



SEDC Controller IP

IP Version: v1.1.0

User Guide

FPGA-IPUG-02290-1.2

December 2025

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language [FAQ 6878](#) for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Contents	3
Abbreviations in This Document.....	7
1. Introduction	8
1.1. Overview of the IP	8
1.2. Quick Facts	8
1.3. IP Support Summary	9
1.4. Features	10
1.5. Licensing Information.....	10
1.5.1. Ordering Part Number.....	10
1.6. Minimum Device Requirements	10
1.7. Naming Conventions	11
1.7.1. Nomenclature.....	11
1.7.2. Signal Names	11
1.7.3. Attribute Names.....	11
2. Functional Description.....	12
2.1. IP Architecture Overview	12
2.1.1. SEDC Hard Block Overview.....	13
2.1.2. SEDC Controller IP Configuration Modes	14
2.1.3. SEDC Controller IP Flow.....	15
2.2. Clocking	16
2.3. Reset	17
2.4. User Interfaces	18
3. IP Parameter Description.....	19
4. Signal Description	20
4.1. Clock Interface	20
4.2. Reset Interface	20
4.3. Control Interface	20
4.4. Monitor Interface.....	22
5. Designing with the IP.....	23
5.1. Generating and Instantiating the IP	23
5.1.1. Generated Files and File Structure	25
5.2. Design Implementation	25
5.3. Specifying the Strategy.....	25
5.4. Running Functional Simulation	25
5.4.1. Simulation Results	28
5.4.2. Limitations of the SEDC Hard Block Simulation Model	29
5.4.3. Testbench Files and Structure	29
5.4.4. Input Stimulus Patterns.....	31
6. Debugging.....	33
7. Design Considerations	34
7.1. Asynchronous Reset (arst_i) Timing Requirements	34
7.2. Assert and Hold auto_correct_i High throughout a Complete SEDC Scan	34
7.3. Assert and Hold continuous_i High throughout Consecutive SEDC Scans	34
7.4. Process Error Information Only when status_update_o Asserts	34
7.5. Availability of Clock Divider Value after Initialization	35
7.6. Add Debouncer on Signal Driven by Mechanical Switch/Button	35
7.7. Implement Reset Synchronizer for Asynchronous Reset Signal.....	35
8. Known Issues	36
8.1. Incorrect Clock Divider Value in Simulation	36
8.2. SEDC Unable to Detect Errors After 1-bit or 2-bit SEI Bitstream Injection.....	36
8.3. Error Bit and Region Location Mismatch Between Hardware and Radiant SEI Bitstream Profile	36
Appendix A. Resource Utilization	38

Appendix B.	Walkthrough of Example Simulation Waveforms	39
B.1.	Port-Driven Dynamic Mode	39
B.2.	Continuous Mode.....	42
B.2.1.	Scenario 1: 1-bit Correctable Error in Continuous Mode (Auto-Correction Mode = ON)	42
B.2.2.	Scenario 2: Multi-bit Error in Continuous Mode (Auto-Correction Mode = ON)	44
Appendix C.	Example of Including +define+LSCC_SEDC_CONTROLLER_RTL_SIM in Generated *.f File	46
Appendix D.	Example Verilog/SystemVerilog Code on Asynchronous Reset Timing Requirements	47
Appendix E.	Example Verilog/SystemVerilog Code for Debouncer	49
Appendix F.	Example Verilog/SystemVerilog Code for Reset Synchronizer	50
References		51
Technical Support Assistance		52
Revision History.....		53

Figures

Figure 2.1. SEDC Controller IP Block Diagram	12
Figure 2.2. SEDC Hard Block System Block Diagram	13
Figure 2.3. SEDC Controller IP Flowchart	15
Figure 2.4. SEDC Controller IP Clock Domain Block Diagram	16
Figure 2.5. SEDC Controller IP Resets Block Diagram	17
Figure 5.1. Module/IP Block Wizard	23
Figure 5.2. IP Configuration	24
Figure 5.3. Check Generated Result	24
Figure 5.4. Simulation Wizard	26
Figure 5.5. Add and Reorder Source	26
Figure 5.6. Parse HDL files for Simulation	27
Figure 5.7. Summary	27
Figure 5.8. Simulation Waveform	28
Figure 5.9. Simulation Completion Log	28
Figure 5.10. Testbench Structure	30
Figure 8.1. Combination of 1-bit and 2-bit SEI Bitstreams	36
Figure 8.2. Example of 1-bit SEI Bitstream	36
Figure 8.3. Reveal Analyzer Waveform for 1-bit SEI Bitstream	37
Figure B.1. Asynchronous Reset and Triggering of SEDC Scan in Continuous Mode	39
Figure B.2. 1-bit Error Occurrence and Manual Error Correction/Resume Scan	39
Figure B.3. CRC Error Occurrence and Auto Resume Scan	40
Figure B.4. End of First SEDC Scan and Triggering/1-bit Error Occurrence of Second SEDC Scan in Continuous Mode	40
Figure B.5. End of Second SEDC Scan and Triggering/Error Occurrences/End of Third SEDC Scan in Continuous and Automatic Error Correction Modes	40
Figure B.6. Triggering/Error Occurrences/End of SEDC Scan in One-Shot Mode	41
Figure B.7. Aborting of SEDC Scan	41
Figure B.8. Asynchronous Reset De-assertion and Triggering of SEDC Scan in Continuous Mode (1-bit Error)	42
Figure B.9. 1-bit Error Detection and Auto-Correction	42
Figure B.10. CRC Error Occurrence and End of First SEDC Scan	43
Figure B.11. Second SEDC Scan in Continuous Mode	43
Figure B.12. Asynchronous Reset De-assertion and Triggering of SEDC Scan in Continuous Mode (Multi-bit Error)	44
Figure B.13. Multi-bit Error Detection and Resume Scan	44
Figure B.14. CRC Error and End of First SEDC Scan (Multi-bit Error)	45
Figure B.15. Second SEDC Scan in Continuous Mode (Multi-bit Error)	45

Tables

Table 1.1. Summary of the SEDC Controller IP	8
Table 1.2. SEDC Controller IP Support Readiness	9
Table 2.1. SEDC Controller IP Modules	12
Table 2.2. SEDC Controller IP Configuration Modes	14
Table 2.3. SEDC Controller IP Processes	15
Table 2.4. SEDC_CLK Frequencies with Different Clock Divider Values	16
Table 2.5. User Interfaces	18
Table 3.1. SEDC Controller IP Attributes	19
Table 4.1. Clock Port	20
Table 4.2. Reset Ports	20
Table 4.3. Control Ports	20
Table 4.4. Monitor Ports	22
Table 5.1. Generated File List	25
Table 5.2. Testbench Components and File List	29
Table 5.3. Input Stimulus Description	31
Table A.1. Resource Utilization	38

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
ASCII	American Standard Code for Information Interchange
ASR	Address Shift Register
CRAM	Configuration Random Access Memory
CRC	Cyclic Redundancy Check
CRC32	Cyclic Redundancy Check – 32 bits
DSR	Data Shift Register
DUT	Design Under Test
EBR	Embedded Block RAM
ECC	Error Correction Code
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose I/O
GUI	Graphical User Interface
HDL	Hardware Description Language
I/O	Input/Output
IP	Intellectual Property
LMMI	Lattice Memory Mapped Interface
LSE	Lattice Synthesis Engine
LUT	Lookup Table
OSC	Oscillator
PDC	Post-Synthesis Design Constraint
PLL	Phase Locked Loop
POR	Power On Reset
RAM	Random Access Memory
RTL	Register Transfer Level
SEC	Soft Error Correction
SED	Soft Error Detection
SEDC	Soft Error Detection/Correction
SEU	Single Event Upset
TMR	Triple Modular Redundancy

1. Introduction

Soft error detection (SED) and soft error correction (SEC), collectively referred to as soft error detection and correction (SEDC), are features that enable the detection and correction of soft errors in the configuration memory of an FPGA device. This document describes the SEDC Controller IP.

1.1. Overview of the IP

The SEDC Controller IP is designed to ensure the effective and reliable operation of the SEDC system by integrating additional soft logic around the SEDC hard block. This integration simplifies the interface and enhances overall functionality, making it easier for users to implement and benefit from advanced error detection and correction features. The IP manages state transitions, drives inputs, captures outputs, and provides internal feedback, all while ensuring robust performance and high reliability through the implementation of triple modular redundancy (TMR).

This IP is suitable for applications requiring high reliability and robust error detection and correction, such as:

- Aerospace and defense: Ensuring system reliability in harsh environments
- Automotive: Enhancing safety and reliability in critical systems
- Industrial automation: Maintaining system integrity in industrial control systems
- Medical devices: Providing reliable operation in life-critical applications
- Telecommunications: Ensuring data integrity in communication systems

1.2. Quick Facts

Table 1.1. Summary of the SEDC Controller IP

IP Requirements	Supported Devices	Lattice Avant™, Certus™-N2
	IP Changes	Refer to the SEDC Controller IP Release Notes (FPGA-RN-02081) .
Resource Utilization	Supported User Interface	General purpose I/O (GPIO)
	Resources	Refer to Appendix A. Resource Utilization .
Design Tool Support	Lattice Implementation	IP Core v1.0.0 – Lattice Radiant™ Software 2025.1 IP Core v1.1.0 – Lattice Radiant Software 2025.1.1
	Synthesis	Synopsys® Synplify Pro® for Lattice
	Simulation	Refer to the Lattice Radiant Software User Guide for the list of supported simulators.

1.3. IP Support Summary

Table 1.2. SEDC Controller IP Support Readiness

IP Version	Radiant Version	Synthesis		Mapping	Place and Route	Timing	Bitstream Generation	Simulation			
		Synplify Pro	LSE					RTL	Post-Synthesis	Post-Route Gate-Level	Post-Route Gate-Level and Timing
1.0.0	2025.1	Yes	—	Yes	Yes	Closed at 80 MHz	Yes	Yes	Yes	Yes	Yes

1.4. Features

Key features of the SEDC Controller IP include:

- Soft IP features (soft logic)
 - Modes: Supports four modes of operation through combinations of the following:
 - One-shot or continuous scanning
 - No error correction or automatic error correction
 - Finite state machine (FSM): Controls the system's operational states and ensures smooth transitions between them
 - Signal management: Efficiently handles the signals sent to and received from the SEDC hard block, ensuring seamless communication
 - Error and status reporting: Monitors the SEDC hard block output ports to provide accurate error and status reporting
 - Simulation support: Supports error injection for easy simulation of system response
 - Triple modular redundancy (TMR): Ensures high reliability by applying TMR to critical registers, protecting against errors
 - Simplified interfaces: Simplifies the Lattice memory mapped interface (LMMI) with internal translation, making it easier for users to implement and interact with the SEDC hard block
- Hard IP features (SEDC hard block)
 - Frame by frame SED check
 - Multiple regions (four regions as hardware default) run in parallel for fast SED and SEC performance
 - 1-bit and multi-bit error detection
 - Error correction code (ECC) to correct 1-bit errors at the frame level
 - 32-bit cyclic redundancy check (CRC32) calculation on the entire configuration RAM (CRAM) in parallel with ECC
 - Programmable SED clock with a wide frequency range
 - Force error capability for system-level simulation

1.5. Licensing Information

The SEDC Controller IP is provided at no additional cost with the Lattice Radiant software.

1.5.1. Ordering Part Number

The SEDC Controller IP does not require an ordering part number.

1.6. Minimum Device Requirements

The minimum device requirements for the SEDC Controller IP are as follows:

- Supports SEDC.
- Meets the resource utilization as captured in [Table A.1](#). Resource utilization for soft logic (LUTs and registers) by this IP is relatively small and primarily dependent on logic optimization by synthesis and place-and-route. Therefore, Lattice does not expect significant differences in resource utilization across different devices.

1.7. Naming Conventions

1.7.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.7.2. Signal Names

Signal names that end with:

- `_n` are active low signals (asserted when value is logic 0)
- `_i` are input signals
- `_o` are output signals

1.7.3. Attribute Names

Attribute names in this document are formatted in title case and italicized (*Attribute Name*).

2. Functional Description

2.1. IP Architecture Overview

Figure 2.1 shows the SEDC Controller IP block diagram. The IP includes several modules as described in Table 2.1. Additionally, an inverter is implemented on the internal connection between the top-level input port `arst_i` and the `LMMIRESET_N` port of the SEDC hard block. This inverter ensures that the appropriate signal level is presented at the SEDC hard block so that reset operations of all modules in the IP are aligned.

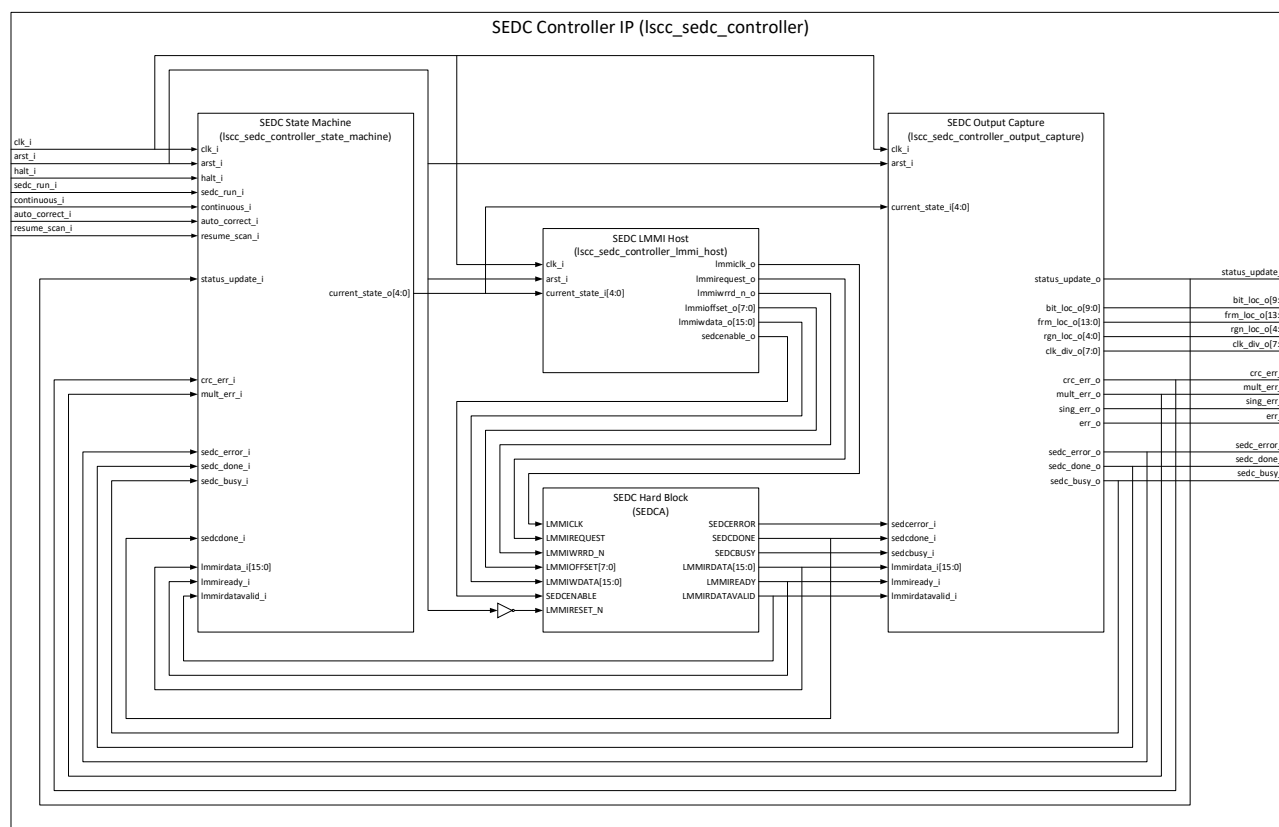


Figure 2.1. SEDC Controller IP Block Diagram

Table 2.1. SEDC Controller IP Modules

Module	Module Name in HDL	Description
SEDC State Machine	<code>lscsc_controller_state_machine</code>	This module contains the FSM that performs state transition based on user input signals and/or feedback signals from the SEDC output capture module and SEDC hard block. Internal registers are implemented with TMR.
SEDC LMMI Host	<code>lscsc_controller_lmli_host</code>	This module drives the inputs of the SEDC hard block based on the state of the FSM. Internal registers are implemented with TMR.
SEDCA Primitive	<code>SEDCA</code>	Primitive representing the SEDC hard block. Refer to the SEDC Hard Block Overview section for more information.
SEDC Output Capture	<code>lscsc_controller_output_capture</code>	This module monitors and captures the output signals from the SEDC hard block into internal registers and reports out the signals to the top-level output ports as necessary, based on the state of the FSM. The module also provides feedback signals to the state machine. Internal registers are implemented with TMR.

2.1.1. SEDC Hard Block Overview

Figure 2.2 shows the system-level view of the SEDC hard block. The SEDC hard block is part of the configuration (sysCONFIG) block in supported FPGA devices. Configuration data is divided into frames across multiple regions, allowing the FPGA device to be programmed either as a whole or in specific regions. The SED hardware reads serial data from the FPGA device configuration memory frame-by-frame in the background while the device is in user function mode and performs ECC calculations on every frame of configuration data.

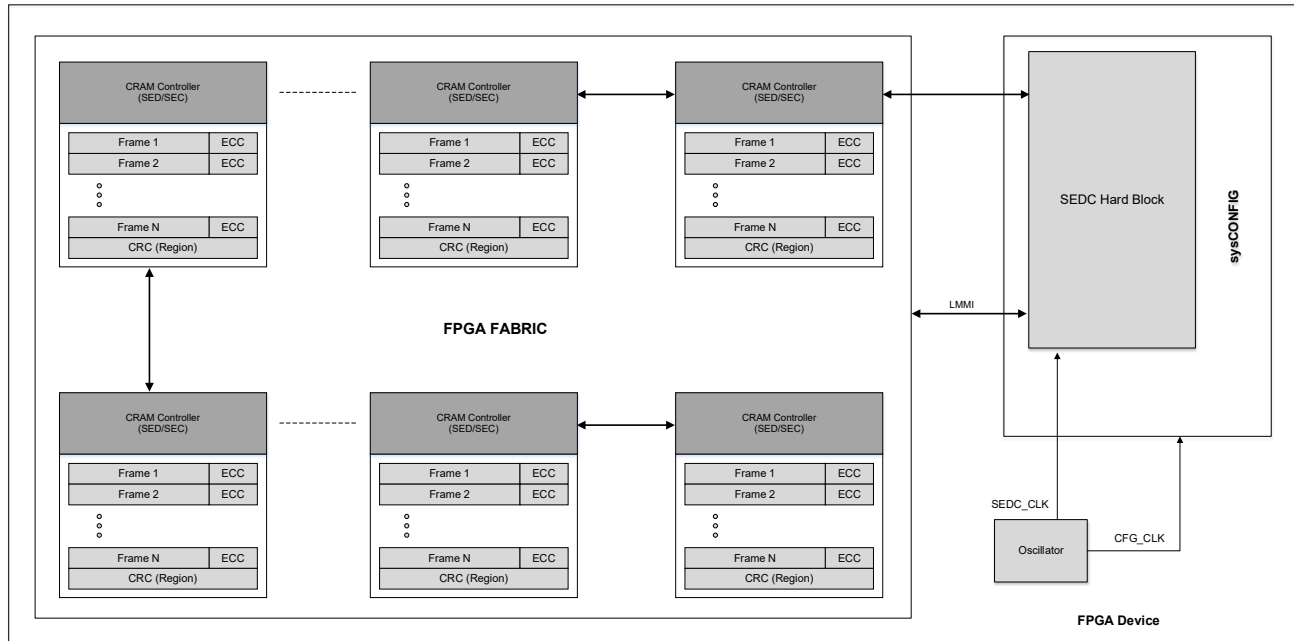


Figure 2.2. SEDC Hard Block System Block Diagram

When a 1-bit error is detected, an error indicator for 1-bit error is generated and the SED resumes operation. If SEC is enabled, the 1-bit error is corrected before the SED resumes operation. The corrected value is rewritten to the frame using ECC information. If multiple 1-bit errors are detected within a frame of configuration data, an error indicator for multi-bit error is generated.

In parallel, cyclic redundancy check (CRC) is calculated for the entire CRAM content along with ECC. After ECC is calculated on all frames of the configuration data, CRC is calculated for the configuration data in the entire device. Full-chip CRC and frame-by-frame ECC calculations do not include the embedded block RAM (EBR). EBRs provide a separate and optional ECC for SED or SEC of the EBR content. Distributed RAM data stored in the CRAM are masked during SED or SEC because RAM content may change during user operation and hence cannot be covered by the SEDC hard block without generating false SEDC errors. The distributed RAM enable bit set in CRAM is covered during SED or SEC.

2.1.2. SEDC Controller IP Configuration Modes

The SEDC Controller IP has four different configuration modes as described in [Table 2.2](#). The SEDC Controller IP only configures the SEDC hard block in one-shot and no auto-correction modes. All other modes are supported through soft logic around the SEDC hard block.

Table 2.2. SEDC Controller IP Configuration Modes

SEDC Mode	Error Correction Mode	Description
Continuous Mode	Auto-Correction Mode	<p>The SEDC Controller IP continuously scans frames. Scanning continues regardless of any errors detected.</p> <p>If a 1-bit error is detected, the IP notifies the user about the error, corrects the error automatically, and resumes scanning frames (without re-checking whether error is corrected) immediately after the correction.</p> <p>If a multi-bit or CRC error is detected, the IP continues scanning frames without halting and notifies the user about the error.</p>
Continuous Mode	No Auto-Correction Mode	<p>The SEDC Controller IP continuously scans frames. Scanning halts only when a 1-bit error is detected and continues after manual intervention.</p> <p>If a 1-bit error is detected, the IP halts and notifies the user about the error. The IP only resumes scanning frames after the error is corrected through manual intervention.</p> <p>If a multi-bit or CRC error is detected, the IP continues scanning frames without halting and notifies the user about the error.</p>
One-Shot Mode	Auto-Correction Mode	<p>The SEDC Controller IP performs a one-time scan of all frames. Scanning continues regardless of any errors detected until all frames are scanned.</p> <p>If a 1-bit error is detected, the IP notifies the user about the error, corrects the error automatically, and resumes scanning frames (without re-checking whether error is corrected) immediately after the correction.</p> <p>If a multi-bit or CRC error is detected, the IP continues scanning frames without halting and notifies the user about the error.</p>
One-Shot Mode	No Auto-Correction Mode	<p>The SEDC Controller IP performs a one-time scan of all frames. Scanning halts only when a 1-bit error is detected and continues after manual intervention until all frames are scanned.</p> <p>If a 1-bit error is detected, the IP halts and notifies the user about the error. The IP only resumes scanning frames after the error is corrected through manual intervention.</p> <p>If a multi-bit or CRC error is detected, the IP continues scanning frames without halting and notifies the user about the error.</p>

2.1.3. SEDC Controller IP Flow

Figure 2.3 shows the high-level flowchart of the SEDC Controller IP. Because of the complexity of the FSM, which comprises over 30 states, this flowchart does not capture details on the complete implementation of the internal FSM. However, this flowchart attempts to facilitate the conceptual understanding of the SEDC Controller IP flow.

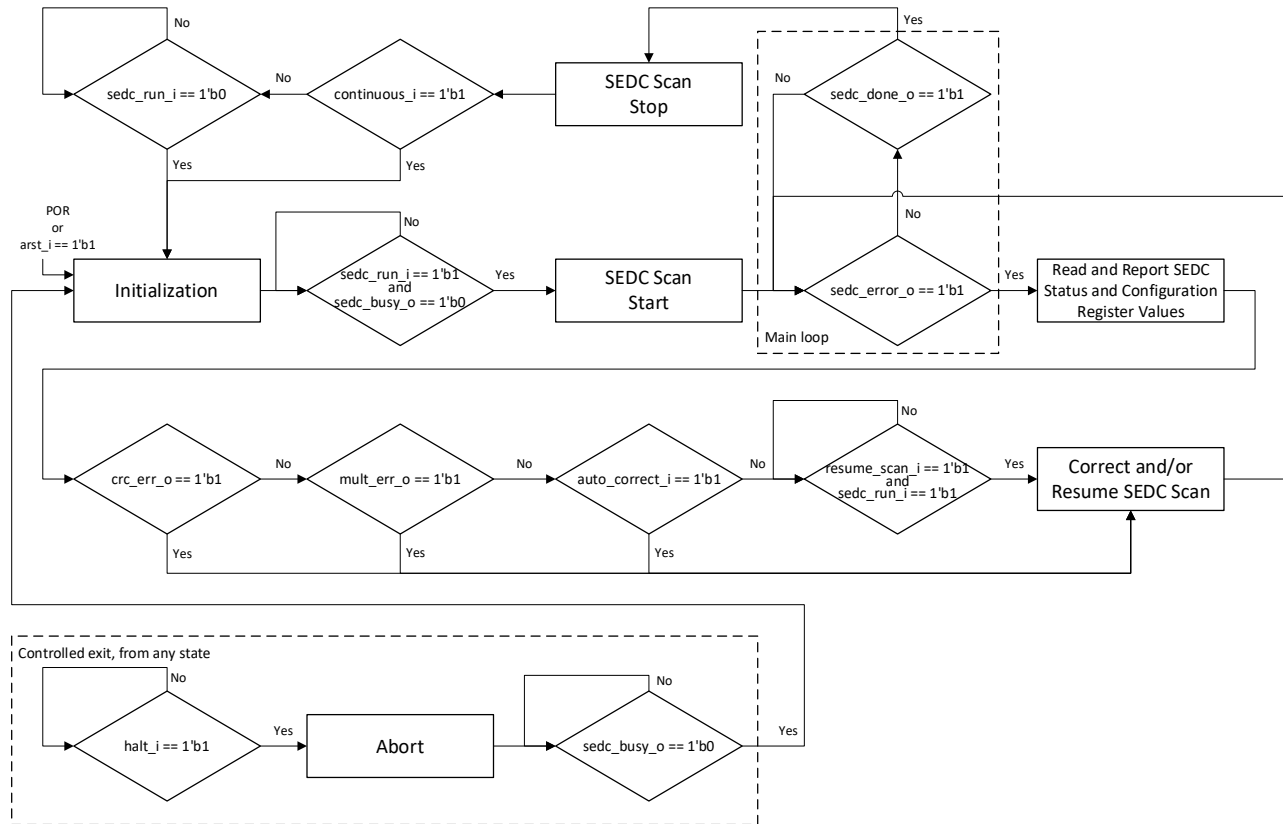


Figure 2.3. SEDC Controller IP Flowchart

Table 2.3 describes each of the processes. For more details on IP behaviors, you can explore the testbench that simulates IP behavior as illustrated by the example waveforms in the [Walkthrough of Example Simulation Waveforms](#) section. Follow the steps in the [Running Functional Simulation](#) section so that the necessary simulation files are included when launching the simulation.

Table 2.3. SEDC Controller IP Processes

Process	Description
Initialization	<ul style="list-style-type: none"> Clears all output registers that drive the output signals <code>crc_err_o</code>, <code>mult_err_o</code>, <code>sing_err_o</code>, <code>err_o</code>, <code>bit_loc_o</code>, <code>frm_loc_o</code>, <code>rgn_loc_o</code>, and <code>clk_div_o</code> to 0. Configures the SEDC hard block through LMMI write transactions. Reads the clock divider value from the SEDC hard block through LMMI read transaction and updates the register that drives the output signal <code>clk_div_o</code>.
SEDC Scan Start	<ul style="list-style-type: none"> Internal soft logic drives the SEDCENABLE signal of the SEDC hard block high to start a SEDC scan. The output signal <code>sedc_busy_o</code> asserts high once the SEDC hard block starts the scan.
Read and Report SEDC Status and Configuration Register Values	<ul style="list-style-type: none"> Reads the SEDC hard block status and configuration register values through LMMI read transactions. Captures these values into output registers that drive the output signals <code>crc_err_o</code>, <code>mult_err_o</code>, <code>sing_err_o</code>, <code>err_o</code>, <code>bit_loc_o</code>, <code>frm_loc_o</code>, <code>rgn_loc_o</code>, and <code>clk_div_o</code>. Asserts the output signal <code>status_update_o</code> high momentarily for one clock cycle after the error status and locations of all output registers are ready to be read.
Correct and/or Resume SEDC Scan	<ul style="list-style-type: none"> Performs error correction and/or resumes scan by informing the SEDC hard block through LMMI write transaction. The output signal <code>sedc_error_o</code> is de-asserted low by the SEDC hard block once it completes the error correction and/or resumes the scan operation.

Process	Description
SEDC Scan Stop	<ul style="list-style-type: none"> Indicates the SEDC hard block has finished the current SEDC scan cycle on all frames. De-asserts the output signal <code>sedc_busy_o</code> low. Asserts the output signal <code>sedc_done_o</code> high momentarily for one clock cycle, immediately after the <code>sedc_busy_o</code> signal is de-asserted low. Internal soft logic drives <code>SEDCENABLE</code> signal of the SEDC hard block low.
Abort	<ul style="list-style-type: none"> Aborts the SEDC Controller IP operation gracefully from any state. Stops all LMMI transactions and/or current SEDC scan. Clears all output registers to 0.

2.2. Clocking

Figure 2.4 shows the high-level block diagram of the clock domain for the SEDC Controller IP. There are two clock domains within the sysCONFIG block namely `SEDC_CLK` and LMMI clock. However, only the LMMI clock is accessible by the SEDC Controller IP. `SEDC_CLK` is hardened to be sourced from the internal oscillator of the FPGA device. The single clock input for the entire IP (`clk_i`) can be driven by an external clock source (through an FPGA I/O pin), PLL, or oscillator. This clock is used to clock all soft logic registers and drive the `LMMICLK` of the SEDC hard block. For the clock frequencies supported by the SEDC Controller IP, refer to Table 4.1. The PLL and oscillator can be instantiated through the FPGA device PLL IP and OSC IP from the Lattice Radiant software IP Catalog, respectively.

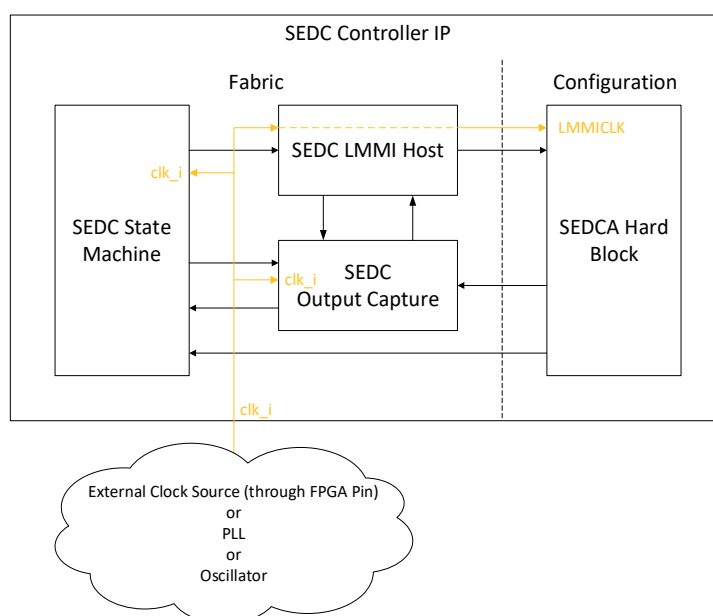


Figure 2.4. SEDC Controller IP Clock Domain Block Diagram

While the `SEDC_CLK` is not accessible by the SEDC Controller IP, its frequency can be adjusted through the *Clock Divider* attribute. Table 2.4 lists the `SEDC_CLK` frequencies with different clock divider values, calculated using the following equation:

$$SEDC_CLK\ (MHz) = \frac{400}{Clock\ Divider\ Value}$$

Table 2.4. `SEDC_CLK` Frequencies with Different Clock Divider Values

Clock Divider Value	SEDC_CLK Frequency (MHz)
2	200
3	133.33
...	...
256	1.56

Clock frequency settings are important for optimizing both performance and power consumption. A slower `clk_i` correspondingly slows the SEDC Controller IP response time. However, the power impact is small because of the SEDC Controller IP's relatively small resource usage (see resource utilization captured in [Table A.1](#)). The selection of `SEDC_CLK` directly impacts the SEDC scan and correction times. For example, with the lowest clock divider value of 2, the `SEDC_CLK` frequency is 200 MHz, resulting in shorter scan and correction times. Conversely, with the highest clock divider value of 256, the `SEDC_CLK` frequency is 1.56 MHz, resulting in longer scan and correction times. While the power impact of the SEDC Controller IP is generally small, it can become significantly more pronounced during continuous operation, making careful clock frequency management crucial for maintaining optimal system performance and efficiency.

2.3. Reset

[Figure 2.5](#) shows the high-level block diagram of the resets for the SEDC Controller IP. There are two reset types namely asynchronous reset (`arst_i`) and synchronous halt (`halt_i`). Both resets can be driven by user logic outside of the IP, GPIO, or other primitives.

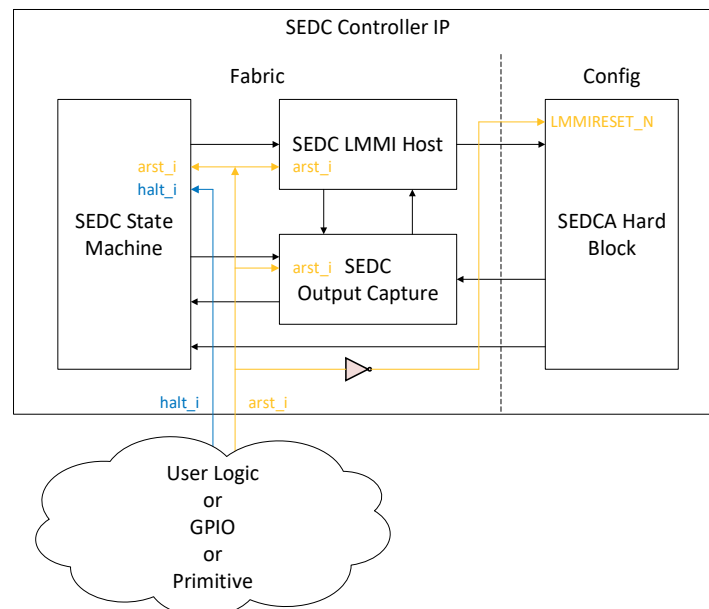


Figure 2.5. SEDC Controller IP Resets Block Diagram

For asynchronous reset, you must assert this reset high and wait at least 60 μ s after the device is configured and has entered user mode before releasing it low. Following this, you can proceed to start the SEDC scan operation. For any subsequent assertions of asynchronous reset, the minimum duration of reset assertion is 84 ns. Asynchronous reset, when asserted high, resets the LMMI logic of the SEDC hard block and immediately clears all the registers in the SEDC state machine, SEDC LMMI host, and SEDC output capture modules to 0. The SEDC LMMI host also drives 0 to all its signals feeding into the SEDC hard block. Note that an inverter is implemented before asynchronous reset is connected to the `LMMIRESET_N` port of the SEDC hard block because `LMMIRESET_N` is an active low signal. If asynchronous reset is asserted high during an on-going SEDC Controller IP operation, the SEDC state machine immediately returns to the initialization state. After the initialization state, the SEDC Controller IP only allows you to start a new SEDC scan after the SEDC hard block is no longer busy.

The halt reset is used to gracefully abort an on-going SEDC Controller IP operation from any state. At the positive clock edge when halt is high, the SEDC state machine enters the abort state, which immediately causes the SEDC LMMI host to stop LMMI transactions and/or the SEDC scan. All registers in the SEDC output capture module are reset to 0. At the immediate positive clock edge after halt is low, the SEDC state machine returns to the initialization state when the SEDC hard block is no longer busy. The minimum duration of reset assertion for halt is one full clock cycle.

2.4. User Interfaces

Table 2.5 lists the available user interfaces of the SEDC Controller IP.

Table 2.5. User Interfaces

User Interface	Communication Standard	Description
Clock	Digital/Binary	A single clock input for the entire IP (clk_i) that can be driven by an external clock source (through GPIO), PLL, or oscillator. Clocks all soft logic registers and drives the LMMICLK of the SEDC hard block.
Reset	Digital/Binary	There are two resets: asynchronous reset (arst_i) and synchronous halt (halt_i). Both resets can be driven by user logic outside of the IP, GPIO, or other primitives. Asynchronous reset resets the LMMI logic of the SEDC hard block and immediately clears all the registers in the SEDC state machine, SEDC LMMI host, and SEDC output capture modules to 0. Halt gracefully aborts an on-going SEDC Controller IP operation from any state.
Control	Digital/Binary	Input signals (sedc_run_i, continuous_i, auto_correct_i, and resume_scan_i) to control IP operations such as: <ul style="list-style-type: none"> Continuous mode <ul style="list-style-type: none"> Executes SEDC scans continuously, one after another. Error correction and resume scan <ul style="list-style-type: none"> Performs error correction on correctable error. Resumes scan upon error detection. Configures SEC as either automatic or manual. IP handles LMMI communications with the SEDC hard block, eliminating need for user involvement.
Monitor	Digital/Binary	Output signals (status_update_o, bit_loc_o, frm_loc_o, rgn_loc_o, clk_div_o, crc_err_o, sing_err_o, mult_err_o, err_o, sedc_error_o, sedc_done_o, and sedc_busy_o) to monitor the IP such as: <ul style="list-style-type: none"> Error status and location <ul style="list-style-type: none"> Reports error status and location information. IP retrieves information from the SEDC hard block through LMMI, eliminating the need for user involvement. Error statuses include 1-bit error (correctable), multi-bit error (uncorrectable), and CRC error. Error locations include bit location, frame location, and region location. Clock divider value <ul style="list-style-type: none"> Reports the clock divider value. IP retrieves information from the SEDC hard block through LMMI, eliminating the need for user involvement. SEDC scan status <ul style="list-style-type: none"> Reports the status of the current SEDC can (busy or done).

3. IP Parameter Description

The configurable attributes of the SEDC Controller IP are shown in the following table. You can configure the IP by setting the attributes accordingly in the IP Catalog Module/IP wizard of the Lattice Radiant software.

Wherever applicable, default values are in bold.

Table 3.1. SEDC Controller IP Attributes

Attribute	Selectable Values	Description
Configuration		
SEDC Mode	Continuous Mode , One-Shot Mode, Port-Driven Dynamic Mode	Specifies the SEDC mode and determines whether the SEDC mode is static (Continuous Mode or One-Shot Mode) or dynamic (Port-Driven Dynamic Mode, switches between static modes). When Port-Driven Dynamic Mode is selected, an additional user interface port <code>continuous_i</code> is exposed.
Error Correction Mode	Auto-Correction Mode , No Auto-Correction Mode, Port-Driven Dynamic Mode	Determines whether the error correction mode is automatic (Auto-Correction Mode), manual (No Auto-Correction Mode), or dynamically switches between the two (Port-Driven Dynamic Mode). When No Auto-Correction Mode is selected, an additional user port interface <code>resume_scan_i</code> is exposed. When Port-Driven Dynamic Mode is selected, two additional user interface ports <code>auto_correct_i</code> and <code>resume_scan_i</code> are exposed.
Clock Divider	2–256	Specifies the clock divider value for the SEDC. The mapping of <i>Clock Divider</i> setting (N) to the hardware value reported in <code>clk_div_o</code> is $N - 1$. For example, when <i>Clock Divider</i> = 2, the <code>clk_div_o</code> readout is 1.
Disable Triple Modular Redundancy	Checked, Unchecked	Specifies whether to disable the TMR on all registers (checked). For highest reliability, Lattice recommends this remain unchecked (TMR enabled).
Clock Frequency (MHz)	0.0000000– 80.0000000	Specifies the clock frequency for port <code>clk_i</code> . For highest reliability, Lattice recommends rounding off to a conservative number.
CRAM Error Injection in Simulation		
Types of Error Injection	No CRAM Error , Simulate Correctable CRAM Error, Simulate Uncorrectable CRAM Error, Simulate CRC CRAM Error	Determines whether to enable the fake error injection capability in simulation, and if so, the type of error to inject.
CRAM Error Location in Simulation		
Bit Error Location	0–1023	Specifies the data shift register (DSR) bit location of the fake error injected. Applicable when <i>Types of Error Injection</i> = Simulate Correctable CRAM Error.
Frame Error Location	0–16383	Specifies the address shift register (ASR) bit location of the fake error injected. Applicable when <i>Types of Error Injection</i> = Simulate Correctable CRAM Error or Simulate Uncorrectable CRAM Error.
Region Error Location	0–31	Specifies the region location of the fake error injected. Applicable when <i>Types of Error Injection</i> = Simulate Correctable CRAM Error or Simulate Uncorrectable CRAM Error.

4. Signal Description

This section describes the SEDC Controller IP ports.

4.1. Clock Interface

Table 4.1. Clock Port

Port	Type	Description
clk_i	Input	Input clock that drives all soft logic registers and the LMMICK of the SEDC hard block. Clock frequency must not exceed 80 MHz.

4.2. Reset Interface

Table 4.2. Reset Ports

Port	Type	Description
arst_i	Input	Asynchronous reset to reset all soft logic registers and the LMMI logic of the SEDC hard block when asserted high. This signal must be asserted high and can only be released low 60 μ s after the device is configured and has entered user mode. Subsequent assertions require a minimum reset assertion duration of 84 ns. The SEDC state machine module proceeds to initialize the SEDC hard block at the immediate positive clock edge after the de-assertion of this signal.
halt_i	Input	Reset to abort the entire IP operation from any state when asserted high. It is synchronous to clk_i. The minimum duration of assertion is one full clock cycle. The SEDC state machine module returns to the initialization state when the SEDC hard block is no longer busy. This occurs at the immediate positive clock edge after de-assertion of this signal.

4.3. Control Interface

Table 4.3. Control Ports

Port	Type	Description
sedc_run_i	Input	Signal to start and run the SEDC scan. This signal must remain high until sedc_busy_o is asserted high. Following this, it can continue to remain high or be de-asserted any time. Requirements in each SEDC Mode: <ul style="list-style-type: none"> Continuous Mode – This signal may remain high from one SEDC scan to the next, as long as you want to run the SEDC scans continuously. One-Shot Mode – This signal must be re-asserted high after the assertion of sedc_done_o to start a new SEDC scan. Port-Driven Dynamic Mode – This signal must be re-asserted high after the assertion of sedc_done_o if continuous_i is low to start a new SEDC scan. If continuous_i is high from one SEDC scan to the next, this signal must also remain high to run the SEDC scans continuously. Requirements in each Error Correction Mode: <ul style="list-style-type: none"> Auto-Correction Mode – This signal can be high or low when an error occurs because it does not impact error correction and/or resumption of the current SEDC scan. No Auto-Correction Mode – This signal must be re-asserted or remain high together with the assertion of resume_scan_i when an error occurs to perform error correction and/or resume the current SEDC scan.

Port	Type	Description
		<ul style="list-style-type: none"> Port-Driven Dynamic Mode – This signal must be re-asserted or remain high together with the assertion of <code>resume_scan_i</code> if <code>auto_correct_i</code> is low when an error occurs to perform error correction and/or resume the current SEDC scan. If <code>auto_correct_i</code> is high when an error occurs, this signal can be high or low because it does not impact error correction and/or resumption of the current SEDC scan.
<code>continuous_i</code>	Input	<p>Signal to run the SEDC scan continuously.</p> <p>1 – Continuous mode 0 – One-shot mode</p> <p>This signal must remain high together with <code>sedc_run_i</code> from one SEDC scan to the next to run the SEDC scans continuously.</p> <p>Exposure of this signal depends on the configuration of <i>SEDC Mode</i>:</p> <ul style="list-style-type: none"> Continuous Mode – This signal is tied off to 1'b1 internally and not exposed as a user interface port. One-Shot Mode – This signal is tied off to 1'b0 internally and not exposed as a user interface port. Port-Driven Dynamic Mode – This signal is exposed as a user interface port.
<code>auto_correct_i</code>	Input	<p>Signal to perform automatic error correction and/or resume the SEDC scan when an error occurs.</p> <p>1 – Auto-Correction Mode is ON 0 – Auto-Correction Mode is OFF</p> <p>This signal must remain high throughout the SEDC scan (if one-shot) or consecutive SEDC scans (if continuous) to ensure smooth automatic error correction and/or resumption of the SEDC scan when an error occurs, while also providing the benefit of minimal latency.</p> <p>Exposure of this signal depends on the configuration of <i>Error Correction Mode</i>:</p> <ul style="list-style-type: none"> Auto-Correction Mode – This signal is tied off to 1'b1 internally and not exposed as a user interface port. No Auto-Correction Mode – This signal is tied off to 1'b0 internally and not exposed as a user interface port. Port-Driven Dynamic Mode – This signal is exposed as a user interface port.
<code>resume_scan_i</code>	Input	<p>Signal to perform error correction and/or resume the SEDC scan manually when a 1-bit error occurs.</p> <p>If Auto-Correction Mode is OFF, whenever 1-bit error is detected, this signal must be asserted/remain high together with <code>sedc_run_i</code> after the assertion of <code>status_update_o</code>.</p> <p>Exposure of this signal depends on the configuration of <i>Error Correction Mode</i>:</p> <ul style="list-style-type: none"> Auto-Correction Mode – This signal is tied off to 1'b0 internally and not exposed as a user interface port. No Auto-Correction Mode – This signal is exposed as a user interface port. Port-Driven Dynamic Mode – This signal is exposed as a user interface port.

4.4. Monitor Interface

Table 4.4. Monitor Ports

Port	Type	Description
status_update_o	Output	Signal to indicate that the status and configuration register outputs are ready to be read, which includes crc_err_o, mult_err_o, sing_err_o, err_o, bit_loc_o, frm_loc_o, and rgn_loc_o. This signal is asserted high momentarily for one clock cycle only. Consume the error statuses and locations immediately when status_update_o is asserted high. Otherwise, the error statuses and locations may become outdated or invalid.
bit_loc_o[9:0]	Output	Signal to indicate the DSR bit location of the last 1-bit error.
frm_loc_o[13:0]	Output	Signal to indicate the ASR frame location of the last 1-bit/multi-bit error.
rgn_loc_o[4:0]	Output	Signal to indicate the region location of the last 1-bit/multi-bit error.
clk_div_o[7:0]	Output	Signal to indicate the clock divider value.
crc_err_o	Output	Signal to indicate the occurrence of a CRC error. Asserted high when CRC error occurs.
mult_err_o	Output	Signal to indicate the occurrence of a multi-bit error. Asserted high when multi-bit error occurs.
sing_err_o	Output	Signal to indicate the occurrence of a 1-bit error. Asserted high when 1-bit error occurs.
err_o	Output	Signal to indicate the occurrence of an error. Asserted high when an error occurs.
sedc_error_o	Output	Signal to indicate whether SEDC has detected an error. Asserted high when an error occurs.
sedc_done_o	Output	Signal to indicate whether the SEDC scan has finished. This signal is asserted high momentarily for one clock cycle only.
sedc_busy_o	Output	Signal to indicate whether the SEDC scan is running. 1 – SEDC scan is running. 0 – SEDC can is idle.

5. Designing with the IP

This section provides information on how to generate the IP Core using the Lattice Radiant software and how to run simulation and synthesis. For more details on the Lattice Radiant software, refer to the Lattice Radiant Software User Guide.

5.1. Generating and Instantiating the IP

You can use the Lattice Radiant software to generate IP modules and integrate them into the device architecture. The steps below describe how to generate the SEDC Controller IP in the Lattice Radiant software.

To generate the SEDC Controller IP:

1. Create a new Lattice Radiant software project or open an existing project.
2. In the **IP Catalog** tab, double-click **SEDC Controller** under **IP on Local > Architecture_Modules** category. The **Module/IP Block Wizard** opens as shown in [Figure 5.1](#). Enter values in the **Component name** and the **Create in** fields and click **Next**.

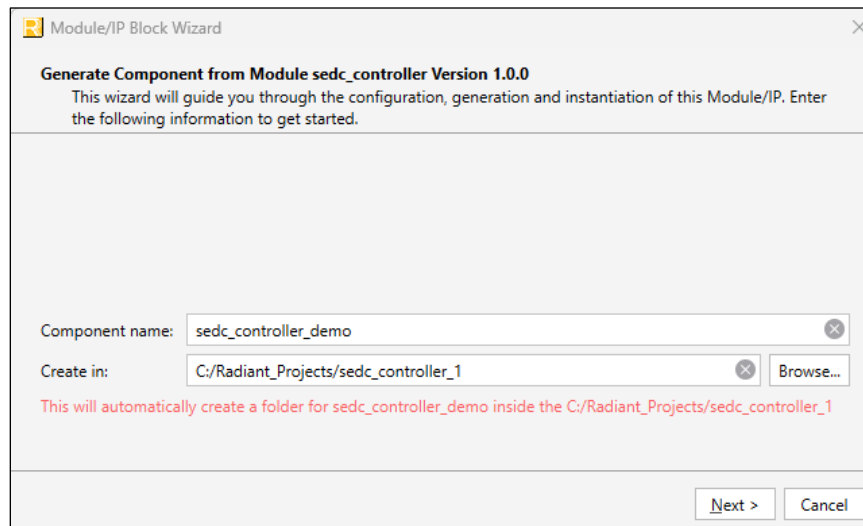


Figure 5.1. Module/IP Block Wizard

3. In the next **Module/IP Block Wizard** window, customize the selected SEDC Controller IP using drop-down lists and check boxes. [Figure 5.2](#) shows an example configuration of the SEDC Controller IP. For details on the configuration options, refer to the [IP Parameter Description](#) section.

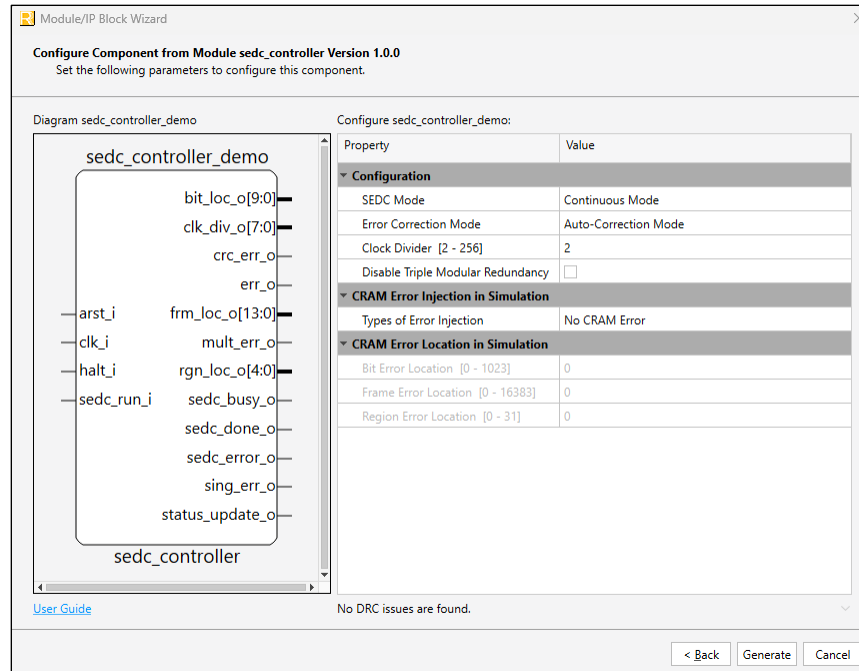


Figure 5.2. IP Configuration

- Click **Generate**. The **Check Generated Result** dialog box opens, showing design block messages and results as shown in Figure 5.3.

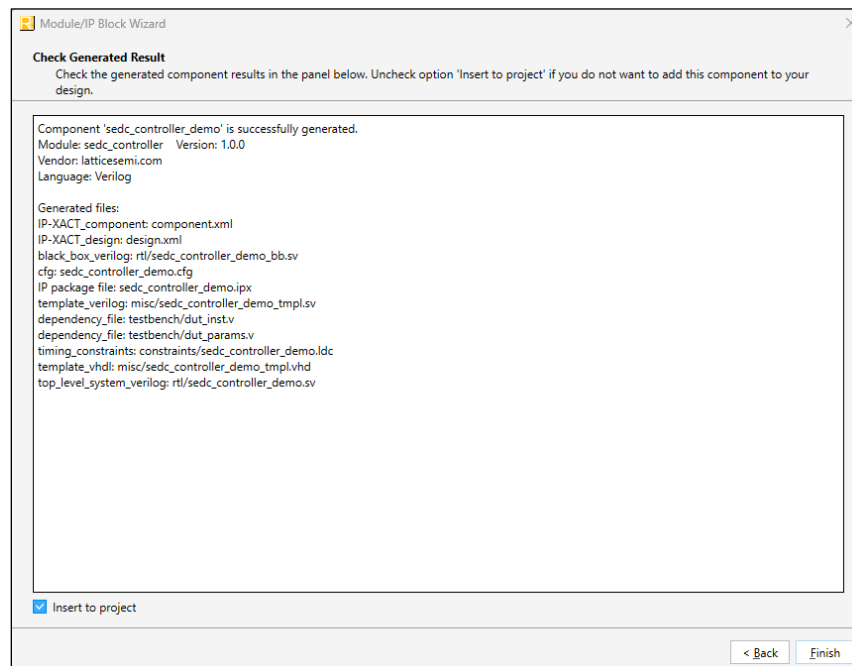


Figure 5.3. Check Generated Result

- Click **Finish**. All the generated files are placed under the directory paths in the **Create in** and the **Component name** fields shown in Figure 5.1.

5.1.1. Generated Files and File Structure

The generated SEDC Controller IP package includes the closed-box (<Component name>_bb.sv) and instance templates (<Component name>_tmpl.sv/vhd) that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file (<Component name>.sv) that can be used as an instantiation template for the module is also provided. You may also use this example as the starting template for your top-level design. The generated files are listed in [Table 5.1](#).

Table 5.1. Generated File List

Attribute	Description
<Component name>.ipx	This file contains the information on the files associated to the generated IP.
<Component name>.cfg	This file contains the parameter values used in IP configuration.
component.xml	Contains the ipxact: component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.sv	This file provides an example RTL top file that instantiates the module.
rtl/<Component name>_bb.v	This file provides the synthesis closed-box.
misc/<Component name>_tmpl.sv misc /<Component name>_tmpl.vhd	These files provide instance templates for the module.

5.2. Design Implementation

Completing your design includes additional steps to specify analog properties, pin assignments, and timing and physical constraints. You can add and edit the constraints using the Device Constraint Editor or by manually creating a PDC file.

Post-synthesis design constraint files (.pdc) contain both timing and non-timing constraint.pdc source files for storing logical timing/physical constraints. Constraints that are added using the Device Constraint Editor are saved to the active .pdc file. The active post-synthesis design constraint file is then used as input for post-synthesis processes.

Refer to the relevant sections in the Lattice Radiant Software User Guide for more information on how to create or edit constraints and how to use the Device Constraint Editor.


5.3. Specifying the Strategy

The Radiant software provides two predefined strategies: Area and Timing. It also enables you to create customized strategies. For details on how to create a new strategy, refer to the Strategies section of the Lattice Radiant Software user guide.

5.4. Running Functional Simulation

You can run functional simulation after the IP is generated.

To run functional simulation:

1. Click the  button located on the **Toolbar** to initiate the **Simulation Wizard** shown in [Figure 5.4](#).

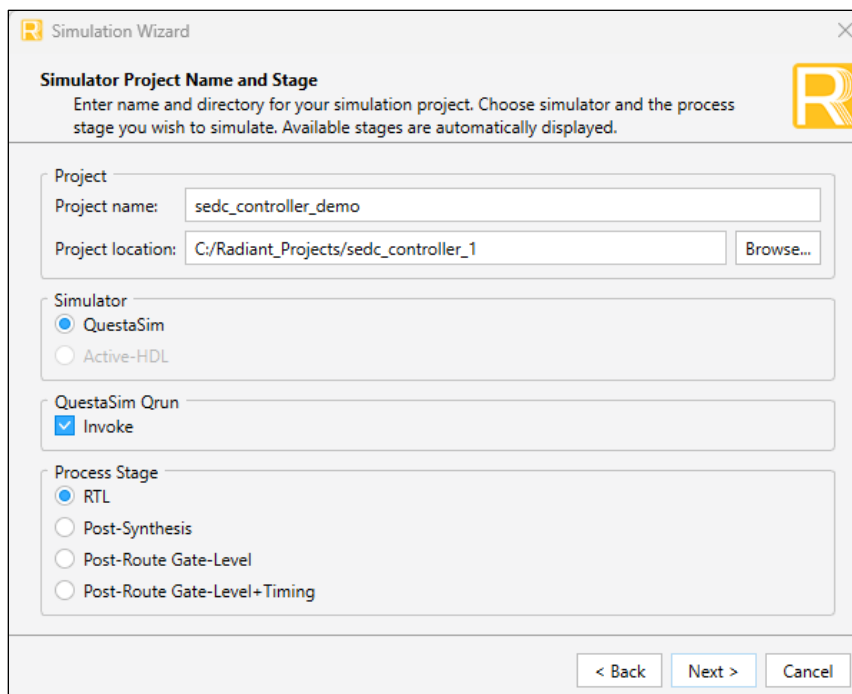


Figure 5.4. Simulation Wizard

2. Click **Next** to open the **Add and Reorder Source** window as shown in Figure 5.5.

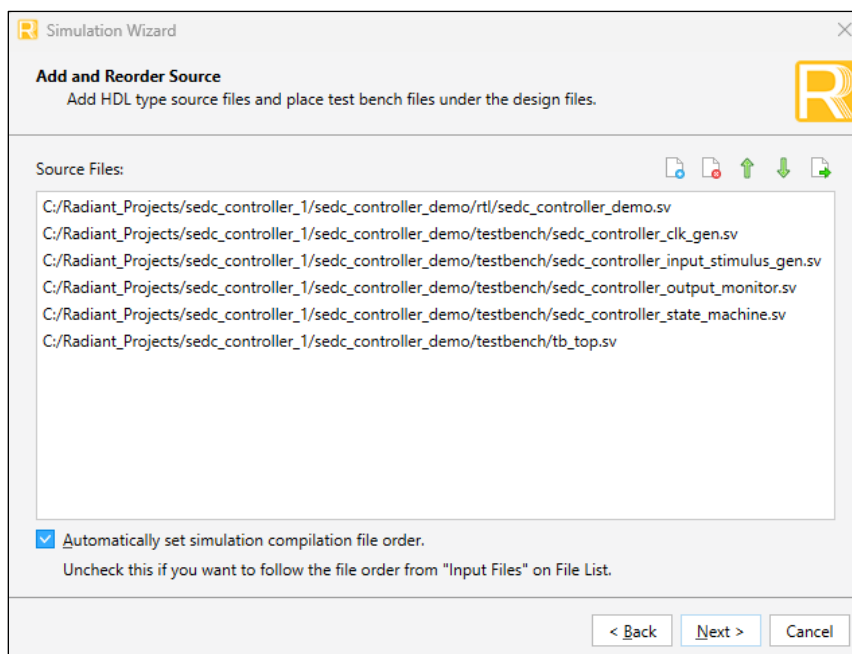


Figure 5.5. Add and Reorder Source

3. Click **Next**. The **Parse HDL files for simulation** window opens as shown in Figure 5.6.

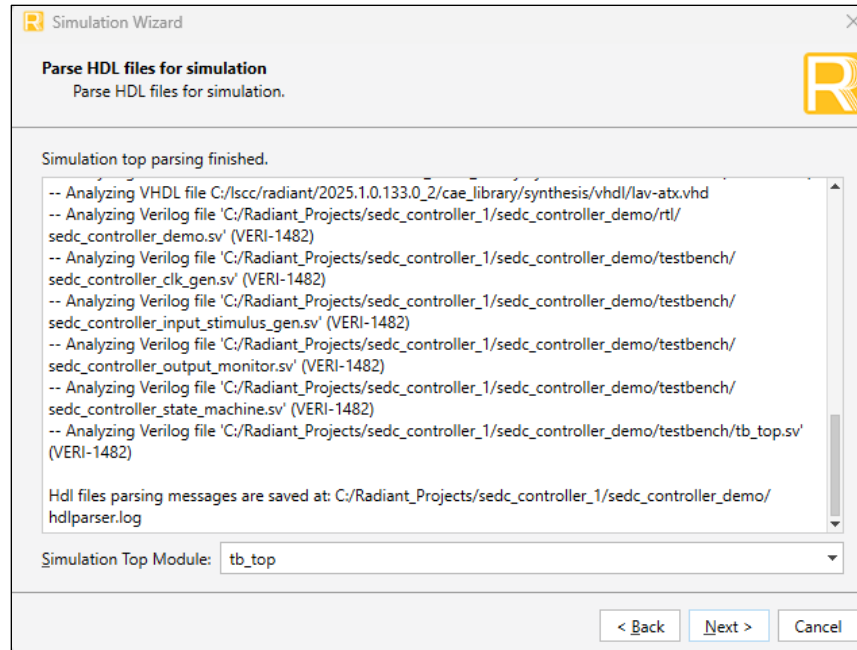


Figure 5.6. Parse HDL files for Simulation

- Click **Next**. The **Summary** window opens as shown in Figure 5.7. Change the time of **Default Run** as shown to run the full simulation.

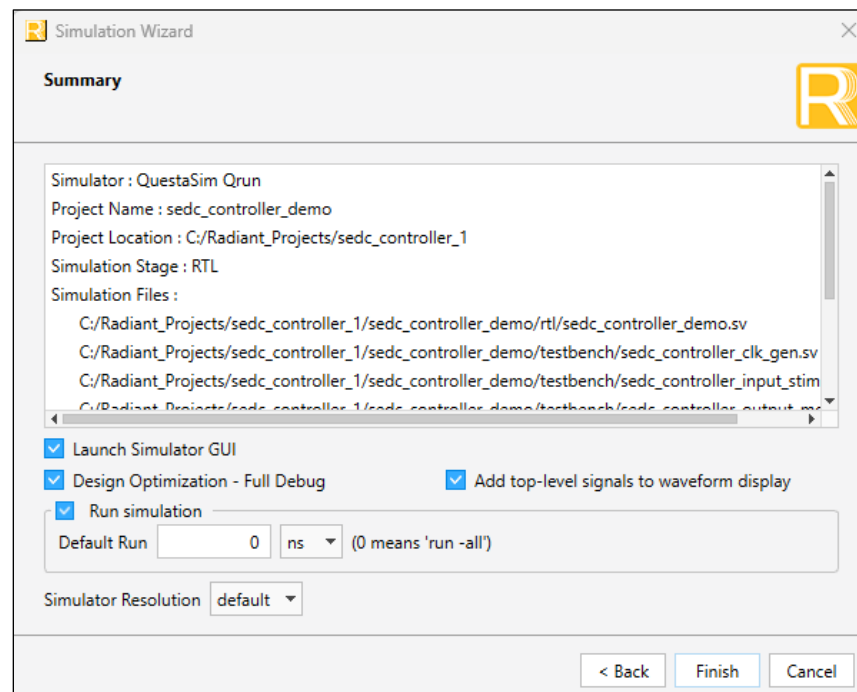


Figure 5.7. Summary

- Click **Finish** to run the simulation. The waveform in Figure 5.8 shows an example simulation result.

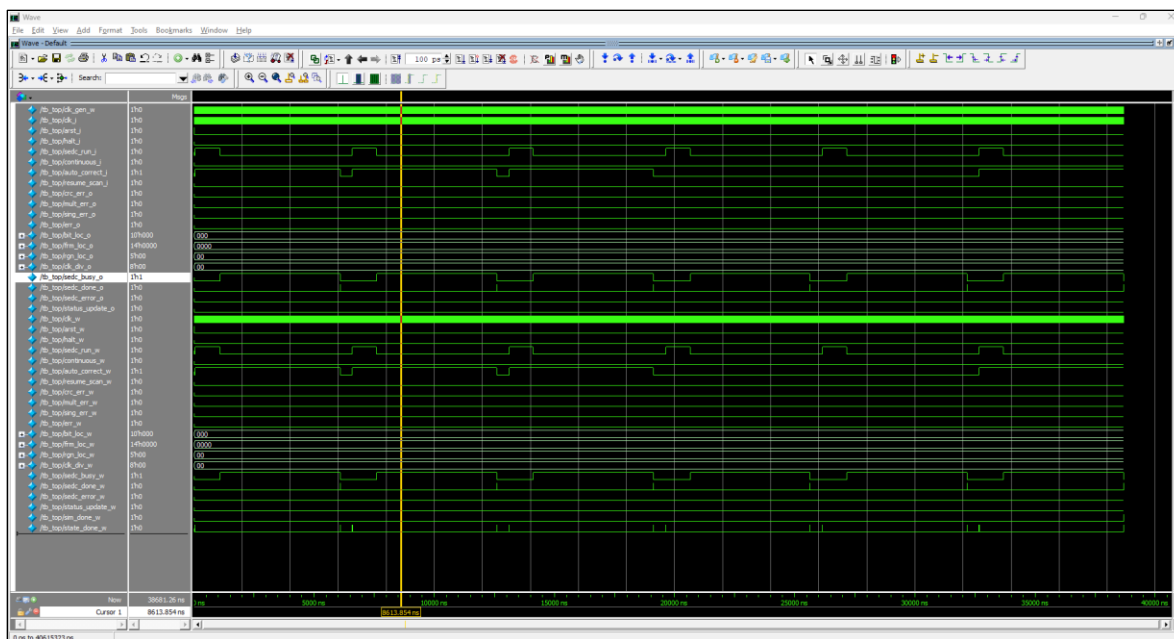


Figure 5.8. Simulation Waveform

5.4.1. Simulation Results

The testbench embedded with the SEDC Controller IP undergoes multiple stimulus patterns to exercise different scenarios, depending on the configured attributes, until the simulation is successfully completed. The Transcript window displays this information as shown in [Figure 5.9](#).

```
Transcript
# run -all
# =====
# 3000 ps: Current State = reset
# =====
# 88000 ps: Current State = run_sedc_with_run_always_high
# =====
# 12223000 ps: Current State = run_sedc_with_run_always_high_and_tmr_err_inj
# =====
# 24373000 ps: Current State = run_sedc_continuously_with_halt
# =====
# 27648000 ps: Current State = stop_all_signals
# =====
# 28153000 ps: Current State = run_sedc_with_run_always_high
# =====
# 34168000 ps: Current State = stop_all_signals
# =====
# 34673000 ps: Current State = run_sedc_with_run_always_high_and_tmr_err_inj
# =====
# 40688000 ps: Current State = stop_all_signals
# =====
# 41193000 ps: Current State = run_sedc_with_run_always_high
# =====
# 47208000 ps: Current State = run_sedc_with_run_always_high_and_tmr_err_inj
# =====
# 53283000 ps: Current State = end
# =====
# ----- SIMULATION FINISHED -----
# Note: $finish : C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/sedc_controller_output_monitor.sv(l437)
# Time: 53282510 ps Iteration: 3 Instance: /tb_top/sedc_controller_output_monitor_inst
# Break in Module sedc_controller_output_monitor at C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/sedc_controller_output_monitor.sv line 1437
V$SIM 6>
```

Figure 5.9. Simulation Completion Log

Some example simulation waveforms are provided and explained in [Appendix B](#). The walkthrough of these simulation waveforms, along with the explanations in the [Limitations of the SEDC Hard Block Simulation Model](#), [Testbench Files and Structure](#), and [Input Stimulus Patterns](#) sections, should facilitate your understanding when you simulate for other IP configurations with the embedded testbench.

5.4.2. Limitations of the SEDC Hard Block Simulation Model

The simulation model of the SEDC hard block is a simplified representation and does not reflect the actual hardware. It is designed to provide a functional demonstration with the following limitations:

- Behaviors are transactionally accurate instead of cycle accurate.
- 1-bit error and multi-bit error injections are mutually exclusive, whereas actual hardware can experience both error types in the same or different SEDC scans.
- A 1-bit error is reported in the first and all subsequent SEDC scans, but actual hardware corrects the error and does not report the error again in subsequent scans unless another 1-bit error occurs in the same error location.
- There is no capability to inject multiple 1-bit errors or multi-bit errors, whereas actual hardware can experience this scenario in the same or different SEDC scans.

5.4.3. Testbench Files and Structure

The SEDC Controller IP comes with a testbench to demonstrate how to drive the input signals of the IP, observe the IP responses, and view the output signals. The testbench generates the appropriate input stimulus to the DUT depending on the configured attributes as described in the [IP Parameter Description](#) section. You may use this testbench as a reference to evaluate how to build your design around the functionality of the SEDC Controller IP. The testbench components and their respective files are described in [Table 5.2](#), and the testbench structure is illustrated in [Figure 5.10](#).

Table 5.2. Testbench Components and File List

Component	File	Description
Testbench Top	tb_top.sv	The top-level testbench module that integrates all other components and coordinates the simulation.
Clock Generator	sedc_controller_clk_gen.sv	Generates the clock signal required for the SEDC Controller IP operation.
Input Stimulus Generator	sedc_controller_input_stimulus_gen.sv	Generates the input stimulus signals based on the configured attributes to test the SEDC Controller IP.
Stimulus State Machine	sedc_controller_stimulus_state_machine.sv	Implements the state machine that controls the sequence of input stimuli to exercise different scenarios.
Output Monitor	sedc_controller_output_monitor.sv	Monitors and captures the output signals from the SEDC Controller IP for verification purposes.
DUT	<Component name>.sv	The DUT, which is the SEDC Controller IP itself, generated by the IP generation process described in the Generating and Instantiating the IP section, allowing you to verify its functionality and performance.

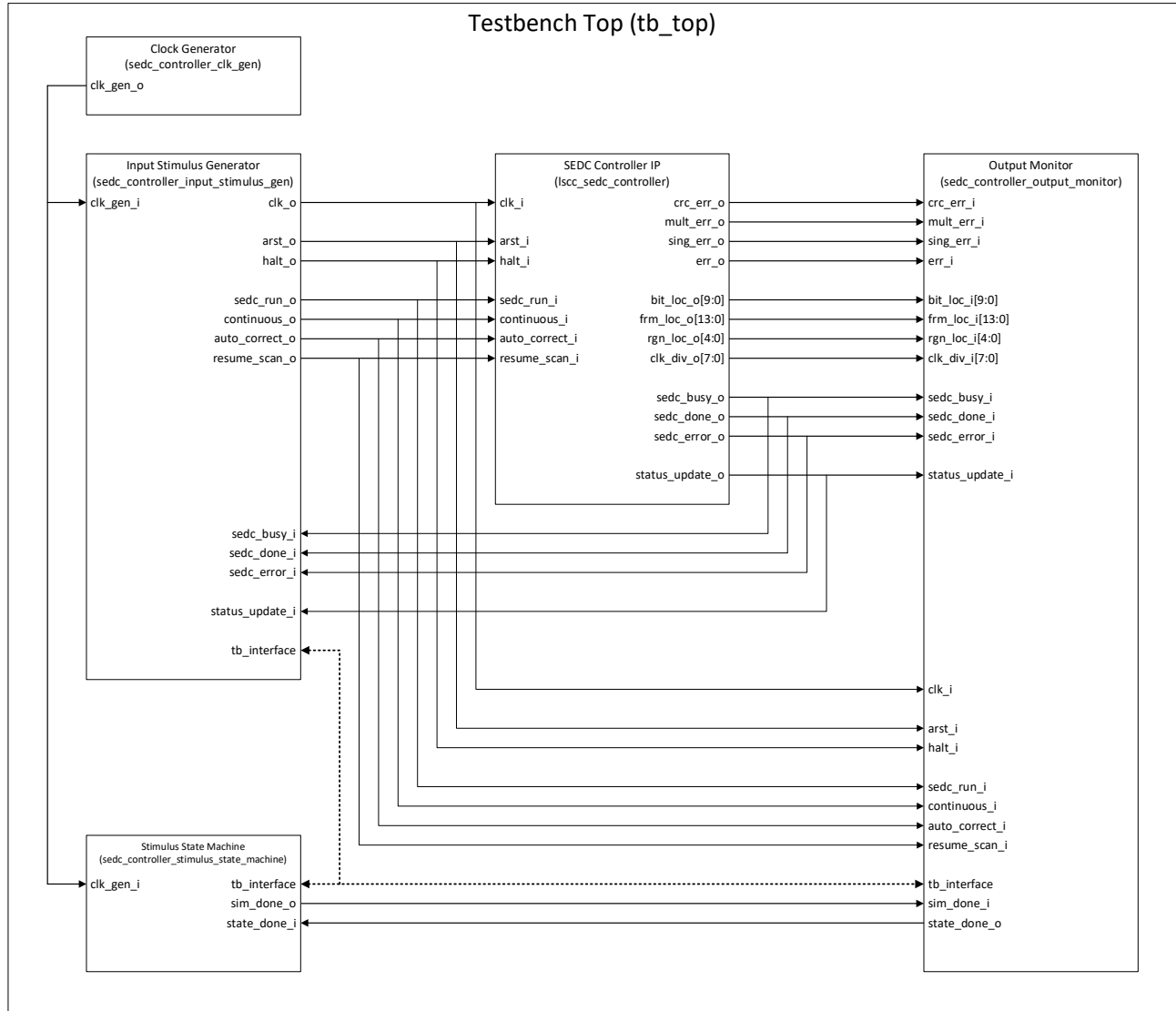


Figure 5.10. Testbench Structure

5.4.4. Input Stimulus Patterns

The input stimulus generator of the testbench consists of different input stimulus patterns as described in [Table 5.3](#). You may add the signal path `tb_top/tb_interface/cur_state_str` to the wave GUI in QuestaSim to monitor the current stimulus state string of the interface.

Table 5.3. Input Stimulus Description

Input Stimulus Pattern Name ^{1, 2}	Description
reset	Asserts the asynchronous reset signal (<code>arst_i</code>) high. Demonstrates the ability to handle asynchronous reset and ensures all internal states and outputs are properly initialized to their default values.
run_sedc_with_run_always_high	Drives the <code>sedc_run_i</code> signal high. Demonstrates continuous operation of the SEDC scan without interruptions.
run_sedc_with_run_always_high_and_tmr_err_inj	Drives the <code>sedc_run_i</code> signal high and injects TMR errors. Demonstrates the ability to handle errors on TMR registers.
run_sedc_continuously_with_halt	Drives the <code>sedc_run_i</code> signal high and asserts the <code>halt_i</code> signal during busy cycles (<code>sedc_busy_o</code> is high). Demonstrates the ability to handle halts during continuous operation.
run_sedc_with_run_always_high_and_manual_resume	Drives the <code>sedc_run_i</code> signal high and asserts the <code>resume_scan_i</code> signal after <code>status_update_o</code> is high. Demonstrates the ability to perform manual error correction/resume scan when an error occurs.
run_sedc_with_run_always_high_manual_resume_and_tmr_err_inj	Drives the <code>sedc_run_i</code> signal high and asserts the <code>resume_scan_i</code> signal after <code>status_update_o</code> is high, with TMR error injection. Demonstrates the ability to handle errors on TMR registers and perform manual error correction/resume scan.
run_sedc_with_run_always_high_and_auto_cor	Drives the <code>sedc_run_i</code> signal high and asserts the <code>auto_correct_i</code> signal. Demonstrates the ability to automatically correct errors during operation.
run_sedc_with_run_always_high_auto_cor_and_tmr_err_inj	Drives the <code>sedc_run_i</code> signal high and asserts the <code>auto_correct_i</code> signal, with TMR error injection. Demonstrates the ability to handle errors on TMR registers and automatically correct errors during operation.
run_sedc_one_shot_with_halt	Drives the <code>sedc_run_i</code> signal high in one-shot mode and asserts the <code>halt_i</code> signal during busy cycles (<code>sedc_busy_o</code> is high). Demonstrates the ability to handle halts during one-shot operation.
run_sedc_one_shot	Drives the <code>sedc_run_i</code> signal high in one-shot mode. Demonstrates the SEDC one scan cycle operation.
run_sedc_one_shot_and_tmr_err_inj	Drives the <code>sedc_run_i</code> signal high in one-shot mode and injects TMR errors. Demonstrates the ability to handle errors on TMR registers during one-shot operation.
run_sedc_one_shot_with_manual_resume	Drives the <code>sedc_run_i</code> signal high in one-shot mode and asserts the <code>resume_scan_i</code> signal after <code>status_update_o</code> is high. Demonstrates the ability to perform manual error correction/resume scan during one-shot operation.
run_sedc_one_shot_with_auto_cor	Drives the <code>sedc_run_i</code> signal high in one-shot mode and asserts the <code>auto_correct_i</code> signal. Demonstrates the ability to automatically correct errors during one-shot operation.
run_sedc_one_shot_with_auto_cor_and_tmr_err_inj	Drives the <code>sedc_run_i</code> signal high in one-shot mode and asserts the <code>auto_correct_i</code> signal, with TMR error injection. Demonstrates the ability to handle errors on TMR registers and automatically correct errors during one-shot operation.
run_sedc_one_shot_with_manual_resume_and_tmr_err_inj	Drives the <code>sedc_run_i</code> signal high in one-shot mode and asserts the <code>resume_scan_i</code> signal after <code>status_update_o</code> is high, with TMR error injection. Demonstrates the ability to handle errors on TMR registers and perform manual error correction/resume scan during one-shot operation.
run_sedc_continuously	Drives the <code>sedc_run_i</code> signal high continuously. Demonstrates continuous operation without interruptions.

Input Stimulus Pattern Name ^{1, 2}	Description
run_scdc_continuously_and_tmr_err_inj	Drives the sedc_run_i signal high continuously and injects TMR errors. Demonstrates the ability to handle errors on TMR registers during continuous operation.
run_scdc_continuously_with_manual_resume	Drives the sedc_run_i signal high continuously and asserts the resume_scan_i signal after status_update_o is high. Demonstrates the ability to perform manual error correction/resume scan during continuous operation.
run_scdc_continuously_with_manual_resume_and_tmr_err_inj	Drives the sedc_run_i signal high continuously and asserts the resume_scan_i signal after status_update_o is high, with TMR error injection. Demonstrates the ability to handle errors on TMR registers and perform manual error correction/resume scan during continuous operation.
run_scdc_continuously_with_auto_cor	Drives the sedc_run_i signal high continuously and asserts the auto_correct_i signal. Demonstrates the ability to automatically correct errors during continuous operation.
run_scdc_continuously_with_auto_cor_and_tmr_err_inj	Drives the sedc_run_i signal high continuously and asserts the auto_correct_i signal, with TMR error injection. Demonstrates the ability to handle errors on TMR registers and automatically correct errors during continuous operation.
stop_all_signals	De-asserts all input signals. Demonstrates the ability to stop all signals and ensure the DUT is in a known idle state.
default_state	Default state with no actions.
end	End state that indicates the end of the simulation.

Notes:

1. If *Disable Triple Modular Redundancy* = Checked in the IP Catalog's Module/IP wizard, *_tmr_err_inj input stimulus patterns will not inject TMR errors.
2. TMR error injection by *_tmr_err_inj input stimulus patterns is guarded with ifdef macros by default because the injection involves hierarchical references that only work for RTL simulation. Refer to the example in [Appendix C](#) on how to add +define+LSCC_SEDC_CONTROLLER_RTL_SIM in the *.f file generated by the Simulation Wizard if you want to observe the behavior. This must only be applied when you have selected **RTL** under **Process Stage** in the Simulation Wizard as shown in [Figure 5.4](#). You must recompile the simulation files for this directive to take effect.

6. Debugging

You may use the Reveal Inserter and Reveal Analyzer tools contained in the Radiant software for design debugging. Reveal continuously monitors signals within the FPGA for specific conditions, which can range from simple to complex. When the trigger condition occurs, Reveal can save signal values preceding, during, and following the event for analysis, including a waveform presentation. The data can be saved to a value change dump file (.vcd), which can be used with tools such as QuestaSim, or to an ASCII tabular format that can be used with tools such as Excel.

Before running the Reveal Analyzer, use the Reveal Inserter to add Reveal modules to your design. In these modules, specify the signals to monitor, define the trigger conditions, and other options. Reveal supports multiple logic analyzer cores using a hard or soft JTAG interface. You can have up to 15 modules, typically one for each clock region of interest. When the modules are set up, regenerate the bitstream data file to program the FPGA.

For more information, refer to the Reveal User Guide for Radiant Software.

7. Design Considerations

7.1. Asynchronous Reset (arst_i) Timing Requirements

To ensure the SEDC Controller IP is properly initialized, it is compulsory that you assert the asynchronous reset signal (arst_i) high for at least 60 μ s after the device is configured and has entered user mode. Once this occurs, you can de-assert the asynchronous reset signal low and start the SEDC scan operation. For any subsequent assertion of the asynchronous reset signal, the minimum duration of assertion is 84 ns.

An example Verilog/SystemVerilog code to implement the 60 μ s and 84 ns requirements is shown in [Appendix D](#).

7.2. Assert and Hold auto_correct_i High throughout a Complete SEDC Scan

To ensure the proper functioning of automatic error correction and scan resumption in the SEDC Controller IP during a complete scan, it is crucial that you assert and hold the auto_correct_i signal high. This action enables the automatic correction and resume scan mechanism in the SEDC Controller IP, ensuring that detected 1-bit correctable errors are promptly corrected (uncorrectable errors are not corrected) during the scan process thereby allowing the scan to resume.

This step is only required if you want to perform automatic error correction for the entire scan. If you do not want to enable automatic error correction for the next scan, you can de-assert the auto_correct_i signal. It is important to note that this functionality is only applicable if you set the *Error Correction Mode* to Port-Driven Dynamic Mode as described in the [IP Parameter Description](#) section. For other settings, this port is tied off internally and not exposed.

Additionally, the auto_correct_i signal must be held high when you drive the sedc_run_i signal high and remain high until the sedc_done_o signal from the IP asserts high, indicating the completion of the scan. Although the flowchart in [Figure 2.3](#) illustrates that the SEDC Controller IP only monitors these signals at the appropriate states (which you have no visibility of at the top-level ports), it is essential that you manage the auto_correct_i and sedc_run_i signals based on the state of sedc_done_o to ensure proper operation.

7.3. Assert and Hold continuous_i High throughout Consecutive SEDC Scans

To ensure seamless operation during consecutive SEDC scans, it is crucial that you assert and hold the continuous_i signal high. This action enables continuous mode in the SEDC Controller IP, allowing it to perform back-to-back scans without interruption.

Additionally, the sedc_run_i signal must be held high to initiate and maintain the scanning process. The scan is considered complete when the sedc_done_o signal asserts high, indicating the end of the current scan. If you want to stop the continuous scanning process starting from a particular scan, you can de-assert the continuous_i signal. It is important to note that this functionality is only applicable if you set the *SEDC Mode* to Port-Driven Dynamic Mode as described in the [IP Parameter Description](#) section. For other settings, this port is tied off internally and not exposed.

Although the flowchart in [Figure 2.3](#) illustrates that the SEDC Controller IP only monitors these signals at the appropriate states (which you have no visibility of at the top-level ports), it is essential that you manage the continuous_i and sedc_run_i signals based on the state of sedc_done_o to ensure proper operation.

7.4. Process Error Information Only when status_update_o Asserts

While error information might be available earlier than the status_update_o signal, depending on the internal state machine states of the SEDC Controller IP, it is not advisable for you to consume this information until status_update_o asserts. This signal indicates that the SEDC Controller IP has updated the status and configuration register outputs, making them ready for consumption. The outputs include crc_err_o, mult_err_o, sing_err_o, err_o, bit_loc_o, frm_loc_o, and rgn_loc_o.

Additionally, you must consume the error information immediately when status_update_o asserts high. Otherwise, the error information may become outdated or invalid. This includes scenarios where subsequent errors occur within the

same SEDC scan and status_update_o asserts high again, or the error information is cleared at the end of the SEDC scan.

By waiting for the assertion of the status_update_o signal, you ensure that you are processing the most accurate and up-to-date error information. Consuming the error information prematurely, before the status_update_o signal asserts, or not consuming the error information promptly when status_update_o asserts, can lead to incorrect error handling and potentially unreliable system behavior.

7.5. Availability of Clock Divider Value after Initialization

The clk_div_o signal, which indicates the clock divider value, is available immediately following the initialization of the SEDC Controller IP. Unlike error-related signals, the status_update_o signal does not assert when clk_div_o is updated. You do not have to wait until a scan is complete to read this value.

7.6. Add Debouncer on Signal Driven by Mechanical Switch/Button

When dealing with signals driven by mechanical switches or buttons, it is essential that you add a debouncer to filter out noise and ensure a clean signal. Mechanical switches often produce spurious signals due to contact bounce, which can cause multiple transitions and lead to erroneous behavior in digital circuits.

A debouncer can be implemented using a shift register, as shown in the example Verilog/SystemVerilog code in [Appendix E](#). This ensures that the signal is stable and reliable, preventing unintended multiple transitions and improving the overall performance of the SEDC Controller IP.

7.7. Implement Reset Synchronizer for Asynchronous Reset Signal

To ensure reliable operation of the asynchronous reset signal (arst_i) within the single clock domain of the SEDC Controller IP, it is essential that you implement a reset synchronizer. This helps to avoid metastability issues and ensures that the reset signal is properly synchronized with the clock.

A typical reset synchronizer can be implemented using a series of flip-flops as shown in the example Verilog/SystemVerilog code in [Appendix F](#). By implementing a reset synchronizer, you can mitigate the risks associated with the asynchronous reset signal and improve the overall reliability of the SEDC Controller IP.

8. Known Issues

8.1. Incorrect Clock Divider Value in Simulation

When simulating the SEDC Controller IP, the clock divider value is always incorrectly reported as 0 on the output port `clk_div_o`. This issue occurs regardless of the functional simulation type namely RTL, Post-Synthesis, Post-Route Gate-Level, or Post-Route Gate-Level+Timing. This is a known issue in the Lattice Radiant software 2025.1 and will be fixed in a future Lattice Radiant software release.

8.2. SEDC Unable to Detect Errors After 1-bit or 2-bit SEI Bitstream Injection

The SEDC hard block fails to detect errors in random 1-bit or 2-bit SEI bitstreams generated using the Radiant SEI tool, after these bitstreams are injected into the device. This issue is not specific to any particular block, bit, or frame location. It is a known issue in the Lattice Radiant software 2025.1 and will be fixed in a future Lattice Radiant software release.

As a workaround, generate additional 1-bit and 2-bit SEI bitstreams targeting different blocks. Aim for eight different combinations of 1-bit and 2-bit SEI bitstreams, an example as shown in [Figure 8.1](#). With more SEI bitstreams, the probability of detecting 1-bit or 2-bit SEDC errors increases when testing on hardware.

	Enable	ID	SEI File Name	Bit	Method	Block	Site	Frame	Bit In Frame
1	<input checked="" type="checkbox"/>	0	impl_1_sei_0	1 Bit	Unused	PFU	R67C80B	data frame 3120	bit 824
2	<input checked="" type="checkbox"/>	1	impl_1_sei_1	1 Bit	Unused	DSP	DSP_CORE_R65C86	data frame 2519	bit 800
3	<input checked="" type="checkbox"/>	2	impl_1_sei_2	1 Bit	Unused	ANY	R29C79F	data frame 3269	bit 364
4	<input checked="" type="checkbox"/>	3	impl_1_sei_3	1 Bit	Random	Routing	N/A	data frame 1362	bit 898
5	<input checked="" type="checkbox"/>	4	impl_1_sei_4	2 Bit	Unused	PFU,DSP	DSP_CORE_R51C176 R52C175A	data frame 3247 data frame 3247	bit 631 bit 632
6	<input checked="" type="checkbox"/>	5	impl_1_sei_5	2 Bit	Unused	PFU,PFU	R43C81F R44C81A	data frame 3009 data frame 3009	bit 533 bit 535
7	<input checked="" type="checkbox"/>	6	impl_1_sei_6	2 Bit	Unused	DSP,DSP	DSP_CORE_R95C282 DSP_CORE_R105C282	data frame 1949 data frame 1949	bit 1163 bit 1284
8	<input checked="" type="checkbox"/>	7	impl_1_sei_7	2 Bit	Random	Unclassified	N/A	data frame 1020 data frame 1020	bit 27 bit 121

Figure 8.1. Combination of 1-bit and 2-bit SEI Bitstreams

8.3. Error Bit and Region Location Mismatch Between Hardware and Radiant SEI Bitstream Profile

The following mismatches are observed in error location reporting:

- Error bit mismatch: This issue randomly occurs in 1-bit SEI bitstreams generated by the Radiant SEI tool. It is not specific to any bit or frame location.
- Error region mismatch: This issue affects all SEI bitstreams generated by the Radiant SEI tool.

As an example, a 1-bit SEI bitstream profile is generated using the Radiant SEI tool for the Avant-X device (LAV-AT-X70-1LFG1156C). The 1-bit error is injected at bit 824 in frame number 3120 located at site R67C80B.

	Enable	ID	SEI File Name	Bit	Method	Block	Site	Frame	Bit In Frame
1	<input checked="" type="checkbox"/>	0	impl_1_sei_0	1 Bit	Unused	PFU	R67C80B	data frame 3120	bit 824

Figure 8.2. Example of 1-bit SEI Bitstream

After injecting the 1-bit SEI bitstream into the Avant-X device, the hardware detects and reports the following error locations through the SEDC Controller IP as shown in the Reveal analyzer waveform in [Figure 8.3](#).

- Error bit location: 868
- Error frame location: 3120
- Error region location: 23

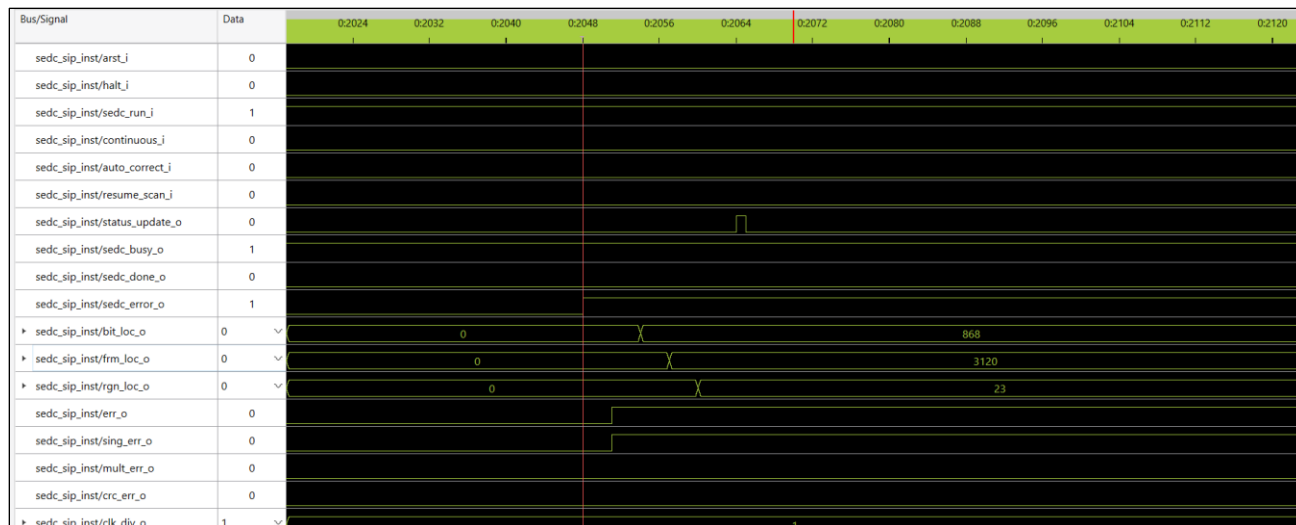


Figure 8.3. Reveal Analyzer Waveform for 1-bit SEI Bitstream

The hardware testing results show mismatches for the error bit and error region locations when compared to the 1-bit SEI bitstream profile. This is a known issue in the Lattice Radiant software 2025.1 and will be fixed in a future Lattice Radiant software release.

As there is currently no workaround for this issue, you can ignore the error locations reported by the SEDC Controller IP.

Appendix A. Resource Utilization

Table A.1 shows a sample resource utilization of the SEDC Controller IP v1.0.0 on LAT-AT-X70-1LFG1156C using the Lattice Radiant software 2025.1 across different IP configurations that impact resource utilization. Note that resource utilization may vary depending on the version of the SEDC Controller IP and Lattice Radiant software used.

Table A.1. Resource Utilization

IP Configuration				Slices	LUTs	Registers	SEDC
SEDC Mode	Error Correction Mode	Disable Triple Modular Redundancy	Clock Frequency (MHz)				
Continuous Mode	Auto-Correction Mode	Unchecked	5.0000000	225/198720	350/397440	204/397440	1/1
Continuous Mode	No Auto-Correction Mode	Unchecked	15.0000000	215/198720	340/397440	201/397440	1/1
Continuous Mode	Port-Driven Dynamic Mode	Unchecked	20.0000000	193/198720	312/397440	201/397440	1/1
One-Shot Mode	Auto-Correction Mode	Unchecked	25.0000000	191/198720	314/397440	201/397440	1/1
One-Shot Mode	No Auto-Correction Mode	Unchecked	30.0000000	211/198720	338/397440	201/397440	1/1
One-Shot Mode	Port-Driven Dynamic Mode	Unchecked	40.0000000	208/198720	338/397440	201/397440	1/1
Port-Driven Dynamic Mode	Auto-Correction Mode	Unchecked	50.0000000	204/198720	318/397440	201/397440	1/1
Port-Driven Dynamic Mode	No Auto-Correction Mode	Unchecked	60.0000000	194/198720	314/397440	201/397440	1/1
Port-Driven Dynamic Mode	Port-Driven Dynamic Mode	Unchecked	80.0000000	231/198720	375/397440	201/397440	1/1
Continuous Mode	Auto-Correction Mode	Checked	5.0000000	117/198720	217/397440	41/397440	1/1
Continuous Mode	No Auto-Correction Mode	Checked	15.0000000	122/198720	235/397440	42/397440	1/1
Continuous Mode	Port-Driven Dynamic Mode	Checked	20.0000000	122/198720	235/397440	42/397440	1/1
One-Shot Mode	Auto-Correction Mode	Checked	25.0000000	127/198720	242/397440	41/397440	1/1
One-Shot Mode	No Auto-Correction Mode	Checked	30.0000000	129/198720	247/397440	42/397440	1/1
One-Shot Mode	Port-Driven Dynamic Mode	Checked	40.0000000	133/198720	255/397440	42/397440	1/1
Port-Driven Dynamic Mode	Auto-Correction Mode	Checked	50.0000000	127/198720	240/397440	41/397440	1/1
Port-Driven Dynamic Mode	No Auto-Correction Mode	Checked	60.0000000	126/198720	243/397440	42/397440	1/1
Port-Driven Dynamic Mode	Port-Driven Dynamic Mode	Checked	80.0000000	122/198720	231/397440	42/397440	1/1

Appendix B. Walkthrough of Example Simulation Waveforms

B.1. Port-Driven Dynamic Mode

The example simulation waveforms provided and explained in this section are based on LAT-AT-X70-1LFG1156C and the IP configuration as follows:

- *SEDC Mode* = Port-Driven Dynamic Mode
- *Error Correction Mode* = Port-Driven Dynamic Mode
- *Types of Error Injection* = Simulate Correctable CRAM Error
- *Bit Error Location* = 124
- *Frame Error Location* = 5678
- *Region Error Location* = 9

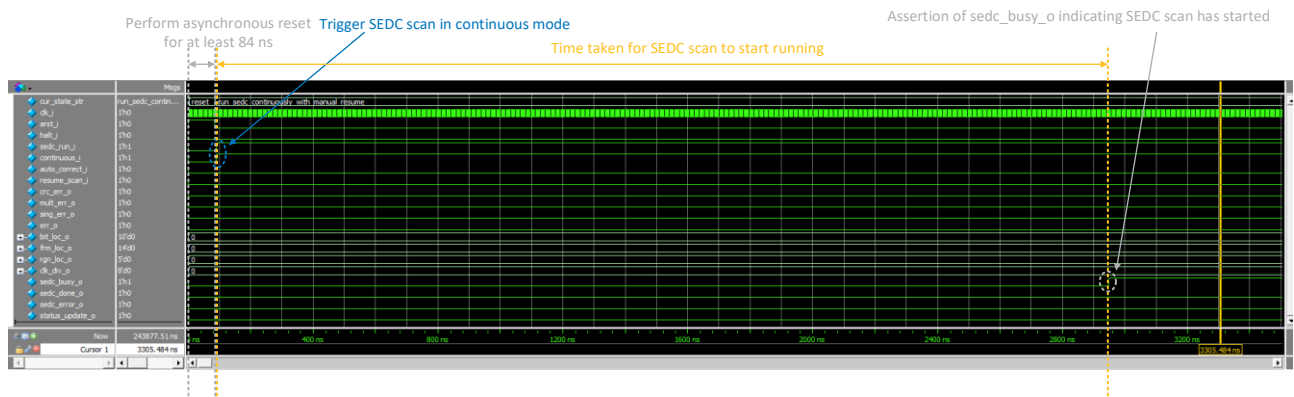


Figure B.1. Asynchronous Reset and Triggering of SEDC Scan in Continuous Mode

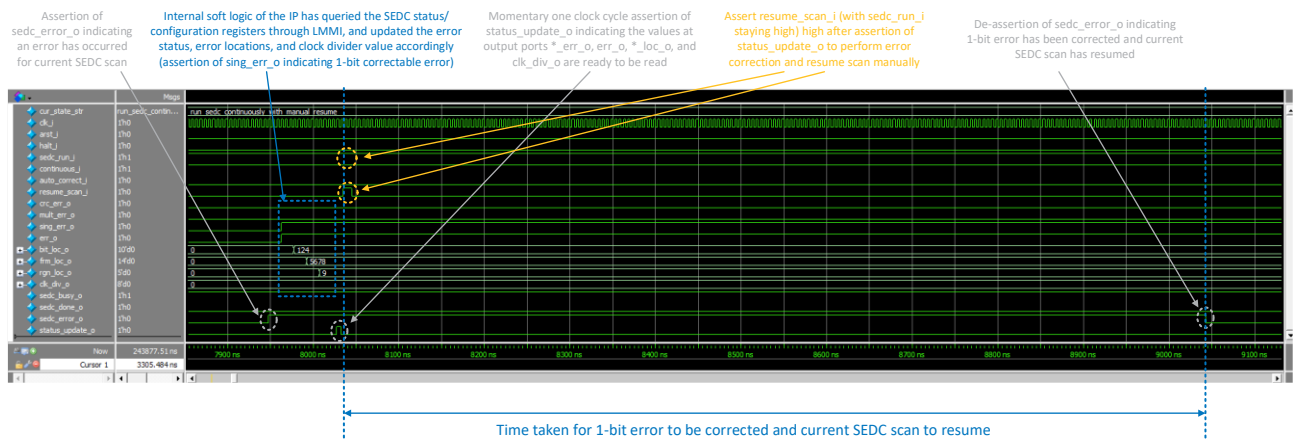


Figure B.2. 1-bit Error Occurrence and Manual Error Correction/Resume Scan

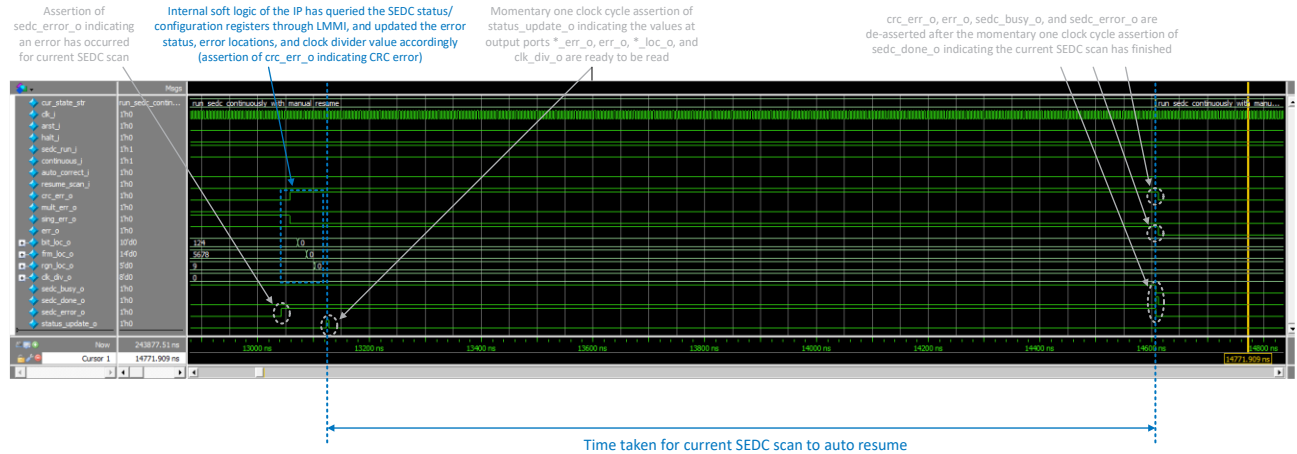


Figure B.3. CRC Error Occurrence and Auto Resume Scan

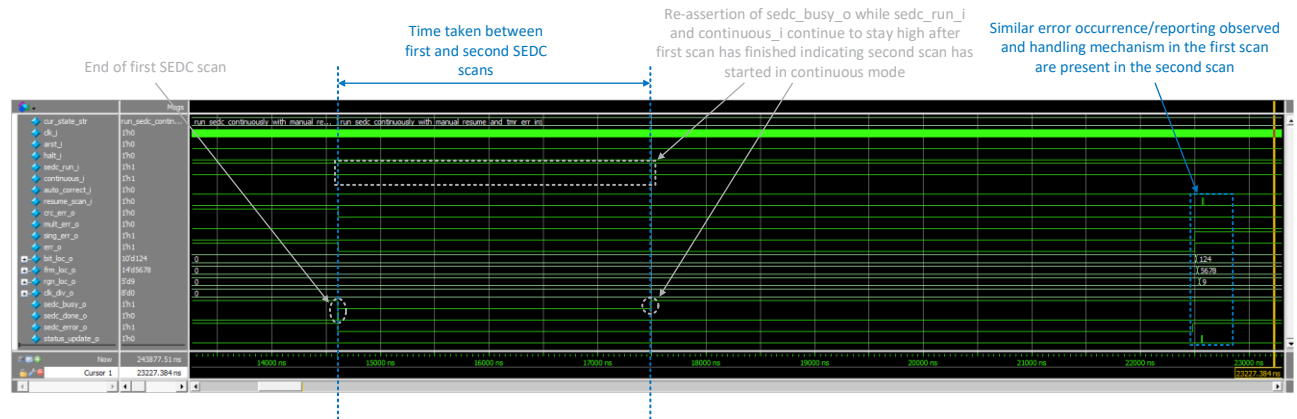


Figure B.4. End of First SEDC Scan and Triggering/1-bit Error Occurrence of Second SEDC Scan in Continuous Mode

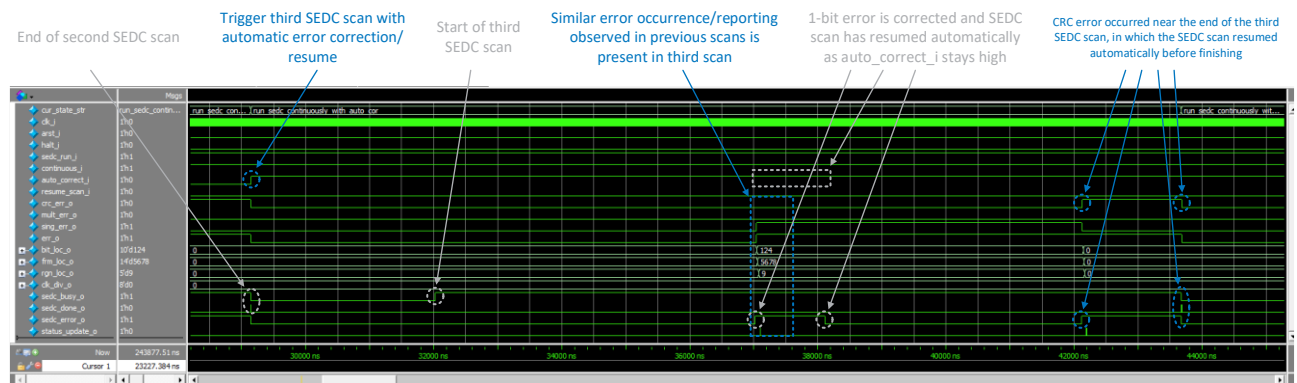


Figure B.5. End of Second SEDC Scan and Triggering/Error Occurrences/End of Third SEDC Scan in Continuous and Automatic Error Correction Modes

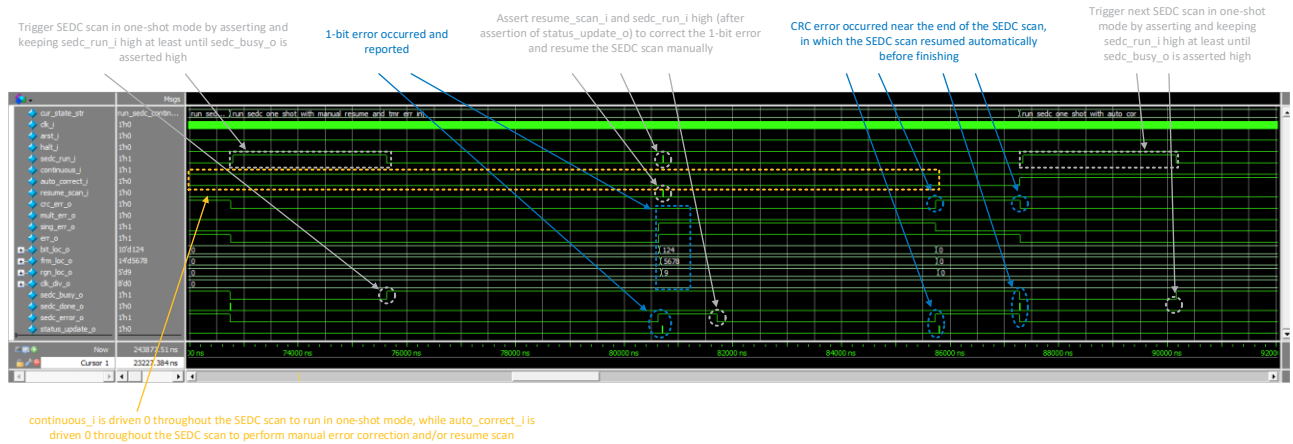


Figure B.6. Triggering/Error Occurrences/End of SEDC Scan in One-Shot Mode

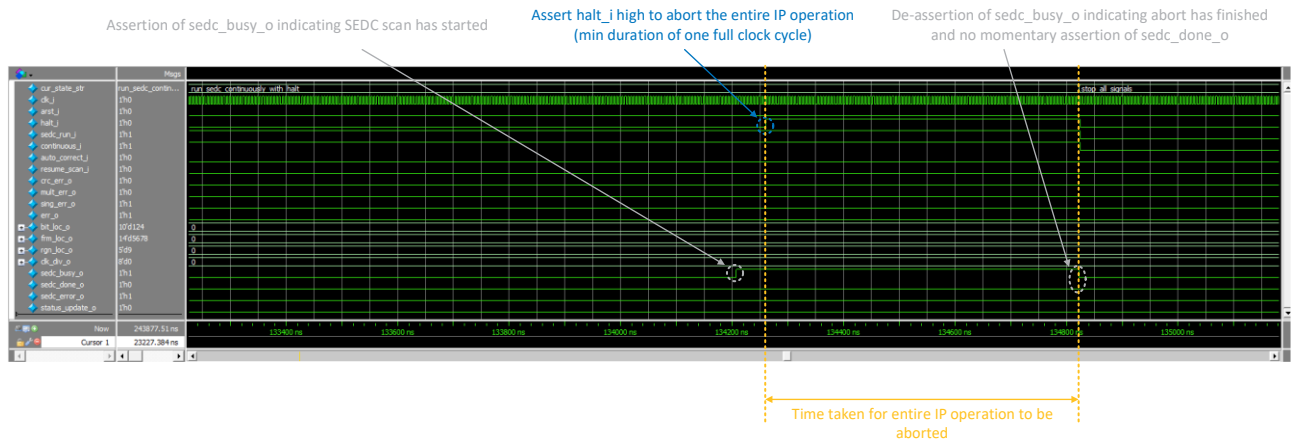


Figure B.7. Aborting of SEDC Scan

B.2. Continuous Mode

B.2.1. Scenario 1: 1-bit Correctable Error in Continuous Mode (Auto-Correction Mode = ON)

The example simulation waveforms provided and explained in this section are based on LAT-AT-X70-1LFG1156C and the IP configuration as follows:

- *SEDC Mode* = Continuous Mode
- *Error Correction Mode* = Auto-Correction Mode
- *Types of Error Injection* = Simulate Correctable CRAM Error
- *Bit Error Location* = 15 (10'h00f)
- *Frame Error Location* = 18 (14'h0012)
- *Region Error Location* = 0

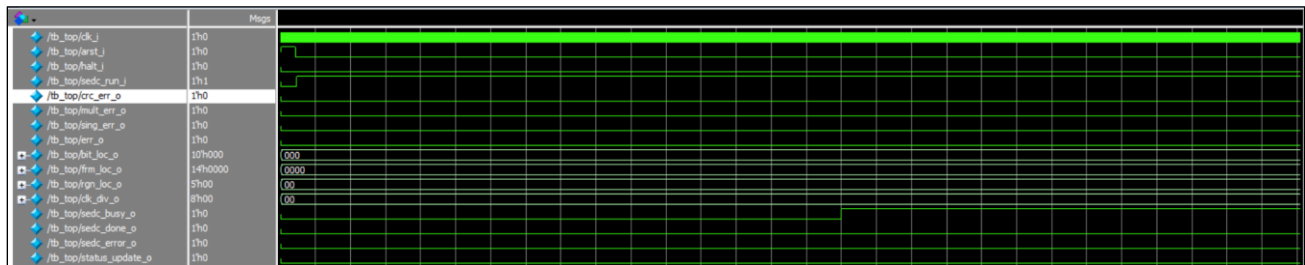


Figure B.8. Asynchronous Reset De-assertion and Triggering of SEDC Scan in Continuous Mode (1-bit Error)

Triggering the Scan:

- After asynchronous reset (arst_i) is de-asserted low, sedc_run_i is asserted high to trigger the SEDC scan.
- sedc_busy_o asserts high to indicate that the scan has started.

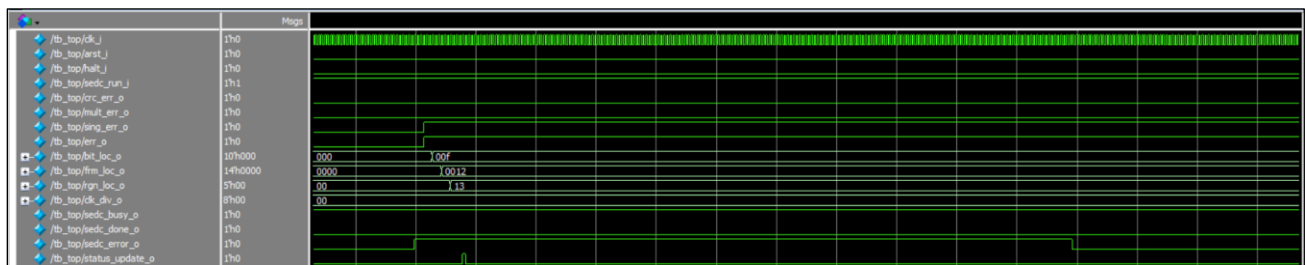


Figure B.9. 1-bit Error Detection and Auto-Correction

Error Detection:

- If an error is detected, sedc_error_o asserts high.
- The internal soft logic of the IP queries the SEDC status and configuration registers through LMMI and updates the following:
 - Error flags: err_o and sing_err_o (asserted high for a correctable 1-bit error)
 - Error location: bit_loc_o[9:0], frm_loc_o[13:0], rgn_loc_o[4:0]
- status_update_o pulses high for one clock cycle to indicate that updated outputs are ready.

Auto-Correction:

- With auto-correction enabled, the SEDC hard IP corrects the 1-bit error and de-asserts sedc_error_o to resume the scan.

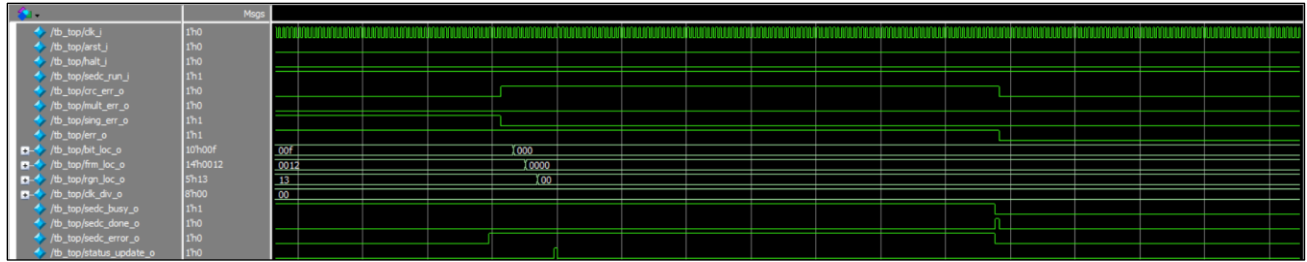


Figure B.10. CRC Error Occurrence and End of First SEDC Scan

CRC Error Near Scan Completion:

- Near the end of the scan, sedc_error_o asserts high again, indicating that an error occurred during this scan.
- crc_err_o asserts high, which is expected whenever a 1-bit or multi-bit error occurs.
- err_o remains high while sing_err_o de-asserts, signaling the CRC error flag.
- Error location outputs are cleared to zero for CRC error reporting.
- status_update_o pulses high for one clock cycle to indicate that updated outputs are ready.

End of Scan:

- After sedc_done_o pulses high for one clock cycle, all flags (crc_err_o, err_o, sedc_busy_o, and sedc_error_o) de-assert low, marking the end of the current scan.

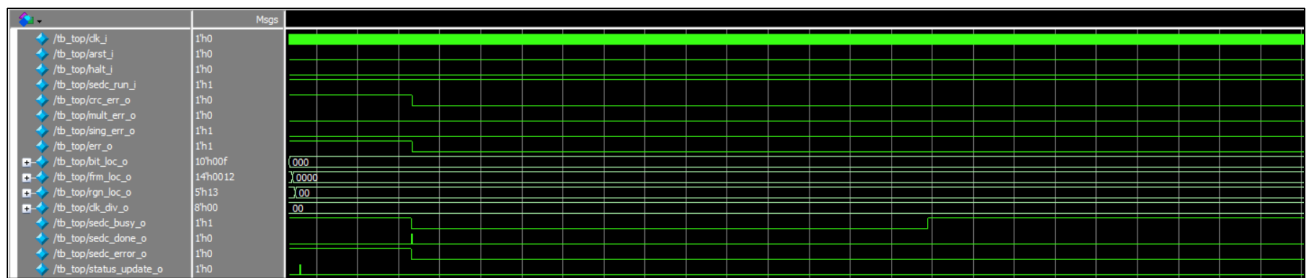


Figure B.11. Second SEDC Scan in Continuous Mode

Continuous Mode Behavior:

- In continuous mode, sedc_busy_o re-asserts while sedc_run_i remains high. This indicates that the next scan has started.

Note: In simulation, a 1-bit error is reported in every scan, but in hardware, the error is corrected and not reported again unless a new error occurs at the same location.

B.2.2. Scenario 2: Multi-bit Error in Continuous Mode (Auto-Correction Mode = ON)

The example simulation waveforms provided and explained in this section are based on LAT-AT-X70-1LFG1156C and the IP configuration as follows:

- *SEDC Mode* = Continuous Mode
- *Error Correction Mode* = Auto-Correction Mode
- *Types of Error Injection* = Simulate Uncorrectable CRAM Error
- *Frame Error Location* = 18 (14'h0012)
- *Region Error Location* = 8 (5'h08)

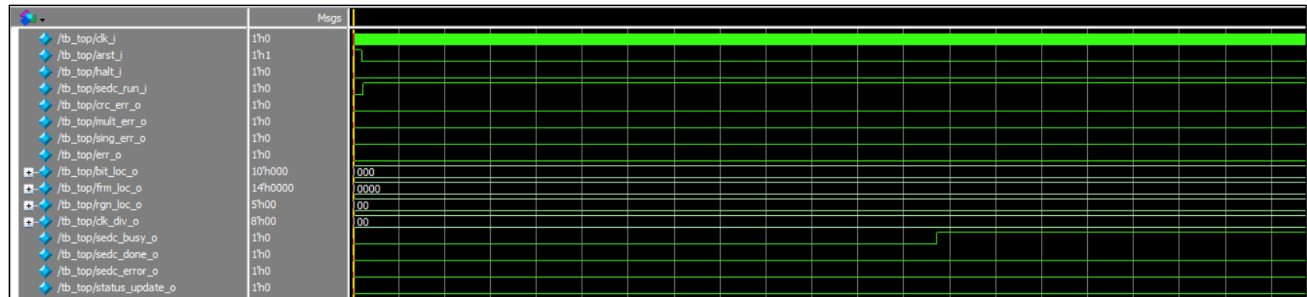


Figure B.12. Asynchronous Reset De-assertion and Triggering of SEDC Scan in Continuous Mode (Multi-bit Error)

Triggering the Scan:

- After asynchronous reset (*arst_i*) is de-asserted low, *sedc_run_i* is asserted high to trigger the SEDC scan.
- *sedc_busy_o* asserts high to indicate that the scan has started.

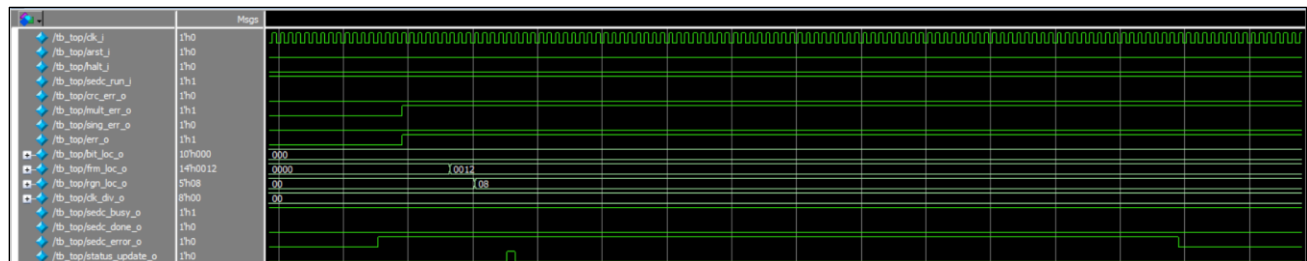


Figure B.13. Multi-bit Error Detection and Resume Scan

Error Detection:

- *sedc_error_o* asserts high when an error is detected.
- The internal soft logic of the IP queries the SEDC status and configuration registers through LMMI and updates the following:
 - Error flags: *err_o* and *mult_err_o* (asserted high for a multi-bit soft error)
 - Error location: *frm_loc_o*[13:0] and *rgn_loc_o*[4:0]
- *status_update_o* pulses high for one clock cycle to indicate that updated values are ready to be read.

Resume Scan:

- Because multi-bit errors are non-correctable, *sedc_error_o* de-asserts to resume the scan.

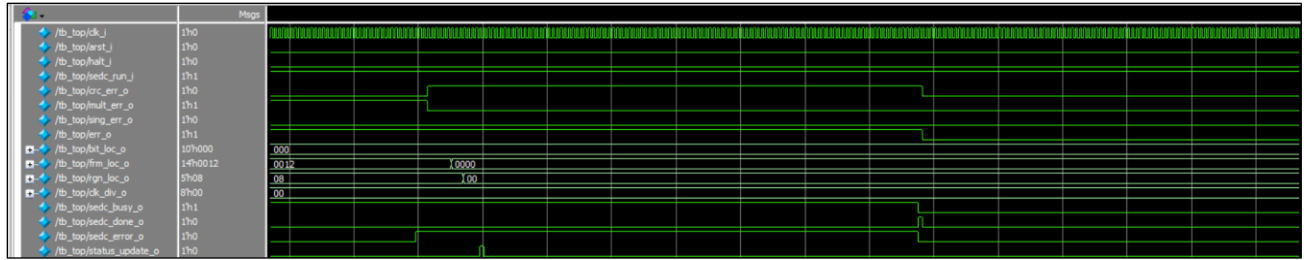


Figure B.14. CRC Error and End of First SEDC Scan (Multi-bit Error)

CRC Error Near Scan Completion:

- Near the end of the scan, sedc_error_o asserts high again, indicating that an error occurred during this scan.
- crc_err_o asserts high, which is expected whenever a 1-bit or multi-bit error occurs.
- err_o remains high while mult_err_o de-asserts, signaling the CRC error flag.
- Error location outputs are cleared to zero for CRC error reporting.
- status_update_o pulses high for one clock cycle to indicate that updated outputs are ready.

End of Scan:

- After sedc_done_o pulses high for one clock cycle, all flags (crc_err_o, err_o, sedc_busy_o, and sedc_error_o) de-assert low, marking the end of the current scan.

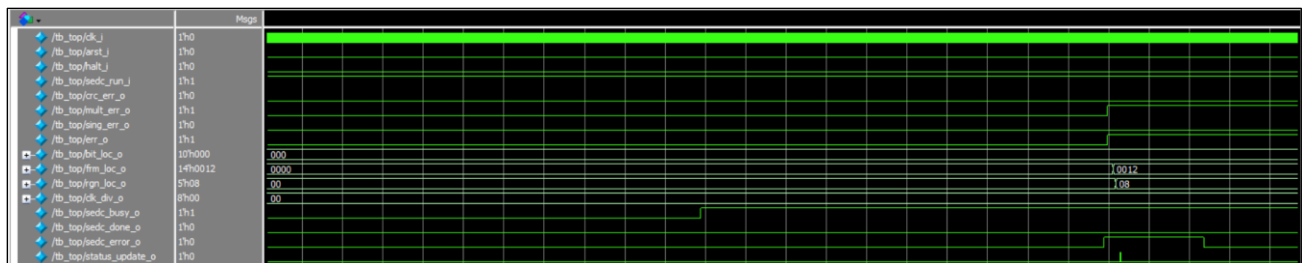


Figure B.15. Second SEDC Scan in Continuous Mode (Multi-bit Error)

Continuous Mode Behavior:

- In continuous mode, sedc_busy_o re-asserts while sedc_run_i remains high. This indicates that the next scan has started.

Note: Multi-bit errors reported in the first scan will continue to be reported in subsequent scans in both simulation and hardware because these errors are uncorrectable.

Appendix C. Example of Including +define+LSCC_SEDC_CONTROLLER_RTL_SIM in Generated *.f File

The following code shows an example of how to include the +define+LSCC_SEDC_CONTROLLER_RTL_SIM directive to simulate the behavior of TMR error injection. In the *.f file generated by the Simulation Wizard, you must add the directive before the file list (ensure that you include this change only if you have selected **RTL** under **Process Stage** in the Simulation Wizard).

```
-L work
-reflib pmi_work
-reflib lav_atx

+define+LSCC_SEDC_CONTROLLER_RTL_SIM

"C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/rtl/sedc_controller_demo.sv"
"C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/sedc_controller_clk_gen.sv"
"C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/sedc_controller_input_s
timulus_gen.sv"
"C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/sedc_controller_output_
monitor.sv"
"C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/sedc_controller_stimulu
s_state_machine.sv"
"C:/Radiant_Projects/sedc_controller_1/sedc_controller_demo/testbench/tb_top.sv"
-sv
-optionset VOPTDEBUG
+noacc+pmi_work.*
+noacc+lav_atx.*

-vopt.options
  -suppress vopt-7033
-end

-gui
-top tb_top
-vsimpl.options
  -suppress vsim-7033,vsim-8630,3009,3389
-end

-do "view wave"
-do "add wave /*"
-do "run -all"
```

Appendix D. Example Verilog/SystemVerilog Code on Asynchronous Reset Timing Requirements

The following code shows an example implementation to meet the asynchronous reset timing requirements in Verilog/SystemVerilog.

```
module async_reset_timing (
    input wire clk_i,          // System clock
    input wire arst_i,         // Asynchronous reset input
    output wire arst_o // Delayed asynchronous reset output
);
    parameter [34:0] CLK_FREQ_HZ = 80000000; // Clock frequency in Hz (80 MHz)
    parameter [34:0] INITIAL_RESET_DELAY_US = 60; // Initial reset delay in µs (60 µs)
    parameter [34:0] MIN_RESET_ASSERTION_NS = 84; // Minimum reset assertion in ns (84 ns)
    localparam [34:0] INITIAL_RESET_DELAY = ((INITIAL_RESET_DELAY_US * CLK_FREQ_HZ) /
1000000) + 1; // Initial reset delay
    localparam [34:0] MIN_RESET_ASSERTION = ((MIN_RESET_ASSERTION_NS * CLK_FREQ_HZ) /
1000000000) + 1; // Minimum reset assertion time

    // Internal registers
    reg [31:0] counter_r;
    reg initial_reset_done_r;
    reg arst_r;

    // Initialize registers
    initial begin
        counter_r = 32'b0;
        initial_reset_done_r = 1'b0;
        arst_r = 1'b1;
    end

    // Wire assignment
    assign arst_o = arst_r;

    always @ (posedge clk_i or posedge arst_i) begin
        if (arst_i) begin
            counter_r <= 32'b0;
            arst_r <= 1'b1;
        end
        else if (!initial_reset_done_r) begin
            if (counter_r < INITIAL_RESET_DELAY) begin
                counter_r <= counter_r + 1;
            end
            else begin
                arst_r <= 1'b0;
                initial_reset_done_r <= 1'b1;
            end
        end
        else begin
            if (counter_r < MIN_RESET_ASSERTION) begin
                counter_r <= counter_r + 1;
            end
            else begin
                arst_r <= 1'b0;
            end
        end
    end
end
```

```
endmodule
```


Appendix E. Example Verilog/SystemVerilog Code for Debouncer

The following code shows an example implementation of a debouncer in Verilog/SystemVerilog.

```
module debounce (
    input wire clock,
    input wire IN,
    output reg OUT
);
    parameter M = 8;
    reg [M:0] shift; // Shift register to wait for stable input

    always @ (posedge clock) begin
        shift <= {shift, IN}; // Shift register
        if (~|shift)
            OUT <= 1'b0;
        else if (&shift)
            OUT <= 1'b1;
        else
            OUT <= OUT;
    end
endmodule
```

Appendix F. Example Verilog/SystemVerilog Code for Reset Synchronizer

The following code shows an example implementation of a reset synchronizer in Verilog/SystemVerilog.

```
module reset_synchronizer (
    input wire arst_i, // Asynchronous reset input
    input wire clk_i,  // Clock input
    output wire arst_o // Synchronized reset output
);

reg [1:0] sync_ff_r; // Synchronizer flip-flops

initial begin
    sync_ff_r = 2'b11;
end

always @(posedge clk_i or posedge arst_i) begin
    if (arst_i) begin
        sync_ff_r <= 2'b11; // Set both flip-flops to 1 on asynchronous reset
    end else begin
        sync_ff_r <= {sync_ff_r[0], 1'b0}; // Shift the reset signal through the flip-flops
    end
end

assign arst_o = sync_ff_r[1]; // Output the synchronized reset signal

endmodule
```

References

- [SEDC Controller IP Release Notes \(FPGA-RN-02081\)](#)
- [Lattice Avant SED/SEC User Guide \(FPGA-TN-02290\)](#)
- [Avant-E](#) web page
- [Avant-G](#) web page
- [Avant-X](#) web page
- [Lattice Radiant Software](#) web page
- [Lattice Solutions IP Cores](#) web page
- [Lattice Solutions Reference Designs](#) web page
- [Lattice Insights](#) for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.2, IP v1.1.0, December 2025

Section	Change Summary
Appendix B. Walkthrough of Example Simulation Waveforms	<ul style="list-style-type: none">Moved existing content into newly created Port-Driven Dynamic Mode section.Added the Continuous Mode section.

Revision 1.1, IP v1.1.0, September 2025

Section	Change Summary
Introduction	Added IP core version to <i>Lattice</i> Implementation in Table 1.1. Summary of the SEDC Controller IP.

Revision 1.0, IP v1.0.0, June 2025

Section	Change Summary
All	Initial release.



www.latticesemi.com