LatticeECP3, ECP5 and ECP5-5G Soft Error Detection (SED)/Correction (SEC) Usage Guide

Technical Note

FPGA-TN-02207-2.0

May 2022

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 associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. 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. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. 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.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CRC	Cycle Redundancy Code
DSP	Digital Signal Processing
EBR	Embedded Block RAM
PFU	Programmable Functional Unit
SEC	Soft Error Correction
SED	Soft Error Detect
SEI	Soft Error Injection
SEU	Single Event Upset
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory

1. Introduction

Soft errors occur when high-energy charged particles alter the stored charge in a memory cell in an electronic circuit. The phenomenon first became an issue in DRAM, requiring error detection and correction for large memory systems in high-reliability applications. As device geometries have continued to shrink, the probability of soft errors in SRAM has become significant for some systems. Designers are using a variety of approaches to minimize the effects of soft errors on system behavior.

SRAM-based FPGAs store logic configuration data in SRAM cells. As the number and density of SRAM cells in an FPGA increase, the probability that a soft error alters the programmed logical behavior of the system increases. A number of approaches have been taken to address this issue, but most involve Intellectual Property (IP) cores that the user instantiates into the logic of their design, using valuable resources and possibly affecting design performance.

This document describes the hardware based Soft Error Detect (SED) approach taken by Lattice Semiconductor for LatticeECP3™ ECP5™ and ECP5-5G™ FPGAs.

Once soft error is detected, Lattice provides an easy way to perform the Soft Error Correction (SEC) without disturbing the functionality of the device. More details are provided in the ECP5 and ECP5-5G Soft Error Correction section.

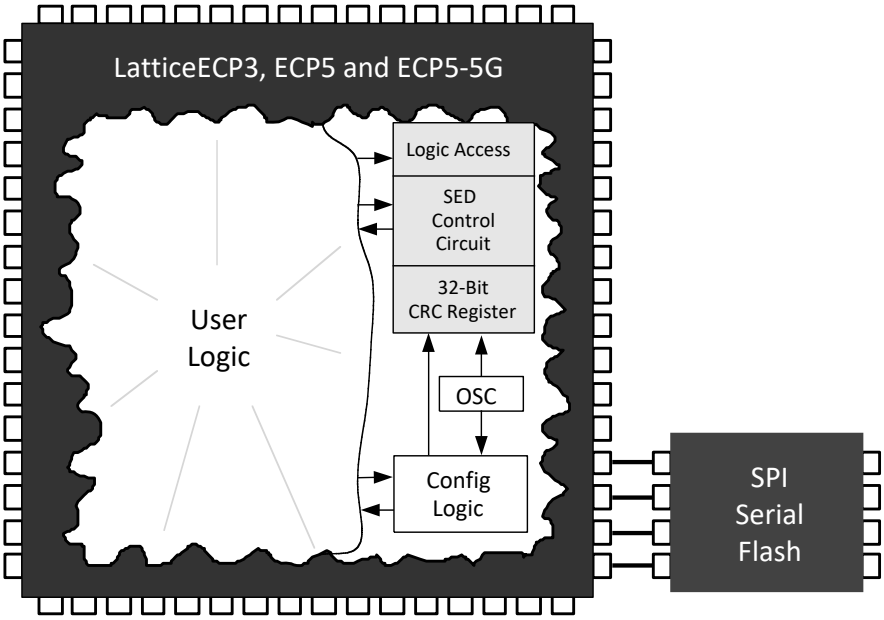
Lattice also provides the tool to help you emulate soft error impact by inserting soft error into the device. More details are provided in the ECP5 and ECP5-5G Soft Error Injection section.

Table 1.1. SED/SEC/SEI Features

	SED	SEC	SEI
LatticeECP3	Yes	—	—
ECP5 and ECP5-5G	Yes	Yes	Yes

2. SED Overview

The SED hardware in the LatticeECP3, ECP5 and ECP5-5G devices consists of an access point to FPGA configuration memory, a controller circuit, and a 32-bit register to store the CRC for a given bitstream (see Figure 2.1). The SED hardware reads serial data from the FPGA's configuration memory and calculates a CRC. The calculated CRC is then compared with the expected CRC that was stored in the 32-bit register. If the CRC values match it indicates that there has been no configuration memory corruption, but if the values differ an error signal is generated. SED checking does not impact the performance or operation of the user logic.



Note: Any kind of configuration memory can be used, including the SPI configuration shown.

Figure 2.1. System Block Diagram

Note that the calculated CRC is based on the particular arrangement of configuration memory for a particular design. Consequently, the expected CRC results cannot be specified until after the design is placed and routed. The Lattice Diamond® bitstream generation software analyzes the configuration of a placed and routed design and updates the 32-bit SED CRC register contents during bitstream generation. EBR and distributed memory contents are ignored in CRC generation.

The following sections describe the LatticeECP3, ECP5 and ECP5-5G SED implementation and flow, along with some sample code with which to get started.

3. SED Hardware Description

As shown in Figure 3.1, the LatticeECP3, ECP5 and ECP5-5G SED hardware has several inputs and outputs that allow you to control, and monitor, SED behavior.

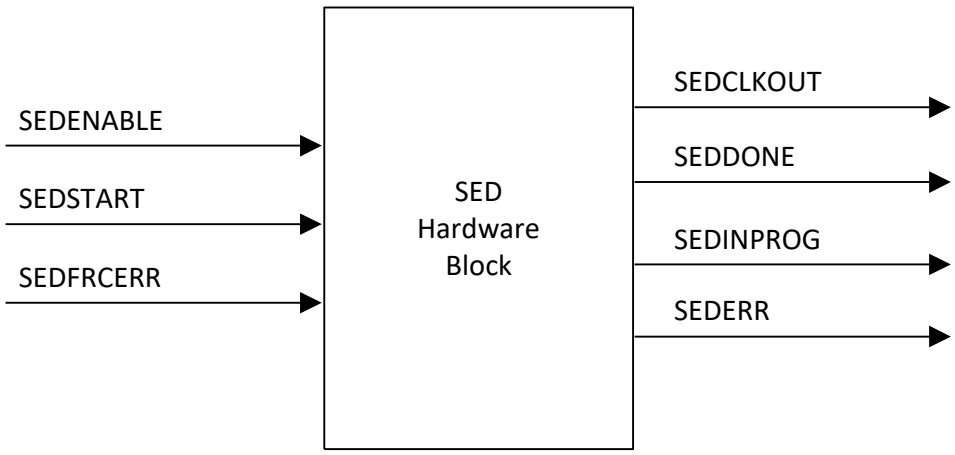


Figure 3.1. Signal Block Diagram

4. SED Signal Descriptions

Table 4.1. SED Signal Descriptions

Signal Name	Direction	Active	Description
SEDENABLE	Input	High	SED enable
SEDCLKOUT	Output	N/A	Output clock
SEDSTART	Input	High	Start SED cycle
SEDINPROG	Output	High	SED cycle is in progress
SEDDONE	Output	High	SED cycle is complete
SEDFRCERR	Input	High	Force an SED error flag
SEDERR	Output	High	SED error flag

4.1. SEDCLK

Clock input to the SED hardware.

This clock is derived from the on-chip oscillator of the LatticeECP3, ECP5 and ECP5-5G devices. The on-chip oscillator's output goes through a divider to create MCLK. MCLK goes through another divider to create SEDCLK in LatticeECP3. The on-chip oscillator's output goes through a separate divider to create SEDCLK in ECP5 and ECP5-5G devices. This is shown in Figure 4.1.

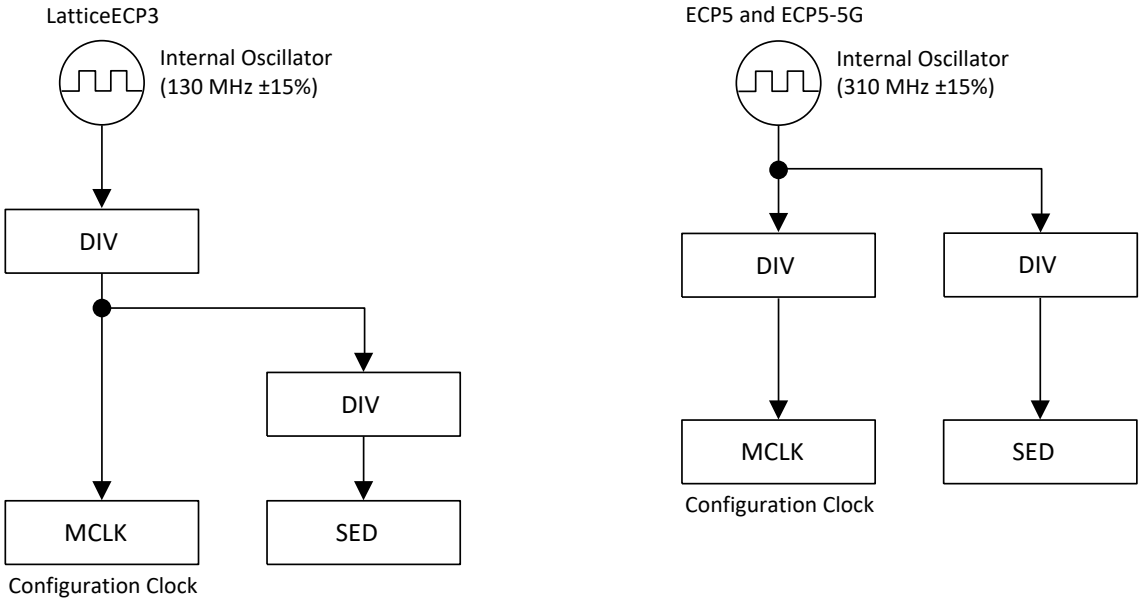


Figure 4.1. Divider Diagram

For LatticeECP3, the software default for MCLK is 2.5 MHz, but this can be modified using the MCCLK_FREQ global preference in Diamond. The divider for SEDCLK can be set to 1 (default), 2, 4, 8, 16 or 32. So the default SEDCLK frequency is 2.5 MHz. The divider value can be set using a parameter, see the example code in SED Sample Code section. Care must be taken to ensure that the SEDCLK setting is above 2.5 MHz.

Table 4.2. Possible LatticeECP3 MCLK Values (MHz)

MHz
2.5
4.3
5.4
6.9
8.1
9.2
10
13
15
20
26
30
33

For ECP5 and ECP5-5G devices, the oscillator frequency is 310 MHz. The divider for SEDCLK can be set to 128 (default), 64, 32, 16, 8 or 5. The default SEDCLK frequency is 2.4 MHz. The SEDCLK frequency can be set using a parameter, see the example code in SED Sample Code section.

Table 4.3. Possible ECP5 and ECP5-5G SEDCLK Frequency

DIV	MHz
128	2.4
64	4.8
32	9.7
16	19.4
8	38.8
5	62.0

4.2. SEDENABLE

Active high input to the SED hardware. It is a level-sensitive signal.

State	Description
1	Enables output of SEDCLKOUT, arms SED hardware.
0	Aborts SED and forces all SED hardware outputs low.

4.3. SEDCLKOUT

Gated version of SEDCLK, SEDCLKOUT is gated by SEDENABLE, therefore you can use this signal to synchronize the inputs.

4.4. SEDSTART

Active high input to the SED hardware. If sedstart is tied high, it keeps rechecking until it is brought low and the current cycle finishes. It can also be pulsed, and can be brought low after SEDINPROG goes high, if desired. It is a level-sensitive signal. This signal has to be synchronized by SEDCLKOUT to prevent spurious assertion of the SEDERR output.

State	Description
1	Start error detection. Must be high a minimum of one SEDCLKIN period.
0	No action.

4.5. SEDFRCERR

Active high input to the SED hardware. As long as this signal is held low, SEDERR stays active. It can be asserted at any time.

State	Description
1	Forces SEDERR high, simulating an SED error.
0	No action.

4.6. SEDINPROG

Active high output from the SED hardware.

State	Description
1	SED checking is in progress, goes high on the clock following SEDSTART high.
0	SED checking is not active.

4.7. SEDDONE

Active high output from the SED hardware.

State	Description
1	SED checking is complete. Reset by a high on SEDSTART or a low on SEDENABLE.
0	SED checking is not complete.

4.8. SEDERR

Active high output from the SED hardware. SEDERR indicates that a CRC mismatch was found between the FPGA's configuration bits and the CRC value held in the 32-bit CRC register. It performs no other action than flagging the mismatch. It is up to the designer to evaluate and react to the assertion of SEDERR.

State	Description
1	SED has detected an error. Reset by SEDENABLE going low.
0	SED has not detected an error.

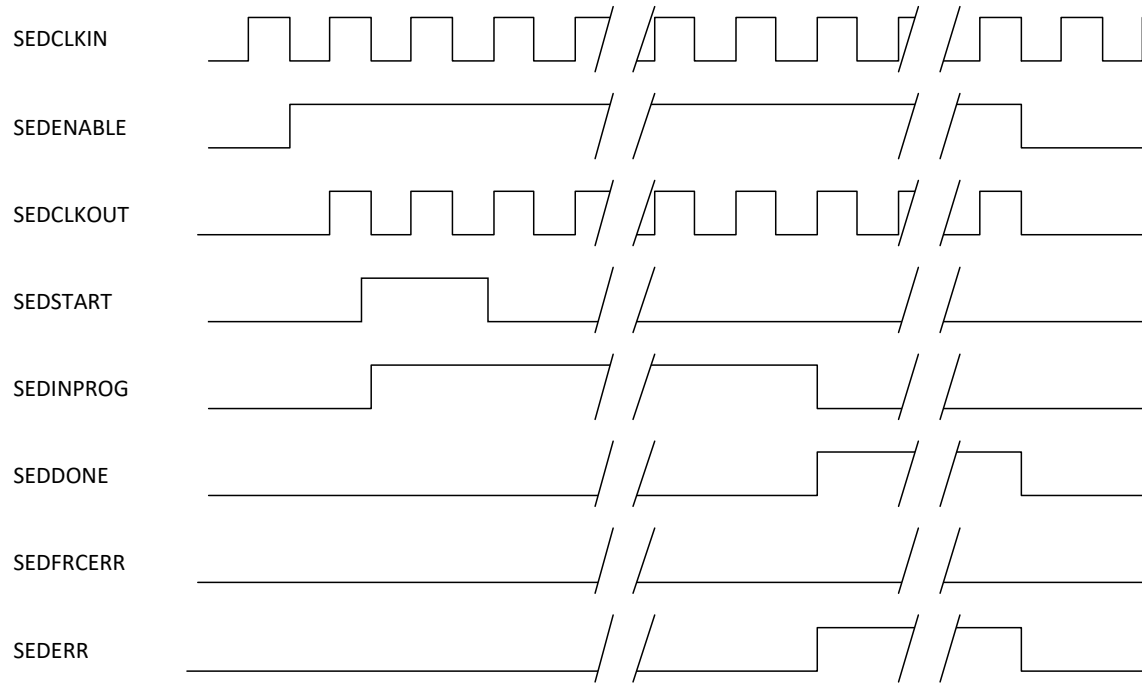
4.9. AUTODONE

Reserved for future use.

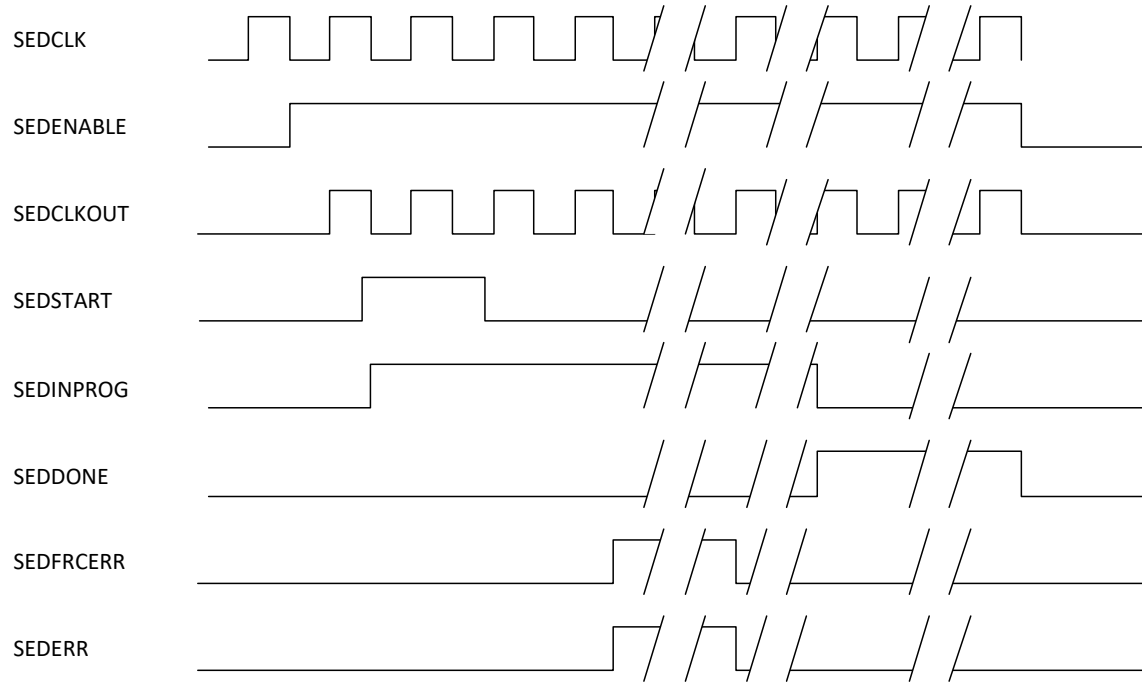
4.10.MCCLK_freq Attribute

Frequency attribute which specifies how fast MCLK is for simulation purposes. For design implementation purposes, MCLK must be set using Diamond Global Preference. Both the simulation and Global Preference setting should match or else there is a DRC error.

5. SED Flow



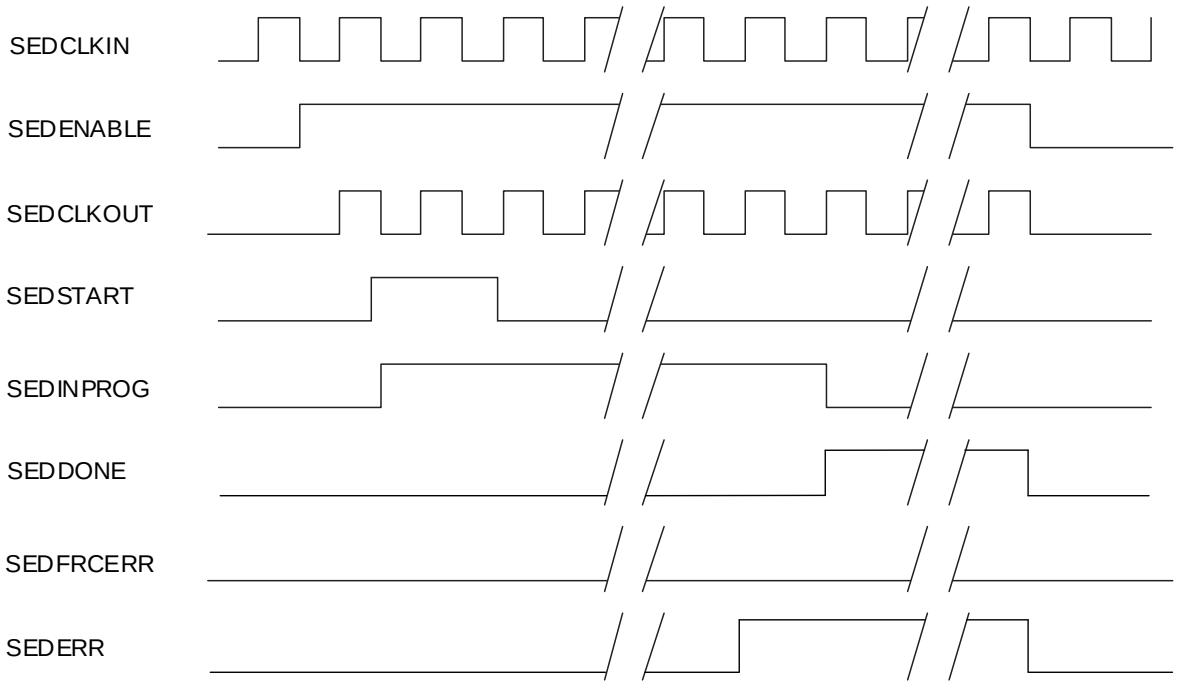
Normal Failure



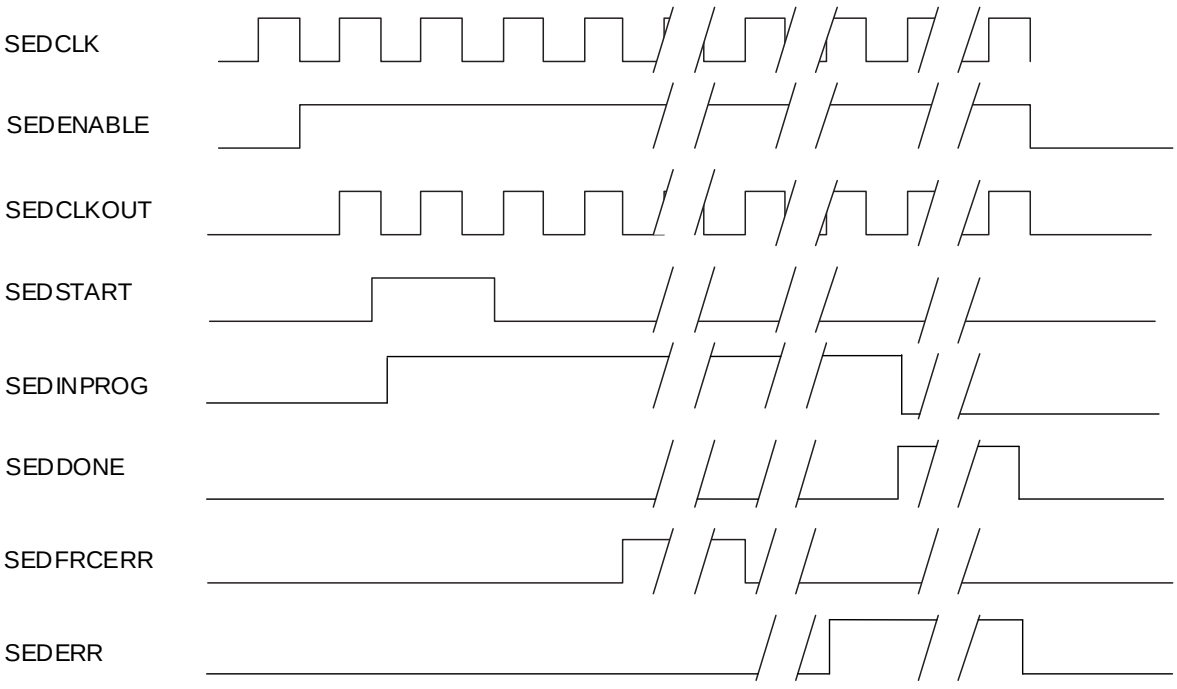
Failure Forced With SEDFRCERR

Note: SED, by its inherent nature, has limited simulation support. In simulation, use the SEDFRCERR signal to force errors.

Figure 5.1. Timing Diagram for ECP3



Normal Failure



Failure Forced with SEDFRCERR

Note: SED, by its inherent nature, has limited simulation support. In simulation, use the SEDFRCERR signal to force errors.

Figure 5.2. Timing Diagram for ECP5 and ECP5-5G

LatticeECP3, ECP5 and ECP5-5G device SED flow:

1. Toggle SEDENABLE after each time the LatticeECP3 is reprogrammed:
   a. Deassert SEDENABLE for one clock cycle.
   b. Assert SEDENABLE for one clock cycle.
   c. Deassert SEDENABLE for one clock cycle.

   This initializes the SED system and must be done prior to testing for soft errors. Following this sequence prevents spurious assertion of the SEDERR output.
2. After initializing the SED logic (step 1 above) SED is ready to use. First, assert SEDENABLE.
3. Assert SEDSTART (active high). SEDINPROG goes the CRC.
4. The calculated CRC is compared with the stored CRC and SEDERR is updated, SEDINPROG goes low, and SEDDONE goes high.
5. When SEDERR is high, it can be reset by bringing SEDENABLE low or reconfiguring the FPGA.

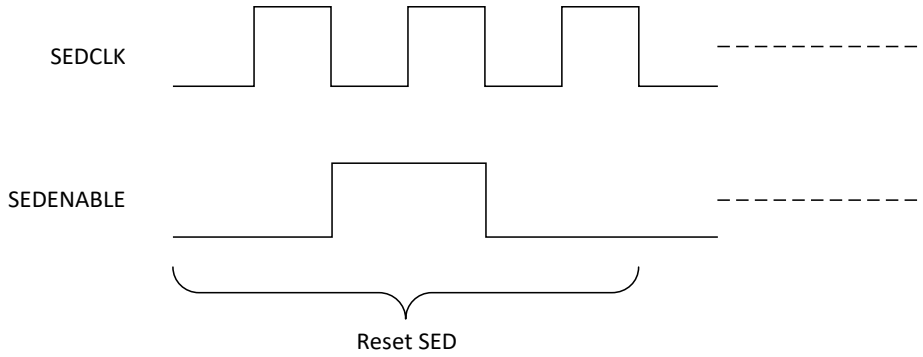


Figure 5.3. SED Initialization

To trigger reconfiguration upon error detection, Figure 5.4 shows one possible solution.

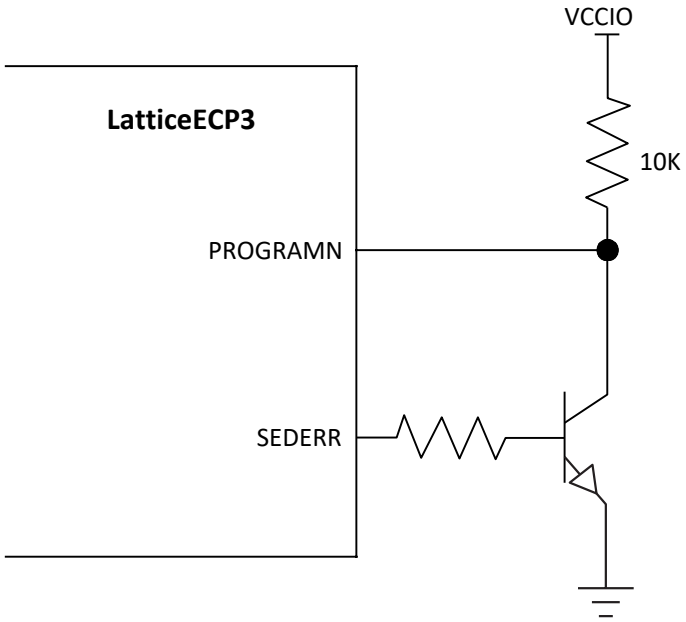


Figure 5.4. Example Schematic

6. SED Run Time

The amount of time needed to perform an SED check depends on the density of the device and the frequency of SEDCLK. There is also some overhead time for calculation, but it is fairly short in comparison. An approximation of the time required can be found by using the formula below.

For LatticeECP3:

Time (max) = Num_configuration_bits (Mb)/(SEDCLK * 0.85) # 15% low side clock

Time (min) = Num_configuration_bits (Mb)/(SEDCLK * 1.15) # 15% high side clock

For ECP5 and ECP5-5G:

Time (max) = Num_configuration_bits (Mb)/(SEDCLK * 0.85) / 8 # 15% low side clock

Time (min) = Num_configuration_bits (Mb)/(SEDCLK * 1.15) / 8 # 15% high side clock

For example, if the design uses a LatticeECP3 with 95K look-up tables and the SEDCLK is the software default of 2.5 MHz:

Time (max) = 19 Mb/(2.5 MHz * 0.85) = 8.95 seconds

Time (min) = 19 Mb/(2.5 MHz * 1.15) = 6.6 seconds

In this example, SED checking takes between 6.6 and ~9 seconds. Remember that this happens in the background and does not affect user logic performance.

Note that the internal oscillator used to generate SEDCLK can vary by ±15%, as shown in the min/max calculations.

The density of the FPGA determines the value of the Num_configuration_bits value. Table 6.1 provides information on the number of configuration bits for LatticeECP3, ECP5 and ECP5-5G FPGAs. SEDCLK is frequency in MHz. Time is in seconds.

Table 6.1. SED Run Time

Density	Bitstream Size (Mb)	Run Time (Seconds)	Run Time (Max)	Run Time (Min)
ECP3-95	19	7.6[1]	9	6.6
LFE5-85	19	0.99[2]	1.23	0.84

Notes:

1. Based on SEDCLK = 2.5 MHz.
2. Based on SEDCLK = 2.4 MHz.

7. SED Sample Code

The following simple example code shows how to instantiate the SED. In the example the outputs of the SED hardware have been routed to FPGA output pins.

7.1. VHDL Example for LatticeECP3

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity SEDCA_VHDL_Test is
port(
       SED_Done : out std_logic;
       SED_In_Prog : out std_logic;
       SED_Clk_Out : out std_logic;
       SED_Out : out std_logic;
       SEDENABLEx : in std_logic;
       SEDSTARTx : in std_logic);
end;

architecture behavioral of SEDCA_VHDL_Test is
       component SEDC
             generic (OSC_DIV : integer :=1 ;
             CHECKALWAYS : string :="DISABLED";
             AUTORECONFIG: string :="OFF" ;
             MCCLK_FREQ : string :="2.5" ;
             DEV_DENSITY : string :="95K" );
port (
       SEDENABLE : in std_logic;
       SEDSTART : in std_logic;
       SEDFRCERR : in std_logic;
       SEDERR : out std_logic;
       SEDDONE : out std_logic;
       SEDINPROG : out std_logic;
       SEDCLKOUT : out std_logic;
       end component;
begin
       inst1: SEDC
       port map (
             SEDENABLE => SEDENABLEx,
             SEDSTART => SEDSTARTx,
             SEDFRCERR => '0',
             SEDERR => SED_Out,
             SEDDONE => SED_Done,
             SEDINPROG => SED_In_Prog,
             SEDCLKOUT => SED_Clk_Out
);
end behavioral;
```

7.2. Verilog Example for LatticeECP3

```
module SEDCA_Verilog_Test( SEDFRCERR, SEDERR, SEDDONE, SEDINPROG, SEDCLKOUT,
  SEDENABLEx, SEDSTARTx);

       input SEDFRCERR, SEDENABLEx, SEDSTARTx;
       output SEDERR, SEDDONE, SEDINPROG, SEDCLKOUT;

SEDC
       (.CHECKALWAYS("DISABLED"),
       .AUTORECONFIG("OFF"),
       .OSC_DIV(1),
       .DEV_DENSITY("95K"),
       .MCCLK_FREQ("2.5"))
sed_ip (
       .SEDENABLE(SEDENABLEx),
       .SEDSTART(SEDSTARTx),
       .SEDFRCERR(SEDFRCERR),
       .SEDERR(SEDERR),
       .SEDDONE(SEDDONE),
       .SEDINPROG(SEDINPROG),
       .SEDCLKOUT(SEDCLKOUT)
       );
endmodule
```

7.3. VHDL Example for ECP5 and ECP5-5G

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity SEDGA_VHDL_Test is
port(
       SED_Done : out std_logic;
       SED_In_Prog : out std_logic;
       SED_Clk_Out : out std_logic;
       SED_Out : out std_logic;
       SEDENABLEx : in std_logic;
       SEDSTARTx : in std_logic);
end;
architecture behavioral of SEDGA_VHDL_Test is
       component SEDGA
       generic (SED_CLK_FREQ : string :="2.4" ;  -- 2.4, 4.8, 9.7, 19.4, 38.8, 62.0
       CHECKALWAYS : string :="DISABLED";
       DEV_DENSITY : string :="85KU" );  --12KU, 12KUM, 25KU, 25KUM, 45KU, 45KUM, 85KU, 85KUM
port (
       SEDENABLE : in std_logic;
       SEDSTART : in std_logic;
       SEDFRCERR : in std_logic;
       SEDERR : out std_logic;
       SEDDONE : out std_logic;
       SEDINPROG : out std_logic;
       SEDCLKOUT : out std_logic);
       end component;
begin
       inst1: SEDGA
       port map (
              SEDENABLE => SEDENABLEx,
              SEDSTART => SEDSTARTx,
              SEDFRCERR => '0',
              SEDERR => SED_Out,
              SEDDONE => SED_Done,
              SEDINPROG => SED_In_Prog,
              SEDCLKOUT => SED_Clk_Out
);
end behavioral;
```

7.4. Verilog Example for ECP5 and ECP5-5G

```
module SEDGA_Verilog_Test( SEDFRCERR, SEDERR, SEDDONE, SEDINPROG, SEDCLKOUT, SEDENABLEx,
SEDSTARTx);
       input SEDFRCERR, SEDENABLEx, SEDSTARTx;
       output SEDERR, SEDDONE, SEDINPROG, SEDCLKOUT;

SEDGA #
       (.SED_CLK_FREQ("2.4"),      // 2.4, 4.8, 9.7, 19.4, 38.8, 62.0
       .CHECKALWAYS("DISABLED"),
       .DEV_DENSITY("85KU"))       // 12KU, 12KUM, 25KU, 25KUM, 45KU, 45KUM, 85KU, 85KUM
sed_ip_inst(
       .SEDENABLE(SEDENABLEx),
       .SEDSTART(SEDSTARTx),
       .SEDFRCERR(SEDFRCERR),
       .SEDERR(SEDERR),
       .SEDDONE(SEDDONE),
       .SEDINPROG(SEDINPROG),
       .SEDCLKOUT(SEDCLKOUT)
       );

endmodule
```

8. ECP5 and ECP5-5G Soft Error Correction

Soft Error Correction (SEC) is performed with background reconfiguration. All bits are rewritten during the reconfiguration, and the erroneous bit is replaced with correct data. During background reconfiguration, writing the same value into configuration SRAM cells does not affect the SRAM cell output.

Upon completion of background reconfiguration, initialization of the device is bypassed, allowing the user functions to be performed during and after reconfiguration.

To use SEC:

1. Select or enable the **BACKGROUND_RECONFIG** option in Diamond Spreadsheet View when you generate the bitstream.
2. Set the synthesis option to disable distributed RAM during the synthesize process.

Once soft error is detected, either inserted or by radiation, there are multiple ways to correct it depending on your preference.

8.1. SPI Master Mode

If you are using SPI Master mode to download bitstream from an external SPI flash to the ECP5 and ECP5-5G device, you can issue a refresh instruction or toggle the PROGRAMN pin to reload the original bitstream from the SPI flash. The DONE pin goes low during the configuration and goes back to high after the configuration.

To reinitialize the device, power cycle the device or issue offline mode instructions into the device to break the background mode.

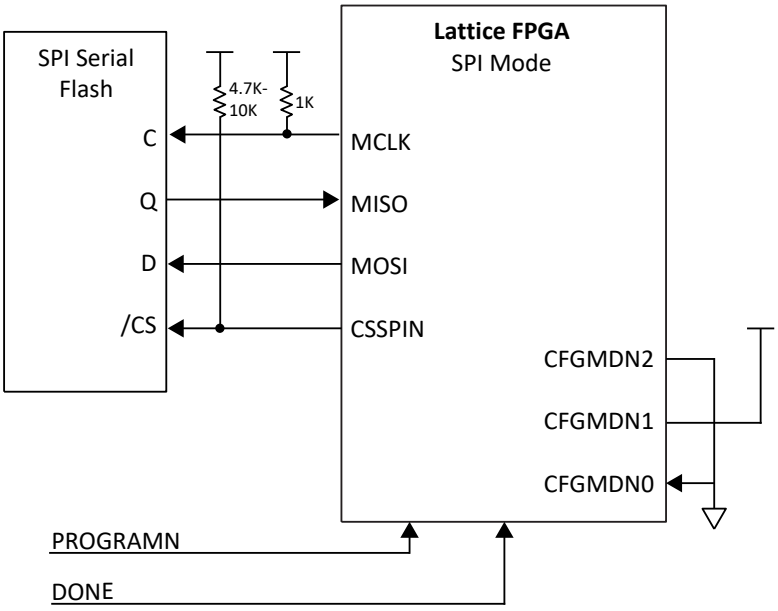


Figure 8.1. ECP5 and ECP5-5G Master SPI Port with SPI Flash

8.2. JTAG Mode

If the bitstream is loaded into the ECP5 and ECP5-5G through an external controller through the JTAG port and you want to re-program the device without the device function being disturbed, select the **XSRAM SEI Fast Program** operation in the Diamond Programmer to create embedded programming and load the original bitstream into the SRAM of the device.

When the ECP5 and ECP5-5G device is in JTAG mode, to re-initialize the device, re-program the device using offline mode, for instance, use the FAST PROGRAM operation.

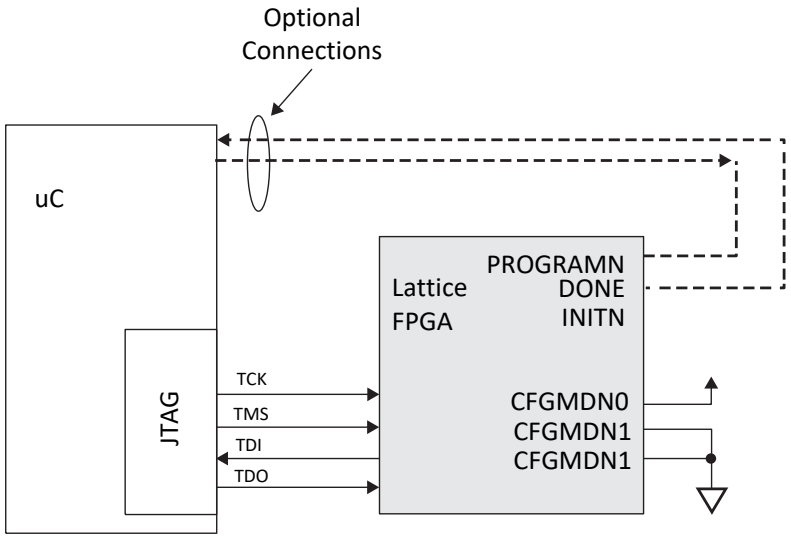


Figure 8.2. JTAG

9. ECP5 and ECP5-5G Soft Error Injection

Lattice Semiconductor provides the Diamond Soft Error Injection (SEI) tool which offers an easy and economical way to emulate the soft error impact to the overall system. This tool allows you to randomly generate and program one or multiple soft errors into the device in background mode without disturbing the device function. Soft Error Injection is only supported when the device configuration mode is being set to slave mode.

To use the Diamond SEI tool:

1. Select or enable the **BACKGROUND_RECONFIG** option in Diamond Spreadsheet View when you generate the bitstream.
2. Set the synthesis option to disable distributed RAM during the synthesize process.
3. Run SEI Editor under Tools. This allows you to create a one frame special bitstream that has one bit different from your original bitstream.

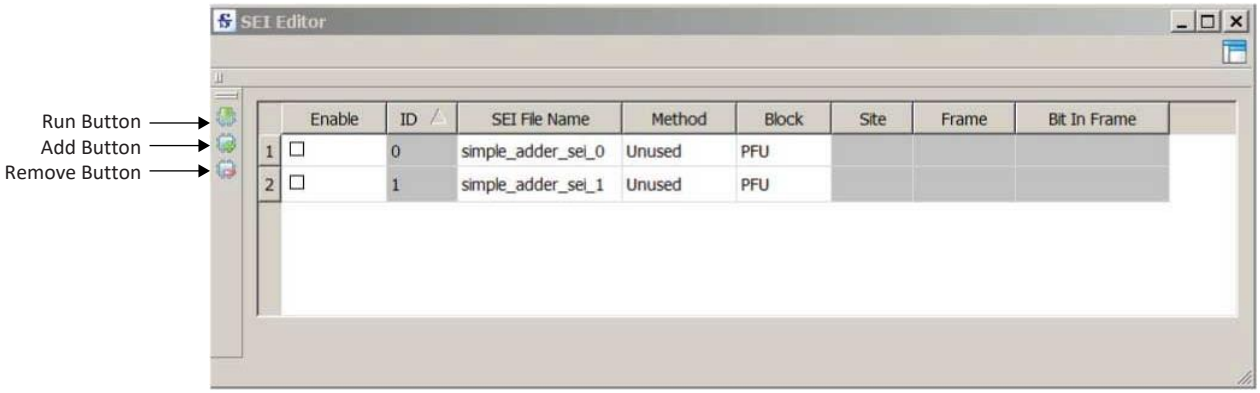


Figure 9.1. SEI Editor

9.1. Method

Unused – Error bit introduced is not used by customer design.

Random – Error bit introduced is random and can be used by customer design.

9.2. Block

Unused – The Block can be selected among PFU, EBR or DSP.

Random – Any functional block including routing. This is not user-selectable.

SEI File Name – Default bitstream name. This may be changed by the user.

ID – Continuous number assigned by software.

Enable – Enable to run. This particular bitstream is generated when you click the Run button.

Add Button – Click this button to add SEI files.

Remove Button – Click this button to remove SEI files.

Run Button – Click this button to generate SEI files.

Note: The gray area cannot be selected.

To program the one-bit-different bitstream, select the *XSRAM SEI Fast Program* operation in Diamond Programmer. Once it is programmed into the device, you can use the general SED routine to detect the soft error. During the SEI Fast Program, you can see the DONE pin go low during the configuration and back to high after the configuration.

Note: While programming the device with soft error bitstream, the SED checking has to stop. You can de-assert SEDENABLE to stop the SED checking.

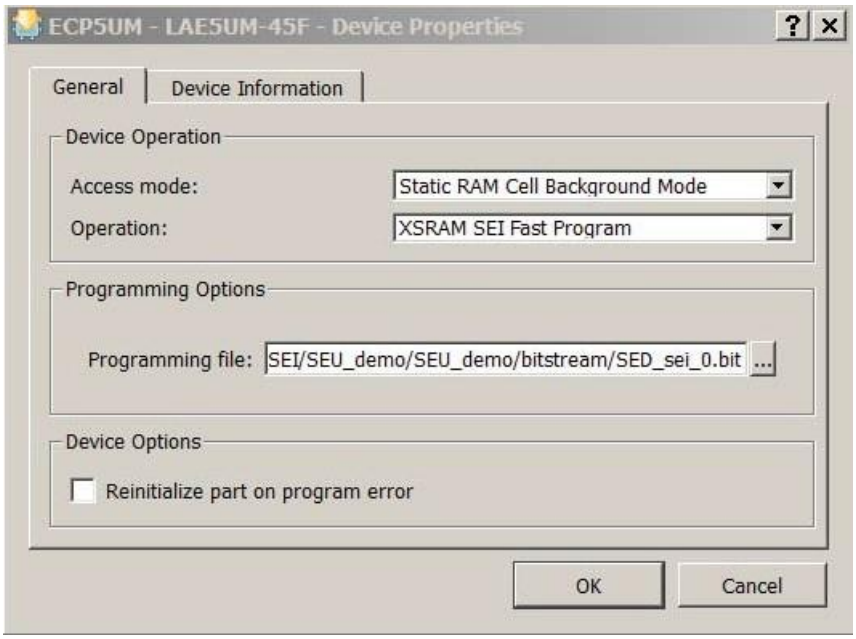


Figure 9.2. Device Properties

For details and step-by-step guide on using the SEI tool, refer to SEU Demo for the ECP5 and ECP5-5G Versa Evaluation Board (UG92).

Technical Support Assistance

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

Revision History

Revision 2.0, May 2022

Section	Change Summary
SED Sample Code	• Updated Section 7.3 VHDL Example for ECP5 and ECP5-5G. Removed 77.5 and 155.0, and added 62.0 to SED_CLK_FREQ. • Updated Section 7.4 Verilog Example for ECP5 and ECP5-5G. Removed 77.5 and 155.0, and added 62.0 to SED_CLK_FREQ.

Revision 1.9, July 2021

Section	Change Summary
SED Sample Code	Updated VHDL Example for ECP5 and ECP5-5G and Verilog Example for ECP5 and ECP5-5G section.
SED Signal Descriptions	Changed MCCLK to MCLK in SEDCLK and MCCLK_freq Attribute section.

Revision 1.8, April 2021

Section	Change Summary
All	• Changed document number from TN1184 to FPGA-TN-02207. • Updated document template.
Disclaimers	Added this section.
SED Flow	Added Figure 5.2 for ECP5 and ECP5-5G.

Revision 1.7, November 2015

Section	Change Summary
All	• Added support for ECP5-5G. • Changed document title to LatticeECP3, ECP5 and ECP5-5G Soft Error Detection (SED)/Correction (SEC) Usage Guide.
Technical Support Assistance	Updated Technical Support Assistance section.

Revision 1.6, March 2015

Section	Change Summary
All	• Changed document title to LatticeECP3 and ECP5 Soft Error Detection (SED)/Correction (SEC) Usage Guide. • Added the ECP5 SEC (Soft Error Correction) section. • Added the ECP5 SEI (Soft Error Injection) section.
SED Signal Descriptions	Updated the SED Signal Descriptions section. • Added information on SEDENABLE and SEDSTART signals.
SED Flow	Updated the SED Flow section. • Updated Figure 5.1. Timing Diagram. Corrected SEDSTART signal.

Revision 1.5, March 2014

Section	Change Summary
All	• Updated to support ECP5 device family. • Changed document title to LatticeECP3 and ECP5 Soft Error Detection (SED) Usage • Changed instances of LatticeECP4UM to ECP5.

Revision 1.4, July 2013

Section	Change Summary
All	• Added support for LatticeECP4UM. • Changed document title to LatticeECP3 and LatticeECP4UM Soft Error Detection (SED) Usage Guide. • Added support for Lattice Diamond in place of ispLEVER.
SED Overview	Updated Figure 2.1. System Block Diagram.
SED Signal Descriptions	• Updated Figure 4.1. Divider Diagram • Updated SEDCLK section. • Converted Figure 4.3, Possible MCLK Values (MHz) to Table 4.2. Possible LatticeECP3 MCLK Values (MHz). • Added Table 4.2, Possible LatticeECP4UM SEDCLK Frequency. • Updated MCCLK_freq Attribute section. Added Diamond Global Preference in place of ispLEVER Design Planner. • Updated Possible MCLK Values (MHz) table.
SED Run Time	Updated SED Run Time section. Added LatticeECP4UM information to context and Table 6.1. SED Run Time.
SED Sample Code	Updated Sample Code section. Added VHDL and Verilog examples fo LatticeECP4UM
Technical Support Assistance	Updated Technical Support Assistance information.

Revision 1.3, February 2012

Section	Change Summary
All	Updated document with new corporate logo.

Revision 1.2, January 2011

Section	Change Summary
SED Signal Descriptions	Removed references to AUTODONE in Sample Code section.

Revision 1.1, November 2009

Section	Change Summary
SED Flow	• Updated SED flow. • Updated Example Schematic diagram.
SED Sample Code	Updated VHDL and Verilog sample code.

Revision 1.0, February 2009

Section	Change Summary
All	Initial release.

www.latticesemi.com