LogiCORE IP Soft Error Mitigation Controller v2.1

DS796 June 22, 2011 **Product Specification**

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. A soft error is an unintended change to the state of memory bits 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 time of 9 ms in mid-sized Virtex®-6 devices (XC6VLX240T using 100 MHz clock) and 15 ms in mid-sized Spartan®-6 devices (XC6SLX45T using 20 MHz clock).
- 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 or replace.
  - Correction by repair method is ECC algorithm based.
  - Correction by replace method is data re-load based.
- Using Xilinx Essential Bits technology, optional error classification to determine if a soft error has affected the function of the user design.
  - Increases uptime by avoiding disruptive recovery approaches for errors that have no real effect on design operation.
  - Reduces effective failures-in-time (FIT).
- Optional error injection to support evaluation of SEM Controller applications.

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family(1)	Virtex-6, Spartan-6				
Resources					
Configuration	**LUTs**	**FFs**	**I/O Pins**	**Block RAMs**	**F_{MAX}**
See Table 5, page 14 for more details.					
Provided with Core					
Documentation	Product Specification User Guide				
Design Files	Controller (NGC)				
Example Design	VHDL and Verilog				
Test Bench	None				
Constraints File	UCF				
Simulation Model	None(2)				
Tested Design Tools					
Design Entry Tools	ISE 13.2				
Simulation	Not Applicable(2)				
Synthesis Tools(3)	Synopsys Synplify PRO Xilinx XST				
Support					
Provided by Xilinx, Inc.					

1. For a complete listing of supported devices, see the release notes for this core. EasyPath devices are not compatible with the error correction by replace method.
2. Functional and Timing simulation of designs that include the controller is supported. However, it is not possible to observe the controller behaviors in simulation. Hardware-based evaluation is required. See Verification and Validation for more information.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Overview

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 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 controller endlessly loops, observing the integrated soft error detection status. When an ECC or CRC error is detected, the controller evaluates the situation to identify the Configuration Memory location involved.

Once this is complete, the controller may optionally correct the soft error by repairing it or by replacing the affected bits. The repair method uses active partial reconfiguration to perform a localized correction of Configuration Memory using a read-modify-write scheme. This method uses 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 controller.

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

When the controller is idle, there is an option to accept input from the user to inject errors into Configuration Memory. This function is useful for testing the integration of the controller into a larger system design. Using the error injection capability, system verification and validation engineers may construct test cases to ensure the complete system responds to soft error events as expected.

The 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.

Controller Interfaces

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

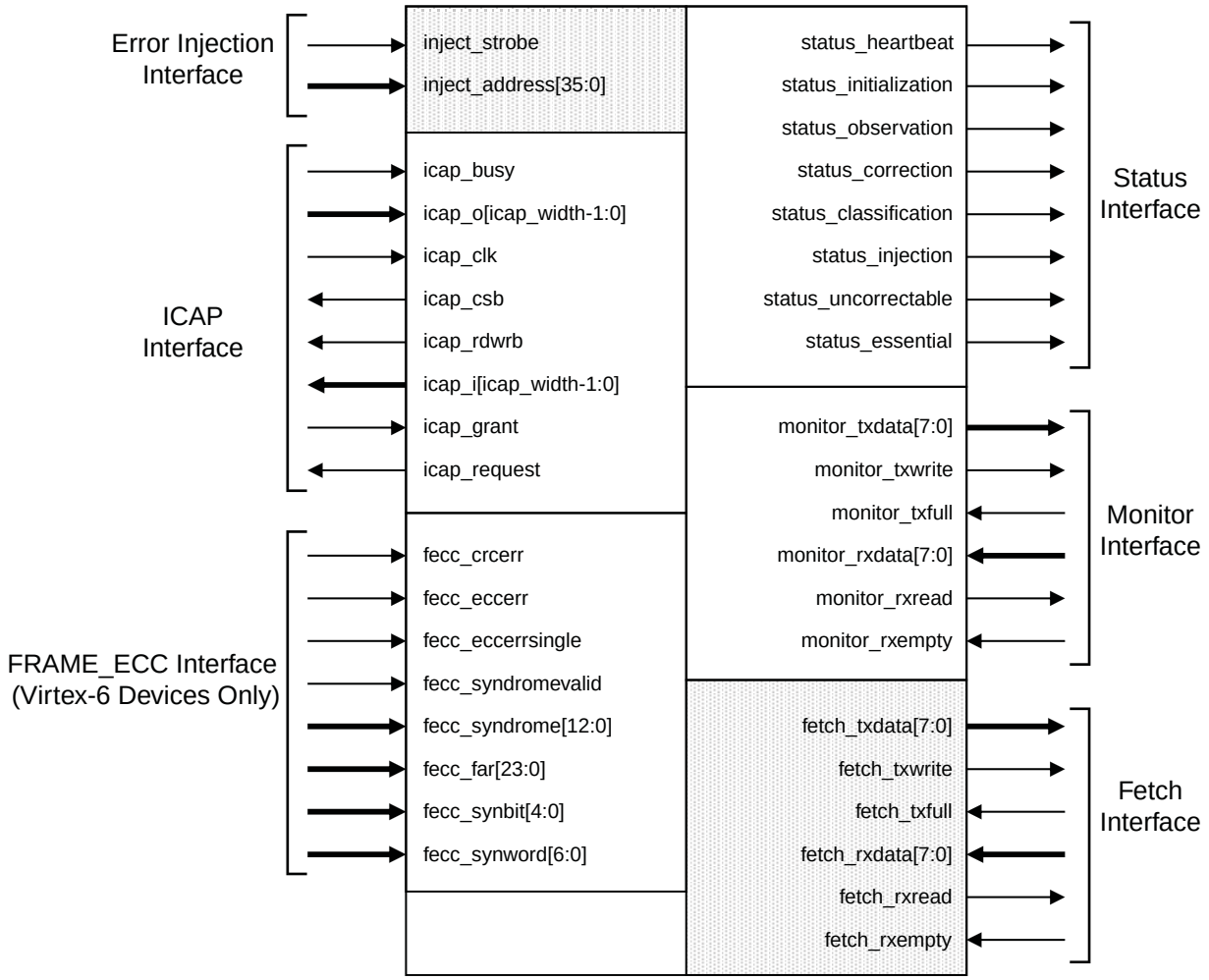

Figure 1: **SEM Controller Ports**

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 UG360, *Virtex-6 FPGA Configuration User Guide*, and UG380, *Spartan-6 FPGA Configuration User Guide*.

FRAME_ECC Interface (Virtex-6 Only)

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 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.

Status Interface

The `status_heartbeat` output provides an indication that the SEM Controller is active. Although the controller mitigates soft errors, it can also be disrupted by soft errors. For example, the controller clock may 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 states may 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 SEM Controller to inject a bit error into Configuration Memory.

The user must provide an error injection address and command on `inject_address[35:0]` and assert `inject_strobe` to indicate an error injection. Error injections may only be requested when the controller is idle.

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

Monitor Interface

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

Fetch Interface

The Fetch Interface provides a mechanism for the SEM 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.

Function

The initialization function of the SEM Controller is to bring itself into a known state. This is achieved by accessing control registers in the FPGA configuration system through the ICAP primitive. The controller must set up the readback process to detect errors. Error events will take one of the following forms:

- The most common error condition is a single bit error in a Configuration Memory frame indicated by a non-zero ECC syndrome with the single-error-detect bit set. If the readback pass is run to completion, the device CRC check will also fail. The readback process reports the complete frame and bit address of the error.
- The second most common error condition is a double bit error in a Configuration Memory frame, also indicated by a non-zero ECC syndrome with the single-error-detect bit not set. If the readback pass is run to completion, the device CRC check will fail. The readback process reports the frame address of the error.
- Uncommon errors involve three or more bits in a Configuration Memory frame. Many of these errors are indicated by a non-zero ECC syndrome, but it is also possible for them to generate ECC syndromes associated with a two-bit error, one-bit error, or no error at all. However, in virtually all cases these uncommon errors will cause the device CRC check to fail.
  - When the ECC syndrome aliases as a two-bit or one-bit error, the controller will treat the error as if it were a two-bit or one-bit error.
  - When the ECC syndrome aliases as no error, the controller will only observe the device CRC check to fail.

The controller will correct 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 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.

Special Case of CRC-Only Failure

- For the case of CRC-only failure, the error is signaled as uncorrectable.

The difference between repair and replace methods becomes clear when considering what happens in the uncommon case of errors of two or more bits. Such errors are not corrected by the repair method. On the other hand, if such errors are corrected by the replace method, the replacement data is always correct regardless of how many bits were corrected in the process, and the error is correctable.

The fundamental tradeoff between repair and replace methods is error correction success rate versus the cost of adding or increasing external data storage requirements. Highly demanding applications may warrant the cost of an increased error correction success rate.

If the controller has not been configured for classification, all errors are unconditionally signaled as essential. Otherwise, the controller uses the stored information about the error locations to perform a look-up in the essential bit data. The controller requests the essential bit data for the corrected frame from the Fetch Interface, and then determines if any of the bits corrected are essential. If one or more bits are essential, the error is signaled as essential. Otherwise, it is signaled as non-essential.

At the completion of correction and classification, the controller returns to observation to wait for the next event.

Error injections are performed by reading back the frame that is to have the error, inverting the desired bit, and writing back the frame. Construction of multi-bit error scenarios is accomplished by repeated single-bit error injections.

Controller Ports

Table 1 details the SEM Controller ports.

Table 1: **SEM Controller Ports**

Name	Sense	Direction	Description
Status Interface			
status_heartbeat	HIGH	OUT	The heartbeat signal is active while status_observation is true. This output will issue a single-cycle high pulse at least once every 128 clock cycles. This signal may be used to implement an external watchdog timer to detect “controller stop” scenarios that may occur if the controller or clock distribution is perturbed 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.
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 to inactive.
status_essential	HIGH	OUT	The essential signal is an error classification status signal. Prior to exiting the classification state, the controller will set 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 will set this signal to reflect the correctability of the error. Then, the controller exits correction state.
Error Injection Interface (Optional)			
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[35:0]	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.
ICAP Interface			
icap_busy	HIGH	IN	Receives BUSY output of ICAP.
icap_o[icap_width-1:0]	HIGH	IN	Receives O output of ICAP. The variable icap_width is equal to 32 for Virtex-6 devices and 16 for Spartan-6 devices.
icap_csb	LOW	OUT	Drives CSB input of ICAP.
icap_rdwrb	LOW	OUT	Drives RDWRB input of ICAP.
icap_i[icap_width-1:0]	HIGH	OUT	Drives I input of ICAP. The variable icap_width is equal to 32 for Virtex-6 devices and 16 for Spartan-6 devices.

Table 1: **SEM Controller Ports**

Name	Sense	Direction	Description
icap_clk	EDGE	IN	Receives the clock for the design. This same clock also must be applied to the CLK input of ICAP. The clock frequency must comply with the ICAP input clock requirements as specified in the target device data sheet.
icap_request	HIGH	OUT	This signal is reserved for future use. Leave this port OPEN.
icap_grant	HIGH	IN	This signal is reserved for future use. Tie this port to VCC.
FRAME_ECC Interface (Virtex-6 Devices Only)			
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[23:0]	HIGH	IN	Receives FAR output of FRAME_ECC.
fecc_synbit[4:0]	HIGH	IN	Receives SYNBIT output of FRAME_ECC.
fecc_synword[6:0]	HIGH	IN	Receives SYNWORD output of FRAME_ECC.
Monitor Interface			
monitor_txdata[7:0]	HIGH	OUT	Parallel transmit data from controller.
monitor_txwrite	HIGH	OUT	Write strobe that 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.
monitor_rxdata[7:0]	HIGH	IN	Parallel receive data from the shim (peripheral).
monitor_rxread	HIGH	OUT	Read strobe that 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 (Optional)			
fetch_txdata[7:0]	HIGH	OUT	Parallel transmit data from controller.
fetch _txwrite	HIGH	OUT	Write strobe that 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 that 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.

System-Level Design Example

To enable rapid understanding and integration, the SEM Controller is delivered within a framework called the system-level design example. This framework contains the controller, as well as a variety of “shims” to external devices. The framework also contains the user application module (userapp), which is a demonstration circuit functionally unrelated to soft error mitigation.

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.

System-Level Design Example Interfaces

Figure 2 shows the system-level design example ports. The ports are clustered into six groups. Shading indicates port groups that only exist in certain configurations.

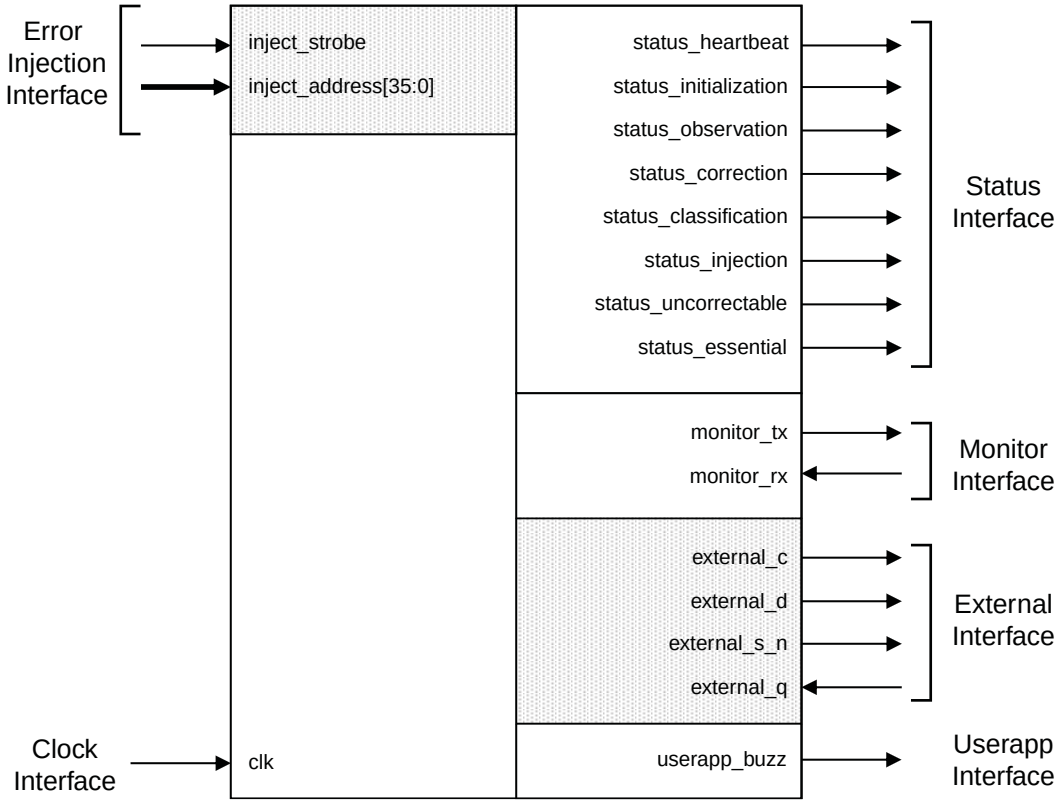

Figure 2: **Design Example Ports**

Clock Interface

The Clock Interface is used to provide a clock to the system-level design example. It is distributed on a global clock buffer to all synchronous logic in the design, including the ICAP and the controller.

Status Interface

The Status Interface is a direct pass-through from the controller Status Interface.

Error Injection Interface

The Error Injection Interface only exists on the system-level design example when error injection is enabled and the user has selected to control it with I/O Pins. In all other cases, this interface is removed. This interface is a direct pass-through to the controller Error Injection Interface. When error injection is enabled and the user has selected to control it with ChipScope, the HID Shim is generated and included in the design. The HID Shim incorporates ChipScope ICON and VIO cores to control error injection.

Monitor Interface

The Monitor Interface is a modified pass-through of the controller Monitor Interface. The system-level design example contains a simple UART that transforms the byte-wide controller Monitor Interface into an RS-232 compatible serial interface. This UART implementation is called the MON Shim.

External Interface

The External Interface is derived from the controller Fetch Interface. The system-level design example contains a simple SPI Master that transforms the byte-wide controller Fetch Interface into a SPI-compatible serial interface. This SPI Master implementation is called the EXT Shim.

Userapp Interface

The Userapp Interface is a direct pass-through from the userapp. When the userapp detects an error condition in itself, it asserts an output. This output is for demonstration purposes only and may be connected to an LED or a buzzer.

System-Level Design Example Function

The system-level design example is a wrapper for the SEM Controller and the shims to external devices. It does not implement any functionality other than to integrate the pieces of the complete solution. Figure 3 is a diagram of the system-level design example. The grey blocks are optional and present when required to support the controller features. Note that for Spartan-6 devices, the FRAME_ECC logic resides in the controller.

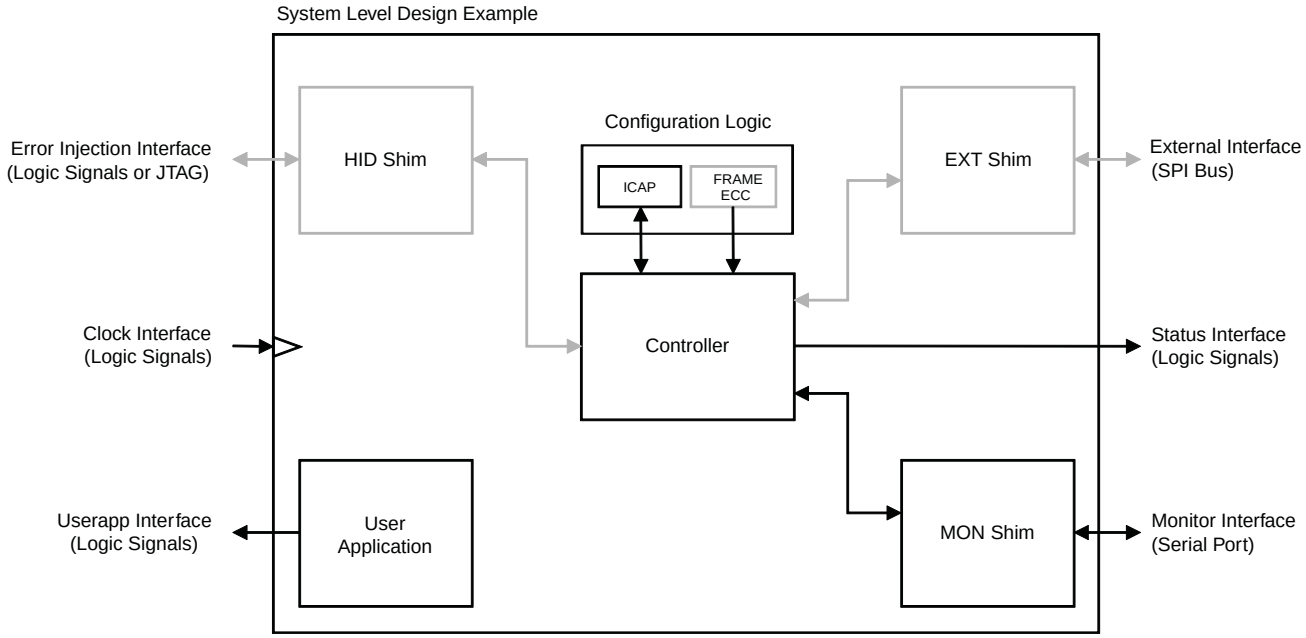

Figure 3: **Design Example Block Diagram**

System-Level Design Example Ports

Table 2 details the design example ports.

Table 2: **Design Example Ports**

Name	Sense	Direction	Description
Clock Interface			
clk	EDGE	IN	Input clock. The entire design is synchronous to this clock. The clock frequency must comply with the ICAP input clock requirements as specified in the target device data sheet. Frequencies below 8 MHz are restricted from use to ensure correct timing of the Monitor Interface.
Status Interface			
status_heartbeat	HIGH	OUT	The heartbeat signal is active while status_observation is true. This output will issue a single-cycle high pulse at least once every 128 clock cycles. This signal may be used to implement an external watchdog timer to detect “controller stop” scenarios that may occur if the controller or clock distribution is perturbed 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.
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 to inactive.
status_essential	HIGH	OUT	The essential signal is an error classification status signal. Prior to exiting the classification state, the controller will set 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 will set this signal to reflect the correctability of the error. Then, the controller exits correction state.
Monitor Interface			
monitor_tx	HIGH	OUT	The monitor_tx signal is an RS-232 compatible status transmission port operating at 9600-8-N-1 with no flow control. Only ASCII characters are sent.
monitor_rx	HIGH	IN	The monitor_rx signal is an RS-232 compatible command reception port operating at 9600-8-N-1 with no flow control. Only ASCII characters are received.
Userapp Interface			
userapp_buzz	HIGH	OUT	This signal is used to indicate the detection of an error in the user application. It is for demonstration purposes only and has no specific use requirement.
External Interface (Optional)			
external_c	HIGH	OUT	The external_c signal is the SPI Bus clock for an external SPI Flash, which can be used to store data for the controller.
external_d	HIGH	OUT	The external_d signal is the SPI Bus “master out, slave in” signal for an external SPI Flash, which can be used to store data for the controller.
external_s_n	LOW	OUT	The external_s_n signal is the SPI Bus select for an external SPI Flash, which can be used to store data for the controller.

Table 2: **Design Example Ports *(Cont'd)***

Name	Sense	Direction	Description
external_q	HIGH	IN	The external_q signal is the SPI Bus "master in, slave out" signal for an external SPI Flash, which can be used to store data for the controller.
Error Injection Interface (Optional)			
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[35:0]	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.

Additional Information for I/O Pin Interfaces

Table 3 provides additional details about the design example's I/O Pin interfaces.

Table 3: **Design Example I/O Pin Interface Details**

Name	Driver Type	Pull	Load	Supply	SelectIO Mode
Clock Interface					
clk	CMOS	NONE	N/A	+2.5V	LVCMOS25
Status Interface					
status_heartbeat	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_initialization	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_observation	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_correction	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_classification	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_injection	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_essential	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
status_uncorrectable	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4
Monitor Interface					
monitor_tx	CMOS	NONE	10 pF	+2.5V	LVCMOS25_S_4
monitor_rx	CMOS	NONE	N/A	+2.5V	LVCMOS25
External Interface (Optional)					
external_c	CMOS	NONE	10 pF	+2.5V	LVCMOS25_F_8
external_d	CMOS	NONE	10 pF	+2.5V	LVCMOS25_F_8
external_s_n	CMOS	NONE	10 pF	+2.5V	LVCMOS25_F_8
external_q	CMOS	NONE	N/A	+2.5V	LVCMOS25
Error Injection Interface (Optional)					
inject_strobe	CMOS	NONE	N/A	+2.5V	LVCMOS25
inject_address[35:0]	CMOS	NONE	N/A	+2.5V	LVCMOS25
Userapp Interface					
userapp_buzz	CMOS	NONE	UNSPECIFIED	+2.5V	LVCMOS25_S_4

Applications

Although the SEM Controller can operate autonomously, most applications will 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 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 controller required? If the controller is required, what features should be used?

When the controller is the best choice for the application, Xilinx recommends that the controller is used as provided, including the system-level design example components for interfacing the controller to external devices. However, these interfaces may be modified if required for the application.

Monitor Interface and Communication

The MON Shim implements a simple UART that bridges between the SEM Controller Monitor Interface and a device with an RS-232 port. If RS-232 communication takes place externally, it may be necessary to use external RS-232 transceivers to attain compatibility with external devices using high amplitude signals.

The MON Shim may be modified to change the supported bit rate, or the shim can be entirely replaced with a different bridge implementing another form of inter-process communication (for example, FIFOs or mailboxes).

Fetch Interface and Data Access

The EXT Shim implements a simple SPI Master that bridges between the SEM Controller Fetch Interface and a SPI Flash device. As SPI communication is expected to take place externally, it may be necessary to use external level translators to attain compatibility with external devices.

The EXT Shim may be modified to support different commands or the shim can be entirely replaced with a different bridge implementing another form of storage system (for example, FIFOs or parallel flash).

The presence of the EXT Shim in the generated core depends on the data storage requirements of the controller as it is configured. Error classification requires external storage of essential bit data. Error correction by replace requires external storage of configuration frame data. When the controller is configured to support one or both of these features, external storage is required.

When the EXT Shim is generated, the system-level design example illustrates how to implement a SPI Flash memory system consisting of a single SPI Flash device. Table 4 lists a recommended SPI Flash device for each supported device and external storage requirement.

Table 4: **External Storage Requirements**

FPGA Device	Error Classification Only	Error Correction by Replacement Only	Error Classification with Error Correction by Replacement
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

Table 4: **External Storage Requirements *(Cont'd)***

FPGA Device	Error Classification Only	Error Correction by Replacement Only	Error Classification with Error Correction by Replacement
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
XC6VSX475T	128 Mbit	128 Mbit	256 Mbit
XC6SLX4	4 Mbit	4 Mbit	8 Mbit
XC6SLX9	4 Mbit	4 Mbit	8 Mbit
XC6SLX16	4 Mbit	4 Mbit	8 Mbit
XC6SLX25T	8 Mbit	8 Mbit	16 Mbit
XC6SLX45T	16 Mbit	16 Mbit	32 Mbit
XC6SLX75T	16 Mbit	16 Mbit	32 Mbit
XC6SLX100T	32 Mbit	32 Mbit	64 Mbit
XC6SLX150T	32 Mbit	32 Mbit	64 Mbit

Figure 4 shows the connectivity between an FPGA device 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 device or I/O bank voltage.

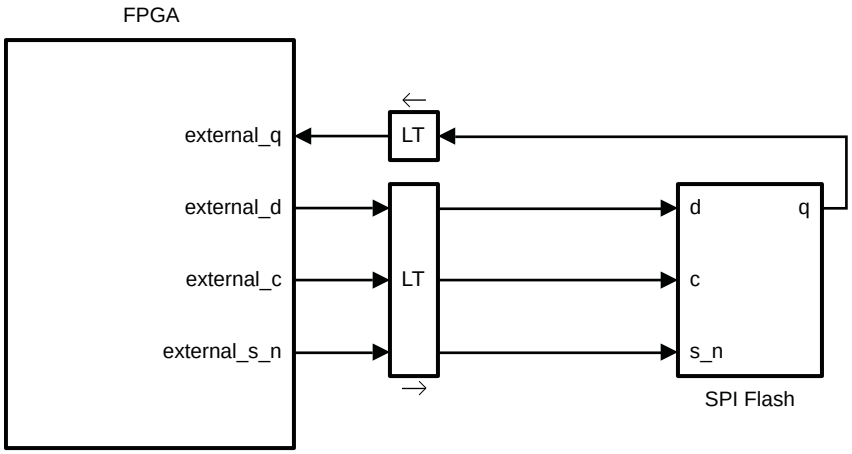

Figure 4: **FPGA Device Connectivity with SPI Flash**

The level translators, if required, must exhibit low propagation delay to maximize the cycle time of the memory system. The general expression of the timing budget is documented in UG764, *LogiCORE IP Soft Error Mitigation Controller User Guide*. This external storage system is additional to, and independent from, the FPGA device configuration method used to load the bitstream into the device.

Resource Requirements

Table 5 describes the resource utilization for supported devices.

Table 5: **Resource Utilization(1)(2)**

FPGA Device	Fmax	IP Core Configuration	LUTs	FFs	I/Os	Block RAMs
Virtex-6 (All)	100 MHz	Complete solution with no optional features	399	307	11	3 RAMB18
		Complete solution with all optional features	455	373	52	3 RAMB18
Virtex-6 -1L (All)	60 MHz	Same as Virtex-6				
Spartan-6 XC6SLX4	20 MHz	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)	12 MHz	Complete solution with no optional features	782	380	11	18 RAMB16
		Complete solution with all optional features	834	446	52	18 RAMB16
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 (All)	4 MHz	Same as Spartan-6				

1. The complete solution is the controller and the example design, which are intended to be used together.
2. The Error Injection Interface is connected to I/Os; use of ChipScope increases LUTs/FFs but decreases I/Os.

Verification and Validation

The controller and example design are verified together, using several methods including an automated hardware testbench 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.

There is currently no support for simulation of the solution. Simulation with a library-based netlist of the controller will properly compile and run, but the controller will not exit the initialization state. Alternatively, customers can use ISim Hardware Co-simulation to simulate their design. Please contact Xilinx for more information.

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

For IDS 13.2 release, this LogiCORE IP product targeting Spartan-6 is in pre-production. Verification and validation activities are on-going and will be completed in a future release.

Support

Xilinx provides technical support for this LogiCORE 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*.

Ordering Information

The Soft Error Mitigation Controller is included at no additional charge with the Xilinx ISE® Design Suite and is provided under the terms of the Xilinx End User License Agreement. The core is generated using the ISE Design Suite CORE Generator software.

For more information, please visit the SEM Controller product page.

Please contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE modules and software. Information about additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
09/21/2010	1.0	Initial Xilinx release.
12/14/2010	2.0	Updated core to v1.2 and ISE to v12.4.
03/01/2011	3.0	Updated core to v1.3 and ISE to v13.1.
06/22/2011	4.0	Updated core to v2.1 and ISE Design Suite to v13.2. Added support for Spartan-6 devices.

Notice of Disclaimer