features	1005	510	56	3 RAMB18, 12 RAMB36
Virtex-7 XC7VX485T	Complete solution with all optional features	1074	510	56	3 RAMB18, 14 RAMB36
Virtex-7 XC7VX550T	Complete solution with all optional features	1043	510	56	3 RAMB18, 16 RAMB36
Virtex-7 XC7V585T	Complete solution with all optional features	1042	510	56	3 RAMB18, 15 RAMB36
Virtex-7 XC7VX690T	Complete solution with all optional features	1179	512	56	3 RAMB18, 20 RAMB36
Virtex-7 XC7VX980T	Complete solution with all optional features	1317	512	56	3 RAMB18, 26 RAMB36
Virtex-7 -2L, Virtex-7Q, Virtex-7Q -2L All Devices	Same as Virtex-7				

a. The complete solution is the SEM Controller and the example design, which are intended to be used together.

b. The Error Injection Interface is connected to I/Os; use of ChipScope increases LUTs/FFs but decreases I/Os.

**Table 2-3: Resource Utilization for Virtex-6 Devices<sup>ab</sup>**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Virtex-6 All Devices	Complete solution with no optional features	402	315	11	3 RAMB18
Virtex-6 All Devices	Complete solution with all optional features	459	383	52	3 RAMB18
Virtex-6 -1L, Virtex-6Q, Virtex-6Q -1L All Devices	Same as Virtex-6				

a. The complete solution is the SEM Controller and the example design, which are intended to be used together.

b. The Error Injection Interface is connected to I/Os; use of ChipScope increases LUTs/FFs but decreases I/Os.

**Table 2-4: Resource Utilization for Spartan-6 Devices<sup>ab</sup>**

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Spartan-6 XC6SLX4	Complete solution with no optional features	764	380	11	4 RAMB16, 1 RAMB8
	Complete solution with all optional features	810	442	52	4 RAMB16, 1 RAMB8
Spartan-6 XC6SLX9	Complete solution with no optional features	764	380	11	4 RAMB16, 1 RAMB8
	Complete solution with all optional features	810	442	52	4 RAMB16, 1 RAMB8
Spartan-6 XC6SLX16	Complete solution with no optional features	764	380	11	6 RAMB16
	Complete solution with all optional features	810	442	52	6 RAMB16
Spartan-6 XC6SLX25(T)	Complete solution with no optional features	782	380	11	7 RAMB16
	Complete solution with all optional features	834	446	52	7 RAMB16
Spartan-6 XC6SLX45(T)	Complete solution with no optional features	782	380	11	10 RAMB16
	Complete solution with all optional features	834	446	52	10 RAMB16
Spartan-6 XC6SLX75(T)	Complete solution with no optional features	764	380	11	15 RAMB16
	Complete solution with all optional features	816	446	52	15 RAMB16
Spartan-6 XC6SLX100(T)	Complete solution with no optional features	782	380	11	18 RAMB16
	Complete solution with all optional features	834	446	52	18 RAMB16

Table 2-4: Resource Utilization for Spartan-6 Devices<sup>ab</sup> (Cont'd)

Device	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Spartan-6 XC6SLX150(T)	Complete solution with no optional features	795	380	11	24 RAMB16
	Complete solution with all optional features	848	446	52	24 RAMB16
Spartan-6 -1L, Spartan-6Q, Spartan-6Q -1L, Automotive Spartan-6 All Devices	Same as Spartan-6				

a. The complete solution is the SEM Controller and the example design, which are intended to be used together.

b. The Error Injection Interface is connected to I/Os; use of ChipScope increases LUTs/FFs but decreases I/Os.

## Performance

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

## Solution Reliability

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

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

## Estimation Data

Xilinx device FIT data is reported in UG116, *Device Reliability Report*. [Table 2-5](#) provides example data for a sample reliability estimation.

**Note:** The data in Table 2-5 is an example. This example data is for illustrative purposes only and must not be used in critical design decisions. See UG116 for current device FIT data.

**Table 2-5: Example Device FIT Data**

Device	Memory Cell Type	Real Time Soft Error Rate FIT/Mbit
7 Series FPGAs	Configuration Memory	100
	Block Memory	100
	Distributed Memory (same as Configuration Memory)	100
	Flip-flops	Unspecified
Virtex-6 FPGAs	Configuration Memory	105
	Block Memory	250
	Distributed Memory (same as Configuration Memory)	105
	Flip-flops	Unspecified
Spartan-6 FPGAs	Configuration Memory	186
	Block Memory	391
	Distributed Memory (same as Configuration Memory)	186
	Flip-flops	Unspecified

Table 2-6 provides an approximate relationship between resources and the number of configuration memory cells associated with each resource.

**Table 2-6: Configuration Bits Per Device Feature**

Device	Device Feature (Includes Routing)	Approximate Number of Configuration Bits
7 Series FPGAs	Logic Slice	1,166
	Block RAM (36Kb)	9,396
	Block RAM (18Kb)	4,698
	I/O Block	2,850
Virtex-6 FPGAs	Logic Slice	1,166
	Block RAM (36Kb)	9,396
	Block RAM (18Kb)	4,698
	I/O Block	2,850
Spartan-6 FPGAs	Logic Slice	1,166
	Block RAM (18Kb)	4,698
	Block RAM (9Kb)	2,349
	I/O Block	2,850

Typically fewer than 10% of configuration memory cells directly impact the active design if a soft error occurs. Therefore, the sample reliability estimation uses a 10% de-rating factor.



## Sample 7 Series Reliability Estimation

The controller and shims use approximately 250 logic slices, 56 I/O blocks, 3 block RAM (18 Kb), and 9 block RAM (36 Kb) in a mid-size XC7K325T device, with all optional features enabled. Consider the configuration bit contribution:

$$\text{Config FIT} = 10\% * (250 * 1,166 + 56 * 2,850 + 3 * 4,698 + 9 * 9,396) * 100 \text{ FIT/Mbit}$$

$$\text{Config FIT} = 5.2 \text{ FIT}$$

The controller and shims use several hundred flip-flops for data, their contribution is ignored due to the small number of bits.

The controller and shims use 65 LUT RAM. The usage breakdown is as follows:

- The MON shim uses 31 LUT RAM for data buffering, but the buffers are generally empty and data corruption not observable. These memory bits are therefore ignored.
- The controller uses 34 LUT RAM for data storage. Errors in used locations are highly likely halt the controller. Approximately 256 memory bits are used.

$$\text{LUT RAM FIT} = 100\% * 256 * 100 \text{ FIT/Mbit}$$

$$\text{LUT RAM FIT} = 0.02 \text{ FIT}$$

The controller uses three block RAM (18 Kb) and nine block RAM (36 Kb). The usage breakdown is:

- An internal buffer uses one block RAM. In the data array, 9600 bits are allocated to data buffers used in correction and classification; a soft error here would only cause potential issue if it occurred *during* mitigation activity. No permanent data resides here; these are therefore ignored. Another 7480 bits are allocated to constant storage; errors in these locations are highly likely to break the controller and must be considered in the analysis. The remaining 1352 bits are unused.
- The controller firmware resides in two block RAMs. The word count is approximately 1932 out of 2048, with at least 336 of the used words only executed one time at system start and therefore ignored. The number of bits considered for the analysis is 28728.

$$\text{Block RAM FIT} = 100\% * 36208 * 100 \text{ FIT/Mbit}$$

$$\text{Block RAM FIT} = 3.5 \text{ FIT}$$

- The enhanced repair algorithm within the controller stores frame-level CRCs in the remaining block RAM (36 Kb). These block RAM contents are protected by built-in ECC of the block RAM, and do contribute to the Block RAM Bit FIT. As computed above, these block RAMs do contribute to the Configuration Bit FIT

The total controller FIT is then:

$$5.2 \text{ FIT} + 0.02 \text{ FIT} + 3.5 \text{ FIT} \approx 8.7 \text{ FIT}$$

### Sample Virtex-6 Reliability Estimation

The controller and shims use approximately 148 logic slices, 52 I/O blocks, and 3 block RAM (18 Kb). Consider the configuration bit contribution:

$$\text{Config FIT} = 10\% * (148 * 1,166 + 52 * 2,850 + 3 * 4,698) * 105 \text{ FIT/Mbit}$$

$$\text{Config FIT} = 3.4 \text{ FIT}$$

The controller and shims use several hundred flip-flops for data, their contribution is ignored due to the small number of bits.

The controller and shims use 65 LUT RAM. The usage breakdown is as follows:

- The MON shim uses 31 LUT RAM for data buffering, but the buffers are generally empty and data corruption not observable. These memory bits are therefore ignored.
- The controller uses 34 LUT RAM for data storage. Errors in used locations are highly likely halt the controller. Approximately 256 memory bits are used.

$$\text{LUT RAM FIT} = 100\% * 256 * 105 \text{ FIT/Mbit}$$

$$\text{LUT RAM FIT} = 0.03 \text{ FIT}$$

The controller uses three block RAM (18 Kb). The usage breakdown is:

- An internal buffer uses one block RAM. In the data array, 10368 bits are allocated to data buffers used in correction and classification; a soft error here would only cause potential issue if it occurred *during* mitigation activity. No permanent data resides here; these are therefore ignored. Another 7552 bits are allocated to constant storage; errors in these locations are highly likely to break the controller and must be considered in the analysis. The remaining 512 bits are unused.
- The controller firmware resides in two block RAMs. The word count is approximately 1550 out of 2048, with at least 150 of the used words only executed one time at system start and therefore ignored. The number of bits considered for the analysis is 25200.

$$\text{Block RAM FIT} = 100\% * 32752 * 250 \text{ FIT/Mbit}$$

$$\text{Block RAM FIT} = 7.9 \text{ FIT}$$

The total controller FIT is then:

$$3.4 \text{ FIT} + 0.03 \text{ FIT} + 7.9 \text{ FIT} \approx 11.3 \text{ FIT}$$

## Sample Spartan-6 Reliability Estimation

The controller and shims use approximately 249 logic slices, 52 I/O blocks, and 10 block RAM (18 Kb) in a mid-size XC6SLX45T device. Consider the configuration bit contribution:

$$\text{Config FIT} = 10\% * (249 * 1,166 + 52 * 2,850 + 10 * 4,698) * 186 \text{ FIT/Mbit}$$

$$\text{Config FIT} = 8.7 \text{ FIT}$$

The controller and shims use several hundred flip-flops for data, their contribution is ignored due to the small number of bits.

The controller and shims use 70 LUT RAM. The usage breakdown is as follows:

- The MON shim uses 36 LUT RAM for data buffering, but the buffers are generally empty and data corruption not observable. These memory bits are therefore ignored.
- The controller uses 34 LUT RAM for data storage. Errors in used locations are highly likely halt the controller. Approximately 256 memory bits are used.

$$\text{LUT RAM FIT} = 100\% * 256 * 186 \text{ FIT/Mbit}$$

$$\text{LUT RAM FIT} = 0.05 \text{ FIT}$$

The controller uses ten block RAM (18 Kb). The usage breakdown is:

- An internal buffer uses one block RAM. In the data array, 3268 bits are allocated to data buffers used in correction and classification; a soft error here would only cause potential issue if it occurred during mitigation activity. No permanent data resides here; these are therefore ignored. Another 6812 bits are allocated to constant storage; errors in these locations are highly likely to break the controller and must be considered in the analysis. The remaining 8352 bits are unused.
- The controller firmware resides in two block RAMs. The word count is approximately 1732 out of 2048, with at least 444 of the used words only executed one time at system start or as debug and therefore ignored. The number of bits considered for the analysis is 23184.
- The soft logic FRAME ECC module within the controller stores the ECC checksums in the remaining block RAM. These block RAM contents are protected by the FRAME ECC, and do not contribute to the Block RAM Bit FIT. As computed above, these block RAMs do contribute to the Configuration Bit FIT.

$$\text{Block RAM FIT} = 100\% * (6812+23184) * 391 \text{ FIT/Mbit}$$

$$\text{Block RAM FIT} = 11.2 \text{ FIT}$$

The total controller FIT is then:

$$8.7 \text{ FIT} + 0.05 \text{ FIT} + 11.2 \text{ FIT} \approx 19.9 \text{ FIT}$$

## Maximum Frequency

The maximum frequency of operation of the SEM Controller is not guaranteed. In no case may the maximum frequency of operation exceed the ICAP Fmax specified in the relevant FPGA data sheet as configuration interface AC timing parameter Frbck. [Table 2-7](#) provides a summary of ICAP Fmax values.

**Table 2-7: ICAP Maximum Frequency**

	Device	ICAP F <sub>MAX</sub>
7 Series FPGAs	Kintex-7	100 MHz <sup>a</sup>
	Kintex-7 -2L	70 MHz
	Kintex-7Q	100 MHz <sup>a</sup>
	Kintex-7Q -2L	70 MHz
	Virtex-7	100 MHz <sup>a</sup>
	Virtex-7 -2L	70 MHz
	Virtex-7Q	100 MHz <sup>a</sup>
	Virtex-7Q -2L	70 MHz
Virtex-6 FPGAs	Virtex-6	100 MHz
	Virtex-6 -1L	60 MHz
	Virtex-6Q	100 MHz
	Virtex-6Q -1L	60 MHz
Spartan-6 FPGAs	Spartan-6 XC6SLX4 to XC6SLX75(T)	50 MHz
	Spartan-6 XC6SLX100(T) to XC6SLX150(T)	35 MHz
	Spartan-6 -1L XC6SLX4 to XC6SLX75(T)	30 MHz
	Spartan-6 -1L XC6SLX100(T) to XC6SLX150(T)	20 MHz
	Spartan-6Q XQ6SLX4 to XQ6SLX75(T)	50 MHz
	Spartan-6Q XQ6SLX100(T) to XQ6SLX150(T)	35 MHz
	Spartan-6Q -1L XQ6SLX4 to XQ6SLX75(T)	30 MHz
	Spartan-6Q -1L XQ6SLX100(T) to XQ6SLX150(T)	20 MHz
	Automotive Spartan-6 XA6SLX4 to XA6SLX75(T)	50 MHz
	Automotive Spartan-6 XA6SLX100(T) to XA6SLX150(T)	35 MHz

a. CORE Generator maximum clock frequency is 70 MHz. See [SEM core release notes](#) for more information.

Other maximum frequency limitations may apply. For more details on determining the maximum frequency of operation for the SEM Controller, see [Interfaces in Chapter 5](#).

## 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 2-8](#) 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 2-8: Start-Up Latency**

Device		Boot Time at ICAP Fmax	Initialization Time at ICAP Fmax (Repair / Replace)	Initialization Time at ICAP Fmax (Enhanced Repair)
7 Series	XC7K70T	110 ms	17.8 ms	2.1 s
	XC7K160T	110 ms	38.8 ms	4.6 s
	XC7K325T	110 ms	71.0 ms	8.5 s
	XC7K355T	110 ms	79.9 ms	9.6 s
	XC7K410T	110 ms	91.5 ms	10.9 s
	XC7K420T	110 ms	93.3 ms	11.1 s
	XC7K480T	110 ms	106.6 ms	12.7 s
	XC7VX330T	110 ms	77.3 ms	9.2 s
	XC7VX415T	110 ms	98.1 ms	11.7 s
	XC7VX485T	110 ms	115.7 ms	13.8 s
	XC7VX550T	110 ms	130.8 ms	15.6 s
	XC7V585T	110 ms	124.4 ms	14.9 s
	XC7VX690T	110 ms	163.5 ms	19.5 s
	XC7VX980T	110 ms	213.3 ms	25.5 s
Virtex-6	XC6VCX75T	110 ms	18.6 ms	N/A
	XC6VCX130T	110 ms	31.2 ms	N/A
	XC6VCX195T	110 ms	45.0 ms	N/A
	XC6VCX240T	110 ms	54.0 ms	N/A
	XC6VHX250T	110 ms	56.1 ms	N/A
	XC6VHX255T	110 ms	56.1 ms	N/A
	XC6VHX380T	110 ms	84.0 ms	N/A
	XC6VHX565T	110 ms	117.9 ms	N/A
	XC6VLX75T	110 ms	18.6 ms	N/A
	XC6VLX130T	110 ms	31.2 ms	N/A
	XC6VLX195T	110 ms	45.0 ms	N/A
	XC6VLX240T	110 ms	54.0 ms	N/A
	XC6VLX365T	110 ms	75.6 ms	N/A
	XC6VLX550T	110 ms	113.1 ms	N/A
	XC6VLX760	110 ms	149.7 ms	N/A
	XC6VSX315T	110 ms	72.3 ms	N/A
XC6VSX475T	110 ms	108.3 ms	N/A	

Table 2-8: Start-Up Latency (Cont'd)

Device		Boot Time at ICAP Fmax	Initialization Time at ICAP Fmax (Repair / Replace)	Initialization Time at ICAP Fmax (Enhanced Repair)
Spartan-6	XC6SLX4	110 ms	8.1 ms	N/A
	XC6SLX9	110 ms	8.1 ms	N/A
	XC6SLX16	110 ms	11.7 ms	N/A
	XC6SLX25(T)	110 ms	19.8 ms	N/A
	XC6SLX45(T)	110 ms	35.7 ms	N/A
	XC6SLX75(T)	110 ms	60.0 ms	N/A
	XC6SLX100(T)	110 ms	113.1 ms	N/A
	XC6SLX150(T)	110 ms	151.8 ms	N/A

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 2-8](#) and [Equation 2-1](#).

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

### 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 2-9](#) illustrates full device scan times.

Table 2-9: Device Scan Times at ICAP Maximum Frequency

	Device	Scan Time at ICAP $F_{MAX}$
7 Series	XC7K70T	5.9 ms
	XC7K160T	12.9 ms
	XC7K325T	23.5 ms
	XC7K355T	26.5 ms
	XC7K410T	30.3 ms
	XC7K420T	30.9 ms
	XC7K480T	35.3 ms
	XC7VX330T	25.6 ms
	XC7VX415T	32.5 ms
	XC7VX485T	38.3 ms
	XC7VX550T	43.3 ms
	XC7V585T	41.2 ms
	XC7VX690T	54.1 ms
	XC7VX980T	70.7 ms
Virtex-6	XC6VCX75T	6.2 ms
	XC6VCX130T	10.4 ms
	XC6VCX195T	15.0 ms
	XC6VCX240T	18.0 ms
	XC6VHX250T	18.7 ms
	XC6VHX255T	18.7 ms
	XC6VHX380T	28.0 ms
	XC6VHX565T	39.3 ms
	XC6VLX75T	6.2 ms
	XC6VLX130T	10.4 ms
	XC6VLX195T	15.0 ms
	XC6VLX240T	18.0 ms
	XC6VLX365T	25.2 ms
	XC6VLX550T	37.7 ms
	XC6VLX760	49.9 ms
	XC6VSX315T	24.1 ms
XC6VSX475T	36.1 ms	



Table 2-9: Device Scan Times at ICAP Maximum Frequency (Cont'd)

	Device	Scan Time at ICAP F <sub>MAX</sub>
Spartan-6	XC6SLX4	2.7 ms
	XC6SLX9	2.7 ms
	XC6SLX16	3.9 ms
	XC6SLX25(T)	6.6 ms
	XC6SLX45(T)	11.9 ms
	XC6SLX75(T)	20.0 ms
	XC6SLX100(T)	37.7 ms
	XC6SLX150(T)	50.6 ms

The device scan time for the target device, at the actual frequency of operation, can be estimated using data from Table 2-9 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<sub>ACTUAL</sub>
- Maximum error detection latency for detection by ECC is Scan Time<sub>ACTUAL</sub>
- Absolute maximum error detection latency for detection by CRC alone is 2.0 \* Scan Time<sub>ACTUAL</sub>

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 2-10](#) provides error correction latency assuming no throttling on the Monitor Interface.

**Table 2-10: Error Correction Latency, No Throttling on Monitor Interface**

Device Family	Correction Mode	Errors in Frame (Correctability)	Error Correction State at ICAP_F <sub>MAX</sub>
7 Series FPGAs	Repair	1-bit (Correctable)	610 μs
		2-bit (Uncorrectable)	20 μs
	Enhanced Repair	1-bit (Correctable)	610 μs
		2-bit (Correctable)	18750 μs
		2-bit (Uncorrectable)	9110 μs
	Replace	Any (Correctable)	830 μs
	Any	CRC-only (Uncorrectable)	10 μs
Virtex-6 FPGAs	Repair	1-bit (Correctable)	490 μs
		2-bit (Uncorrectable)	20 μs
	Replace	Any (Correctable)	660 μs
	Any	CRC-only (Uncorrectable)	10 μs
Spartan-6 LX4 through LX75(T)	Repair	1-bit (Correctable)	370 μs
		2-bit (Uncorrectable)	35 μs
	Any	CRC-only (Uncorrectable)	15 μs
Spartan-6 LX100(T) through LX150(T)	Repair	1-bit (Correctable)	525 μs
		2-bit (Uncorrectable)	50 μs
	Any	CRC-only (Uncorrectable)	25 μs

a. 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 2-10](#) 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 2-11](#) provides error classification latency assuming no throttling on the Monitor Interface.

**Table 2-11: Error Classification Latency, No Throttling on Monitor Interface**

Device Family	Correction Mode	Errors in Frame (Correctability)	Classification Mode	Error Classification State at ICAP_F <sub>MAX</sub>
7 Series FPGAs	Any	Correctable	Enabled	750 μs
	Any	Uncorrectable	Disabled	10 μs
	Any	Uncorrectable	Any	10 μs
Virtex-6 FPGAs	Any	Correctable	Enabled	610 μs
	Any	Correctable	Disabled	10 μs
	Any	Uncorrectable	Any	10 μs
Spartan-6 LX4 through LX75(T)	Repair	Correctable	Enabled	435 μs
		Correctable	Disabled	10 μs
		Uncorrectable	Any	10 μs
Spartan-6 LX100(T) through LX150(T)	Repair	Correctable	Enabled	620 μs
		Correctable	Disabled	15 μs
	Any	Uncorrectable	Any	15 μs

The error classification latency at the actual frequency of operation can be estimated using data from [Table 2-11](#) 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 XC6VLX240T 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.

$$DetectionLatency = 0.5 \cdot ScanTime_{ACTUAL} = 0.5 \cdot 18.0ms \cdot \left[ \frac{100MHz}{66MHz} \right] = 13.636ms \quad \text{Equation 2-6}$$

$$CorrectionLatency = 490\mu s \cdot \left[ \frac{100MHz}{66MHz} \right] = 0.742ms \quad \text{Equation 2-7}$$

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

$$MitigationLatency = 13.636ms + 0.742ms + 0.015ms = 14.393ms \quad \text{Equation 2-9}$$

The second sample estimation illustrates the calculation of error mitigation latency for a two-bit error by the solution implemented in an XC6VLX240T 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.

$$DetectionLatency = 0.5 \cdot ScanTime_{ACTUAL} = 0.5 \cdot 18.0ms \cdot \left[ \frac{100MHz}{66MHz} \right] = 13.636ms \quad \text{Equation 2-10}$$

$$CorrectionLatency = 660\mu s \cdot \left[ \frac{100MHz}{66MHz} \right] = 1.000ms \quad \text{Equation 2-11}$$

$$ClassificationLatency = 610\mu s \cdot \left[ \frac{100MHz}{66MHz} \right] = 0.924ms \quad \text{Equation 2-12}$$

$$MitigationLatency = 13.636ms + 1.000ms + 0.924ms = 15.560ms \quad \text{Equation 2-13}$$

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 one byte payload:

*Equation 2-14*

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

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.

# 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.

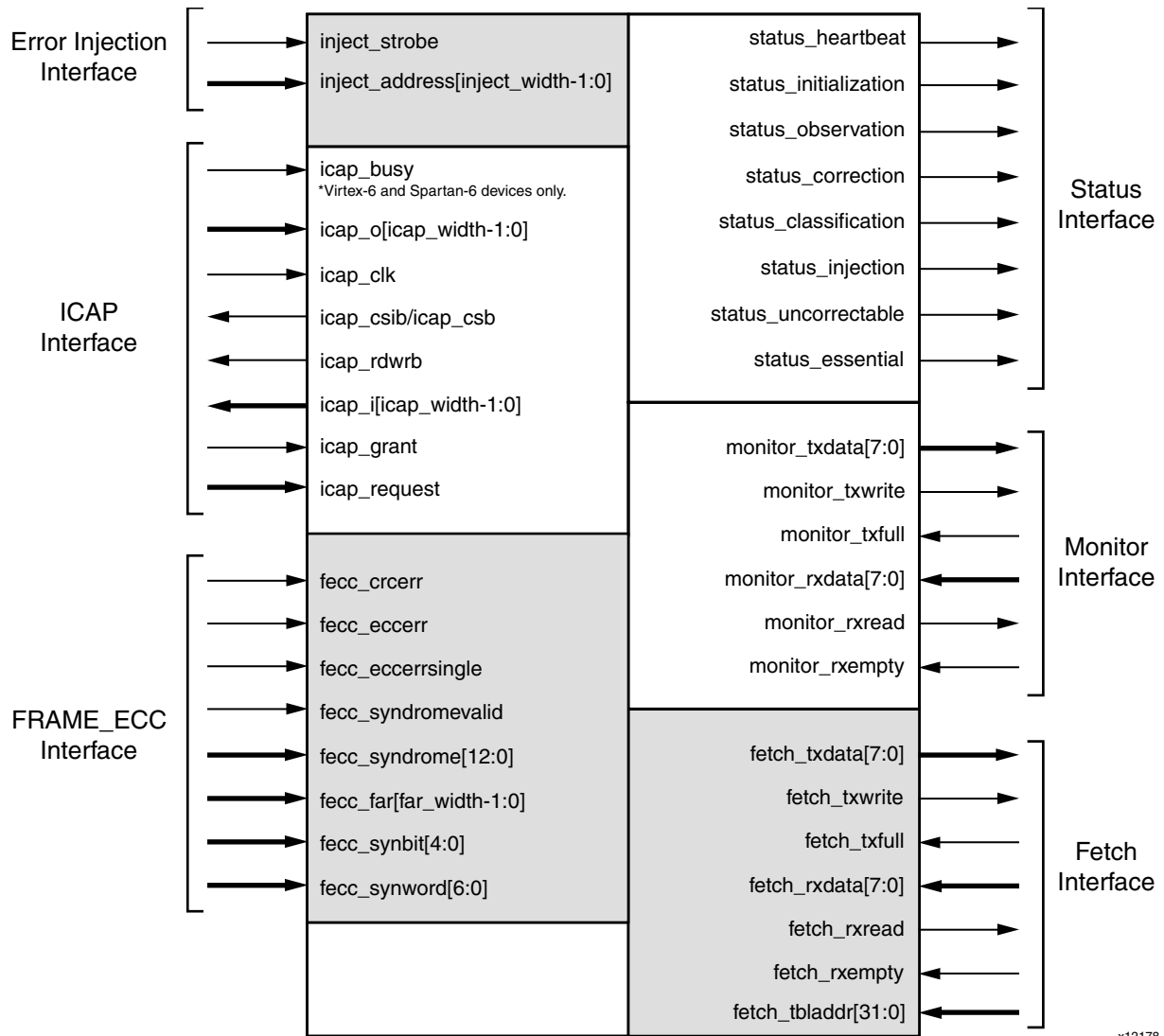


Figure 2-1: SEM Controller Ports

The SEM Controller has no reset input or output. It automatically initializes itself with an internal synchronous reset derived from the de-assertion 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 UG470, *7 Series FPGAs Configuration User Guide*, UG360, *Virtex-6 FPGA Configuration User Guide*, and UG380, *Spartan®-6 FPGA Configuration User Guide*.

For 7 series devices, `icap_busy` is not part of the ICAP Interface.

Table 2-12: ICAP Interface Signals

Name	Sense	Direction	Description
<code>icap_busy</code>	HIGH	IN	Receives BUSY output of ICAP. For 7 series devices, <code>icap_busy</code> is not part of the ICAP Interface.
<code>icap_o[icap_width-1:0]</code>	HIGH	IN	Receives O output of ICAP. The variable <code>icap_width</code> is equal to 32 for 7 series and Virtex-6 devices and 16 for Spartan-6 devices.
<code>icap_csib / icap_csb</code>	LOW	OUT	Drives CSIB (7 series) / CSB (Virtex-6 and Spartan-6) input of ICAP.
<code>icap_rdwrb</code>	LOW	OUT	Drives RDWRB input of ICAP.
<code>icap_i[icap_width-1:0]</code>	HIGH	OUT	Drives I input of ICAP. The variable <code>icap_width</code> is equal to 32 for 7 series and Virtex-6 devices and 16 for Spartan-6 devices.
<code>icap_clk</code>	EDGE	IN	Receives the clock for the design. This same clock also must be applied to the CLK input of ICAP. The clock frequency must comply with the ICAP input clock requirements as specified in the target device data sheet.
<code>icap_request</code>	HIGH	OUT	This signal is reserved for future use. Leave this port OPEN.
<code>icap_grant</code>	HIGH	IN	This signal is reserved for future use. Tie this port to VCC.

## 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 UG470, *7 Series FPGAs Configuration User Guide*, and UG360, *Virtex-6 FPGA Configuration User Guide*.

For Spartan-6 devices, the FRAME\_ECC primitive and the soft error detection functionality are implemented in soft logic within the SEM Controller. This logic implements the same

capabilities as present in Virtex-6 FPGAs. As a result, no FRAME\_ECC Interface exists on implementations of the SEM Controller for Spartan-6 devices.

**Table 2-13: FRAME\_ECC Interface Signals**

Name	Sense	Direction	Description
fecc_crcerr	HIGH	IN	Receives CRCERROR output of FRAME_ECC.
fecc_eccerr	HIGH	IN	Receives ECCERROR output of FRAME_ECC.
fecc_eccerrsingle	HIGH	IN	Receives ECCERRORSINGLE output of FRAME_ECC.
fecc_syndromevalid	HIGH	IN	Receives SYNDROMEVALID output of FRAME_ECC.
fecc_syndrome[12:0]	HIGH	IN	Receives SYNDROME output of FRAME_ECC.
fecc_far[far_width-1:0]	HIGH	IN	Receives FAR output of FRAME_ECC. The variable for far_width is 26 for 7 series devices and 24 for Virtex-6 devices.
fecc_synbit[4:0]	HIGH	IN	Receives SYNBIT output of FRAME_ECC.
fecc_synword[6:0]	HIGH	IN	Receives SYNWORD output of FRAME_ECC.

Advanced users can monitor the FRAME\_ECC Interface to provide an early warning of soft error detection.

Monitoring of the FRAME\_ECC Interface offers additional advantage when the SEM Controller is configured for error correction by repair, with error classification disabled. In this configuration, the user can implement a custom error classification algorithm that executes in parallel with the SEM Controller error correction algorithm, providing a very low delay between error detection and the completion of error classification.

## Status Interface

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

**Table 2-14: Status Interface Signals**

Name	Sense	Direction	Description
status_heartbeat	HIGH	OUT	The heartbeat signal is active while status_observation is TRUE. This output issues a single-cycle high pulse at least once every 128 clock cycles for 7 series and Virtex-6 devices, and at least once every 512 clock cycles for Spartan-6 devices. 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 FALSE, the behavior of the heartbeat signal is unspecified.
status_initialization	HIGH	OUT	The initialization signal is active during controller initialization, which occurs one time after the design begins operation.



**Table 2-14: Status Interface Signals**

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

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

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 2-15: Error Injection Interface Signals*

Name	Sense	Direction	Description
inject_strobe	HIGH	IN	The error injection control is used to indicate an error injection request. The inject_strobe signal should be pulsed high for one cycle 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 [inject_width-1:0]	HIGH	IN	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. For 7 series devices, the variable inject_width is 40, and for Virtex-6 and Spartan-6 devices, the variable is 36.

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 5](#).

## 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 2-16: Monitor Interface Signals*

Name	Sense	Direction	Description
monitor_txddata[7:0]	HIGH	OUT	Parallel transmit data from controller.
monitor_txwrite	HIGH	OUT	Write strobe, qualifies validity of parallel transmit data.
monitor_txfull	HIGH	IN	This signal implements flow control on the transmit channel, from the shim (peripheral) to the controller.

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

<b>Name</b>	<b>Sense</b>	<b>Direction</b>	<b>Description</b>
monitor_rxdata[7:0]	HIGH	IN	Parallel receive data from the shim (peripheral).
monitor_rxread	HIGH	OUT	Read strobe, acknowledges receipt of parallel receive data.
monitor_rxempty	HIGH	IN	This signal implements flow control on the receive channel, from the shim (peripheral) to the controller.

## 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 2-17: Fetch Interface Signals**

<b>Name</b>	<b>Sense</b>	<b>Direction</b>	<b>Description</b>
fetch_txdata[7:0]	HIGH	OUT	Parallel transmit data from controller.
fetch_txwrite	HIGH	OUT	Write strobe, qualifies validity of parallel transmit data.
fetch_txfull	HIGH	IN	This signal implements flow control on the transmit channel, from the shim (peripheral) to the controller.
fetch_rxdata[7:0]	HIGH	IN	Parallel receive data from the shim (peripheral).
fetch_rxread	HIGH	OUT	Read strobe, acknowledges receipt of parallel receive data.
fetch_rxempty	HIGH	IN	This signal implements flow control on the receive channel, from the shim (peripheral) to the controller.
fetch_tbladdr[31:0]	HIGH	IN	Used to specify the starting address of the controller data in the external source.

# Customizing and Generating the Core

This chapter provides instructions for generating the SEM Controller. The generation instructions are annotated with detailed information about each of the available options.

---

## Creating a Project in ISE Design Suite

First, create a project using the Xilinx CORE Generator software. For detailed information on starting and using the CORE Generator software, see the Xilinx CORE Generator User Guide. Perform the following steps:

1. Start the CORE Generator software.
2. Choose **File > New Project** from the menu.
3. Using the file requestor dialog box:
  - a. Navigate to the desired project directory.
  - b. Modify the project file name, if desired.
  - c. Click **Save**.
4. Set the Part Options in the Project Options dialog box:
  - a. Select the target family, device, package, and speed grade.  
  
Example: Kintex-7, xc7k325t-ffg900-1  
  
Note: If an unsupported family is selected, the IP core will not appear in the IP Catalog.
  - b. Click **Apply**.
5. Set the Generation Options in the Project Options dialog box:
  - a. For Flow, select Design Entry in either VHDL or Verilog.
  - b. For Flow Settings, select either ISE (for XST) or Synplicity (for Synplify Pro).
  - c. Click **Okay**.

After creating the project, the IP core will be available for selection in the IP Catalog, located at **FPGA Features and Design > Soft Error Mitigation > Soft Error Mitigation**.

# Customizing and Generating the Core in ISE Design Suite

Locate the IP core in the IP Catalog and click it once to select it. Important information regarding the solution is displayed in the main window. Review this information before proceeding.

In the Actions section of the main window, click the Customize and Generate link. This launches page one of the solution customization dialog box, shown in [Figure 3-1](#).

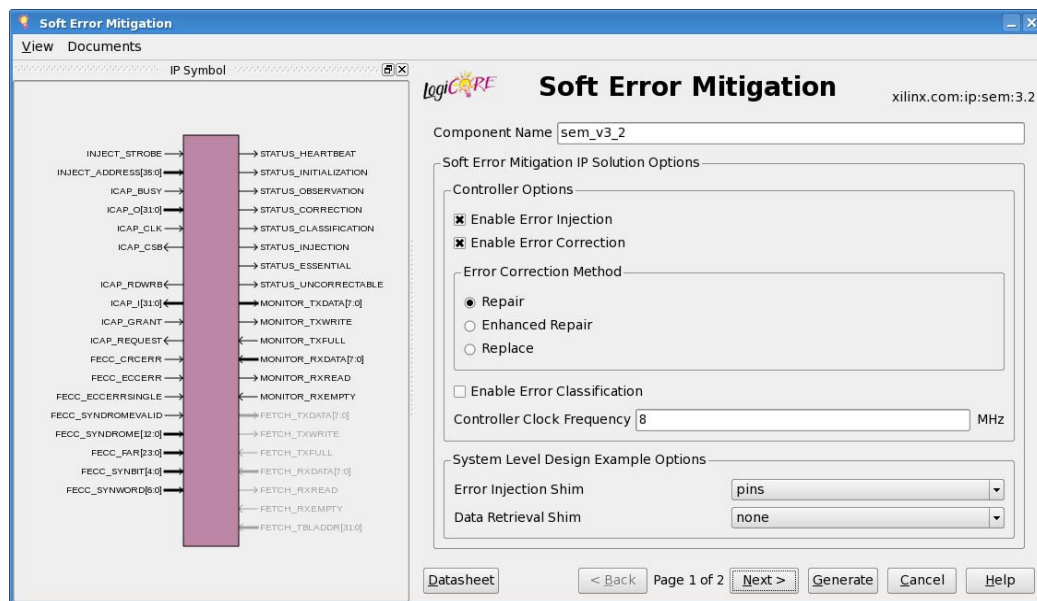


Figure 3-1: Solution Customization Dialog Box Page 1

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 will be 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\_v3\_2" 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 will exist on the component, given the current option settings. Ports that will be generated are drawn in black. This diagram is automatically updated when the option settings are modified.

## Controller Options: Enable Error Injection

The Enable Error Injection checkbox 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 commands from the user. 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 will not perform any error injections. Note that 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 ignore them.

## Controller Options: Enable Error Correction

The Enable Error Correction checkbox 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 selected by the user. 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 will 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 checkbox 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 may 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 bitgen 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 will look up the error location. Depending on the information in the table, the controller will either report the error as essential or non-essential. If a detected error cannot be corrected, this is because the error cannot be located. Therefore, the controller conservatively reports the error as essential because it has no way to look up data to indicate otherwise.

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

Note that 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 5](#). 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 Xilinx ChipScope tool

When selecting ChipScope analyzer to control the Error Injection Interface, the ChipScope tool ICON and ChipScope tool VIO cores are not included. They must be generated (separately) with their output products placed in the example design directory. The necessary ChipScope tool core settings are described at the end of this chapter.

## 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 page two of the of the solution customization dialog box by clicking Next. This page is shown in [Figure 3-2](#).

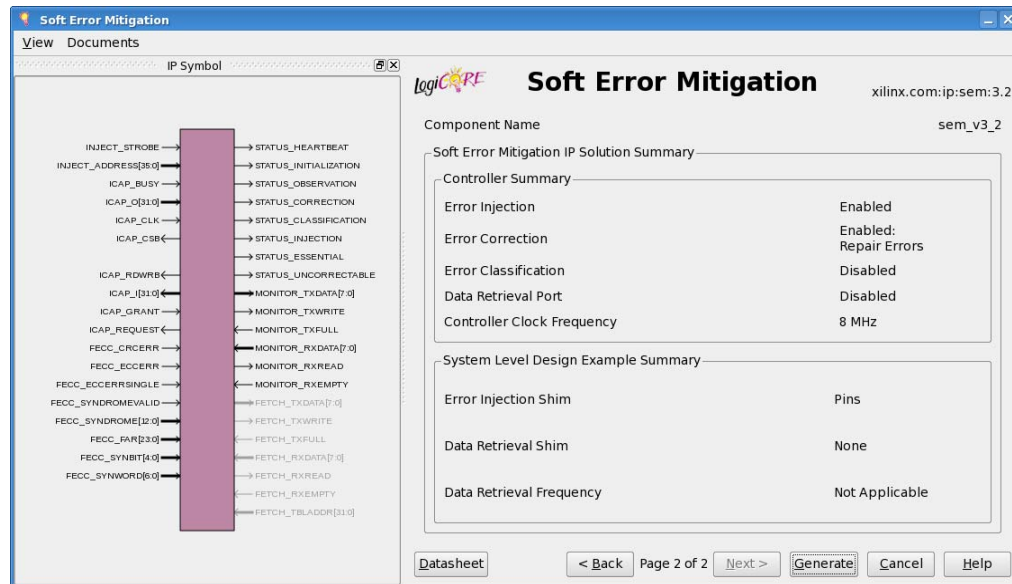


Figure 3-2: Solution Customization Dialog Box Page 2



Review the summary to confirm each option is correct. Return to the previous page, if necessary, to correct or change options prior to generation. After the options are reviewed and correct, proceed to generate the solution.

## Generating the Solution







Click Generate to begin the generation process. The CORE Generator software will generate and deliver a customized solution based on the provided option settings. When this process has completed, a final dialog box with important information regarding the solution will appear. Review this information before exiting the CORE Generator software to review the delivered files.

## Output Generation

The SEM Controller deliverables are organized in the directory structure shown below. Click a directory name to go to the description of the directory and the files it contains.

-  [<project directory>](#)  
Top-level project directory; name is user-defined
-  [<project directory>/<component name>](#)

Release notes file

-  [<component name>/doc](#)  
Product documentation
-  [<component name>/example design](#)  
Verilog or VHDL design files
-  [<component name>/implement](#)  
Implementation script files
-  [<component name>/implement/results](#)  
Results directory, created after implementation scripts are run, and contains implement script results
-  [<component name>/implement/synplify](#)  
Synthesis results when Synplify Pro is used
-  [<component name>/implement/xst](#)  
Synthesis results when XST is used

## <project directory>

The <project directory> contains all the CORE Generator project files.

Table 3-1: Project Directory

Name	Description
<project_dir>	
<component_name>.xco	Configuration options file.
<component_name>.ngc	SEM Controller implementation netlist.
<component_name>.{v vhd}	SEM Controller simulation netlist.
<component_name>.{veo vho}	SEM Controller instantiation template.
<component_name>_flist.txt	List of files delivered with the solution.

[Back to Top](#)

## <project directory>/<component name>

The component name directory contains the release notes in the readme file provided with the core, which can include tool requirements, updates, and issue resolution.

Table 3-2: Component Name Directory

Name	Description
<project_dir>/<component_name>	
sem_v3_2_readme.txt	Release notes file.

[Back to Top](#)

## <component name>/doc

The doc directory contains the PDF documentation provided with the core.

**Table 3-3: Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
pg036_sem.pdf	SEM Controller Product Guide

[Back to Top](#)

## <component name>/example design

The example design directory contains the example design files provided with the core.

**Table 3-4: Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
sem_cfg.{v vhd}	Example design configuration logic.
sem_example.{v vhd}	Example design top level.
sem_example.ucf	Example design constraints file.
sem_mon.{v vhd}	Example design MON shim.
sem_mon_fifo.{v vhd}	Example design MON shim FIFO sub-block.
sem_mon_piso.{v vhd}	Example design MON shim PISO sub-block.
sem_mon_sipo.{v vhd}	Example design MON shim SIPO sub-block.
sem_ext.{v vhd}	Example design EXT shim. This file is only generated if error classification or error correction by replace are enabled.
sem_ext_byte.{v vhd}	Example design EXT shim byte transfer sub-block. This file is only generated if error classification or error correction by replace are enabled.
sem_hid.{v vhd}	Example design HID shim. This file is only generated if error injection using the ChipScope tool is enabled.
icon_bscan_bufg.{v vhd}	Example design HID shim sub-block. This file is only generated for Virtex-6 or Spartan-6 devices if error injection using the ChipScope tool is enabled.

[Back to Top](#)

## <component name>/implement

The implement directory contains the core implementation script files.

Table 3-5: Implement Directory

Name	Description
<project_dir>/<component_name>/implement	
implement.bat	Windows batch file implementation script.
implement.sh	Linux batch file implementation script.
makedata.tcl	Bitgen data formatter for essential bit lookup table and correction by replace data. This file is only generated if error classification or error correction by replace are enabled.
synplify.prj	Synplify Pro synthesis script, only generated if the project option flow setting is Synplicity.
xst.prj	XST synthesis project, only generated if the project option flow setting is ISE Design Suite.
xst.scr	XST synthesis script, only generated if the project option flow setting is ISE Design Suite.

[Back to Top](#)

## <component name>/implement/results

The results directory is created by the implement script. The implementation results are placed in the results directory.

## <component name>/implement/synplify

The synplify directory is created by the implementation script. The synthesis results are placed in the synplify directory.

## <component name>/implement/xst

The xst directory is created by the implementation script. The synthesis results are placed in the xst directory.

# Generating and Using ChipScope Tool Files

This section describes requirements for ChipScope tool files necessary to support the optional error injection feature in the specific customizations where the ChipScope tool is selected as the error injection shim.

## ChipScope Tool Files for 7 Series FPGAs

The ChipScope tool ICON core must be generated for the target device, using the default core name "chipscope\_icon", with the following parameters:

```
ENABLE_JTAG_BUFG = TRUE
NUMBER_CONTROL_PORTS = 1
USE_EXT_BSCAN = FALSE
USE_SOFTBSCAN= FALSE
USE_UNUSED_BSCAN = FALSE
```

The USER\_SCAN\_CHAIN can be set as desired. After the ICON core is generated, the `chipscope_icon.ngc` and `chipscope_icon.{v|vhd}` files must be copied to the example design directory.

The ChipScope tool VIO core must be generated for the target device, using the default core name "chipscope\_vio", with the following parameters:

```
ENABLE_ASYNCHRONOUS_INPUT_PORT = FALSE
ENABLE_ASYNCHRONOUS_OUTPUT_PORT = FALSE
ENABLE_SYNCHRONOUS_INPUT_PORT = TRUE
ENABLE_SYNCHRONOUS_OUTPUT_PORT = TRUE
INVERT_CLOCK_INPUT = FALSE
SYNCHRONOUS_INPUT_PORT_WIDTH = 8
SYNCHRONOUS_OUTPUT_PORT_WIDTH = 41
```

After the VIO core is generated, `chipscope_vio.ngc` and `chipscope_vio.{v|vhd}` files must be copied to the example design directory. The instantiation and interconnection of the ICON and VIO components inside the HID shim can be inspected, if desired. The mapping of the VIO synchronous input and synchronous output ports is:

```
sync_in[7] receives status_heartbeat
sync_in[6] receives status_uncorrectable
sync_in[5] receives status_essential
sync_in[4] receives status_injection
sync_in[3] receives status_classification
sync_in[2] receives status_correction
sync_in[1] receives status_observation
sync_in[0] receives status_initialization
sync_out[40] drives inject_strobe
sync_out[39:0] drives inject_address[39:0]
```

## ChipScope Tool Files for Virtex-6 and Spartan-6 FPGAs

The ChipScope ICON core must be generated for the target device, using the default core name "chipscope\_icon", with the following parameters:

```
ENABLE_JTAG_BUFG = TRUE
NUMBER_CONTROL_PORTS = 1
USE_EXT_BSCAN = FALSE
USE_SOFTBSCAN= FALSE
USE_UNUSED_BSCAN = FALSE
```

The USER\_SCAN\_CHAIN can be set as desired. After the ICON core is generated, the `chipscope_icon.ngc` and `chipscope_icon.{v|vhd}` files must be copied to the example design directory.

The ChipScope tool VIO core must be generated for the target device, using the default core name "chipscope\_vio", with the following parameters:

```
ENABLE_ASYNCHRONOUS_INPUT_PORT = FALSE
ENABLE_ASYNCHRONOUS_OUTPUT_PORT = FALSE
ENABLE_SYNCHRONOUS_INPUT_PORT = TRUE
ENABLE_SYNCHRONOUS_OUTPUT_PORT = TRUE
INVERT_CLOCK_INPUT = FALSE
SYNCHRONOUS_INPUT_PORT_WIDTH = 9
SYNCHRONOUS_OUTPUT_PORT_WIDTH = 37
```

After the VIO core is generated, `chipscope_vio.ngc` and `chipscope_vio.{v|vhd}` files must be copied to the example design directory.

The instantiation and interconnection of the ICON and VIO components inside the HID shim can be inspected, if desired. The mapping of the VIO synchronous input and synchronous output ports is:

```
sync_in[8] is reserved, and is tied low
sync_in[7] receives status_heartbeat
sync_in[6] receives status_uncorrectable
sync_in[5] receives status_essential
sync_in[4] receives status_injection
sync_in[3] receives status_classification
sync_in[2] receives status_correction
sync_in[1] receives status_observation
sync_in[0] receives status_initialization
sync_out[36] drives inject_strobe
sync_out[35:0] drives inject_address[35:0]
```

## Using ChipScope Analyzer

A project must be created in the ChipScope Analyzer software. The status signals received by the synchronous input port should be represented with LEDs. The `inject_address` output value should be represented as HEX for data entry. The `inject_strobe` control output must be represented as a three-cycle PULSE output for proper operation.

# Example Design

This section overviews the function of the SEM Controller system-level design example and the interfaces it exposes. The system-level design example encapsulates the controller and various shims that serve to interface the controller to other devices. These shims can include I/O Pins, ChipScope, I/50 Interfaces, Memory Controllers, or application-specific system management interfaces.

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

---

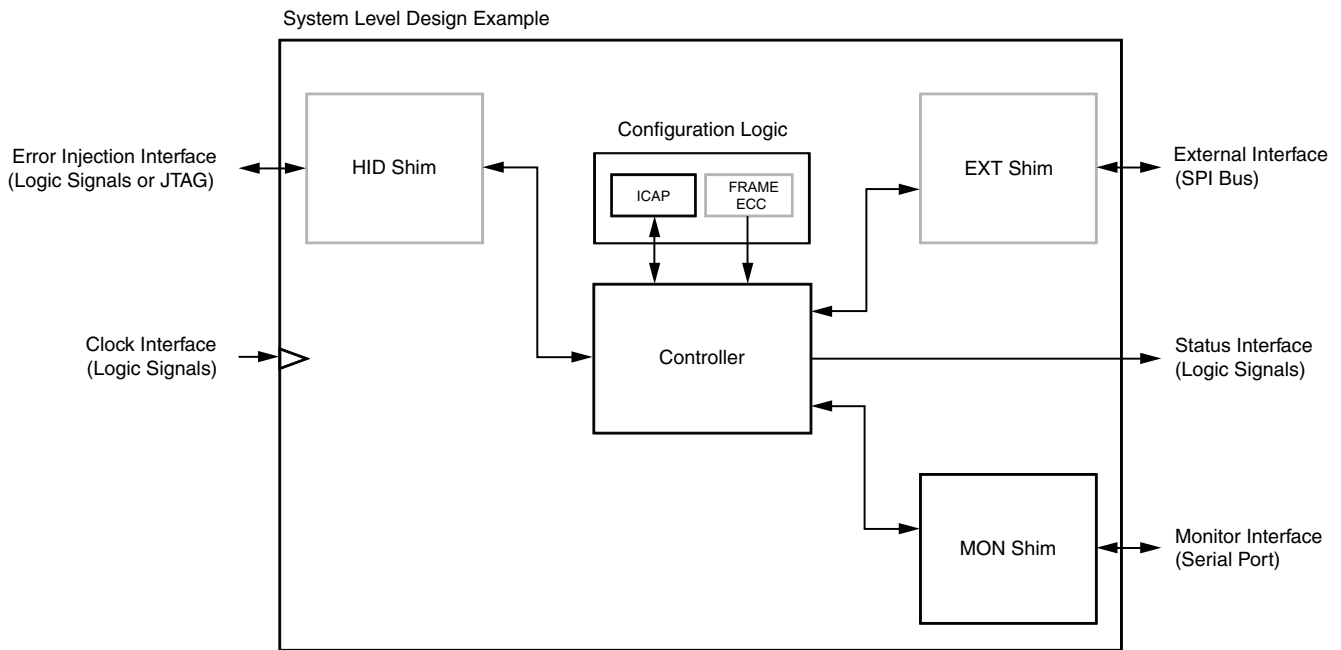
## Functions

In addition to 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 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.
- The EXT shim, 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.
- The HID shim, a bridge between the controller and an interface device. The resulting interface can be used to exchange commands and status with the controller. This shim is only present in certain controller configurations.



Figure 4-1 shows a block diagram of the system-level design example. The blocks drawn in gray only exist in certain configurations.



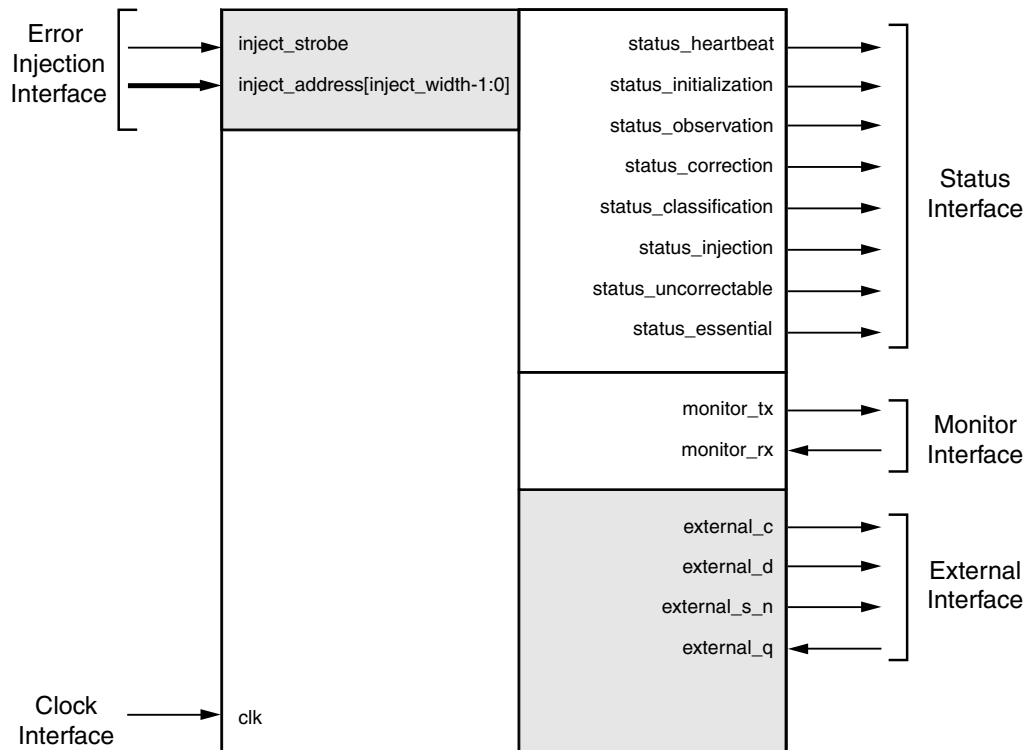
X12177

Figure 4-1: Example Design Block Diagram

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 4-2 shows the example design ports. The ports are clustered into six groups. The groups shaded in gray only exist in certain configurations.



X12176

Figure 4-2: Example Design Ports

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 the controller. 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 elements.

Table 4-1: 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 the controller (a byte stream of ASCII codes) for serial transmission. Similarly, the shim de-serializes command information presented to the controller (a bitstream of ASCII codes) for parallel presentation to the controller.

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 5, 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 may be necessary depending on system requirements.

Table 4-2: 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 the controller requires 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 the controller 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 the controller 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 5, 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 may be necessary depending on system requirements.

*Table 4-3: 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 the controller supports error injection and the HID shim is set to I/O Pins. This interface is a direct pass-through from the controller. See [Chapter 2, Error Injection Interface](#) for a description of this interface.

When the controller supports error injection and the HID shim is set to ChipScope, or when error injection is disabled, this interface is absent.

---

## 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 will compile, but the controller will not exit the initialization state.

Hardware-based evaluation of the controller behaviors is required. Alternatively, customers can use ISim Hardware Co-simulation to evaluate their design.

---

## Demonstration Test Bench

No simulation test bench is provided with the example design.

---

## Implementation in ISE Design Suite

After generating the solution, the resulting controller netlist and example design files can be used with a standard design flow.

Synthesis of designs that instantiate the controller is supported for Synopsys Synplify Pro and Xilinx XST. The synthesis output can be processed using the Xilinx implementation tools. This chapter outlines the example design synthesis and implementation scripts.

Implementation of the controller, when configured to use the optional error classification function or the optional correction by replace function, requires a large amount of system RAM. Xilinx recommends the use of a 64-bit operating system with 16 Gb or more of system RAM.

To synthesize and implement the example design, open a console window and type the following:

### Windows

```
ms-dos> cd <project directory>\<component name>\implement
ms-dos> implement.bat
```

### Linux

```
% cd <project directory>/<component name>/implement
% ./implement.sh
```

These commands change directory and execute an implementation script. The script synthesizes the design, performs physical implementation, and generates programming files. The result files are placed in the results directory. The following is an outline of the scripted processing sequence:

1. If the solution requires ChipScope analyzer files in the example design directory, the script verifies the files exist. If they do not exist, the script exits.
2. The script removes intermediate and result files from previous implementation runs.
3. The script synthesizes the project using Synopsys Synplify Pro or Xilinx XST based on the project option settings used when the solution was generated.
4. The script performs a physical implementation:
  - a. Xilinx ngdbuild application builds a design database
  - b. Xilinx map application maps the design to the target device
  - c. Xilinx par application places and routes the design
  - d. Xilinx trce application performs static timing analysis
  - e. Xilinx netgen application generates a timing simulation model

- f. Xilinx bitgen application generates programming files
5. 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 bitgen output files into a SPI Flash programming file.
6. The script removes some intermediate files to clean up the results directory.

The script file starts from the example design source and the controller netlist and results in programming files. It is possible to use the Xilinx ISE Design Suite graphical design environment to implement the example design, provided the implementation tool options used in the script are preserved and the additional TCL script is invoked manually, if it is required.

---

## 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 may 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 bitgen application.

The format of the data is required to be binary, using the full data set(s) generated by bitgen. 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.

- 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)
- The remaining bytes are reserved, filled with zero

A pointer value of 0x00000000 is used if a particular block of data is not present. After the table, the essential bit data and replacement data can be located at any address provided each data set is contiguous.

The TCL script, which post processes the bitgen 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
- 

## Directory and File Contents

The example design directory contents are the same as the files shown in [Output Generation in Chapter 3](#).

# 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 60](#).
  - 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 may be limited if standard baud rates are desired.
  - A sample analysis is presented in [Monitor Interface, page 59](#).

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 the controller is doing.
- Flags: Identifies the type of error detected.
- Heartbeat: Indicates the FPGA configuration system is active.

Only the heartbeat event is unique to the Status Interface. The other information is also available on the Monitor Interface.

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. The Status Interface is the most reliable event reporting mechanism because it has the least amount of logic associated with it. Although it is possible to receive and interpret the status signals inside the FPGA, this is discouraged because it reduces the overall reliability of the solution.

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 signals in the Status Interface are generated by sequential logic processes in the controller 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 5-1](#). 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.

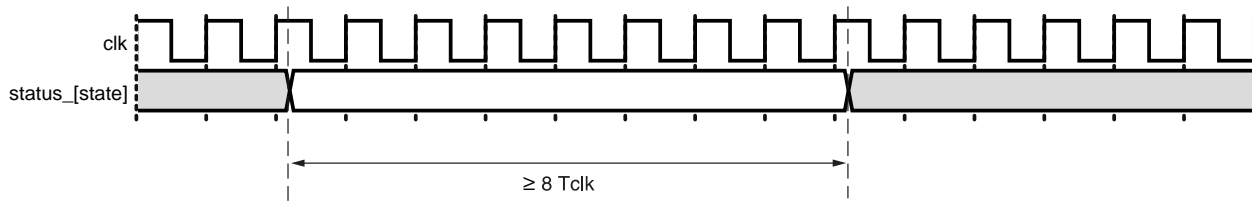


Figure 5-1: Status Interface State Signals Switching Characteristics

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 5-2 and Figure 5-3. 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.

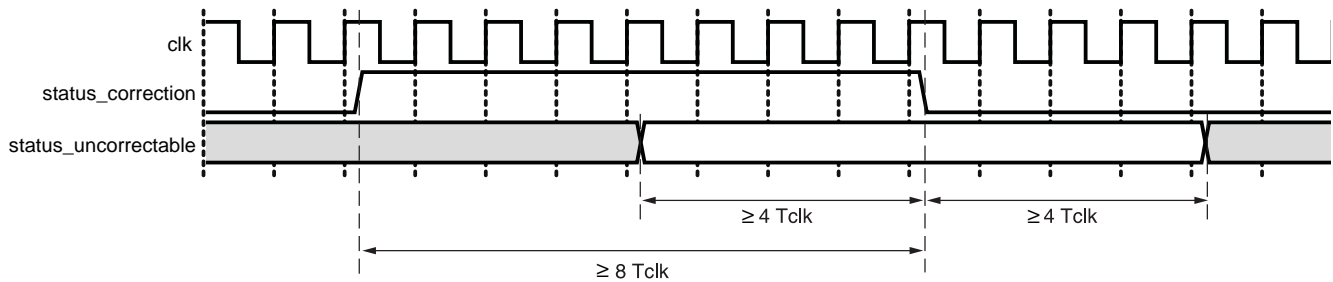


Figure 5-2: Status Interface Uncorrectable Flag Switching Characteristics

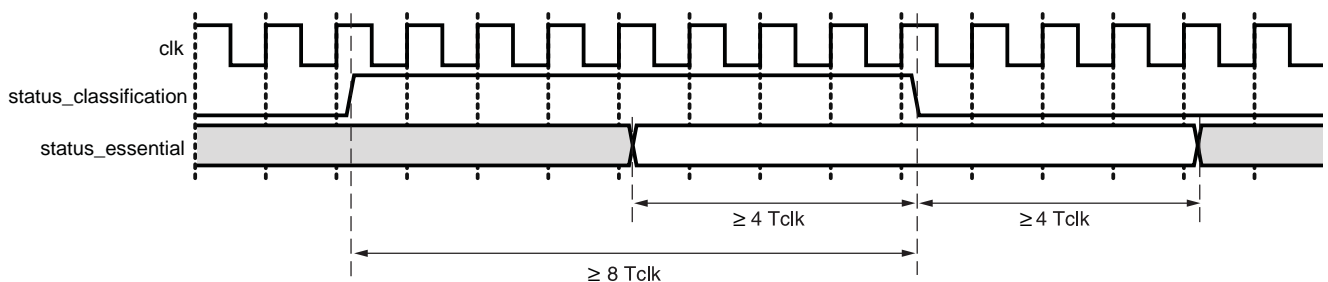


Figure 5-3: Status Interface Essential Flag Switching Characteristics

The switching behavior of the heartbeat signal `status_heartbeat` is illustrated in Figure 5-4. This signal is a direct output from the integrated silicon readback process, and is active during the observation state. Upon entering the observation state, the heartbeat signal will become active when the integrated silicon 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 other states, the heartbeat signal will be active, but the switching behavior is not guaranteed.

The first heartbeat after entering the observation state should occur within the one readback scan. An entire scan depends on the device and frequency. See [Table 2-9](#) to determine how long a readback scan takes for the device being used.

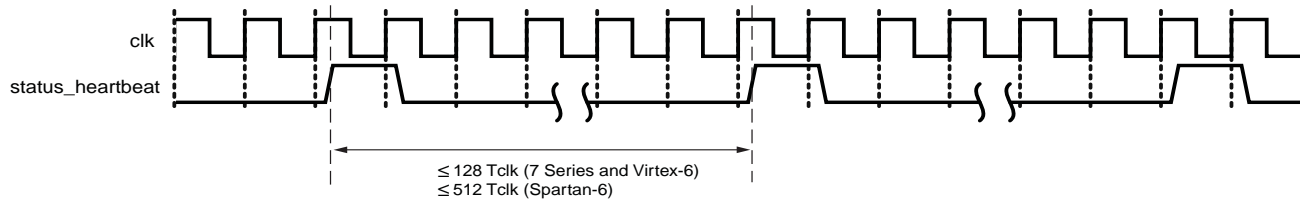


Figure 5-4: Status Interface Heartbeat Switching Characteristics

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 5-5](#) shows the switching behavior, and is representative of both transmit and receive.



Figure 5-5: 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:

$$V\_ENABLETIME = \text{round to integer} \left[ \frac{\text{input clock frequency}}{16 \times \text{nominal bitrate}} \right] - 1 \quad \text{Equation 5-1}$$

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.

$$V\_ENABLETIME = \text{round to integer} \left[ \frac{66000000}{16 \times 115200} \right] - 1 = 35 \quad \text{Equation 5-2}$$

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.

## 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 5-1](#) provides an SPI flash device recommendation for each supported FPGA:

**Table 5-1: External Storage Requirements**

Device	Error Classification Only	Error Correction by Replacement Only	Error Classification with Error Correction by Replacement
XC7K70T	32 Mbit	32 Mbit	64 Mbit
XC7K160T	64 Mbit	64Mbit	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
XC7V585T	128 Mbit	128 Mbit	256 Mbit
XC7VX690T	256 Mbit	256 Mbit	512 Mbit
XC7VX980T	256 Mbit	256 Mbit	512 Mbit
XC6VCX75T	32 Mbit	32 Mbit	64 Mbit
XC6VCX130T	32 Mbit	32 Mbit	64 Mbit
XC6VCX195T	64 Mbit	64 Mbit	128 Mbit
XC6VCX240T	64 Mbit	64 Mbit	128 Mbit
XC6VHX250T	64 Mbit	64 Mbit	128 Mbit
XC6VHX255T	64 Mbit	64 Mbit	128 Mbit
XC6VHX380T	128 Mbit	128 Mbit	256 Mbit
XC6VHX565T	128 Mbit	128 Mbit	256 Mbit
XC6VLX75T	32 Mbit	32 Mbit	64 Mbit
XC6VLX130T	32 Mbit	32 Mbit	64 Mbit
XC6VLX195T	64 Mbit	64 Mbit	128 Mbit
XC6VLX240T	64 Mbit	64 Mbit	128 Mbit
XC6VLX365T	128 Mbit	128 Mbit	256 Mbit
XC6VLX550T	128 Mbit	128 Mbit	256 Mbit
XC6VLX760	256 Mbit	256 Mbit	512 Mbit
XC6VSX315T	128 Mbit	128 Mbit	256 Mbit

Table 5-1: External Storage Requirements (Cont'd)

Device	Error Classification Only	Error Correction by Replacement Only	Error Classification with Error Correction by Replacement
XC6VSX475T	128 Mbit	128 Mbit	256 Mbit
XC6SLX4	4 Mbit	N/A	N/A
XC6SLX9	4 Mbit	N/A	N/A
XC6SLX16	4 Mbit	N/A	N/A
XC6SLX25T	8 Mbit	N/A	N/A
XC6SLX45T	16 Mbit	N/A	N/A
XC6SLX75T	16 Mbit	N/A	N/A
XC6SLX100T	32 Mbit	N/A	N/A
XC6SLX150T	32 Mbit	N/A	N/A

The SPI flash device must support the standard “fast read” command 0x0B. If the SPI flash density is larger than 128 Mbit, it must also support a 32-bit addressing extension, enabled with “enable 32-bit addressing” command 0xB7.

Figure 5-6 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 may not be available depending on the selected FPGA or I/O bank voltage.

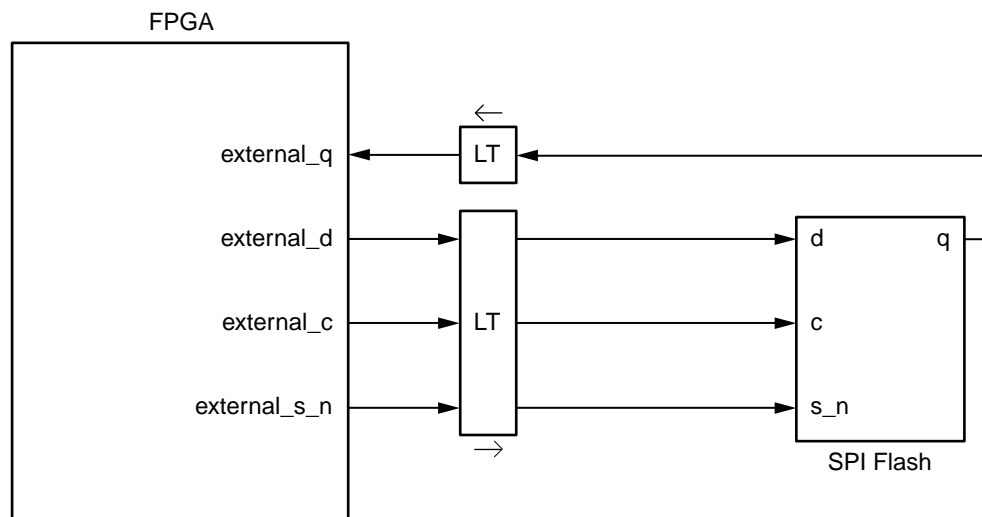


Figure 5-6: SPI Flash Device Connection, Including Level Translators

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.

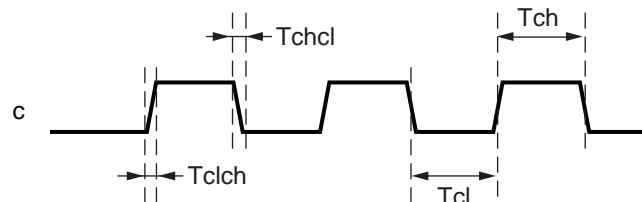


Figure 5-7: SPI Flash Device Input Clock Requirements

The following parameters, shown in Figure 5-7, 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 will exhibit 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  $\geq 2$  ns? Yes
2. Check: Is  $T_{chcl} \geq T_{fall}$ ? Is  $33$  ns  $\geq 2$  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:  $T_{clk} \geq T_{fall} + T_{cl}$  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 will 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.

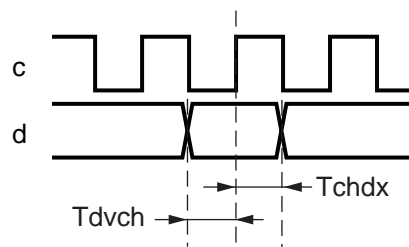


Figure 5-8: SPI Flash Device Input Data Capture Requirements



The following parameters, shown in [Figure 5-8](#), 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 5-9.

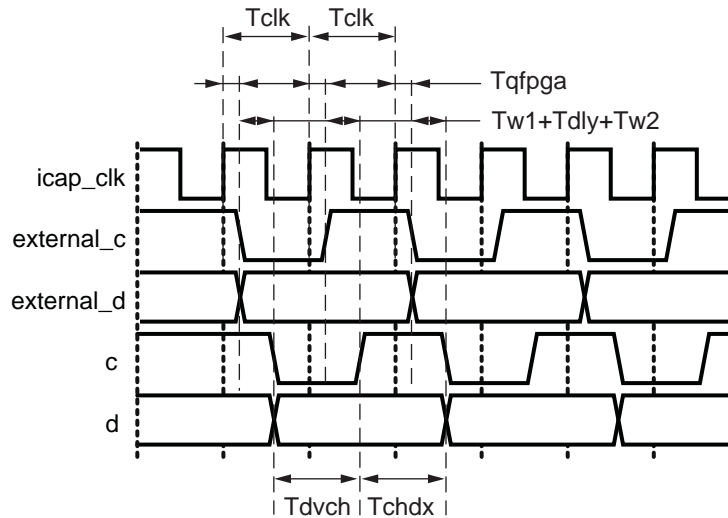


Figure 5-9: Input Data Capture Timing

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:**

- $T_{dvch} = 2 \text{ ns}$  (from SPI flash data sheet)
  - $T_{chdx} = 5 \text{ ns}$  (from SPI flash data sheet)
1. Calculate:  $T_{clk} \geq T_{dvch}$  requires  $T_{clk} \geq 2 \text{ ns}$
  2. Calculate:  $T_{clk} \geq T_{chdx}$  requires  $T_{clk} \geq 5 \text{ ns}$

These requirements on  $T_{clk}$  indicate that the SPI Transmit Waveform and Timing Budget will 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 will exhibit 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.

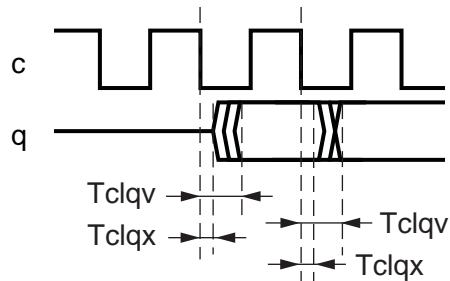


Figure 5-10: SPI Flash Device Output Data Switching Characteristics

The following parameters, shown in [Figure 5-10](#), 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 5-11 and Figure 5-12.

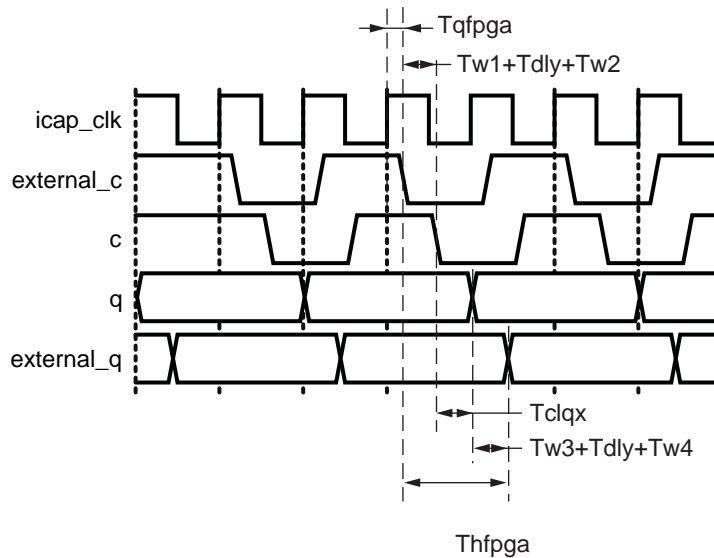


Figure 5-11: Output Data Capture Timing (Hold Analysis)

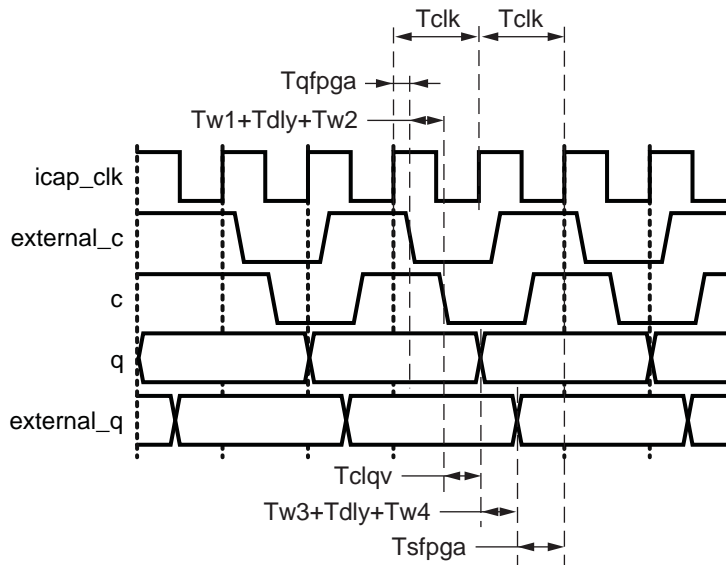


Figure 5-12: Output Data Capture Timing (Setup Analysis)

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:

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

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:

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

**Example:**

- $T_{clqv} = 8$  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 that results from the implementation of the system-level design example in the FPGA targeted for use in the application. To generate the necessary reports, the timing analyzer must be run using the “-fastpaths” option.

The examples that follow are excerpts from the timing analyzer report generated from a Virtex-6 device implementation of the system-level design example. The purpose of the examples are to illustrate where to find the required information.

Locate  $T_{qfpga}$  by searching the timing report for flip-flop to pad maximum path timing analysis, where the destination pad is identified as "external\_c".

- $T_{qfpga} = \text{I/O Datapath Delay (external\_c)}$
- $T_{qfpga} = 3.360 \text{ ns, maximum}$

-----  
Slack: 13.042ns (requirement - (clock arrival + clock path + datapath + uncertainty))

Source: example\_ext/example\_ext\_byte/ext\_c\_ofd (FF)

Destination: external\_c (PAD)

Source Clock: icap\_clk rising at 0.000ns

Requirement: 20.000ns

**Datapath Delay: 3.360ns (Levels of Logic = 1)**

Clock Path Delay: 3.573ns (Levels of Logic = 2)

Clock Uncertainty: 0.025ns

Clock Uncertainty: 0.025ns ((TSJ<sup>2</sup> + TIJ<sup>2</sup>)<sup>1/2</sup> + DJ) / 2 + PE

Total System Jitter (TSJ): 0.050ns

Total Input Jitter (TIJ): 0.000ns

Discrete Jitter (DJ): 0.000ns

Phase Error (PE): 0.000ns

Maximum Clock Path at Slow Process Corner:

clk to example\_ext/example\_ext\_byte/ext\_c\_ofd

Location	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)
U23.I	Tiopi	0.702	clk clk clk_IBUFG	
BUFGCTRL_X0Y0.I0	net (fanout=1)	0.938	clk_IBUFG	
BUFGCTRL_X0Y0.O	Tbgcko_O	0.092	example_bufg	example_bufg
OLOGIC_X0Y194.CLK	net (fanout=283)	1.841	icap_clk	

-----  
Total 3.573ns (0.794ns logic, 2.779ns route)  
(22.2% logic, 77.8% route)

Maximum Datapath at Slow Process Corner:

example\_ext/example\_ext\_byte/ext\_c\_ofd to external\_c

Location	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)
OLOGIC_X0Y194.OQ	Tockq	0.625	external_c_OBUF	example_ext/example_ext_byte/ext_c_ofd
E33.O	net (fanout=1)	0.002	external_c_OBUF	
E33.PAD	Tioup	2.733	external_c	external_c_OBUF external_c

-----  
**Total 3.360ns** (3.358ns logic, 0.002ns route)  
(99.9% logic, 0.1% route)  
-----

Locate  $T_{qfpga}$  by searching the timing report for flip-flop to pad minimum path timing analysis, where the destination pad is identified as "external\_c".

- $T_{qfpga}$  = I/O Datapath Delay (external\_c)
- $T_{qfpga}$  = 1.473 ns, minimum

-----  
 Delay (fastest paths): 2.951ns (clock arrival + clock path + datapath - uncertainty)

Source: example\_ext/example\_ext\_byte/ext\_c\_ofd (FF)

Destination: external\_c (PAD)

Source Clock: icap\_clk rising at 0.000ns

**Datapath Delay: 1.473ns (Levels of Logic = 1)**

Clock Path Delay: 1.503ns (Levels of Logic = 2)

Clock Uncertainty: 0.025ns

Clock Uncertainty: 0.025ns ((TSJ<sup>2</sup> + TIJ<sup>2</sup>)<sup>1/2</sup> + DJ) / 2 + PE

Total System Jitter (TSJ): 0.050ns

Total Input Jitter (TIJ): 0.000ns

Discrete Jitter (DJ): 0.000ns

Phase Error (PE): 0.000ns

Minimum Clock Path at Fast Process Corner:

clk to example\_ext/example\_ext\_byte/ext\_c\_ofd

Location	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)
-----				
U23.I	Tiopi	0.316	clk clk	clk_IBUFG
BUFGCTRL_X0Y0.I0	net (fanout=1)	0.367	clk_IBUFG	
BUFGCTRL_X0Y0.O	Tbgcko_0	0.033	example_bufg	example_bufg
OLOGIC_X0Y194.CLK	net (fanout=283)	0.787	icap_clk	
-----				
<b>Total</b>		<b>1.503ns</b>	(0.349ns logic, 1.154ns route)	
			(23.2% logic, 76.8% route)	

-----

U23.I Tiopi 0.316 clk clk clk\_IBUFG

BUFGCTRL\_X0Y0.I0 net (fanout=1) 0.367 clk\_IBUFG

BUFGCTRL\_X0Y0.O Tbgcko\_0 0.033 example\_bufg example\_bufg

OLOGIC\_X0Y194.CLK net (fanout=283) 0.787 icap\_clk

-----

Total 1.503ns (0.349ns logic, 1.154ns route)

(23.2% logic, 76.8% route)

Minimum Datapath at Fast Process Corner:

example\_ext/example\_ext\_byte/ext\_c\_ofd to external\_c

Location	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)
-----				
OLOGIC_X0Y194.OQ	Tockq	0.215	external_c_OBUF	example_ext/example_ext_byte/ext_c_ofd
E33.O	net (fanout=1)	0.002	external_c_OBUF	
E33.PAD	Tioop	1.256	external_c	external_c_OBUF external_c
-----				
<b>Total</b>		<b>1.473ns</b>	(1.471ns logic, 0.002ns route)	
			(99.9% logic, 0.1% route)	

-----

OLOGIC\_X0Y194.OQ Tockq 0.215 external\_c\_OBUF example\_ext/example\_ext\_byte/ext\_c\_ofd

E33.O net (fanout=1) 0.002 external\_c\_OBUF

E33.PAD Tioop 1.256 external\_c external\_c\_OBUF external\_c

-----

Total 1.473ns (1.471ns logic, 0.002ns route)

(99.9% logic, 0.1% route)

-----

Locate  $T_{sfpga}$  by searching the timing report for pad to flip-flop maximum path timing analysis, where the source pad is identified as "external\_q".

- $T_{sfpga}$  = I/O Datapath Delay (external\_q)
- $T_{sfpga}$  = 4.943 ns, maximum

-----  
 Slack (setup): 18.398ns (requirement - (datapath - clock path - clock arrival + uncertainty))

Source: external\_q (PAD)  
 Destination: example\_ext/example\_ext\_byte/ext\_q\_ifd (FF)  
 Destination Clock: icap\_clk rising at 0.000ns  
 Requirement: 20.000ns  
**Datapath Delay: 4.943ns (Levels of Logic = 1)**  
 Clock Path Delay: 3.366ns (Levels of Logic = 2)  
 Clock Uncertainty: 0.025ns

Clock Uncertainty: 0.025ns ((TSJ<sup>2</sup> + TIJ<sup>2</sup>)<sup>1/2</sup> + DJ) / 2 + PE  
 Total System Jitter (TSJ): 0.050ns  
 Total Input Jitter (TIJ): 0.000ns  
 Discrete Jitter (DJ): 0.000ns  
 Phase Error (PE): 0.000ns

Maximum Datapath at Slow Process Corner:

external\_q to example\_ext/example\_ext\_byte/ext\_q\_ifd  
 Location Delay type Delay(ns) Physical Resource Logical Resource(s)

-----  
 C33.I Tiopi 0.766 external\_q external\_q external\_q\_IBUF  
 ILOGIC\_X0Y187.DDLY net (fanout=1) 4.046 external\_q\_IBUF  
 ILOGIC\_X0Y187.CLK Tidockd 0.131 fetch\_rxddata<0> example\_ext/example\_ext\_byte/ext\_q\_ifd

-----  
**Total 4.943ns** (0.897ns logic, 4.046ns route)  
 (18.1% logic, 81.9% route)

Minimum Clock Path at Slow Process Corner:

clk to example\_ext/example\_ext\_byte/ext\_q\_ifd  
 Location Delay type Delay(ns) Physical Resource Logical Resource(s)

-----  
 U23.I Tiopi 0.670 clk clk clk\_IBUFG  
 BUFGCTRL\_X0Y0.I0 net (fanout=1) 0.865 clk\_IBUFG  
 BUFGCTRL\_X0Y0.O Tbgcko\_O 0.087 example\_bufg example\_bufg  
 ILOGIC\_X0Y187.CLK net (fanout=283) 1.744 icap\_clk

-----  
**Total 3.366ns** (0.757ns logic, 2.609ns route)  
 (22.5% logic, 77.5% route)  
 -----



Locate  $T_{\text{hfpga}}$  by searching the timing report for pad to flip-flop minimum path timing analysis, where the source pad is identified as "external\_q".

- $T_{\text{hfpga}}$  = I/O Datapath Delay (external\_q)
- $T_{\text{hfpga}}$  = -1.800 ns, minimum

```

-----
Slack (hold): 0.131ns (requirement - (clock path + clock arrival + uncertainty - datapath))
Source:      external_q (PAD)
Destination: example_ext/example_ext_byte/ext_q_ifd (FF)
Destination Clock: icap_clk rising at 0.000ns
Requirement: 0.000ns
Datapath Delay: 1.800ns (Levels of Logic = 1)
Clock Path Delay: 1.644ns (Levels of Logic = 2)
Clock Uncertainty: 0.025ns

Clock Uncertainty: 0.025ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
Total System Jitter (TSJ): 0.050ns
Total Input Jitter (TIJ): 0.000ns
Discrete Jitter (DJ): 0.000ns
Phase Error (PE): 0.000ns

Minimum Datapath at Fast Process Corner:
external_q to example_ext/example_ext_byte/ext_q_ifd
Location      Delay type  Delay(ns) Physical Resource Logical Resource(s)
-----
C33.I      Tiopi      0.371 external_q external_q external_q_IBUF
ILOGIC_X0Y187.DDLY net (fanout=1) 1.432 external_q_IBUF
ILOGIC_X0Y187.CLK Tiocdd (-Th) 0.003 fetch_rxdata<0> example_ext_byte/ext_q_ifd
-----
Total          1.800ns (0.368ns logic, 1.432ns route)
                  (20.4% logic, 79.6% route)

Maximum Clock Path at Fast Process Corner:
clk to example_ext/example_ext_byte/ext_q_ifd
Location      Delay type  Delay(ns) Physical Resource Logical Resource(s)
-----
U23.I      Tiopi      0.371 clk clk clk_IBUFG
BUFGCTRL_X0Y0.I0 net (fanout=1) 0.397 clk_IBUFG
BUFGCTRL_X0Y0.O Tbgcko_0 0.035 example_bufg example_bufg
ILOGIC_X0Y187.CLK net (fanout=283) 0.841 icap_clk
-----
Total          1.644ns (0.406ns logic, 1.238ns route)
                  (24.7% logic, 75.3% route)
-----

```

Check:

- Is  $T_{hfpga} \leq T_{qfpga,min} + T_{clq}$ ?
- Is  $-1.800 \text{ ns} \leq 1.473 \text{ ns} + 0 \text{ ns}$ ?
- Is  $-1.800 \text{ ns} \leq 1.473 \text{ ns}$ ? YES

Calculate:

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

requires

$$T_{clk} \geq 0.5 * (3.360 \text{ ns} + 1 \text{ ns} + 3 \text{ ns} + 1 \text{ ns} + 8 \text{ ns} + 1 \text{ ns} + 3 \text{ ns} + 1 \text{ ns} + 4.943 \text{ ns})$$

or

$$T_{clk} \geq 13.152 \text{ ns}$$

The hold requirement is satisfied, and the requirement on  $T_{clk}$  indicates that the SPI Receive Waveform and Timing Budget will restrict the system-level design example input clock cycle time to be 13.152 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 may pose on the maximum frequency of the system-level design example input clock.

### Example:

Using the results from the preceding examples in this section, the memory interface is functional, and the most stringent requirement on  $T_{clk}$  is that  $T_{clk} = 13.152 \text{ ns}$ , as the memory interface only works when the input clock frequency is 76.037 MHz or lower. Other input clock frequency limits, such as the ICAP maximum clock frequency and the system-level example maximum clock frequency, must also be considered.

## 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 four types of commands:

- Command to enter Idle state (halt normal scanning to inject errors)

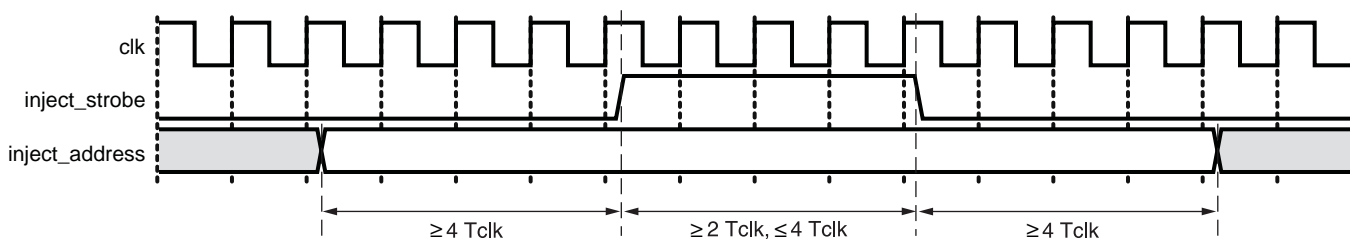
- Command to enter Observation state (resume normal scanning)
- Inject error at physical frame address
- Inject error at linear frame address

These commands are not unique to the Error Injection Interface; they are also available on the Monitor Interface.

In many cases, desired 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; in fact, this is exactly what is done when the ChipScope analyzer 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 the controller using the strobe to enable an input register. The timing requirements shown in [Figure 5-13](#) must be observed to ensure successful data capture.



**Figure 5-13: Error Injection Interface Timing Requirements**

The clock signal shown in [Figure 5-13](#) is meant only to illustrate the waveform time scale. While the pulse widths are specified in terms of clock cycles, the external device generating and driving the error injection signals need not be synchronous to the clock signal.

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 will be ignored and no error will be detected in the observation state.

## 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.

## Controller Activity

### Initialization

During FPGA configuration, the controller is held in reset 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 initializes. During initialization state, `status_initialization` is TRUE.

This initialization includes some internal housekeeping, as well as directly observable events such as the generation of a status report on the Monitor Interface. Initialization includes:

- A delay to wait for availability of the configuration system through ICAP
- A first readback cycle where the frame-level ECC checksums are computed
- A second readback cycle where the device-level CRC checksum is computed
- An additional readback cycle where an additional checksum is computed (only executed if Virtex-6 or 7 series devices are targeted)
- An additional readback cycle where 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 error injections can be performed. This command causes the controller to transition to the idle state.

The "status report" command is not frequently used; it provides some diagnostic information, and can be helpful as a mechanism to "ping" the controller without idling it. 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. At this point, 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 de-assertion 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 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 is not frequently used; it 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.

## 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 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 5-14](#) through [Figure 5-17](#), with "X" representing a "don't care."

3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0  
9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 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 5-14: 7 Series Enter Idle State Command*

3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0  
9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 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 5-15: 7 Series Enter Observation State Command*

3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0  
5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 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 5-16: Virtex-6 and Spartan-6 Enter Idle State Command*

3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0  
5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 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 5-17: Virtex-6 and Spartan-6 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.

### 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 formats for linear frame address are shown in Figure 5-18 through Figure 5-20, with borders marking nibble boundaries and shading marking separate bit fields in the command.

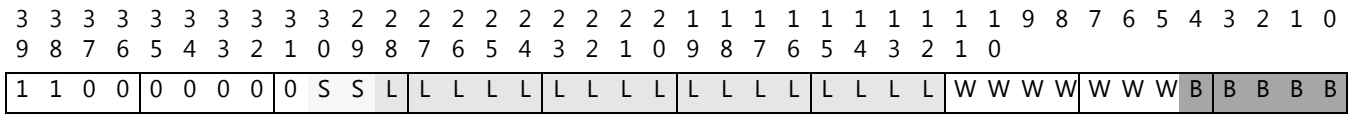


Figure 5-18: 7 Series Error Injection Command (Linear Frame Address)

Where:

- SS = SLR number (2-bit)
- LLLLLLLLLLLLLLLLLL = linear frame address (17-bit)
- WWWWWWW = word address (7-bit)
- BBBBB = bit address (5-bit)

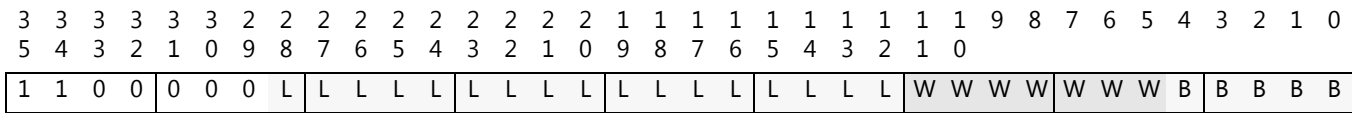


Figure 5-19: Virtex-6 Error Injection Command (Linear Frame Address)

Where:

- LLLLLLLLLLLLLLLLLL = linear frame address (17-bit)
- WWWWWWW = word address (7-bit)
- BBBBB = bit address (5-bit)

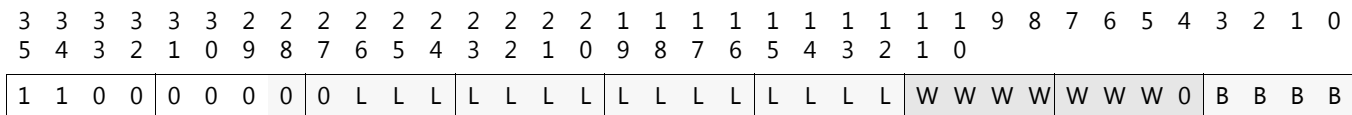


Figure 5-20: Spartan-6 Error Injection Command (Linear Frame Address)

Where:

- LLLLLLLLLLLLLLLLLL = linear frame address (15-bit)
- WWWWWWW = word address (7-bit)



- BBBB = bit address (4-bit)

An error injection command using physical frame address has the form shown in Figure 5-21 through Figure 5-23.

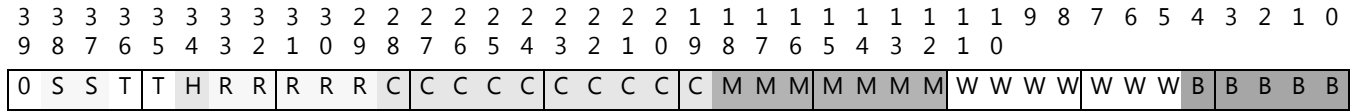


Figure 5-21: 7 Series Error Injection Command (Physical Frame Address)

Where:

- SS = SLR number (2-bit)
- TT = block type (2-bit)
- H = half address (1-bit)
- RRRRR = row address (5-bit)
- CCCCCCCCC = column address (10-bit)
- MMMMMMM = minor address (7-bit)
- WWWWWW = word address (7-bit)
- BBBB = bit address (5-bit)

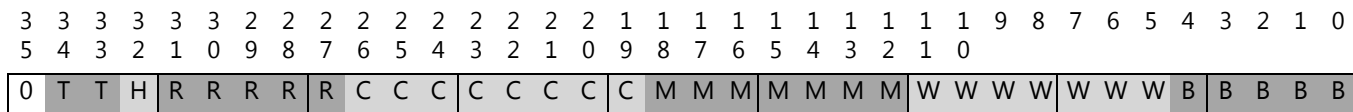


Figure 5-22: Virtex-6 Error Injection Command (Physical Frame Address)

Where:

- TT = block type (2-bit)
- H = half address (1-bit)
- RRRRR = row address (5-bit)
- CCCCCC = column address (8-bit)
- MMMMMMM = minor address (7-bit)
- WWWWWW = word address (7-bit)
- BBBB = bit address (5-bit)

Completion of the command is indicated on the Status Interface by the de-assertion of the `status_injection` output indicating the controller has exited the error injection state.

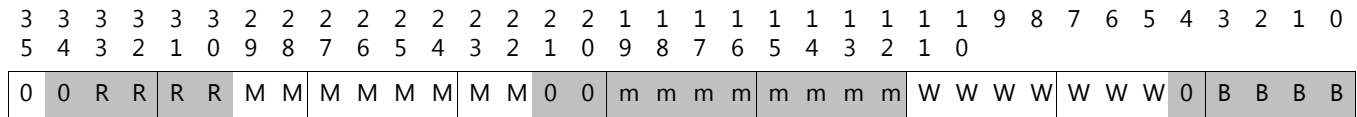


Figure 5-23: Spartan-6 Error Injection Command (Physical Frame Address)

Where:

- RRRR = row address (4-bit)
- MMMMMMMM = major address (8-bit)
- mmmmmmmm = minor address (8-bit)
- WWWWWWW = word address (7-bit)
- BBBB = bit address (4-bit)

## Monitor Interface Commands

The Monitor Interface commands define what a user 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 {9-digit hex value}    Virtex-6 and Spartan-6
N {10-digit hex value}   7 Series
```

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.

## Monitor Interface Messages

The Monitor Interface messages define what messages a user 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 only once when the controller first starts. The 7 series device initialization report looks like this:

```

X7_SEM_V3_2      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
    
```

The Virtex-6 and Spartan-6 device initialization reports are virtually identical except the controller reports its name and version as V6\_SEM\_V3\_2 and S6\_SEM\_V3\_2, respectively.

### 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 5-2: State Change Report Decoding

Report String	State Name
SC 00	Idle
SC 01	Initialization
SC 02	Observation
SC 04	Correction
SC 08	Classification

Table 5-2: State Change Report Decoding

Report String	State Name
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:

```
HALT
```

### 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 5-3: 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 7 series device status report has the following format:

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

The Virtex-6 and Spartan-6 device status reports are virtually identical except that the Maximum Frame is reported as a 6-digit hex value and the SLR Number is not reported.

### 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 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. Controller reports physical frame address and linear frame address.

```
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}
```

The Virtex-6 and Spartan-6 device error detection reports are virtually identical except that frame addresses are reported as 6-digit hex values.

## 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.

The general form of the report for a correctable, non-essential event is:

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

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}
```

The general form of the report for an uncorrectable event, or when classification is disabled, is shorter. In these cases, the error is assumed to be essential because no information is available to suggest otherwise, and there is little to report:

SC 08  
FC 60

Bit 6, essential is set

---

## Systems

Although the soft error mitigation solution 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 controller and determines if additional actions are necessary (for example, reconfigure the device or reboot the application). The nature of application-level supervisory functions varies from design to design.

As noted previously, there is a small, yet finite possibility that the soft error mitigation solution is disrupted by a soft error. The solution has indicators of general health that the application-level supervisory function can elect to monitor:

- The controller heartbeat, `status_heartbeat`: This signal is a direct output from the soft error mitigation solution. This signal exhibits pulses which indicate the FPGA configuration system readback process is active. If, during the observation state, these pulses cease for no apparent reason, the application-level supervisory function should conclude that the FPGA configuration system readback process has experienced a fault. This is an uncorrectable, essential error.
- The hardware CRC failure indicator, `INIT_B` (7 series and Virtex-6 devices only): This signal is a direct output from the hardware readback process. If the readback process detects a hardware CRC failure, it asserts `INIT_B`. A transient assertion of `INIT_B` is expected during FPGA configuration and the controller initialization process, after error injections, and in some cases when soft error events occur. If, during observation state, `INIT_B` indicates an error and the controller does not transition into correction state after the expected duration of a complete readback cycle, the application-level supervisory function should conclude the controller has experienced a fault. This is an uncorrectable, essential error.

The recommended interface between the soft error mitigation solution and the application-level supervisory function for normal communication is the serial interface supported by the MON shim. Using the MON shim introduces a small amount of logic, but drastically reduces the number of I/O required. As a result, the MON shim offers higher reliability than using the direct logic signal based interfaces.

---

## Customizations

The system-level design example encapsulates the controller and various shims that serve to interface the controller to other devices. These shims can include I/O Pins, ChipScope, 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 user 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 user 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 ChipScope 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 57](#) and [Error Injection Interface, page 74](#).

### 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 59](#) 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. When making a FIFO replacement, note that an “empty” flag is the logical inverse of a “data present” flag.

## Replace with Alternate Function

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 59](#). The data exchanged over this interface is ASCII. For a summary of the status and command formats, see [Monitor Interface Commands, page 82](#) and [Monitor Interface Messages, page 83](#).

[Figure 5-24](#) 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.

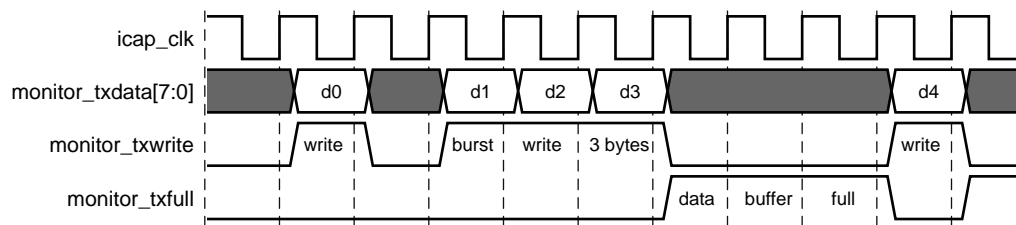


Figure 5-24: Monitor Interface Transmit Protocol

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 will stall, 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 5-25 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_rxddata[7:0]` and acknowledges reception by asserting `monitor_rxread` for a single clock cycle.

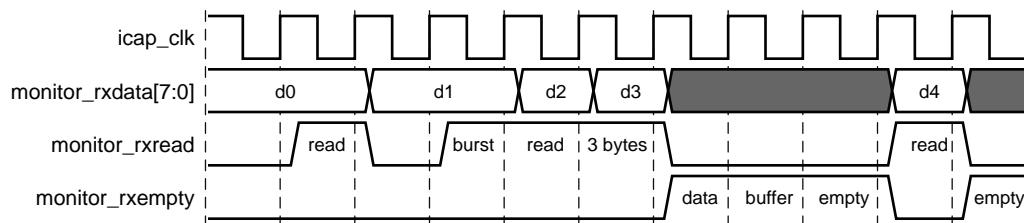


Figure 5-25: Monitor Interface Receive Protocol

The controller can perform burst reads by obtaining a data sequence from `monitor_rxddata[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.

## 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 may 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 the controller:

- Leave all controller monitor interface outputs "open" or "unconnected."
- Connect the controller's `monitor_txfull` input to logic zero.
- Connect the controller's `monitor_rxempty` input to logic one.
- Connect the controller's `monitor_rxddata[7:0]` input 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.

### Replace with Alternate Function

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 5-26 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.

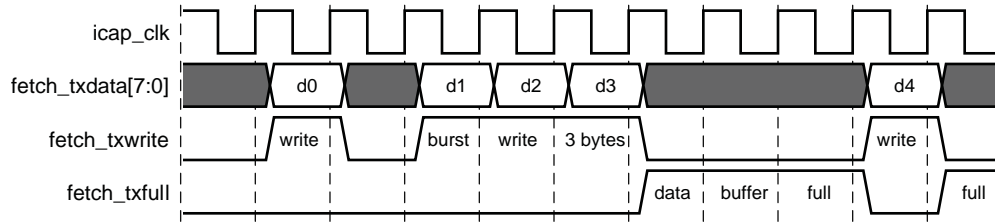


Figure 5-26: Fetch Interface Transmit Protocol

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 5-27 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_rxddata[7:0]` and acknowledges reception by asserting `fetch_rxread` for a single clock cycle.

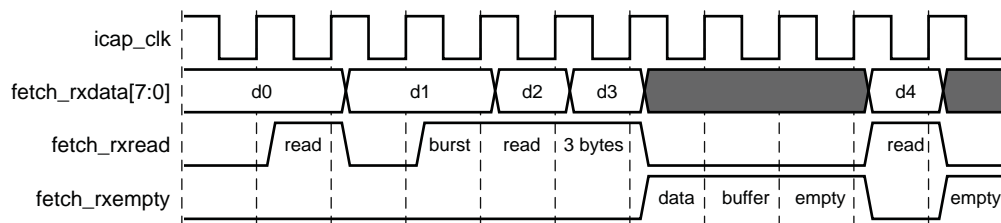


Figure 5-27: Fetch Interface Receive Protocol

The controller can perform burst reads by obtaining a data sequence from `fetch_rxddata[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 4](#).

During the initialization state, the controller purges the receive buffer by reading and discarding data until it is empty. The controller then performs two more reads “beyond empty.” The condition of a receive buffer read while the receive buffer is empty is a signal for the peripheral to reset itself. This provides a mechanism to synchronize the command-response protocol. 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.

---

## 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 bitgen 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.

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

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

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

In many cases (for example, LUTRAM functions) it is possible for the user design to implement data protection on these bits for purposes of soft error mitigation. Another

approach is to modify the user design to eliminate the use of features that introduce configuration memory masking.

## Virtex-6 FPGAs

Virtex-6 FPGAs implement fine grain masking at a resource level. This means individual resources, when configured for dynamic operation, have their configuration memory bits masked. Only the required memory bits are masked, without impacting unrelated memory bits. The masked bits are no longer monitored by the controller.

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

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

## Spartan-6 FPGAs

Spartan-6 FPGAs implement coarse grain masking at a frame level. This means individual resources, when configured for dynamic operation, result in one or more frames of configuration memory being masked. The masked bits are no longer monitored by the controller.

This over-masking can compromise error detection and error injection capability of the controller. The easiest way to avoid over-masking is to reduce the use of functions that involve dynamic memory bits. However, complete elimination may not be possible, and it can be undesirable because it prohibits use of powerful Spartan-6 FPGA features. Generally, a better approach is to understand the masking rules and then apply placement constraints to cluster resources that involve dynamic memory bits into the same configuration memory frames. Consult UG380, *Xilinx Spartan-6 FPGA Configuration User Guide* for additional information.

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 overwrite the contents of dynamic state elements. A side effect is that error injections into masked resources do not result in error detections, although the memory contents have changed.

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 estimated controller FIT. Although the impact is small, it is best to strive for high reliability unless it poses a significant burden.

When additional logic exists on the clock path (for example, clock management blocks, or logic-based clock division) care must be taken to guarantee by design that the maximum clock frequency 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 may 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.
- 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 will not exit the initialization state. Hardware-based evaluation of the controller behaviors is required. Alternatively, customers can use ISim Hardware Co-simulation to simulate their design.
- Use of bitstream security (encryption and authentication) is not supported by the controller.
- 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 device and standard SPI flash devices, level translation may be required in the design of the SPI memory system.
- ICAP Arbitration, ICAP Switchover, and Partial Reconfiguration 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.
- In implementations for Spartan-6 FPGAs, neither Suspend mode nor PLL DRP can be used.
- Controller implementations for Spartan-6 FPGAs operate only on soft errors in Type 0 configuration frames. See UG380, *Xilinx Spartan-6 FPGA Configuration User Guide* for information on Spartan-6 configuration frame types.

- Controller implementations for Virtex-6 FPGAs operate on soft errors in Type 0, Type 2, and Type 3 configuration frames. See UG360, *Xilinx Virtex-6 FPGA Configuration User Guide* for information on Virtex-6 configuration frame types.
- Controller implementations for 7 series FPGAs operate on soft errors in Type 0, Type 2, and Type 3 configuration frames. See UG470, *Xilinx 7 Series FPGA Configuration User Guide*, for information on 7 series configuration frame types.

# Constraining the Core

---

## 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 a user constraints file (UCF).

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

Constraints may 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 User Constraints File

Although the UCF delivered with each generated solution shares the same overall structure and sequence of constraints, the contents may vary based on options set at generation. The sections that follow define the structure and sequence of constraints using a Virtex-6 device implementation as an example.

### Device Selection Constraint

The first section of the UCF specifies the device for the implementation tools to target, including the part, package, and speed grade. The device in the UCF reflects the device chosen in the CORE Generator software project. An example device selection constraint is:

```
CONFIG PART = XC6VLX240T-FF1156-1 ;
```

Although the controller itself is designed to function in any of the supported devices, some details of the system-level example design depend on the selected part. For this reason, this

constraint should not be modified. To target a different device, return to the CORE Generator software project, change the device, and re-generate the solution.

## 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 UCF, this constraint is placed on the system-level design example clock input and propagated into the controller. The constraint is discussed in [Example Design Constraints](#).

The signal paths between the controller and the FPGA configuration system primitives must be considered as synchronous paths. By default, the paths between the ICAP 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 (when present), 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:

```
INST "example_cfg/example_frame_ecc" TPSYNC = FECC_SPECIAL;
TIMESPEC "TS_FECC_SYNC" = FROM "FECC_SPECIAL" TO FFS(*) 7000 ps;
TIMESPEC "TS_FECC_PADS" = FROM "FECC_SPECIAL" TO PADS(*) 20000 ps;
```

## 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:

```
NET "clk" IOSTANDARD=LVCOS25 | PERIOD=10000 ps;
```

The second group is for the Status Interface, applying an I/O standard and time name, so that an output timing constraint can be applied to the interface. The output timing constraint is set at two times the period constraint.

```
NET "status_initialization" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_observation" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_correction" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_classification" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_injection" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_uncorrectable" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_essential" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;
NET "status_heartbeat" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=STAPINS;

TIMEGRP "STAPINS" OFFSET = OUT 20000 ps AFTER "clk";
```

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

```
INST "example_mon/example_mon_sipo/sync_reg" IOB = TRUE;
INST "example_mon/example_mon_piso/pipeline_serial" IOB = TRUE;

NET "monitor_tx" IOSTANDARD=LVCOS25 | DRIVE=4 | SLEW=SLOW | TNM=SERPINS;
NET "monitor_rx" IOSTANDARD=LVCOS25 | TNM=SERPINS;

TIMEGRP "SERPINS" OFFSET = IN 20000 ps VALID 40000 ps BEFORE "clk";
TIMEGRP "SERPINS" OFFSET = OUT 20000 ps AFTER "clk";
```

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 time name, so that input and output timing constraints can be applied 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.

```
INST "example_ext/example_ext_byte/ext_c_ofd" IOB = TRUE;
INST "example_ext/example_ext_byte/ext_d_ofd" IOB = TRUE;
INST "example_ext/example_ext_byte/ext_q_ifd" IOB = TRUE;
INST "example_ext/ext_s_ofd" IOB = TRUE;

NET "external_c" IOSTANDARD=LVCOS25 | DRIVE=8 | SLEW=FAST | TNM=SPIPINS;
NET "external_d" IOSTANDARD=LVCOS25 | DRIVE=8 | SLEW=FAST | TNM=SPIPINS;
NET "external_s_n" IOSTANDARD=LVCOS25 | DRIVE=8 | SLEW=FAST | TNM=SPIPINS;
NET "external_q" IOSTANDARD=LVCOS25 | TNM=SPIPINS ;

TIMEGRP "SPIPINS" OFFSET = IN 20000 ps VALID 40000 ps BEFORE "clk";
TIMEGRP "SPIPINS" OFFSET = OUT 20000 ps AFTER "clk";
```

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 time name, so that an input timing constraint can be applied to the interface. The input timing constraint is set at two times the period constraint.

```
NET "inject_strobe" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[0]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[1]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[2]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[3]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[4]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[5]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[6]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[7]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[8]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[9]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
```

```

NET "inject_address[10]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[11]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[12]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[13]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[14]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[15]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[16]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[17]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[18]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[19]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[20]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[21]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[22]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[23]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[24]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[25]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[26]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[27]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[28]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[29]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[30]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[31]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[32]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[33]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[34]" IOSTANDARD=LVCOS25 | TNM=INJPINS;
NET "inject_address[35]" IOSTANDARD=LVCOS25 | TNM=INJPINS;

```

```

TIMEGRP "INJPINS" OFFSET = IN 20000 ps VALID 40000 ps BEFORE "clk";

```

The following constraints in the UCF implement an area group to place portions of the system-level design example into a bounded region of the selected device. The instances included in the area group depend on the options set at generation. The range values vary depending on device selection. The area group is defined so that component packing is closed to resources from outside the group, but the unused component locations are open to placement of unrelated logic.

The area group 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 area group forces tighter packing and generates a resource usage summary that is helpful in estimating the FIT of the system-level design example.

```

INST "example_wrapper/*" AREA_GROUP = "SEM_CONTROLLER" ;
INST "example_mon/*" AREA_GROUP = "SEM_CONTROLLER" ;
INST "example_ext/*" AREA_GROUP = "SEM_CONTROLLER" ;

AREA_GROUP "SEM_CONTROLLER" RANGE = SLICE_X84Y115:SLICE_X99Y124 ;
AREA_GROUP "SEM_CONTROLLER" RANGE = RAMB18_X4Y46:RAMB18_X4Y49 ;
AREA_GROUP "SEM_CONTROLLER" GROUP = CLOSED ;
AREA_GROUP "SEM_CONTROLLER" PLACE = OPEN ;

```

The final constraints in the UCF 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.

```
## NET "clk" LOC = " ";

## NET "external_c" LOC = " ";
## NET "external_d" LOC = " ";
## NET "external_q" LOC = " ";
## NET "external_s_n" LOC = " ";

## NET "monitor_tx" LOC = " ";
## NET "monitor_rx" LOC = " ";

## NET "status_initialization" LOC = " ";
## NET "status_observation" LOC = " ";
## NET "status_correction" LOC = " ";
## NET "status_classification" LOC = " ";
## NET "status_injection" LOC = " ";
## NET "status_uncorrectable" LOC = " ";
## NET "status_essential" LOC = " ";
## NET "status_heartbeat" LOC = " ";

## NET "inject_strobe" LOC = " ";
## NET "inject_address[0]" LOC = " ";
## NET "inject_address[1]" LOC = " ";
## NET "inject_address[2]" LOC = " ";
## NET "inject_address[3]" LOC = " ";
## NET "inject_address[4]" LOC = " ";
## NET "inject_address[5]" LOC = " ";
## NET "inject_address[6]" LOC = " ";
## NET "inject_address[7]" LOC = " ";
## NET "inject_address[8]" LOC = " ";
## NET "inject_address[9]" LOC = " ";
## NET "inject_address[10]" LOC = " ";
## NET "inject_address[11]" LOC = " ";
## NET "inject_address[12]" LOC = " ";
## NET "inject_address[13]" LOC = " ";
## NET "inject_address[14]" LOC = " ";
## NET "inject_address[15]" LOC = " ";
## NET "inject_address[16]" LOC = " ";
## NET "inject_address[17]" LOC = " ";
## NET "inject_address[18]" LOC = " ";
## NET "inject_address[19]" LOC = " ";
## NET "inject_address[20]" LOC = " ";
## NET "inject_address[21]" LOC = " ";
## NET "inject_address[22]" LOC = " ";
## NET "inject_address[23]" LOC = " ";
## NET "inject_address[24]" LOC = " ";
## NET "inject_address[25]" LOC = " ";
## NET "inject_address[26]" LOC = " ";
## NET "inject_address[27]" LOC = " ";
## NET "inject_address[28]" LOC = " ";
## NET "inject_address[29]" LOC = " ";
## NET "inject_address[30]" LOC = " ";
## NET "inject_address[31]" LOC = " ";
## NET "inject_address[32]" LOC = " ";
## NET "inject_address[33]" LOC = " ";
## NET "inject_address[34]" LOC = " ";
## NET "inject_address[35]" LOC = " ";
```

---

## 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.



# 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.

Xilinx completely verifies this LogiCORE IP product for production use in production Virtex-6 and Spartan-6 FPGA devices.

For the ISE Design Suite 14.1 release, this LogiCORE IP product targeting 7 series devices (excluding devices implemented with stacked silicon interconnect (SSI) technology) is in pre-production. Verification and validation activities are on-going and will be completed in a future release.

---

## 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

---

## 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 (Virtex-6 devices)
    - Complete testing across the supported device list (Spartan-6 devices)
    - Sample testing across the supported device list for all sub-families (7 series devices)
- 

## 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.

# Migrating

This appendix describes migrating from the previous version of the IP to the current version.

---

## Customization and Generation Changes

There are no customization or core generation changes for this version.

---

## Port Changes

A new input port was added to the Fetch Interface of the SEM Controller. This port is named `fetch_tbladdr[31:0]` and is only present in configurations of the controller that include the Fetch Interface. This port provides a mechanism to specify the starting address of the external data. By default, this port is tied to zero in the example design.

---

## Functionality Changes

There are no functionality changes for this version.

# Debugging

This appendix contains details about resolving problems and issues with the SEM core.

---

## Release Notes

Always check the online release notes for the most up to date information regarding any potential known issues discovered with the IP. The release notes are at:

[www.xilinx.com/support/answers/44541.htm](http://www.xilinx.com/support/answers/44541.htm)

---

## 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_V3_2
SC 01
FS 02
ICAP OK
RDBK OK
INIT OK
SC 02
O>
```

The first line lists the device and core version, and it varies depending on the target device used. For example, the Virtex-6 SEM controller would issue V6\_SEM\_V3\_2 while the Spartan-6 SEM controller would issue S6\_SEM\_V3\_2. 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 [Chapter 3, Customizing and Generating the Core](#). Also verify that the frequency of the actual clock provided to the SEM controller, coupled with the MON shim V\_ENABLE\_TIME parameter value, yield a standard baud rate and that the terminal program communication settings match the bit rate. This is described in [Equation 5-1](#) and [Equation 5-2](#).

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

```
X7_SEM_V3_2
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.
- Ensure that no other process is blocking the ICAP.
  - Ensure 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.

---

## 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.

# Additional Resources

---

## Xilinx Resources

For support resources such as answers, documentation, downloads, and forums, see the Xilinx Support website at:

[www.xilinx.com/support](http://www.xilinx.com/support).

For a glossary of technical terms used in Xilinx documentation, see:

[www.xilinx.com/company/terms.htm](http://www.xilinx.com/company/terms.htm).

---

## Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

---

## References

These documents provide supplemental material useful with this user guide:

1. UG470, *7 Series FPGAs Configuration User Guide*
2. UG360, *Xilinx Virtex-6 FPGA Configuration User Guide*
3. UG380, *Xilinx Spartan-6 FPGA Configuration User Guide*
4. UG116, *Xilinx Device Reliability Report*
5. WP286, *Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits*

---

## Technical Support

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

See the IP Release Notes Guide ([XTP025](#)) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

---

## Ordering Information

The core is provided under the [Xilinx End User License Agreement](#) and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the [SEM product page](#).

Contact your local Xilinx [sales representative](#) for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx [IP Center](#).

---

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
4/24/12	1.0	Initial Xilinx release. Replaces DS796, <i>LogiCORE IP Soft Error Mitigation Controller Data Sheet</i> , and UG764, <i>LogiCORE IP Soft Error Mitigation Controller User Guide</i> .

---

## Notice of Disclaimer

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 the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; 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 Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.