LatticeXP2 Soft Error Detection (SED) Usage Guide

Technical Note

FPGA-TN-02253-2.2

March 2021

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 and with all faults, and all risk associated with such information is entirely with 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
EBR	Embedded Block RAM
PFU	Programmable Functional Unit
SEC	Soft Error Correction
SED	Soft Error Detect
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 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. The LatticeXP2 devices have a hardware implemented soft error detector which does not affect performance or heat dissipation of the devices.

This document describes the hardware based soft error detect (SED) approach taken by Lattice Semiconductor for LatticeXP2™ FPGAs.

2. SED Overview

The SED hardware in the LatticeXP2 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 data that is read, and the CRC that is calculated, does not include EBR memory or PFUs used as RAM. 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.

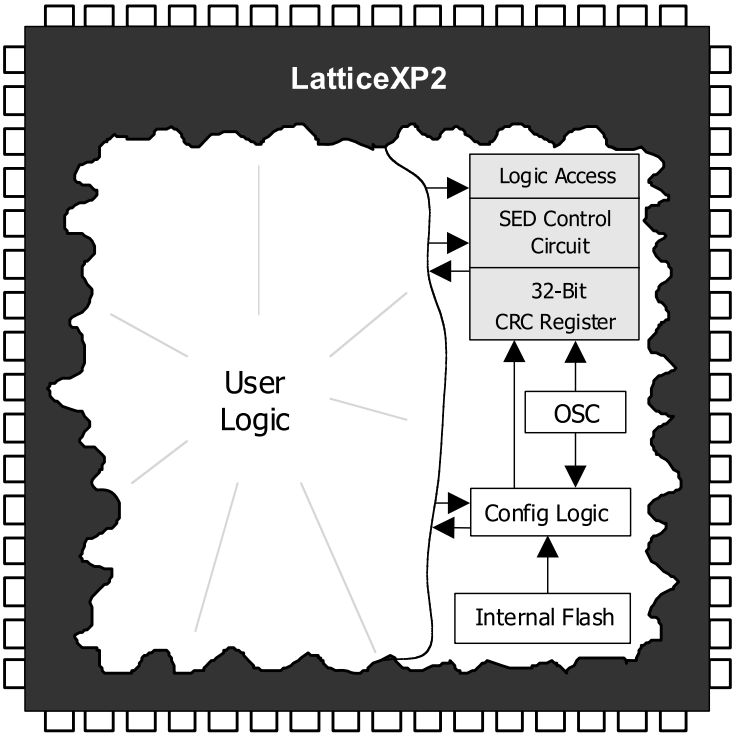

Figure 2.1. System Block Diagram

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 ispLEVER® bitstream generation software analyzes the configuration of a placed and routed design and updates the 32-bit SED CRC register contents during bitstream generation.

The following sections describe the LatticeXP2 SED implementation and flow, along with some sample code to get started with.

3. SED Limitations

SED should only be run when the logic of the device is held in a steady state condition to prevent false error Indications.

All background programming instructions are not available while the SED is in progress due to resource contention. If a normal (not background) Flash programming command or SRAM configuration command is run the SRAM CRC Error check will be terminated. Refer to PCN 02B-12 for further details.

4. Basic SED and One-shot SED Modes

4.1. Basic SED

Basic SED checks the CRC for all bits. For Basic SED (SEDBA), the inputs are SEDCLKIN, SEDENABLE, SEDSTART, and SEDFRCERRN. The output signals are SEDCLKOUT, SEDDONE, SEDINPROG, and SEDERR.

Once an error is detected the SEDERR signal will stay high. SED supports the following Soft Error Corrections (SEC): “Do Nothing” or on-demand user reconfiguration by pulling the PROGAMN pin low from another device.

4.2. One-Shot SED

The One-Shot SED setting is based on the One-Shot Fuse. The module (SEDBB) has no input ports. The output signals are SEDDONE, SEDINPROG, and SEDERR. At a minimum, the user must connect SEDERR to an I/O pin in order to detect an error.

4.3. Hardware Description

As shown in Figure 4.1, the LatticeXP2 SED hardware has several inputs and outputs that allow the user to control, and monitor, SED behavior.

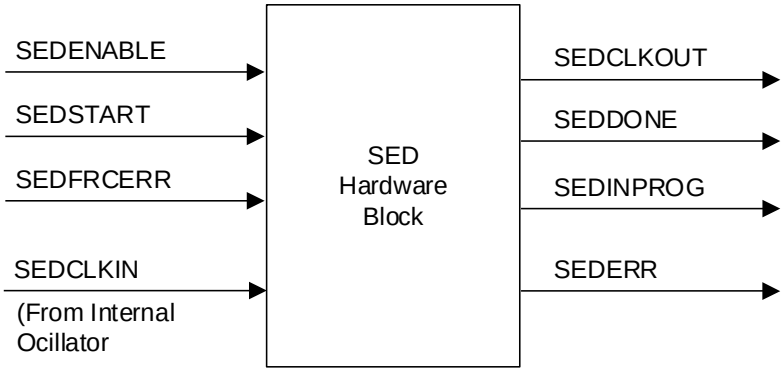

Figure 4.1. Signal Block Diagram

5. SED Signal Descriptions

Table 5.1. SED Signal Descriptions

Signal Name	Direction	Active	Description
SEDCLKIN	Input	N/A	Clock
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

5.1. SEDCLKIN

Clock input to the SED hardware.

When external SPI configuration is used, this clock is derived from the LatticeXP2's on-chip oscillator. The on-chip oscillator's output goes through a divider to create MCCLK. MCCLK goes through another divider to create SEDCLKIN.

The software default for MCCLK is 2.5 MHz, but this can be modified using the MCCLK_FREQ global preference in ispLEVER's pre-map Design Planner (see LatticeXP2 sysCONFIG Usage Guide (TN1141) for supported values of MCCLK). It has a range of 2.5 MHz to 66 MHz.

The divider for SEDCLKIN can be set to 1, 2, 4, 8, 16 or 32. The default is 1, so the default SEDCLKIN frequency is 2.5 MHz. The divider value can be set using a parameter, see the example code at the end of this document.

If internal Flash configuration mode is used, SEDCLKIN can only be set to 3.1 MHz with a divider setting of 1.

Note that SEDCLKIN is an internally generated signal, so it should not be included as an input in the user design.

See the examples at the end of this document. Also note that while inputs to the SED block are clocked using SEDCLKIN, no attempt has been made to synchronize between clock domains. If this is a concern for a particular design then the designer will need to provide synchronization.

5.2. OSC_DIV

Options: 1, 2, 4, 8, 16 or 32 for external configuration. Only 1 can be selected for internal configuration. The CLK that drives the SED module will be set by MCCLK/OSC_DIV.

5.3. DEV_DENSITY

Options: "17K" (default), "5K", "8K", "30K", "40K" for device density selection.

5.4. SEDENABLE

Level-sensitive signal which starts SED checking.

Table 5.2. SEDENABLE

State	Description
	Enables output of SEDCLKOUT, arms SED hardware.

5.5. SEDCLKOUT

Gated version of SEDCLK, SEDCLKOUT is gated by SEDENABLE.

5.6. SEDSTART

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 5.3. SEDSTART

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

5.7. SEDFRCERR

Active high input to the SED hardware, sampled on the rising edge of SEDCLKIN.

Table 5.4. SEDFRCERR

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

5.8. SEDINPROG

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT

Table 5.5. SEDINPROG

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

5.9. SEDDONE

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 5.6. SEDDONE

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

5.10. SEDERR

Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 5.7. SEDERR

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

6. SED Flow

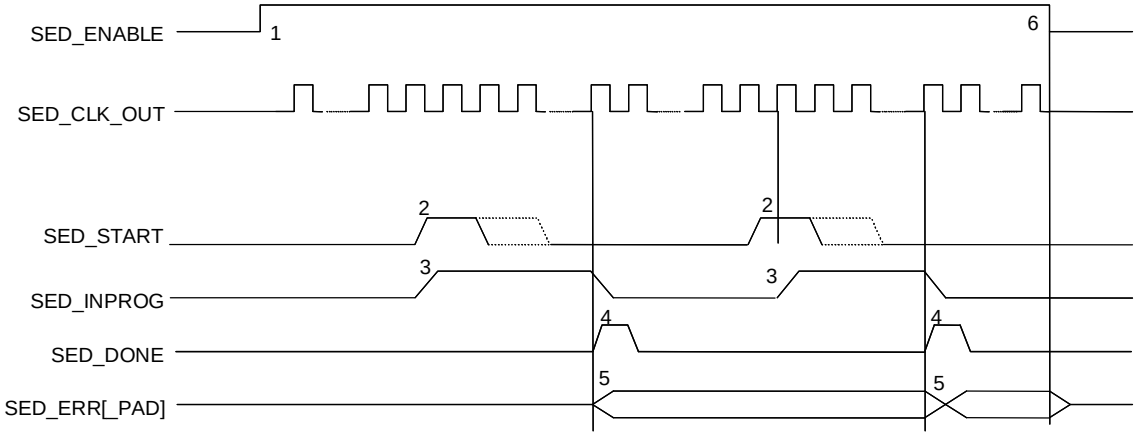

Figure 6.1. Timing Diagram

The general SED flow is as follows.

1. User logic sets SEDENABLE high. This signal may be tied high if desired.
2. User logic sets SEDSTART high. SEDINPROG goes high. If SEDDONE is already high it is driven low. SEDSTART may be tied high to enable continuous SED checking.
3. SED starts reading back data from the configuration SRAM.
4. SED finishes checking. SEDERR is updated, SEDINPROG goes low, and SEDDONE goes high.
5. If SEDERR is driven high there are only two ways to reset it, drive SEDENABLE low or reconfigure the FPGA.
6. SEDENABLE goes low when/if the user specifies, and SED is no longer in use.

The user has two choices when an error is detected, ignore the error, and possibly log it, or reconfigure the FPGA. Reconfiguration can be accomplished by driving the PROGRAMN pin low. This can be done by externally connecting a GPIO pin to PROGRAMN.

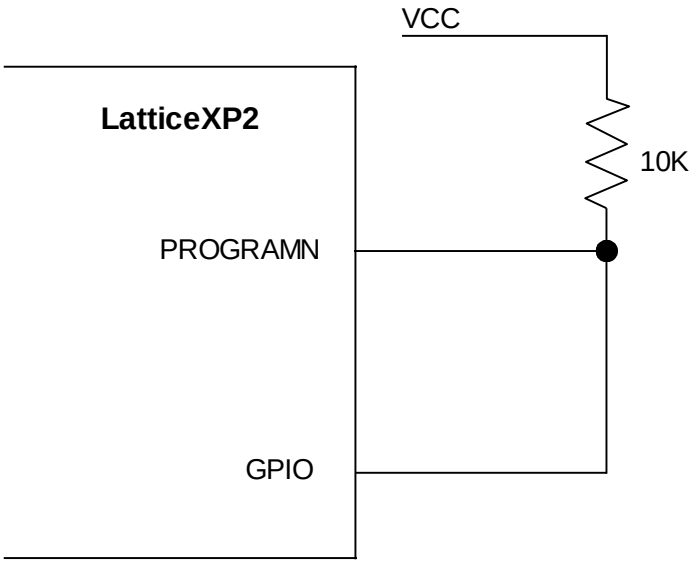

Figure 6.2. Example Schematic

7. SED Run Time

The amount of time needed to perform an SED check depends on the density of the device and the frequency of SEDCLKIN. 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.

Maxbits / SEDCLKIN = Time

Maxbits is in mega-bits and depends on the density of the FPGA (see Table 16-8). SEDCLKIN is frequency in MHz. Time is in seconds

For example, for a design using a LatticeXP2 with 5K look-up tables and the SEDCLKIN is the software default of 3.1 MHz:

1.236 Mbits / 3.1 MHz = 398.71 ms

In this example, SED checking will take approximately 398.71 ms. Remember that this happens in the background and does not affect user logic performance.

Note that the internal oscillator used to generate SEDCLKIN can vary by ±30%.

Table 7.1. SED Run Time

Device	XP2-5K	XP2-8K	XP2-17K	XP2-30K	XP2-40K
Density	1.236 M	1.954 M	3.636 M	5.964 M	8.304 M
66 MHz	18.7 ms	29.6 ms	55.1 ms	90.4 ms	126.2 ms
50 MHz	24.7 ms	39.1 ms	72.7 ms	119.3 ms	166.1 ms
3.1 MHz	399 ms	624 ms	1.173 s	1.924 s	2.679 s
2.5 MHz	495 ms	782 ms	1.455 s	2.395 s	3.325 s

8. Sample Code

The following simple example code shows how to instantiate the SED. In the example the SED is always on and always running, and the outputs of the SED hardware have been routed to FPGA output pins. Note that the SEDBA primitive is part of ispLEVER 6.1 or later.

8.1. Basic SED VHDL Example

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

entity example is
   port (
        sed_done: out std_logic;
        sed_in_prog: out std_logic;
        sed_clk_out: out std_logic;
        sed_out: out std_logic);
end;

architecture behavioral of example is

    component SEDBA -- SED component
        GENERIC(OSC_DIV      : integer := 1;
           DEV_DENSITY: string  := "17K");
        port(
           SEDENABLE: in std_logic;
           SEDSTART: in std_logic;
           SEDFRCERRN: in std_logic;
           SEDERR: out std_logic;
           SEDDONE: out std_logic;
           SEDINPROG: out std_logic;
           SEDCLKOUT: out std_logic) ;

    end component;

begin

    inst1: SEDBA
    generic map (OSC_DIV=> "1",
       DEV_DENSITY=> "17k")
    port map (
          SEDENABLE=> '1', -- tied high
          SEDSTART=> '1', -- tied high
          SEDFRCERRN=> '1', -- tied high
          SEDERR=> sed_out, -- wired to an output
          SEDDONE=> sed_done, -- wired to an output
          SEDINPROG=> sed_in_prog, -- wired to an output
          SEDCLKOUT=> sed_clk_out ) ; -- wired to an output);

end behavioral;
```

8.2. One Shot SED in VHDL

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

entity example is
    port (
        sed_done : out std_logic;
        sed_in_prog : out std_logic;
        sed_out : out std_logic);
end;

architecture behavioral of example is
   component SEDBB -- This is for One Shot SED
     GENERIC(OSC_DIV        : integer := 1;
             DEV_DENSITY    : string  := "17K");
     port (
        SEDDONE : out std_logic;
        SEDINPROG : out std_logic;
        SEDERR : out std_logic
        );
end component;

begin

     isnt1: SEDBB
     generic map (OSC_DIV=> “1”,
                  DEV_DENSITY=> "17k")
     port map (
        SEDERR => sed_out, -- wired to an output
        SEDDONE => sed_done, -- wired to an output
        SEDINPROG => sed_in_prog); -- wired to an output
end behavioral ;
```

8.3. Basic SED Verilog Example

```
module example (
    sed_done,
    sed_in_prog,
    sed_clk_out,
    sed_out) ;

output sed_done;
output sed_in_prog;
output sed_clk_out;
output sed_out;

assign V_hi = 1'b1;
assign V_lo = 1'b0;

SEDBA
     #(.OSC_DIV (1),
        .DEV_DENSITY ("17K"))

SED_IP(
      .SEDENABLE(V_hi), // always high
      .SEDSTART(V_hi),    // always high
      .SEDFRCERRN(V_hi), // always high
      .SEDERR(sed_out),   // wired to an output
      .SEDDONE(sed_done), // wired to an output
      .SEDINPROG(sed_in_prog), // wired to an output
      .SEDCLKOUT(sed_clk_out)); // wired to an output

endmodule
```

8.4. One-Shot SED in Verilog

```
module example (
     sed_done,
     sed_in_prog,
     sed_clk_out,
     sed_out) ;

output sed_done;
output sed_in_Prog;
output sed_clk_out;
output sed_out;

assign V_hi = 1'b1;
assign V_lo = 1'b0;

     SEDBB
          #(.OSC_DIV (1),
             .DEV_DENSITY ("17K"))

       SED_IP(
             .SEDDONE(sed_done),
             .SEDINPROG(sed_inprog),
             .SEDERR(sed_out)
             );
endmodule

module SEDBB (SEDERR, SEDDONE, SEDINPROG);
    output SEDERR, SEDDONE, SEDINPROG ;
endmodule
```

Technical Support Assistance

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

Revision History

Revision 2.2, March 2021

Section	Change Summary
All	• Changed document number from TN1130 to FPGA-TN-02253. • Updated document template.
Disclaimers	Added this section.
Acronyms in This Document	Added this section.
SED Signal Descriptions	Added DEV_DENSITY section.
Sample Code	Updated codes in this section.

Revision 2.1, October 2012

Section	Change Summary
SED Limitations	Added this section.

Revision 2.0, September 2009

Section	Change Summary
Sample Code	Updated Basic SED VHDL Example code and One-Shot SED in VHDL code.”.

Revision 1.9, January 2009

Section	Change Summary
SED Flow	Updated Basic SED Verilog Example code.
Sample Code	Updated One-Shot SED in Verilog code.

Revision 1.8, January 2009

Section	Change Summary
SED Flow	Updated text section. Added Example Schematic diagram.

Revision 1.7, August 2008

Section	Change Summary
SED Signal Descriptions	Updated SEDCLKIN and OSC_DIV text sections.
SED Run Time	Updated text section and table.

Revision 1.6, July 2008

Section	Change Summary
SED Flow	Added footnote to timing diagram.

Revision 1.5, April 2008

Section	Change Summary
SED Signal Descriptions	Corrected “SEDFRCERR” to read “SEDFRCERRN”.

Revision 1.4, March 2008

Section	Change Summary
SED Signal Descriptions	Updated SEDCLKIN and OSC_DIV text sections and SEDENABLE table.

Revision 1.3, February 2008

Section	Change Summary
SED Flow	Updated Timing Diagram

Revision 1.2, January 2008

Section	Change Summary
SED Signal Descriptions	Updated OSC_DIV text section.
SED Flow	Updated text section.

Revision 1.1, January 2008

Section	Change Summary
Sample Code	Updated code in the following sections: Basic SED VHDL Example, One Shot SED in VHDL, Basic SED Verilog Example, One Shot SED in Verilog.

Revision 1.0, February 2007

Section	Change Summary
All	Initial release.

www.latticesemi.com