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

November 2015 Technical Note TN1184

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 will alter 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 SEC (Soft Error Correction) section.

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

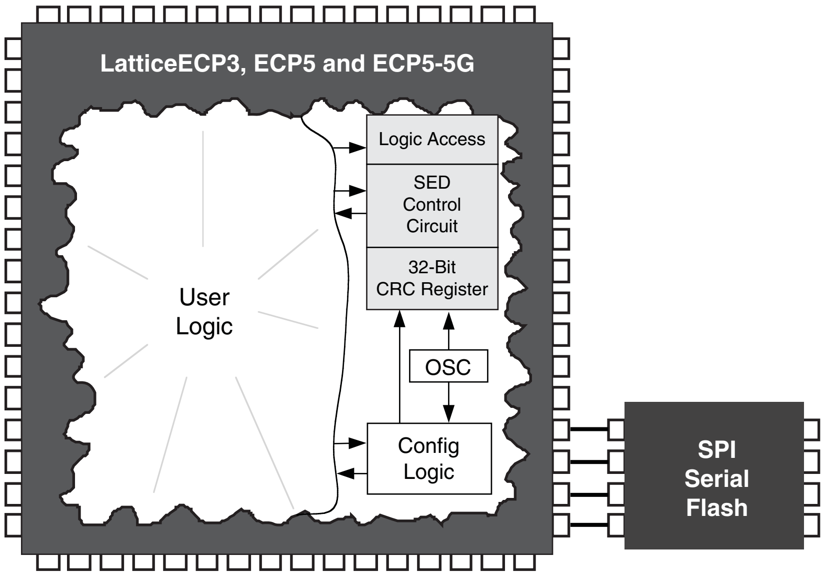
Table 1. SED/SEC/SEI Features

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

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 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.

Figure 1. System Block Diagram[1]



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

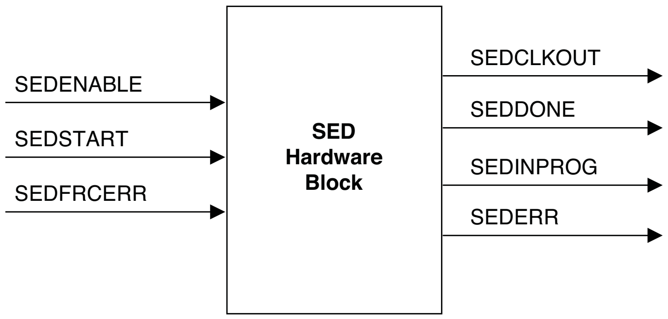
Note that the calculated CRC is based on the particular arrangement of configuration memory for a particular design. Consequently, the expected CRC results cannot be specified until after the design is placed and routed. The Lattice Diamond® bitstream generation software analyzes the configuration of 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 to get started with.

SED Hardware Description

As shown in Figure 2, the LatticeECP3, ECP5 and ECP5-5G SED hardware has several inputs and outputs that allow the user to control, and monitor, SED behavior.

Figure 2. Signal Block Diagram



SED Signal Descriptions

Table 2. SED Signal Descriptions

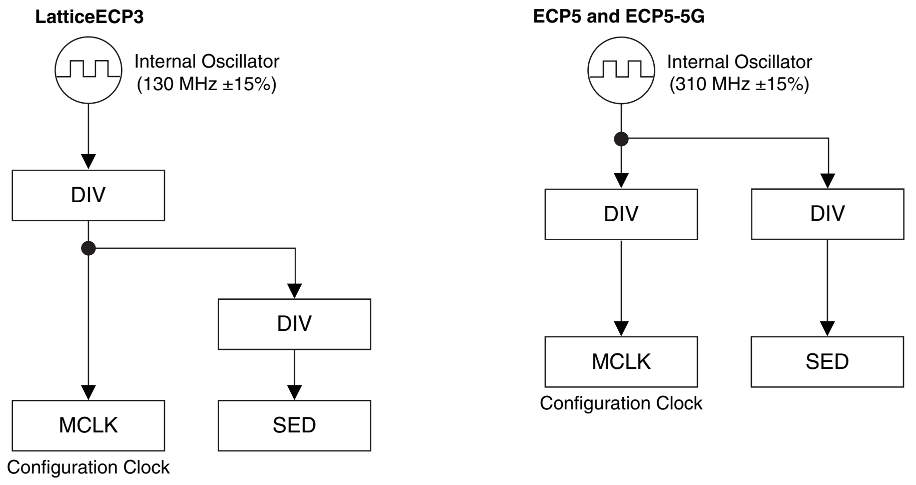
Signal Name	Direction	Active	Description
SEDENABLE	Input	High	SED enable
SEDCLKOUT	Output	N/A	Output clock
SEDSTART	Input	High	Start SED cycle
SEDINPROG	Output	High	SED cycle is in progress
SEDDONE	Output	High	SED cycle is complete
SEDFRCERR	Input	High	Force an SED error flag
SEDERR	Output	High	SED error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

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 MCCLK. MCCLK 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 3.

Figure 3. Divider Diagram



For LatticeECP3, the software default for MCCLK 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 at the end of this document. Care must be taken to ensure that the SEDCLK setting is above 2.5 MHz.

Table 3. 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. so the default SEDCLK frequency is 2.4 MHz. The SEDCLK frequency can be set using a parameter, see the example code at the end of this document.

Table 4. 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

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.

SEDCLKOUT

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

SEDSTART

Active high input to the SED hardware. If sedstart is tied high, it will keep 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.

SEDFRCERR

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

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

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.

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.

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.

AUTODONE

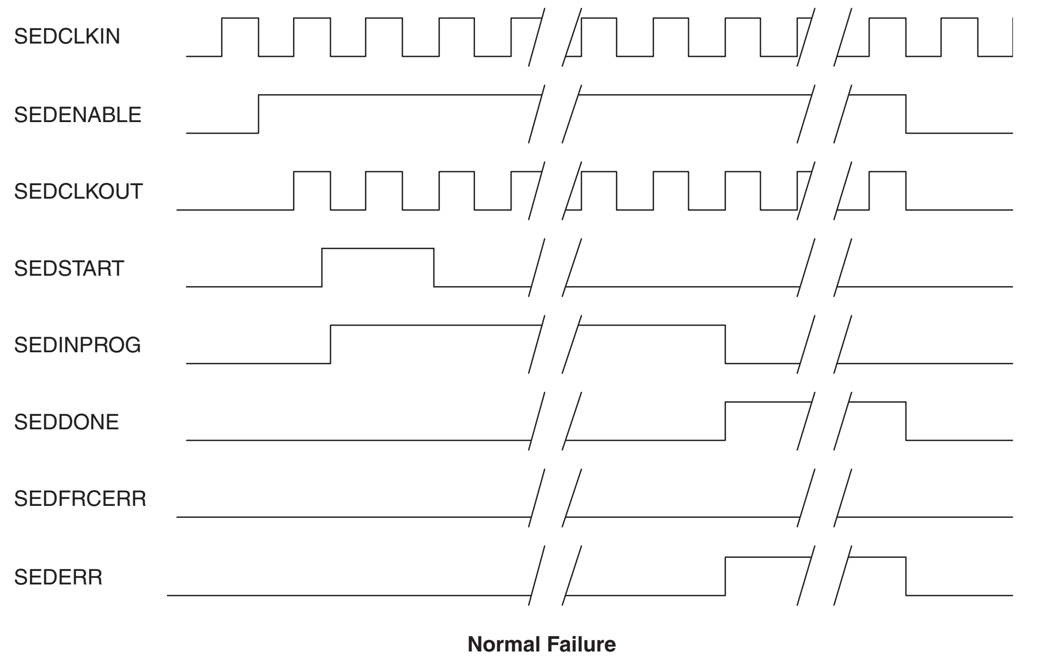
Reserved for future use.

MCCLK_freq Attribute

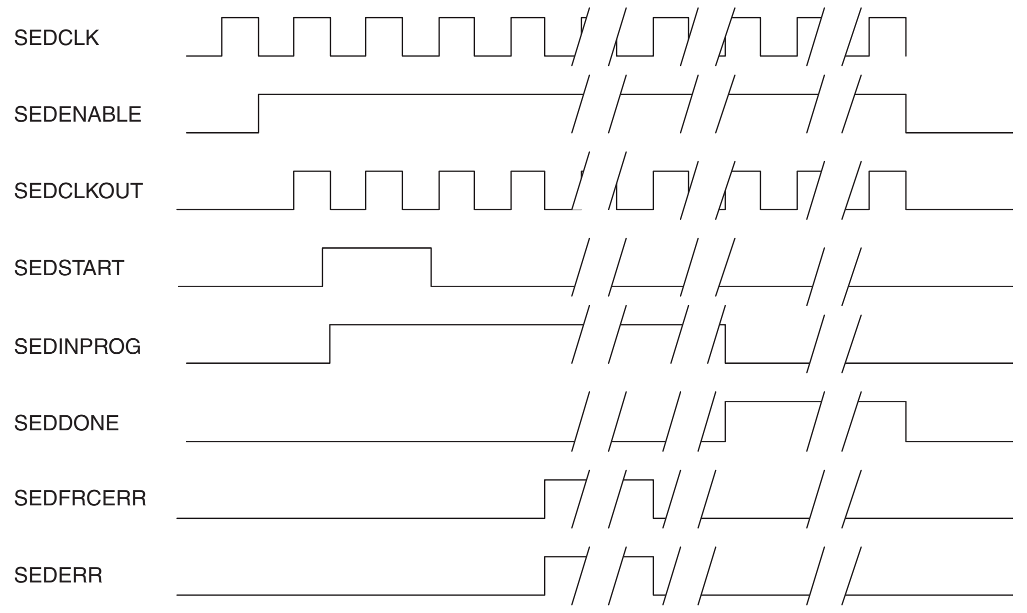
Frequency attribute which specifies how fast MCCLK is for simulation purposes. For design implementation purposes, MCCLK must be set using Diamond Global Preference. Both the simulation and Global Preference setting should match or else there will be a DRC error.

SED Flow

Figure 4. Timing Diagram



Normal Failure



Failure Forced With SEDFRCERR

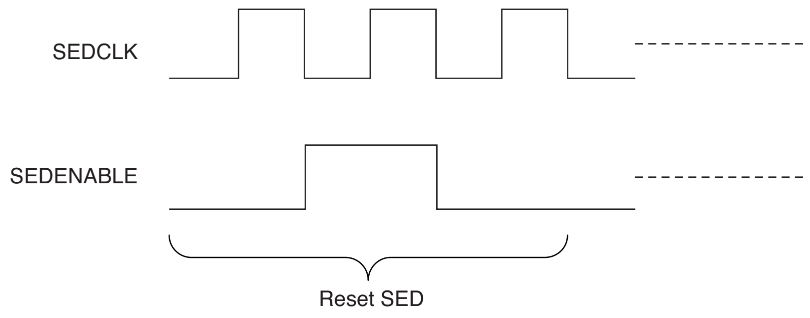
Note: SED, by its inherent nature, has limited simulation support. In simulation, use the SEDFRCERR signal to force errors.

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.

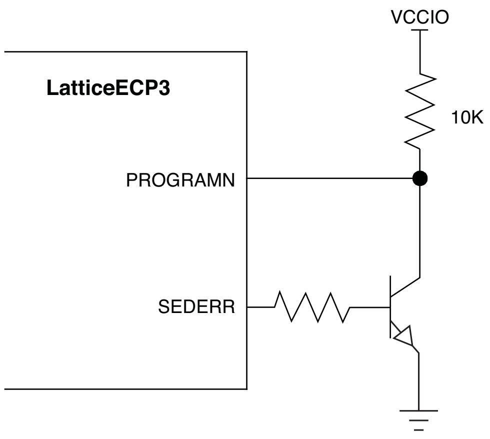
   This initializes the SED system and must be done prior to testing for soft errors. Following this sequence prevents spurious assertion of the SEDERR output.
2. After initializing the SED logic (step 1 above) SED is ready to use. First, assert SEDENABLE.
3. Assert SEDSTART (active high). SEDINPROG will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

Figure 5. SED Initialization



To trigger reconfiguration upon error detection, Figure 6 shows one possible solution.

Figure 6. Example Schematic



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 will also be 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 Mbits / (2.5 MHz * 0.85) = 8.95 seconds

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

In this example, SED checking will take 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 5 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 5. 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

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

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.

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;
```

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
```

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 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 SEDGA_VHDL_Test is
      component SEDGA
        generic (   SED_CLK_FREQ : string :="2.4" ;
        CHECKALWAYS : string :="DISABLED";
        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: 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;
```

Verilog Example for ECP5 and ECP5-5G

```
module SEDGA_Verilog_Test( SEDFRCERR, SEDERR, SEDDONE, SEDINPROG, SEDCLKOUT, SEDEN-
ABLEx, SEDSTARTx);

      input SEDFRCERR, SEDENABLEx, SEDSTARTx;

      output SEDERR, SEDDONE, SEDINPROG, SEDCLKOUT;

SEDGA

      (.SED_CLK_FREQ("2.4")

      .CHECKALWAYS("DISABLED"),

      .DEV_DENSITY("95K"))

sed_ip(

      .SEDENABLE(SEDENABLEx),

      .SEDSTART(SEDSTARTx),

      .SEDFRCERR(SEDFRCERR),

      .SEDERR(SEDERR),

      .SEDDONE(SEDDONE),

      .SEDINPROG(SEDINPROG),

      .SEDCLKOUT(SEDCLKOUT)

      );

endmodule
```

ECP5 and ECP5-5G SEC (Soft Error Correction)

SEC error correction 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.

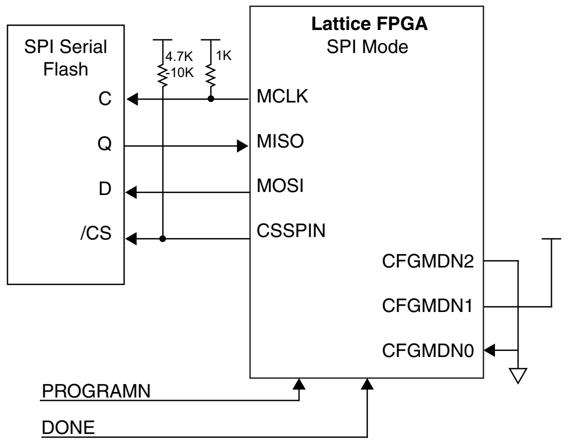
Once soft error is detected, either inserted or by radiation, there are multiple ways to correct it depending on your preference.

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 7. ECP5 and ECP5-5G Master SPI Port with SPI Flash

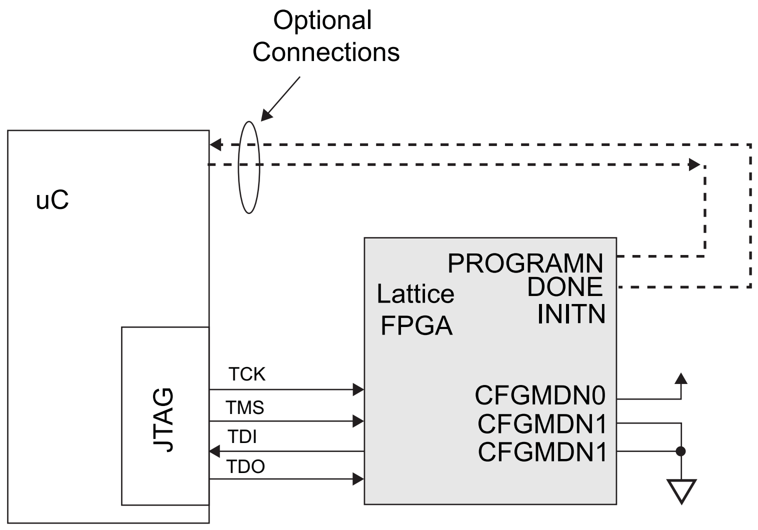


JTAG Mode

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. JTAG



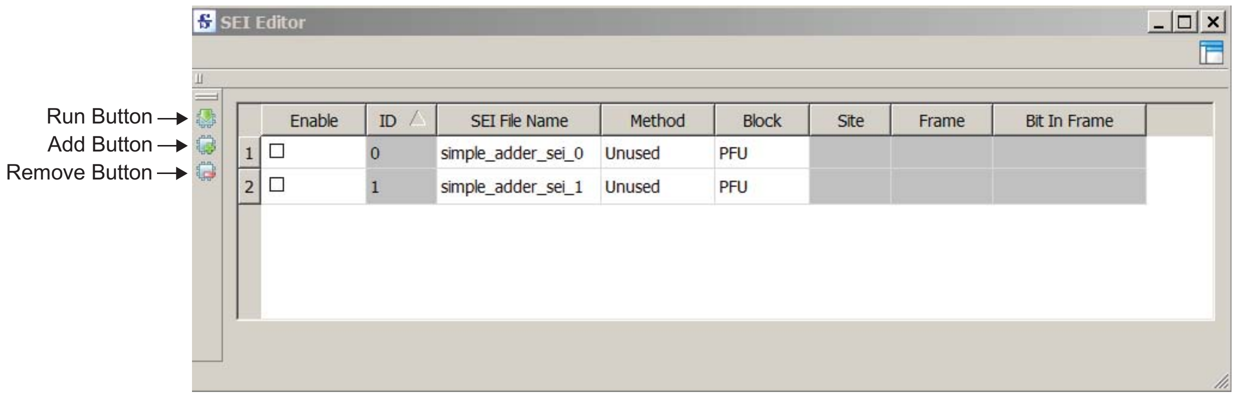
ECP5 and ECP5-5G SEI (Soft Error Injection)

Lattice Semiconductor provides the Diamond 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. SEI (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. SEI Editor



	Enable	ID	SEI File Name	Method	Block	Site	Frame	Bit In Frame
1	☐	0	simple_adder_sei_0	Unused	PFU			
2	☐	1	simple_adder_sei_1	Unused	PFU			

Method
— Unused – Error bit introduced is not used by customer design.
— Random – Error bit introduced is random and can be used by customer design.

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.

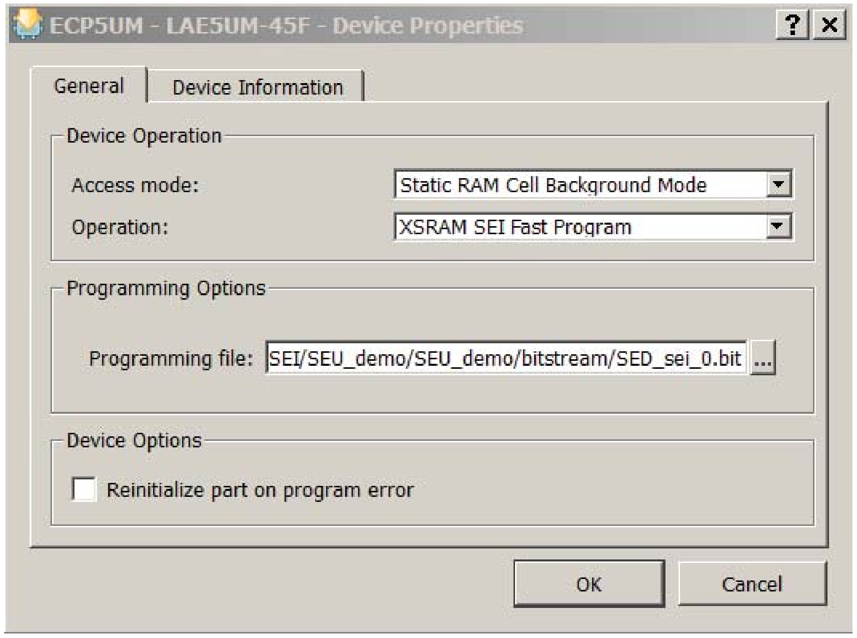
Run Button – Click this button to generate SEI files.

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 will 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 will have to stop. You can de-assert SEDENABLE to stop the SED checking.

Figure 10. Device Properties



Access mode: Static RAM Cell Background Mode
Operation: XSRAM SEI Fast Program
Programming file: SEI/SEU_demo/SEU_demo/bitstream/SED_sei_0.bit
☐ Reinitialize part on program error

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

Technical Support Assistance

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

Revision History

Date	Version	Change Summary
November 2015	1.7	Added support for ECP5-5G.
		Changed document title to LatticeECP3, ECP5 and ECP5-5G Soft Error Detection (SED)/Correction (SEC) Usage Guide.
		Updated Technical Support Assistance section.
March 2015	1.6	Changed document title to LatticeECP3 and ECP5 Soft Error Detection (SED)/Correction (SEC) Usage Guide.
		Updated the SED Signal Descriptions section. — Added information on SEDENABLE and SEDSTART signals.
		Updated the SED Flow section. — Updated Figure 4, Timing Diagram. Corrected SEDSTART signal.
		Added the ECP5 SEC (Soft Error Correction) section.
		Added the ECP5 SEI (Soft Error Injection) section.
March 2014	01.5	Updated to support ECP5 device family.
		Changed document title to LatticeECP3 and ECP5 Soft Error Detection (SED) Usage Guide.
		Changed instances of LatticeECP4UM to ECP5.
July 2013	01.4	Added support for LatticeECP4UM.
		Changed document title to LatticeECP3 and LatticeECP4UM Soft Error Detection (SED) Usage Guide.
		Updated Figure 16-1, System Block Diagram.
		Added support for Lattice Diamond in place of ispLEVER.
		Updated Figure 16-3, Divider Diagram.
		Updated SEDCLK section. • Converted Figure 16-4. Possible MCLK Values (MHz) to Table 16-2, Possible LatticeECP3 MCLK Values (MHz). • Added Table 16-3, Possible LatticeECP4UM SEDCLK Frequency.
		Updated MCCLK_freq Attribute section. Added Diamond Global Preference in place of ispLEVER Design Planner.
		Updated SED Run Time section. Added LatticeECP4UM information to context and Table 16-4, SED Run Time.
		Updated Sample Code section. Added VHDL and Verilog examples fo LatticeECP4UM.
		Updated Technical Support Assistance information.
		Updated Possible MCLK Values (MHz) table.
		Updated Technical Support Assistance information.
February 2012	01.3	Updated document with new corporate logo.
January 2011	01.2	Removed references to AUTODONE in Sample Code section.
November 2009	01.1	Updated SED flow.
		Updated Example Schematic diagram.
		Updated VHDL and Verilog sample code.
February 2009	01.0	Initial release.