LatticeECP2/M Soft Error Detection (SED) Usage Guide

June 2013 **Technical Note TN1113**

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.

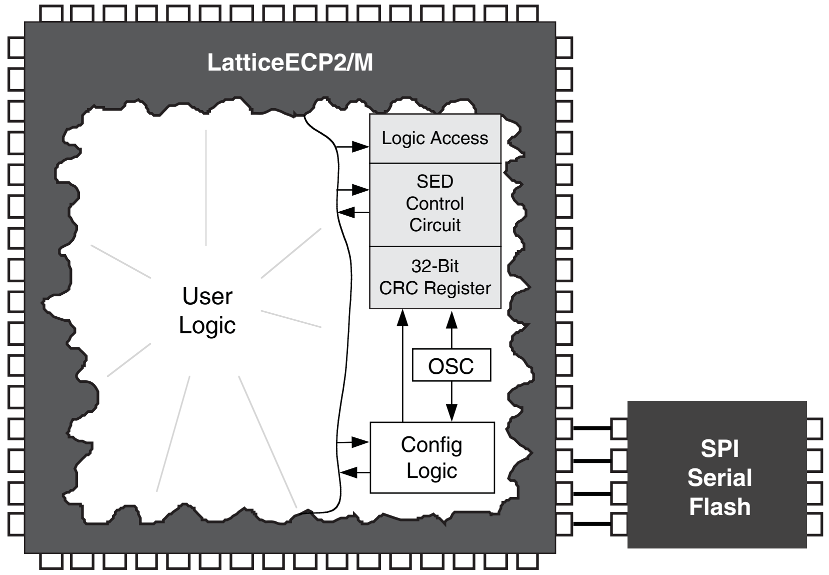
SRAM-based FPGAs store logic configuration data in SRAM cells. As the number and density of SRAM cells in an FPGA increase, the probability that a soft error will alter the programmed logical behavior of the system increases. A number of approaches have been taken to address this issue, but most involve 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 LatticeECP2™ and LatticeECP2M™ FPGAs.

SED Overview

The SED hardware in the LatticeECP2/M 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 17-1). 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 17-1. 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.

The following sections describe the LatticeECP2/M SED implementation and flow, along with some sample code to get started with.

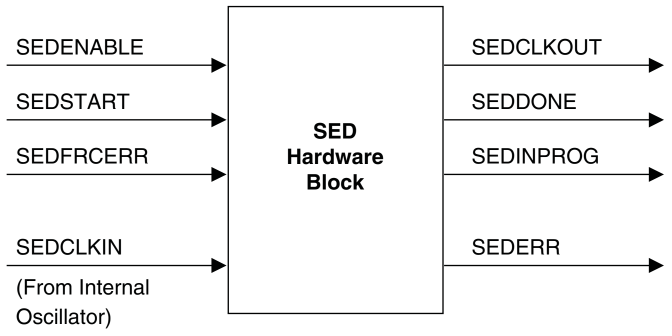
SED Limitations

SED should only be run when the logic of the device is held in a steady state condition to prevent false error indications. If a normal SRAM configuration command is run the SRAM CRC Error check will be terminated. Refer to PCN 02B-12 for further details.

Hardware Description

As shown in Figure 17-2, the LatticeECP2/M SED hardware has several inputs and outputs that allow the user to control, and monitor, SED behavior.

Figure 17-2. Signal Block Diagram



Signal Descriptions

Table 17-1. 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

SEDCLKIN

Clock input to the SED hardware.

This clock is derived from the LatticeECP2/M on-chip oscillator. The on-chip oscillator 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 the Global Preferences tab of the Diamond Spreadsheet View (see TN1108, LatticeECP2/M sysCONFIG Usage Guide, for possible values of MCCLK).

The divider for SEDCLKIN can be set to 1, 2, 4, 8, 16 or 32. The software 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. Care must be taken to ensure that the SEDCLKIN setting is at least 20 MHz. Refer to Appendix A for details on MCCLK and SEDCLKIN frequencies.

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.

SEDENABLE

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

Table 17-2. SEDENABLE

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

SEDCLKOUT

Gated version of SEDCLKIN, SEDCLKOUT is gated by SEDENABLE.

SEDSTART

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

Table 17-3. SEDSTART

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, sampled on the rising edge of SEDCLKIN.

Table 17-4. SEDFRCERR

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

SEDINPROG

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

Table 17-5. SEDINPROG

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, clocked out on the rising edge of SEDCLKOUT.

Table 17-6. 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.

SEDERR

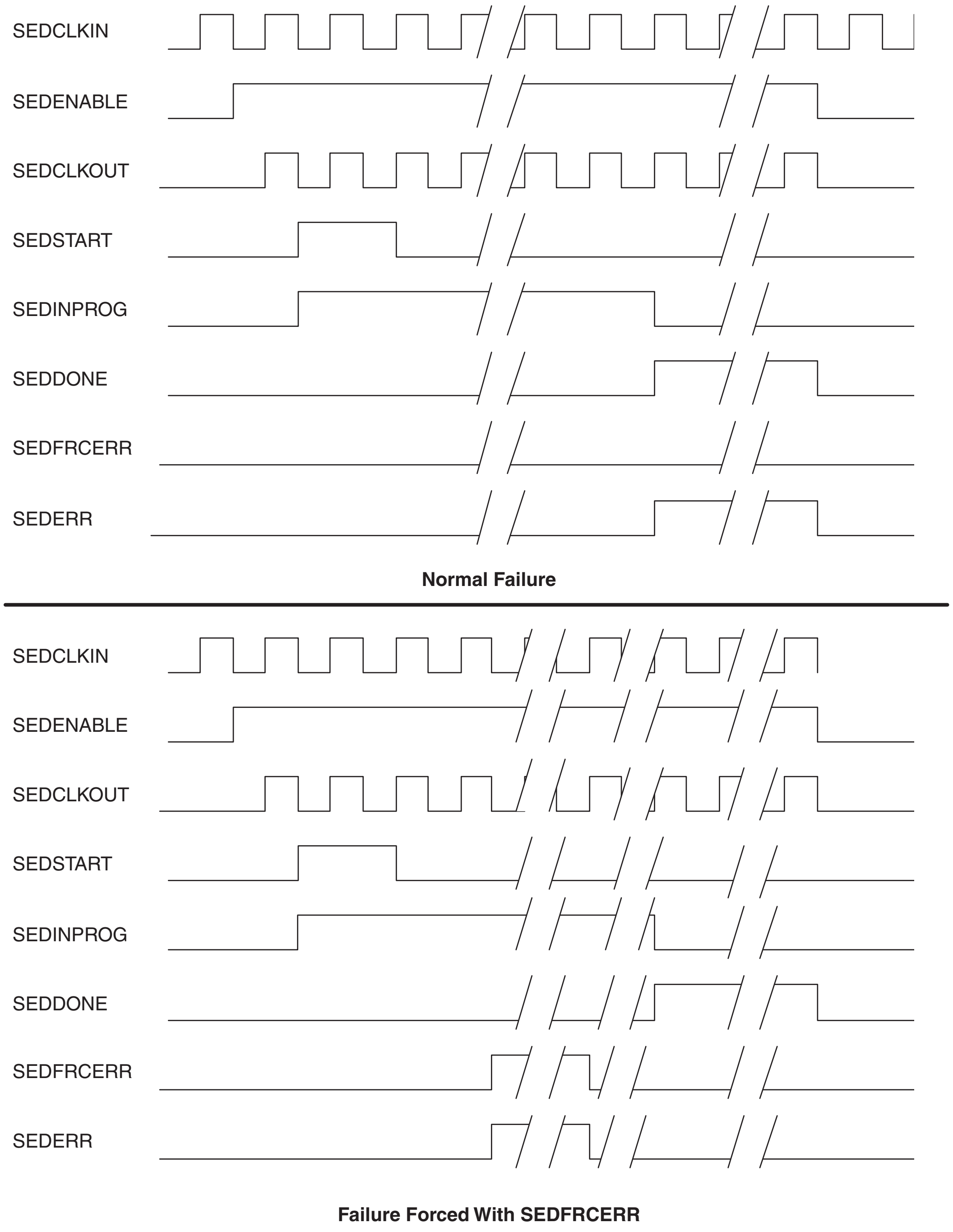
Active high output from the SED hardware, clocked out on the rising edge of SEDCLKOUT.

Table 17-7. SEDERR

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

SED Flow

Figure 17-3. Timing Diagram



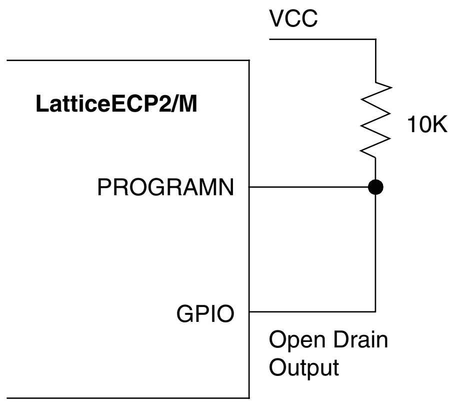
The general SED flow is as follows.

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.

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 with external logic or by wiring one of the FPGA's general purpose I/Os to the PROGRAMN pin and toggling the pin with user logic, perhaps something as simple as inverting SEDERR. If a general purpose I/O is tied to PROGRAMN it is recommended that the I/O Type be set to open drain and an external pull-up resistor be connected to the pin.

Figure 17-4. 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 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 following formula:

Maxbits / SEDCLKIN = Time

Maxbits is in mega-bits and depends on the density of the FPGA (see Table 17-8). SEDCLKIN is frequency in MHz. Time is in seconds

For example, if the design is using a LatticeECP2 with 50K look-up tables and the SEDCLKIN is set in the software to be 20 MHz:

8.9 Mbits / 20 MHz = 0.445 seconds

In this example, SED checking will take approximately 0.445 seconds. 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 17-8. SED Run Time

Density	Bitstream Size (Mb)	Run Time[1] (ms)
ECP2-6	1.5	75
ECP2-12	2.9	145
ECP2-20	4.5	225
ECP2-35	6.3	315
ECP2-50	8.9	445
ECP2-70	13.3	665
ECP2M-20	5.9	295
ECP2M-35	9.8	490
ECP2M-50	15.8	790
ECP2M-70	19.8	990
ECP2M-100	25.6	1280

1. Based on SEDCLKIN = 20 MHz.

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 SEDAA primitive is part of ispLEVER 6.0 or later.

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  SEDAA -- SED component
      generic (OSC_DIV : integer := 1); -- set SEDCLKIN divider
      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
```

```
      isnt1: SEDAA
      generic map (OSC_DIV=> "1")
      port map (
        SEDENABLE    => '1`,    -- tied high
        SEDSTART     => '1`,    -- tied high
        SEDFRCERR    => '0`,   -- tied low
        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 ;
```

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;

SEDAA
     #(.OSC_DIV(1))

SED_IP(
      .SEDENABLE(V_hi), // always high
      .SEDSTART(V_hi),    // always high
      .SEDFRCERR(V_lo), // always low
      .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
```

Technical Support Assistance

e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2006	01.0	Initial release.
April 2006	01.1	Fixed VHDL code
September 2006	01.2	Changed FPGA family naming to show support for LatticeECP2M.
April 2007	01.3	Updated SED Flow timing diagram.
September 2007	01.4	Updated SEDCLKIN frequency setting.
February 2008	01.5	Updated Timing Diagram.
July 2008	01.6	Added note to SED Flow timing diagram.
January 2009	01.7	Updated Verilog Example code.
September 2009	01.8	Updated VHDL Example in the Sample Code section.
April 2010	01.9	Changed minimum operating frequency of SEDCLKIN to be at least 20 MHz.
		Updated SED Run Time table.
		Corrected formula for data in SED Run Time table to use SEDCLKIN = 20 MHz.
January 2012	02.0	Added Appendix A.
October 2012	02.1	Added SED Limitations section.
		Updated document with new corporate logo.
June 2013	02.2	Updated Technical Support Assistance information.

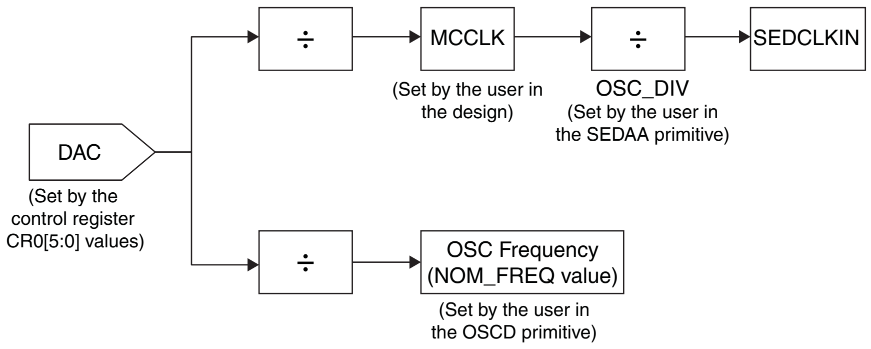
Appendix A. Calculating Exact MCCLK and SEDCLKIN Values

This appendix deals with the interdependency of the SEDCLKIN, MCCLK and the internal oscillator frequency. Figure 17-5 shows the hardware block diagram, including the internal oscillator, the MCCLK divider and the SEDCLKIN divider. The base frequency is generated by the DAC, which generates one out of four base frequencies. The internal oscillator frequency and the MCCLK and SEDCLKIN frequencies are generated by applying proper divider values to the DAC base frequency. The DAC is capable of generating only one of the four values: 260 MHz, 270 MHz, 300 MHz or 330 MHz. The selection of the base DAC frequency depends upon the internal oscillator frequency and this, in turn, affects the MCCLK and SEDCLKIN frequencies. The DAC base frequency is set by the control register CR0[5:0] values. The CR0 values can be observed in the Lattice ispVM™ System software. To see CR0 values in the ispVM tool, select **ispTools > ispVM Editors > "Control Register0 Editor.."**. The CR0[5:0] values corresponding to the DAC base frequency include:

- 011110 for 260 MHz
- 010001 for 270 MHZ
- 001001 for 300 MHz
- 000001 for 330 MHz

See the ispVM Help system for more details on the ControlRegister0 Editor.

Figure 17-5. Internal Clocking Scheme


÷
MCCLK
÷
SEDCLKIN
(Set by the user in the design)
OSC_DIV
(Set by the user in the SEDAA primitive)
DAC
(Set by the control register CR0[5:0] values)
÷
OSC Frequency (NOM_FREQ value)
(Set by the user in the OSCD primitive)

The CR0 values are selected based on the selection of valid NOM_FREQ values of the OSCD primitive for the internal oscillator. See the LatticeECP2/M Family Data Sheet for more details on the OSCD primitive.

Tables 17-9 to 17-12 show valid NOM_FREQ values and corresponding CR0[5:0] values, resulting in one of the four DAC base frequencies.

Table 17-9. NOM_FREQ Values and CR0[5:0] Values that Result in a 260 MHz DAC Base Frequency

NOM_FREQ in OSCD Primitive (MHz)	Resulting CR0[5:0] Setting	Resulting Frequency (MHz)
2.5	011110	260
4.3		
5.4		
6.9		
8.1		
9.2		
10		
13		
26		

Table 17-10. NOM_FREQ Values and CR0[5:0] Values that Result in a 270 MHz DAC Base Frequency

NOM_FREQ in OSCD Primitive (MHz)	Resulting CR0[5:0] Setting	Resulting Frequency (MHz)
15	010001	270
34		
45		

Table 17-11. NOM_FREQ Values and CR0[5:0] Values that Result in a 300 MHz DAC Base Frequency

NOM_FREQ in OSCD Primitive (MHz)	Resulting CR0[5:0] Setting	Resulting Frequency (MHz)
30	001001	300

Table 17-12. NOM_FREQ Values and CR0[5:0] Values that Result in a 330 MHz DAC Base Frequency

NOM_FREQ in OSCD Primitive (MHz)	Resulting CR0[5:0] Setting	Resulting Frequency (MHz)
20	000001	330
41		
55		

The default value of NOM_FREQ is 2.5 MHz, or when the OSCD primitive is not explicitly instantiated. In that case, the CR0[5:0] value is 011110, resulting in 260 MHz of DAC base frequency. If the OSCD primitive is instantiated in the design, and the NOM_FREQ value is 30 MHz, then as per Table 17-10, the tool will set the CR[5:0] value as 001001, which results in a DAC base frequency of 300 MHz.

Calculating MCCLK Frequency

The MCCLK_FREQ is 2.5 MHz by default; however, this can be set in the attributes of the SED HDL component (see sample code section or Diamond help files) and Global Constraints tab of the Diamond Spreadsheet View. Whenever the user selects a particular MCCLK frequency, the most appropriate divider value is selected, based on the DAC base frequency. As mentioned in the previous section, the DAC base frequency depends upon the OSCD NOM_FREQ value selected. Table 17-13 shows the divider values to generate the nearest MCCLK frequency, based on the DAC frequency.

Table 17-13. Divider Values to Generate Nearest MCCLK Frequency, Based on DAC Frequency

	CR0[5:0] Setting	**011110**	**010001**	**001001**	**000001**
	DAC Base Frequency (MHz)	**260**	**270**	**300**	**330**
MCCLK_FREQ (MHz)	2.5	104	108	120	128
	4.3	60	62	70	76
	5.4	48	50	56	62
	6.9	38	40	44	48
	8.1	32	34	38	40
	9.2	28	30	32	36
	10	26	28	30	34
	13	20	20	24	26
	15	18	18	20	22
	20	14	14	16	16
	26	10	10	12	12
	30	8	10	10	12
	34	8	8	8	10
	41	6	6	6	8
	45	6	6	6	8
	55	6	6	6	8
	60	4	4	6	6
	130	2	2	2	2

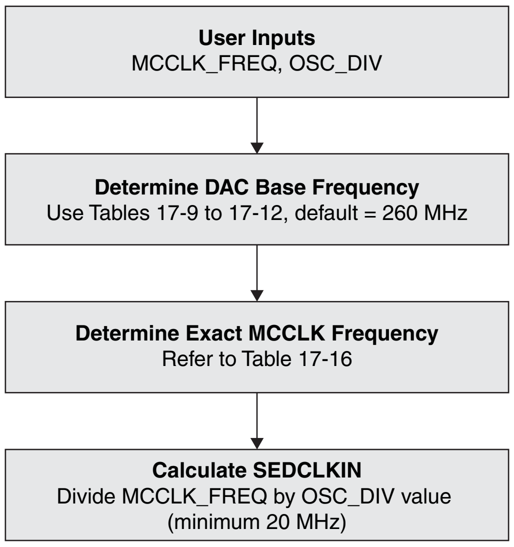
Table 17-13 shows all the valid divider values which exist in the LatticeECP2/M device. Because of these integer divider values, the exact MCCLK frequencies may not be observed in several cases. The actual MCCLK frequency is obtained by dividing the DAC base frequency by the divider value under that column, while the row corresponds to the desired MCCLK frequency.

For example, if the desired MCCLK value is 9.2 MHz, and the DAC base frequency is 300 MHz (CR0[5:0] = 001001), then the divider value would be 32. Thus, the MCCLK value would be 300/32 = 9. Similarly, with a DAC base frequency of 260 MHz, if MCCLK_ FREQ is 41, 45 or 55, the same divider value is selected, resulting in the same output frequency.

Calculating the SEDCLKIN

Once the MCCLK frequency has been calculated, the SEDCLKIN is calculated by dividing the MCCLK by a valid set of divider values, as defined by OSC_DIV parameter of the SEDAA primitive. In effect, SEDCLKIN = MCCLK/OSC_DIV. Due to the discrete nature of divider values and four different DAC Base frequencies, the exact SEDCLKIN cannot be expected. The variation can go up to ±30% as mentioned in the DC and Switching Characteristics section of the LatticeECP2/M Family Data Sheet. Figure 17-6 shows the steps required to calculate the SEDCLKIN frequency.

Figure 17-6. Calculating SEDCLKIN Frequency


User Inputs
MCCLK_FREQ, OSC_DIV
Determine DAC Base Frequency
Use Tables 17-9 to 17-12, default = 260 MHz
Determine Exact MCCLK Frequency
Refer to Table 17-16
Calculate SEDCLKIN
Divide MCCLK_FREQ by OSC_DIV value
(minimum 20 MHz)