AN 737: SEU Detection and Recovery in Arria® 10 Devices

Contents

1. Overview

This application note describes the implementation of Arria® 10 single event upset (SEU) detection and recovery features by presenting the following information.

- Error detection and correction feature architecture in Arria 10 devices.
- General implementation guidelines for error detection cyclic redundancy check (EDCRC) and error correction feature.
- General implementation guidelines for embedded memory error correction code (ECC) feature.
- Arria 10 EDCRC reference design with detailed development flow.

Related Information

- Test Methodology of Error Detection and Recovery using CRC in Intel FPGA Devices
  Provides more information about SEU detection and recovery in Arria II, Stratix® III, Stratix® IV, Arria V, Cyclone® V, and Stratix® V devices.
- Advanced SEU Detection Intel® FPGA IP User Guide
  Provides more information about hierarchy tagging and sensitivity processing using the Advanced SEU Detection Intel® FPGA IP.
- Fault Injection Intel® FPGA IP Core User Guide
  Provides more information about injecting soft error to simulate SEU using the Fault Injection Intel® FPGA IP.
- SEU Mitigation for Arria 10 Devices
  Provides more information about Arria 10 SEU features.
- Error Message Register Unloader Intel® FPGA IP Core User Guide
  Provides more information about retrieving and storing the error message register using the Error Message Register Unloader Intel® FPGA IP.
- Arria 10 EDCRC Reference Design Files
  Reference design files that you need to apply steps and compilation described in *Creating Arria 10 SEU Fault Injection and Hierarchy Tagging Design with Qsys*.
- Complete Arria 10 EDCRC Reference Design Files
  Precompiled reference design files ready for design testing in *Design Testing with Fault Injection Debugger*.
- Arria 10 GX FPGA Development Kit

2. Quartus® Prime Software SEU FIT Reports

The Quartus® Prime software generates reports that contain the parameters involved in SEU FIT calculations and the result of these calculations for each component. These reports are available only for licensed users.

2.1. Enabling the Projected SEU FIT by Component Usage Report

The Quartus Prime Fitter generates the Projected SEU FIT by Component Usage report. The Quartus Prime software only generates reports for designs that successfully pass place and route.

To enable the report:

1. Obtain and install the SEU license.
2. Add the following assignments to your project's `.qsf` file:

```
set_global_assignment –name ENABLE_ADV_SEU_DETECTION ON
set_global_assignment –name SEU_FIT_REPORT ON
```

2.2. SEU FIT Parameters Report

The SEU FIT Parameters report shows the environmental assumptions that influence the FIT/Mb values.

Figure 1. SEU FIT Parameters

- **Altitude** represents the default altitude (above sea-level).
- **Neutron Flux Multiplier** is the relative flux for the default location, which is New York City per JESD specification. The default is 1. Change the setting by adding the following assignment to your `.qsf` file:

```
set_global_assignment -name RELATIVE_NEUTRON_FLUX <relative_flux>
```

  Note: You can compute scaled values using the JESD published equations for altitude, latitude, and longitude. Websites, such as www.seutest.com, can make this computation for you.

- **Alpha Flux** is the default for standard Intel packages; you cannot override the default.

  Note: When you change the relative **Neutron Flux Multiplier**, the Quartus Prime software only scales the neutron component of FIT. Location does not affect the Alpha flux.

Related Information

Soft-error Testing Resources
Provides soft-error testing resources and calculator.

2.3. Projected SEU FIT by Component Usage Report

The Projected SEU FIT by Component Usage report shows the different components (or cell types) that comprise the total FIT rate, and SEU FIT calculation results.

An Intel FPGA's sensitivity to soft errors varies by process technology, component type, and your design choices when implementing the component (such as tradeoffs between area/delay and SEU rates). The report shows all bits (the raw FIT), utilized bits (only resources the design actually uses), and the ECC-mitigated bits.

Figure 2. Projected SEU FIT by Component Usage Report

Projected SEU FIT by Component Usage

<<Filter>>

	Component	Raw	Utilized	w/ECC	AVF 0.5	AVF 0.25
1	Configuration (CRAM)	8800	69	69	35	18
1	--Logic	2371	15	15	8	4
2	--Routing	6415	54	54	27	14
3	--I/O config	14	<1	<1	<1	<1
2	RAM	12133	2573	2306	1153	576
1	--HARD-IP (E.G. PCIe)	899	267	0	0	0
2	--Embedded RAM	11234	2306	2306	1153	576
3	--MLAB (LUTRAM)	0	0	0	0	0
3	Registers	3213	65	65	34	17
1	--Hard-IP FF	1185	39	39	20	10
2	--DSP/M20K FF	840	23	23	12	6
3	--Design FF	1188	3	3	2	<1
4						
5	TOTAL	24146	2707	2440	1222	611

2.3.1. Component FIT Rates

The Projected SEU FIT by Component report shows FIT for the following components:

- SRAM embedded memory in embedded processors hard IP and M20K or M10K blocks
- CRAM used for LUT masks and routing configuration bits
- LABs in MLAB mode
- I/O configuration registers, which the FPGA implements differently than CRAM and design flipflops
- Standard flipflops the design uses in the address and data registers of M20K blocks, in DSP blocks, and in hard IP
- User flipflops the design implements in logic cells (ALMs or LEs)

2.3.2. Raw FIT

The Quartus Prime Projected SEU FIT by Component Usage report provides raw FIT data. Raw FIT is the FIT rate of the FPGA if the design uses every component. Raw FIT data is not design specific.

Note: The Altera Reliability Report, available on the Altera FPGA web site, also provides reliability data and testing procedures for Altera FPGA devices.

The Quartus Prime software computes the FIT for each component using (component Mb × intrinsic FIT/Mb × Neutron Flux Multiplier) for the device family and process node. (For flip flops, "Mb" represents a million flip flops.)

To give the worst-case raw FIT, the report assumes the maximum amount of CRAM that implements MLABs in the device. Thus, the CRAM raw FIT is the sum of CRAM and MLAB entries.

Note: The Quartus Prime software counts device bits for target devices using different parameter information than the Reliability Report. Therefore, the Quartus Prime RAW SEU FIT rate is different from the EDCRC max FIT Rate based on the Reliability Report data. The EDCRC max FIT rate based on the Reliability Report data represents the likelihood of error in the CRAM indicated by CRC_ERROR pin.

Related Information

FPGA Functional Analysis, Quality & Reliability Support

2.3.3. Utilized FIT

The **Utilized** column shows FIT calculations considering only resources that the design actually uses. Since SEU events in unused resources do not affect the FPGA, you can safely ignore these bits for resiliency statistics.

Additionally, the **Utilized** column discounts unused memory bits. For example, implementing a 16 × 16 memory in an M20K block uses only 256 bits of the 20 Kb.

Note: The Error Detection flag and the Projected SEU FIT by Component report do not distinguish between critical bit upsets, such as fundamental control logic, or non critical bit upsets, such as initialization logic that executes only once in the design. Apply hierarchy tags at the system level to filter out less important logic errors.

The Projected SEU FIT by Component report's **Utilized** CRAM FIT represents provable deflation of the FIT rate to account for CRAM upsets that do not matter to the design. Thus, the SEU incidence is always higher than the utilized FIT rate.

Related Information

Designating the Sensitivity of the Design Hierarchy, AN 866: Mitigating and Debugging Single Event Upsets in Quartus Prime Standard Edition

2.3.3.1. Comparing .smh Critical Bits Report to Utilized Bit Count

The number of design critical bits that the Compiler reports during `.smh` generation correlates to the utilized bits in the report, but it is not the same value. The difference occurs because the `.smh` file includes all bits in a resource, even when the resource usage is partial.

2.3.3.2. Considerations for Small Designs

The raw FIT for the entire device is always correct. In contrast, the utilized FIT is very conservative, and only becomes accurate for designs that reasonably fill up the chosen device. FPGAs contain overhead, such as the configuration state machine, the clock network control logic, and the I/O calibration block. These infrastructure blocks contain flip flops, memories, and sometimes I/O configuration blocks.

The Projected SEU FIT by Component report includes the constant overhead for GPIO and HSSI calibration circuitry for first I/O block or transceiver the design uses. Because of this overhead, the FIT of a 1-transceiver design is much higher than 1/10 the FIT of a 10-transceiver design. However, a trivial design such as "a single AND gate plus flipflop" could use so few bits that its CRAM FIT rate is 0.01, which the report rounds to zero.

2.3.4. Mitigated FIT

You can lower FIT by reducing the observed FIT rate, such as by enabling ECC. You can also use the optional M20K ECC to mitigate FIT, as well as the (not optional) hard processor ECC and other hard IP such as memory controllers, PCIe, and I/O calibration blocks.

The Projected SEU FIT by Component Usage report's **w/ECC** column represents the FPGA's lowest guaranteed, provable FIT rate that the Quartus Prime software can calculate. ECC does not affect CRAM and flipflop rates; therefore, the data in the **w/ECC** column for these components is the same as the in **Utilized** column.

The ECC code strength varies with the device family. In Arria 10 devices, the M20K block can correct up to two errors, and the FIT rate beyond two (not corrected) is small enough to be negligible in the total.

2.3.5. Architectural Vulnerability Factor

The Single Event Functional Interrupt (SEFI) ratio measures bit errors due to SEU strikes versus functional interrupts. Minimizing this ratio improves SEU mitigation.

10% SEFI factors are A typical specification to deflate the raw FIT to that observed in practice. For reference, the last two columns in the Projected SEU FIT by Component Usage report show AVF deflations for a conservative SEFI of 50% and a moderate SEFI of 25%.

SEFI represents a combination of factors. A utilization + ECC factor of 40% and AVF of 25% thus represents a global SEFI factor of 10%, because 0.4 × 0.25 = 0.1. An end-to-end SEFI factor of 10% is typical for a full design.

Related Information

Understanding Single Event Functional Interrupts in FPGA Designs

3. Arria 10 Error Detection and Correction Feature Architecture

3.1. Error Detection and Correction for CRAM

3.1.1. Error Detection Cyclic Redundancy Check

In user mode, the contents of the configured configuration RAM (CRAM) bits can be affected by soft errors. These soft errors, which are caused by an ionizing particle, are not common in Altera FPGA devices. However, high-reliability applications that require error-free device operation may require your design to consider these errors.

The hardened on-chip EDCRC circuitry allows you to perform the following operations without any impact on the fitting or performance of the device:

- Auto-detection of cyclic redundancy check (CRC) errors during configuration.
- Optional soft errors (SEU and multiple bit upset) detection and identification in user mode.
- Fast soft error detection. The error detection speed is improved.
- Two types of check-bits:
  - Frame-based check-bits—stored in CRAM and used to verify the integrity of the frame.
  - Column-based check-bits—stored in registers and used to protect integrity of all frames.

During error detection in user mode, a number of EDCRC engines run in parallel for Arria 10 devices. The number of error detection CRC engines depends on the frame length—total bits in a frame.

Each column-based error detection CRC engine reads 128 bits from each frame and processes within four cycles. To detect errors, the error detection CRC engine needs to read back all frames.

Figure 3. Block Diagram for Error Detection in User Mode

The block diagram shows the registers and data flow in user mode.

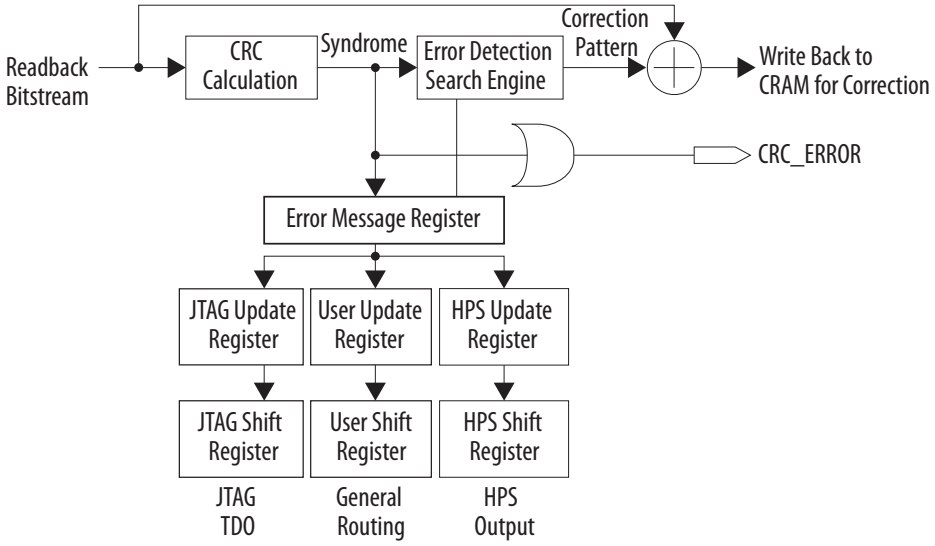


Table 1. Error Detection Registers

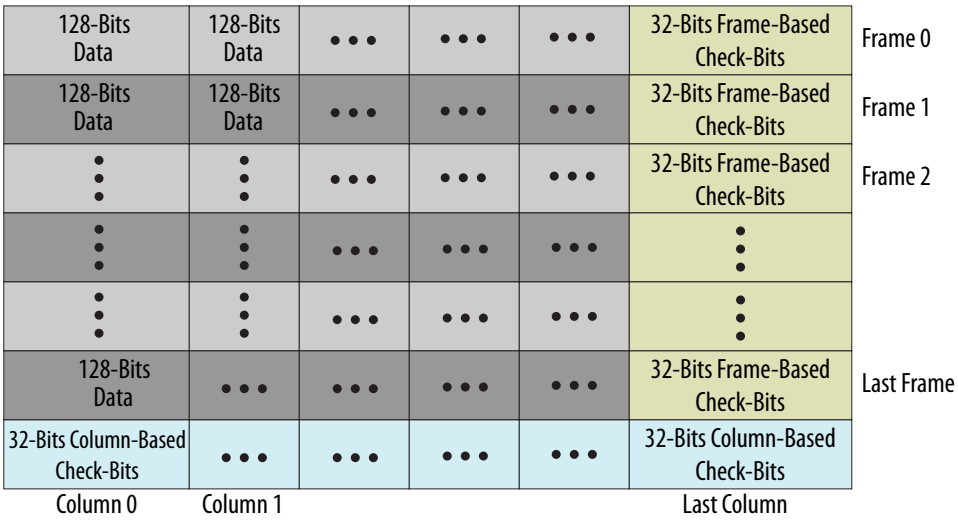
Name	Description
Error message registers (EMR)	Contains error details for single-bit and double-adjacent errors. The error detection circuitry updates this register each time the circuitry detects an error.
User update register	This register is automatically updated with the contents of the EMR one clock cycle after the contents of this register are validated. The user update register includes a clock enable, which must be asserted before its contents are written to the user shift register. This requirement ensures that the user update register is not overwritten when its contents are being read by the user shift register.
User shift register	This register allows user logic to access the contents of the user update register via the core interface. You can use the Error Message Register Unloader Intel® FPGA IP core to shift-out the EMR information through user shift register. For more information, refer to related information.
JTAG update register	This register is automatically updated with the contents of the EMR one clock cycle after the content of this register is validated. The JTAG update register includes a clock enable, which must be asserted before its contents are written to the JTAG shift register. This requirement ensures that the JTAG update register is not overwritten when its contents are being read by the JTAG shift register.
JTAG shift register	This register allows you to access the contents of the JTAG update register via the JTAG interface using the `SHIFT_EDERROR_REG` JTAG instruction.
Hard Processor System (HPS) update register	This register is automatically updated with the contents of the EMR one clock cycle after the content of this register is validated. The (HPS) update register includes a clock enable, which must be asserted before its contents are written to the HPS shift register. This requirement ensures that the HPS update register is not overwritten when its contents are being read by the HPS shift register.
HPS shift register	This register allows you to access the contents of the HPS update register via the HPS interface.

Related Information

Error Message Register Unloader Intel FPGA IP Core User Guide
Provides more information about using the Error Message Register Unloader Intel FPGA IP.

3.1.1.1. Column-Based and Frame-Based Check-Bits

Figure 4. Column-Based and Frame-Based Check-Bits



EDCRC Check-Bits Updates

Frame-based check-bits are calculated on-chip during configuration. Column-based check-bits are updated after configuration.

When you enable the EDCRC feature, after the device enters user mode, the EDCRC function starts reading CRAM frames. The data collected from the read-back frame is validated against the frame-based check-bits.

After the initial frame-based verification is completed, the column-based check-bits is calculated based on the respective column CRAM. The EDCRC hard block recalculates the column-based check-bits in one of the following scenarios:

- FPGA re-configuration
- After successful partial reconfiguration (PR) session
- After configuration via protocol (CvP) session

3.1.1.2. Error Message Register

The EMR contains information on the error type, the location of the error, and the actual syndrome. This register is 78 bits wide in Arria 10 devices. The EMR does not identify the location bits for uncorrectable errors. The location of the errors consists of the frame number, double word location and bit location within the frame and column.

You can shift out the contents of the register through the following:

- EMR Unloader IP core—core interface
- SHIFT_EDERROR_REG JTAG instruction—JTAG interface
- HPS Shift register—HPS interface

Figure 5. Error Message Register Map

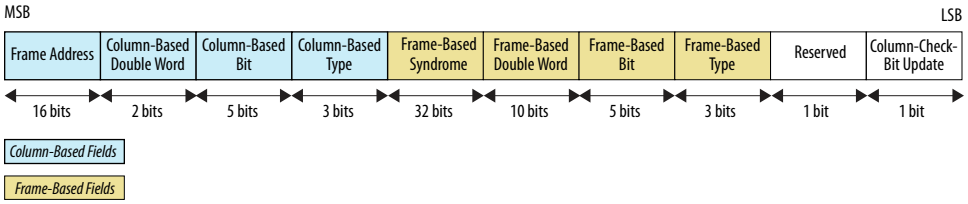


Table 2. Error Message Register Width and Description

Note: Refer to the *Correctable and Uncorrectable Error Cases* table to determine whether the error is correctable or uncorrectable.

Name	Width (Bits)	Description
Frame Address	16	Frame Number of the error location
Column-Based Double Word	2	There are 4 double words per frame in a column. It indicates the double word location of the error
Column-Based Bits	5	Error location within 32-bit double word
Column-Based Type	3	Types of error shown in the *Error Type in EMR* table.
Frame-Based syndrome register	32	Contains the 32-bit CRC signature calculated for the current frame. If the CRC value is 0, the CRC_ERROR pin is driven low to indicate no error. Otherwise, the pin is pulled high.
Frame-Based Double Word	10	Double word location within the CRAM frame.
Frame-Based Bit	5	Error location within 32-bit double word
Frame-Based Type	3	Types of error shown in the *Error Type in EMR* table.
Reserved	1	Reserved bit
Column-Based Check-Bits Update	1	Logic high if there is error encountered during the column check-bits update stage. The CRC_ERROR pin will be asserted and stay high until the FPGA is reconfigured.

Related Information

- Reading EMR using JTAG Interface on page 16
- Reading EMR using EMR Unloader IP Core on page 16
- Reading EMR using HPS on page 16
- Correctable and Uncorrectable Error Cases

3.1.1.2.1. Error Type in EMR

Table 3. Error Type in EMR

The following table lists the possible error types reported in the error type field in the EMR.

Note: Refer to the *Correctable and Uncorrectable Error Cases* table to determine whether the error is correctable or uncorrectable.

Error Types	Bit 2	Bit 1	Bit 0	Description
Frame-based	0	0	0	No error
	0	0	1	Single-bit error
	0	1	X	Double-adjacent error
	1	1	1	Uncorrectable error
Column-Based	0	0	0	No error
	0	0	1	Single bit error
	0	1	X	Double-adjacent error in a same frame
	1	0	X	Double-adjacent error in a different frame
	1	1	0	Double-adjacent error in a different frame
	1	1	1	Uncorrectable error

Related Information

- CRC_ERROR Pin Behavior on page 18
- Error Detection Frequency, SEU Mitigation for Arria 10 Devices
  Provides more information about Arria 10 SEU error detection frequency.
- Correctable and Uncorrectable Error Cases
- Correctable and Uncorrectable Error Cases

3.1.2. Recovering from CRC Errors

Arria 10 devices support the internal scrubbing capability. The internal scrubbing feature corrects correctable CRAM upsets automatically when an upset is detected. However, internal scrubbing can not fix the FPGA to a known good state. The time between the error and completion of scrubbing can be tens of millisecond. This duration represents thousands of clock cycles in which the corrupted data was written to memory or status registers. It is a good practice to always follow any SEU event with a soft-reset to bring the FPGA operation to a known good state.

If a soft-reset is unable to bring the FPGA to a known good state, you can reconfigure the device to rewrite the CRAM and reinitialize the design registers. The system that hosts the Arria 10 device must control the device reconfiguration. When reconfiguration completes successfully, the Arria 10 device operates as intended.

3.2. Memory Blocks Error Correction Code Support

ECC allows you to detect and correct data errors at the output of the memory. ECC can perform single-error correction, double-adjacent-error correction, and triple-adjacent-error detection in a 32-bit word. However, ECC cannot detect four or more errors.

The M20K blocks have built-in support for ECC when in x32-wide simple dual-port mode:

- The M20K runs slower than non-ECC simple-dual port mode when ECC is engaged. However, you can enable optional ECC pipeline registers before the output decoder to achieve higher performance compared to non-pipeline ECC mode at the expense of one cycle of latency.
- The M20K ECC status is communicated with two ECC status flag signals—e (error) and ue (uncorrectable error). The status flags are part of the regular output from the memory block. When ECC is engaged, you cannot access two of the parity bits because the ECC status flag replaces them.

4. Guidelines for Error Detection CRC and Error Correction Feature

4.1. Error Detection

4.1.1. Enabling Error Detection

There are two methods to turn on Arria 10 error detection CRC feature based on your application needs.

- If your design detects and reads the EMR using user logic, you need to instantiate the EMR Unloader IP core which will automatically turn the EDCRC feature on.
- If you want to monitor SEU with the external host and do not need to read the EMR from user logic, you can turn on EDCRC feature by enabling CRC_ERROR pin in your Quartus Prime project.

Related Information

Error Message Register Unloader Intel FPGA IP Core User Guide
Provides more information about using the Error Message Register Unloader Intel FPGA IP.

4.1.1.1. Enabling the Error Detection CRC_ERROR Pin

To enable the CRC_ERROR pin for external host monitoring purpose, perform the following steps:

1. On the **Assignments** menu, click **Device**.
2. Click **Device and Pin Options** and select the **Error Detection CRC** at the left panel.
3. Check the **Enable Error Detection CRC_ERROR pin**.
4. Select the EDCRC clock divisor from the list of **Divide error check frequency by**.

   Note: This option provides you with a flexibility to run the EDCRC at a slower speed. However, Intel recommends you to set to the smallest EDCRC clock divisor. Setting a high divisor can impact the error detection time performance. Refer to Arria 10 Handbook SEU Mitigation chapter of the Arria 10 handbook for detection time specification.
5. Check the **Enable open drain on CRC_ERROR pin** if you have an external pull up resistor on your board.
6. Click OK.

Related Information

SEU Mitigation for Arria 10 Devices
Provides more information about Arria 10 SEU error detection time.

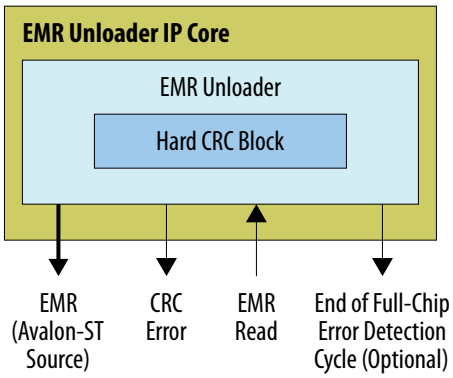
4.1.2. Reading EMR

4.1.2.1. Reading EMR using EMR Unloader IP Core

You can instantiate the EMR Unloader IP core to detect SEU and unload EMR content in user logic. The EDCRC feature will be turned on automatically when the EMR Unloader IP core is instantiated. EMR Unloader IP core helps you to read the EMR whenever there is an SEU event by:

- Unloading the EMR via core logic
- Accessing the hard CRC Block
- Providing access to the user logic to read the EMR data

Figure 6. EMR Unloader IP Core Block Diagram



Related Information

Error Message Register Unloader Intel FPGA IP Core User Guide
Provides more information about retrieving and storing the error message register using the Error Message Register Unloader Intel FPGA IP.

4.1.2.2. Reading EMR using HPS

The FPGA Manager in the HPS has the ability to monitor the CRC_ERROR status pin and to retrieve the error symptom, location and type. You can choose to enable the CRC error interrupt from the FPGA Manager, followed by CRC error information extraction from respective registers.

Related Information

FPGA Manager, Arria 10 Hard Processor System Technical Reference Manual

4.1.2.3. Reading EMR using JTAG Interface

To unload the contents of the EMR using a JTAG port, use the SHIFT_EDERROR_REG JTAG instruction. This JTAG instruction connects the EMR to the JTAG pin in the error detection block between the TDI and TDO pins. You can execute the instruction whenever the CRC_ERROR pin goes high. You must unload the contents of the EMR before the register is overwritten by the information of the next CRC error.

Table 4. SHIFT_EDERROR_REG JTAG Instruction

JTAG Instruction	Instruction Code	Description
SHIFT_EDERROR_REG	00 0001 0111	The JTAG instruction connects the EMR to the JTAG pin in the error detection block between TDI and TDO pins.

The following shows the Jam™ Standard Test and Programming Language (STAPL) Format File (**.jam**) used to execute the SHIFT_EDERROR_REG JTAG instruction to unload the contents of the EMR.

Example 1. Example of .jam File to Unload the Contents of the EMR for Arria 10 Device

```
ACTION UNLOAD_EMR = EXECUTE;
DATA EMR_DATA;
BOOLEAN out[78];
ENDDATA;
PROCEDURE EXECUTE USES EMR_DATA;
DRSTOP IDLE;
IRSTOP IDLE;
STATE IDLE;
IRSCAN 10, $017;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 78,$0, CAPTURE out[77..0];
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;
PRINT " ";
PRINT "Data read out from the ";

PRINT "EMR_Register :"    , out[77], out[76], out[75], out[74], out[73],
out[72], out[71], out[70], out[69], out[68], out[67], out[66], out[65], out[64],
out[63],out[62], " ", out[61], out[60], " ", out[59], out[58], out[57], out[56],
out[55], " ", out[54], out[53], out[52], " ", out[51], out[50], out[49],
out[48], out[47], out[46], out[45], out[44], out[43], out[42], out[41], out[40],
out[39], out[38], out[37], out[36], out[35], out[34], out[33], out[32], out[31],
out[30], out[29], out[28], out[27], out[26], out[25], out[24], out[23], out[22],
out[21], out[20], " ", out[19], out[18], out[17], out[16], out[15], out[14],
out[13], out[12], out[11], out[10], " ", out[9] , out[8], out[7], out[6],
out[5], " ", out[4], out[3], out[2], " ", out[1], " ", out[0];

'PRINT " ";

PRINT "Frame Address :", out[77], out[76], out[75], out[74], out[73], out[72],
out[71], out[70], out[69], out[68], out[67], out[66], out[65], out[64], out[63],
out[62];
PRINT "Column-Based Double Word Location :", out[61], out[60];
PRINT "Column-Based Bit :", out[59], out[58], out[57], out[56], out[55];
PRINT "Column-Based Type :", out[54], out[53], out[52];
PRINT "Frame-Based Syndrome :" , out[51], out[50], out[49], out[48], out[47],
out[46], out[45], out[44], out[43], out[42], out[41], out[40], out[39], out[38],
out[37], out[36], out[35], out[34], out[33], out[32], out[31], out[30], out[29],
out[28], out[27], out[26], out[25], out[24], out[23], out[22], out[21], out[20];
PRINT "Frame-Based Double Word Location :", out[19], out[18], out[17], out[16],
out[15], out[14], out[13], out[12], out[11], out[10];
PRINT "Frame-Based Bit :", out[9] , out[8], out[7], out[6], out[5];
PRINT "Frame-Based Type :", out[4], out[3], out[2];
PRINT "Reserved bit :", out[1];
PRINT "Column-based EDCRC Check Bits Update:", out[0];
STATE IDLE;

EXIT 0;
ENDPROC;
```

Related Information

- SEU Mitigation for Arria 10 Devices
  Provides more information about Arria 10 SEU features.

- Modifying Single-Device .jam Files for Use in a Multi-Device JTAG Chain on page 41

4.2. Enabling Error Correction (Internal Scrubbing)

Arria 10 supports the internal scrubbing feature to automatically scrub away the flipped bit induced by the SEU. To enable the internal scrubbing feature, follow these steps:

1. On the **Assignments** menu, click **Device**.
2. Click **Device and Pin Options** and select the **Error Detection CRC** tab.
3. Turn on **Enable internal scrubbing**.
4. Click **OK**.

4.3. Interpreting CRC_ERROR

It is important to determine the error type when an SEU is detected. This section explains the CRC_ERROR pin behavior and how to interpret whether the error type is correctable or uncorrectable.

4.3.1. CRC_ERROR Pin Behavior

The Arria 10 fast EDCRC feature runs all the column-based check-bits engine in parallel. When an SEU is detected, the column-based check-bits asserts the CRC_ERROR, the detected frame location is then passed to the frame-based check-bits to further localize the affected bit. This process causes the CRC_ERROR pin to assert twice. Column-based check-bits assert the first CRC_ERROR pulse and followed by the frame-based check-bits asserting the second pulse.

In Arria 10, as soon as an SEU is detected, the CRC_ERROR is asserted high and remains high until the EMR is ready to be read. You can unload the EMR data as soon as the CRC_ERROR pin goes low. Once EMR data is unloaded, can determine the error type and the affected location. With these information you can decide how your system should respond to the specific SEU event.

Figure 7. Fast EDCRC Process Flow Chart

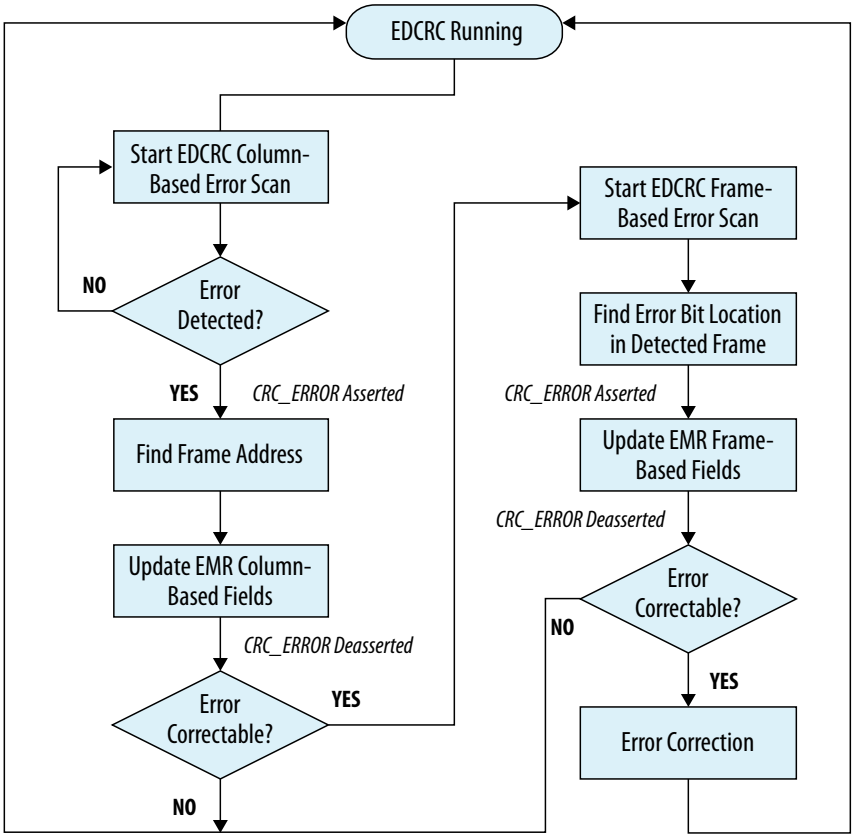
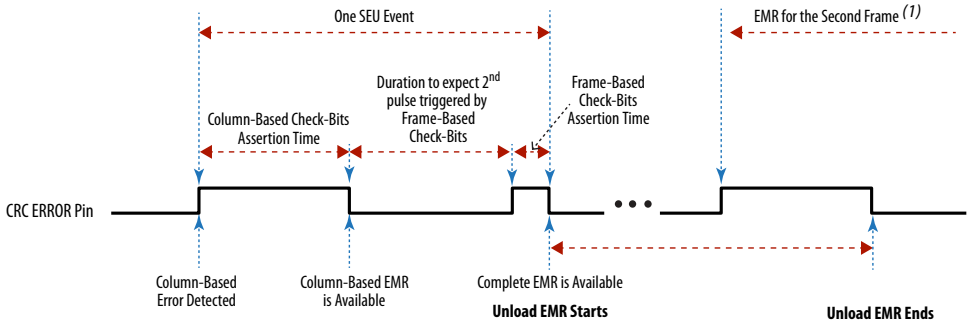


Figure 8. Timing Diagram for Column-Based Check-Bits

If the error is correctable, in most cases, there is a second pulse in a single SEU event .There are cases where the error is uncorrectable when the CRC_ERROR pin asserts 2 pulses, refer to Correctable and Uncorrectable Error for complete correctable and uncorrectable error cases. The complete EMR is only available at the falling edge of the second pulse.

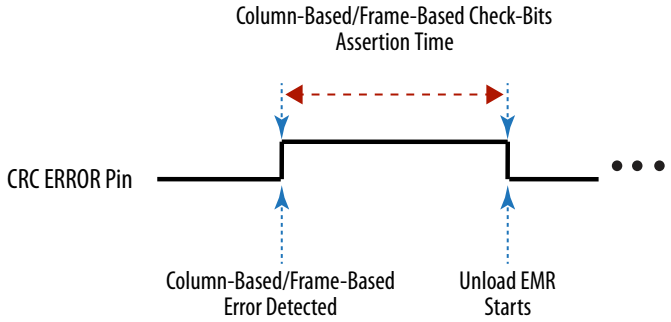


(1) In a rare event of correctable double-adjacent error located in different frames.

In the rare event of an uncorrectable and un-locatable error, the CRC_ERROR signal is asserted only once. There is no second pulse assertion by frame-based check-bits due to the uncorrectable error location cannot be located. The statistical likelihood of uncorrectable multi-bit SEU is less than one in 10,000 years for a device in typical environmental conditions.

Figure 9. Timing Diagram for Column-Based or Frame-Based Check-Bits

Example of CRC_ERROR pin behavior for column-based/frame-based check-bits with a single pulse observed in one SEU event.



Related Information

- Error Type in EMR on page 13
- Error Detection Frequency, SEU Mitigation for Arria 10 Devices
  Provides more information about Arria 10 SEU error detection frequency.
- Correctable and Uncorrectable Error Cases

4.3.2. Correctable and Uncorrectable Error

When an SEU is detected, you can read the EMR data to determine whether the error is correctable or uncorrectable. Intel recommends you to use Altera EMR Unloader IP core in your design. The Altera EMR Unloader IP core interprets the error and reports it at the output.

Table 5. Correctable and Uncorrectable Error Cases

The table summarizes the correctable and uncorrectable error cases. You do not need to determine whether the current EDCRC operation is in frame-based check-bits or column-based check-bits but you need to know how to interpret the error type of the column-based or frame-based. If the EMR Unloader IP core reports the error type other than 3'b111, the error is correctable and the error will be scrubbed if you turned on internal scrubbing.

Case	EDCRC Operation	CRC_ERROR Pulse	Column-Based Field	Frame-Based Field	Correctable	Remark
A(1)	Frame-based check-bits	1	All 0's	Type = 3'b001 or Type = 3'b010 & bit ≠ 5'h1F or Type = 3'b011 & bit = 5'h1F	Yes	Error will be corrected if internal scrubbing is turned On
B(1)	Frame-based check-bits	1	All 0's	Type = 3'b111 or Type = 3'b010 & bit = 5'h1F or Type = 3'b011 & bit ≠ 5'h1F	No	The frame-based check-bits will retry for 2 times and enter dead state where CRC_ERROR

continued...

(1) Case can occur only when the Frame-based CRC error is detected after the CRAM is configured, such as FPGA configuration, partial PR or CvP. The SEU event is statistically impossible to happen during CRAM configuration, such cases are to cover other problems such as corrupted configuration data or bad CRAM that unable to hold the correct bit setting.

Case	EDCRC Operation	CRC_ERROR Pulse	Column-Based Field	Frame-Based Field	Correctable	Remark
				EMR Unloader IP core will set type = 3'b111 if any of above condition met		stays high until FPGA reconfiguration
C	Stuck in dead state	1 pulse and stay high after 2nd assertion	EMR Unloader IP core set Type = 3'b111	EMR Unloader IP core set type = 3'b111	No	CRC_ERROR stays high until FPGA reconfiguration. Refer Case B to understand how EDCRC can stuck in dead state
D	Column-based check-bits	1	Type = 3'b111 or Type = 3'b010 & bit = 5'h1F or Type = 3'b011 & bit ≠ 5'h00 EMR Unloader IP core will set type = 3'b111 if any of above condition met	All 0's	No	Detected uncorrectable error during column-based check-bits
E	Column-based check-bits and frame-based check-bits	2	Any type except: Type = 3'b111 or Type = 3'b010 & bit = 5'h1F or Type = 3'b011 & bit ≠ 5'h00	Type = 3'b111 or Type = 3'b010 & bit = 5'h1F or Type = 3'b011 & bit ≠ 5'h1F EMR Unloader IP core will set type = 3'b111 if any of above condition met	No	Detected uncorrectable error
F	Column-based check-bits and frame-based check-bits	2	Any type except: Type = 3'b111 or Type = 3'b010 & bit = 5'h1F or Type = 3'b011 & bit ≠ 5'h00	Any type except: Type = 3'b111 or Type = 3'b010 & bit = 5'h1F or Type = 3'b011 & bit ≠ 5'h1F	Yes	Error will be corrected if internal scrubbing is turned On

5. Guidelines for Embedded Memory ECC Feature

The Arria 10 FIFO Intel FPGA IP supports embedded memory ECC for M20K memory blocks. The built-in ECC feature in the Arria 10 devices can perform:

- Single-error detection and correction
- Double-adjacent-error detection and correction
- Triple-adjacent-error detection

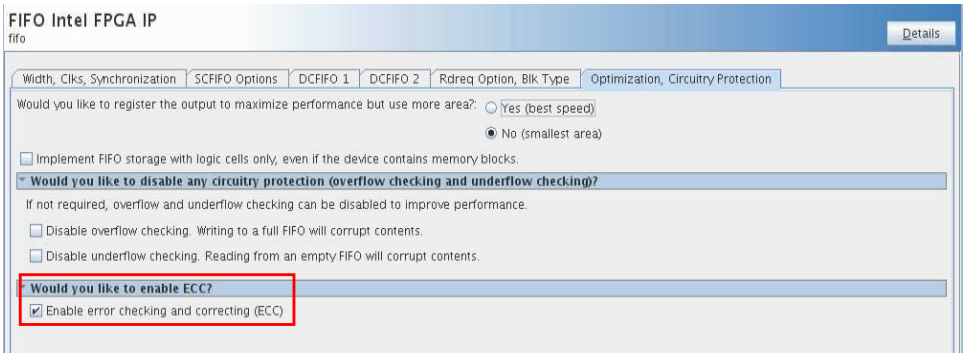
You can turn on FIFO Embedded ECC feature by enabling `enable_ecc` parameter in the FIFO Intel FPGA IP parameter editor.

Note: Embedded memory ECC feature is only available for M20K memory block type.

Note: The embedded memory ECC supports variable data width. When ECC is enabled, RAM combines multiple M20K blocks in the configuration of 32 (width) x 512 (depth) to fulfill your instantiation. The unused data width will be tied to the V_{CC} internally.

Note: The embedded memory ECC feature is not supported in mixed-width mode.

Figure 10. ECC Option in FIFO Intel FPGA IP Parameter Editor



When you enable the ECC feature, a 2-bit wide error correction status port (`eccstatus[1:0]`) will be created in the generated FIFO entity. These status bits indicate whether the data that is read from the memory has an error in single-bit with correction, fatal error with no correction, or no error bit.

- 00: No error
- 01: Illegal
- 10: A correctable error occurred and the error has been corrected at the outputs; however, the memory array has not been updated.
- 11: An uncorrectable error occurred and uncorrectable data appears at the output

Related Information

FIFO Intel FPGA IP User Guide

6. Arria 10 EDCRC Reference Design

The EDCRC reference design demonstrates the following main SEU detection and recovery for Arria 10:

- Instantiating various SEU-related IP cores such as EMR Unloader IP core, Advanced SEU Detection IP core, and Fault Injection IP core
- Demonstrating how the Advanced SEU Detection IP core retrieves the SMH information from the EPCQ-L with Serial Flash Controller IP core(2)
- Integrating the reference design into your system and characterize your system response to the SEU event with the Intel Fault Injection feature.

Related Information

- Arria 10 EDCRC Reference Design Files
  Reference design files that you need to apply steps and compilation described in *Creating Arria 10 SEU Fault Injection and Hierarchy Tagging Design with Qsys*.
- Complete Arria 10 EDCRC Reference Design Files
  Precompiled reference design files ready for design testing in *Design Testing with Fault Injection Debugger*.
- Arria 10 GX FPGA Development Kit

6.1. System Requirements

This reference design is targeted for the following hardware and software:

- Arria 10 development kit that is using 10AX115S2F45I2SG device.
- Quartus Prime software version 16.0

  Note: You must have a licensed version of the Quartus Prime software to generate SMH files.

Note: You can tweak some setting in this design if you wish to test on other Arria 10 devices. For example, you can change the device to other Arria 10 part, set other clock source frequency, and clock source pin assignment.

(2) You can only use EPCQ-L to store SMH and access with Serial Flash Controller when you set your MSEL pin to Active Serial Configuration.

6.2. Creating Arria 10 SEU Fault Injection and Hierarchy Tagging Design with Qsys

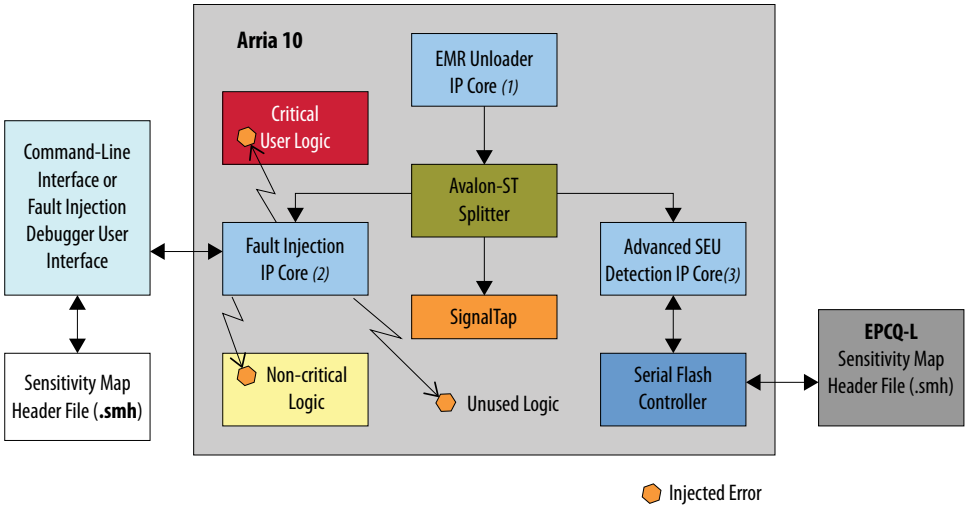
The `a10-seu.zip` reference design consists of:

- `a10_seu.qar`—the project archive file
- `top.v`—the top level module of the project
- `top.sdc`—the timing constraint file
- `top.stp`—the Signal Tap file

Note: The `a10-seu-complete.zip` consists of a fully compiled and output files-ready reference design. You can refer directly to Design Testing with Fault Injection Debugger on page 33 if you choose to use this complete design as a reference.

In this design, you will use Platform Designer (Standard) to connect the Intel SEU-related IP cores together. IP core to be connected are EMR Unloader IP core, Fault Injection IP core and Advanced SEU Detection IP core. Some other IP cores are also needed to make the design complete, which are Altera IOPLL IP core, AVST Splitter and Serial Flash Controller IP core.

Figure 11. Arria 10 SEU Fault Injection and Hierarchy Tagging Design



Notes:
1. The Fault Injection IP core and Advanced SEU Detection IP core read the EMR from EMR Unloader IP core .
2. The Fault Injection IP core flips the bits of the targeted logic.
3. The Advanced SEU Detection IP core flag the affected region by reading the .smh file stored in EPCQ-L.

Related Information

Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices

6.2.1. Starting Quartus Prime Software and Opening the Reference Design Project

The Quartus Prime project serves as an easy starting point for this reference design development flow. The Quartus Prime project contains all setting and design files required to create the `.sof`.

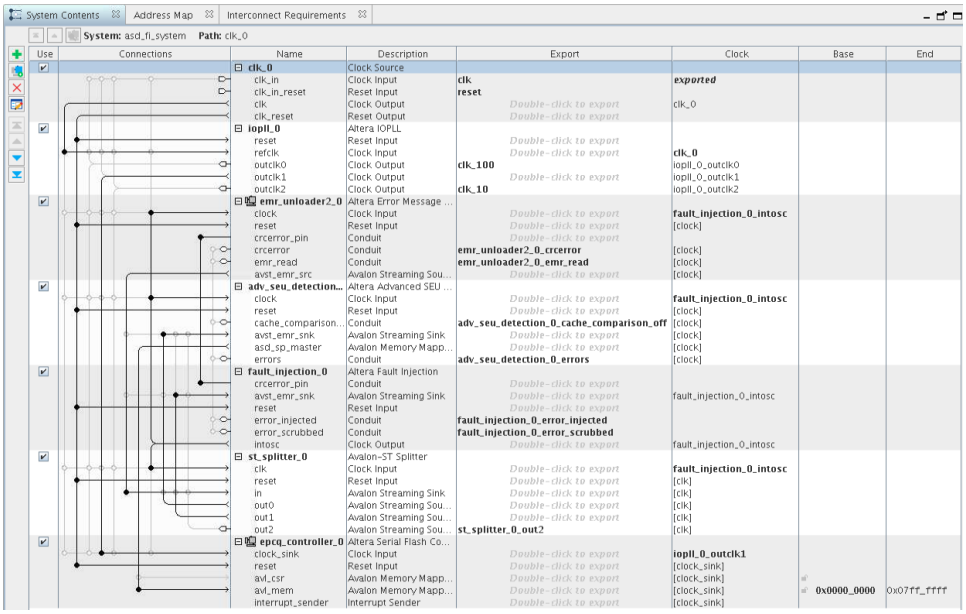
To open the Quartus Prime project, perform the following steps:

1. In the Quartus Prime software, click **Open Existing Project** on the splash screen, or on the **File** menu, click **Open Project**. The **Open Project** dialog box appears.
2. Browse to the `<qar file directory>` where you store your `.qar` file.
3. Select the file `a10_seu.qar` and click **Open**.
4. Change the destination folder name if required, or leave it default as `<qar file directory>/a10_seu_restored`. Click **OK**.

6.2.2. Creating New Qsys System

To create a new Qsys system, click **Qsys** on the **Tools** menu in the Quartus Prime software. Qsys starts and displays the **System Contents** tab.

Figure 12. Complete IP Core settings and Connections in Qsys



6.2.2.1. Specifying Target FPGA and Clock Settings

To specify target FPGA and clock settings in Qsys, perform the following steps:

1. Click **Device Family** in **View** menu, select the **Device Family** that matches the Arria 10 device you are targeting. Warning will appear if the selected device family does not match Quartus Prime project settings, you need to make sure your selected device in Quartus Prime project settings match to your selected **Device Family** in Qsys.
2. On the **System Contents** tab, double click the `clk_0` component. In the **Parameters** tab for `clk_0`, set the **Clock frequency** to `50MHz`.

   Next, you begin to add other IP cores to the Qsys system.

6.2.2.2. Adding Altera IOPLL IP Core

You must instantiate Altera IOPLL IP core in this reference design to generate 3 different clock sources, 10MHz, 20MHz and 100MHz. To add the Altera IOPLL IP core, perform the following steps:

1. On the **IP Catalog** Tab, expand **Basic Functions**, expand **Clock**; **PLLs and Resets**, **PLL**, and then click **Altera IOPLL**.
2. Click **Add**. The **Altera IOPLL** parameter editor appears.
3. On **PLL** tab, at **General** section, set the **Reference Clock Frequency** to 50.
4. Uncheck **Enable locked output port**.
5. At **Output Clocks** section, set **Number Of Clocks** to 3.
6. Set the clocks as the following:
   a. For `outclk0`, set the **Clock Name** to `clk_100`and set the **Desired Frequency** to `100MHz`.
   b. For `outclk1`, set the **Clock Name** to `clk_20` and set the **Desired Frequency** to `20MHz`.
   c. For `outclk2`, set the **Clock Name** to `clk_10` and set the **Desired Frequency** to `10MHz`.
7. Click **Finish** to return to **Qsys**.
8. On **System Contents** tab, an instance of the **iopll_0** appears in the system contents table.
9. Connect the `clk` port of the `clk_0` clock source to the `refclk` port of the `iopll_0`.
10. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of the `iopll_0`.
11. Double click the `outclk2` of the `iopll_0` at **Export** column to export `outclk2` as the clock source for other component outside of this Qsys system. Rename the exported signal as `clk_10`.
12. Double click the `outclk0` of the `iopll_0` at **Export** column to export `outclk0` as the clock source for other component outside of this Qsys system. Rename the exported signal as `clk_100`.

6.2.2.3. Adding EMR Unloader IP Core

You must instantiate EMR Unloader IP core to unload the EMR whenever there is SEU event. To add the EMR Unloader IP core, perform the following steps:

1. On the **IP Catalog** tab, expand **Basic Functions**, expand **Configuration and Programming**, and then click **Altera Error Message Register Unloader**.
2. Click **Add**. The **Altera Error Message Register Unloader** parameter editor appears.
3. In **CRC error check clock divisor** list, select **2**.
4. Check the **Input clock is driven from Internal Oscillator**. This reference example uses Internal Oscillator to drive EMR Unloader IP core.
5. Click **Finish** to return to Qsys. On **System Contents** tab, an instance of the **emr_unloader2_0** appears in the system contents table.
6. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of `emr_unloader2_0`.
7. Double click the `crcerror`, and `emr_read` of `emr_unloader2_0` at **Export** column to export them for external access. Leave the name as default.

6.2.2.4. Adding Advance SEU Detection IP Core

You must instantiate the ASD IP core for sensitivity processing and to validate the hierarchy tagging feature. To add ASD IP core, perform the following steps:

1. On the **IP Catalog tab**, expand **Basic Functions**, expand **Configuration and Programming**, and then click **Altera Advanced SEU Detection**.
2. Click **Add**. The **Altera Advanced SEU Detection** parameter editor appears.
3. Leave the **CRC error cache depth** list default selection at 8.
4. Set **Largest ASD region ID used** to 3.
5. Check the **Use on-chip sensitivity processing**.
6. Set **Memory interface address width** to 32.
7. Set **Sensitivity Data start address** to `0x02000000`.
8. Click **Finish** to return to Qsys. On **System Contents** tab, an instance of the **adv_seu_detection_0** appears in the system contents table.
9. Connect the `clk_reset` port of the `clk_0` clock source to the `reset` port of the `adv_seu_detection_0`.
10. Double click the `cache_comparison_off`, and `errors` port of `adv_seu_detection_0` at **Export** column to export them for external access, leave the name default.

6.2.2.5. Adding Fault Injection IP Core

You must instantiate the Fault Injection IP core to inject the fault to the CRAM. The faults can be a Single Bit Error (SBE), Double Adjacent Error (DAE) or Uncorrectable Multi Bit Error (UMBE). To add Fault Injection IP core, perform the following steps:

1. On the **IP Catalog** tab, expand **Basic Functions**, expand **Configuration and Programming**, and then click **Altera Fault Injection**.
2. Click **Add**.**The Altera Fault Injection** parameter editor appears.
3. Click **Finish** to return to **Qsys**.On the **System Contents** tab, an instance of the **fault_injection_0** appears in the system contents table.
4. Connect `clk_reset` port of the `clk_0` clock source to the `reset` port of the `fault_injection_0`.
5. Connect `intosc` port of the `fault_injection_0` to `clock` port of `emr_unloader2_0`.
6. Connect `intosc` port of the `fault_injection_0` to `clock` port of `adv_seu_detection_0`.
7. Connect `crcerror_pin` port of `emr_unloader2_0` to `crcerror_pin` port of `fault_injection_0`.
8. Double click the `error_injected`, and `error_scrubbed` of the `fault_injection_0` at **Export** column to export them for external access, leave the name default.

6.2.2.6. Adding Avalon-ST Splitter

EMR Unloader core sends the EMR data to the downstream IP cores with Avalon-ST protocol. Both ASD IP core and Fault Injection IP core require EMR data from EMR Unloader core. You need to instantiate the Avalon-ST Splitter to distribute the EMR data from EMR Unloader to ASD IP core and Fault Injection IP core. To add the Avalon-ST Splitter, perform the following steps:

1. On the **IP Catalog** tab, expand **Basic Functions**, expand **Bridges and Adaptors**, expand **Streaming**, and click **Avalon-ST Splitter**.
2. Click **Add**. The **Avalon-ST Splitter** parameter editor appears.
3. Set **NUMBER_OF_OUTPUTS** to 3.
4. Check only **USE_VALID**, **USE_ERROR** and **USE_DATA**, uncheck all other check boxes.
5. Set **DATA_WIDTH** to 119.
6. Set **ERROR_WIDTH** to 1.
7. Set **BITS_PER_SYMBOL** to 119.
8. Click **Finish** to return to Qsys. On the **System Contents** tab, an instance of the **st_splitter_0** appears in the system contents table.
9. Connect `clk_reset` port of the `clk_0` clock source to `reset` port of `st_splitter_0`.
10. Connect `intosc` port of the `fault_injection_0` to `clk` port of `st_splitter_0`.

11. Connect `avst_emr_src` port of `emr_unloader2_0` to `in` port of `st_splitter_0`.
12. Connect `out0` port of `st_splitter_0` to `avst_emr_snk` port of `adv_seu_detection_0`.
13. Connect `out1` port of `st_splitter_0` to `avst_emr_snk` port of `fault_injection_0`.
14. Double click the `out2` port of the `st_splitter_0` at **Export** column to export it for external access, leave the name default. This port will be used for Signal Tap purpose to read the EMR value after the fault injection.

6.2.2.7. Adding Serial Flash Controller

You must use the Serial Flash Controller IP core to access to the EPCQ-L1024 that stores the SMH file in this reference design. The ASD IP core reads the SMH data from EPCQ-L1024 via Serial Flash Controller IP core. To add the Serial Flash Controller, perform the following steps:

1. On the **IP Catalog** tab, expand **Basic Functions**, expand **Configuration and Programming**, and then click **Altera Serial Flash Controller**.
2. Click **Add**. The **Altera Serial Flash Controller** parameter editor appears. Set the parameters as the follows:
   a. On **Configuration device type** list, select **EPCQL1024**.
   b. On **Choose I/O mode**, select **QUAD**.
   c. On **Number of Chip Selects used** list, select **1**.
3. Click **Finish** to return to **Qsys**. On the **System Content** tab, an instance of the **epcq_controller_0** appears in the system contents table.
4. Connect `outclk1` port of `iopll_0` to `clock_sink` port of `epcq_controller_0`.

   Note: The Fmax for Serial Flash Controller is 25MHz
5. Connect `clk_reset` port of `clk_0 clock` source to `reset` port of `epcq_controller_0`.
6. Connect `asd_sp_master` port of `adv_seu_detection_0` to `avl_mem` port of `epcq_controller_0`.

6.2.3. Generating Qsys System

To generate the Qsys system, perform the following steps:

1. Click **Generate HDL** from **Generate** menu.
2. Click **Generate**. Click **Yes** when the **Save Changes?** dialog box appears.
3. Type `asd_fi_system` in the **File name** box and click **Save**. The **Generate** dialog box appears and system generation process begins.
4. Click **Close** to close the dialog box.
5. On the **File** menu, click **Exit** to close **Qsys** and return to the **Quartus Prime** software.

   You are ready to integrate the Qsys system into Quartus Prime project.

6.2.4. Integrating Qsys System into Quartus Prime Project

To complete the reference design, you must perform the following tasks:

- Generate In-System Source and Probe (ISSP) IP core
- Quartus Prime project setting and add the following files (provided in download package) to the project:
  - `Top.v`—instantiate the Qsys system module and connect all other IP cores
  - `Top.stp`—monitor some key signals with Signal Tap tool
  - `Top.sdc`—timing constraint
- Assign ASD regions to up counter and down counter
- Assign FPGA device and pin locations
- Compile the project

6.2.4.1. Generating In-System Source and Probe IP Core

To generate ISSP, perform the following steps:

1. On **IP Catalog**, expand **Basic Functions**, expand **Simulation; Debug and Verification**, expand **Debug and Performance** and double click **Altera In-System Sources and Probes**.
2. **IP Parameter Editor** appears, key in `issp` in **Entity name**, click **OK**.
3. Set **Probe Port Width [0..511]** to `0`.
4. Set **Source Port Width [0..511]** to `4`.
5. Leave default to all other setting.
6. Click **Generate HDL** from **Generate Menu**, click **Generate**.
7. Click **close** and click **Exit** from **File** menu.
8. Click **Yes** if prompted to add the Quartus Prime IP File to the project.

6.2.4.2. Quartus Prime Project Settings

To set the Quartus Prime project setting, add the top level file, Signal Tap file and SDC file to the project, perform the following steps:

1. Click **Device** at **Assignments** menu, and then click **Device and Pin Options** in **Device** dialog box.
2. Under **Configuration Category**, select **Active Serial x4** for the **Configuration scheme**.
3. Under **Error Detection CRC Category**, check the **Enable Error Detection CRC_ERROR pin**.
4. Leave **Enable internal scrubbing** uncheck.

   Note: You can enable **Enable internal scrubbing** during internal scrubbing feature tryout.
5. Set the **Divide error check frequency by** list to 2.
6. Check the **Generate SEU sensitivity map file** (`.smh`).
7. Click **OK** to exit **Device and Pin Options** dialog box.

8. Click **OK** again to exit **Device** dialog box.
9. Click **Settings** at **Assignments** menu, select **Files** category at left panel, add `top.v`, `top.stp` and `top.sdc` to the project.
10. Select **TimeQuest Timing Analyzer** category at left panel, add the `top.sdc` to SDC files to include in the project.
11. Select **Signal Tap Logic Analyzer** category at left panel, check **Enable Signal Tap Logic Analyzer** and select the `top.stp` as the **Signal Tap File** name.
12. Click **OK** to close the **Settings** window.
13. Click **Processing** Menu, click **Start ➤ Analysis and Synthesis**.

6.2.4.3. Assigning ASD Regions

This reference design uses 3 ASD regions. To assign the ASD regions, perform the following steps:

1. At **Project Navigator** window, select **Hierarchy**, expand top, right click **down_counter:down_counter_inst**, select **Design Partition**, **Set as Design Partition**.
2. Repeat step 1 for **up_counter:up_counter_inst** to set the **Design Partition**.
3. In the **Design Partition Window** , set the **Netlist Type** and **ASD Region** for the following **Partition Name**:

Partition Name[3]	Netlist Type	ASD Region[4]
Top	Source File	1
down_counter:down_counter_inst	Source File	2
up_counter:up_counter_inst	Source File	3

6.2.4.4. Assigning FPGA Pin Location

To assign the clock source pin to your design, perform the following steps:

1. Launch the **Pin Planner** from the **Assignment** menu.
2. Assign **AU33** to `inclk` input.
3. Close the **Pin Planner**.

6.2.4.5. Compiling the Project

You must compile the project to generate the `.sof` file and `.smh` file. To compile the project, perform the following steps:

1. Click **Start Compilation** in the **Processing** menu.

(3) You can toggle **Design Partition Window** on or off from **Assignments** menu or enter the shortcut key Alt+D.

(4) To make **ASD Region** column visible in **Design Partition Window**, right click the header of the table and check **ASD Region**.

The full compilation process begins and this may take a while to complete the compilation.

2. After the compilation complete, you will get the `.sof` file and `.smh` file in the `output_files` folder, you need these files for hardware verification later.

6.3. Design Testing with Fault Injection Debugger

The following are the main steps to test your reference design:

1. Convert `.sof` file and `.smh` file to `.jic` file.
2. Program `.jic` file to EPCQ-L.
3. Launch Signal Tap Logic Analyzer and Fault Inject Debugger.
4. Configure the `.sof` to Arria 10 and reading `.smh` file with Fault Injection Debugger.
5. Start Signal Tap to monitor the signal and injecting an error with Fault Injection Debugger.
6. Observe the Signal Tap output.

This section will go through some simple steps to inject faults to the CRAM. For more information about the Fault Injection Debugger, refer to Fault Injection Debugger User Guide.

Related Information

Debugging Single Event Upset Using the Fault Injection Debugger, AN 866: Mitigating and Debugging Single Event Upsets in Quartus Prime Standard Edition
Provides more information about using the Fault Injection Debugger.

6.3.1. Converting .sof File and .smh File to .jic File

To program `.sof` file and `.smh` file into EPCQ-L, you must convert them to a `.jic` file. The converted `.jic` file is consist of:

- The bit stream forArria 10 FPGA configuration in Active Serial mode upon power up
- The `.smh` file content at certain offset that you can define in Convert Programming File tool

To convert, perform the following steps:

1. Go to your `output_files` folder, duplicate the `top.smh` file and rename it to `top.hex`.

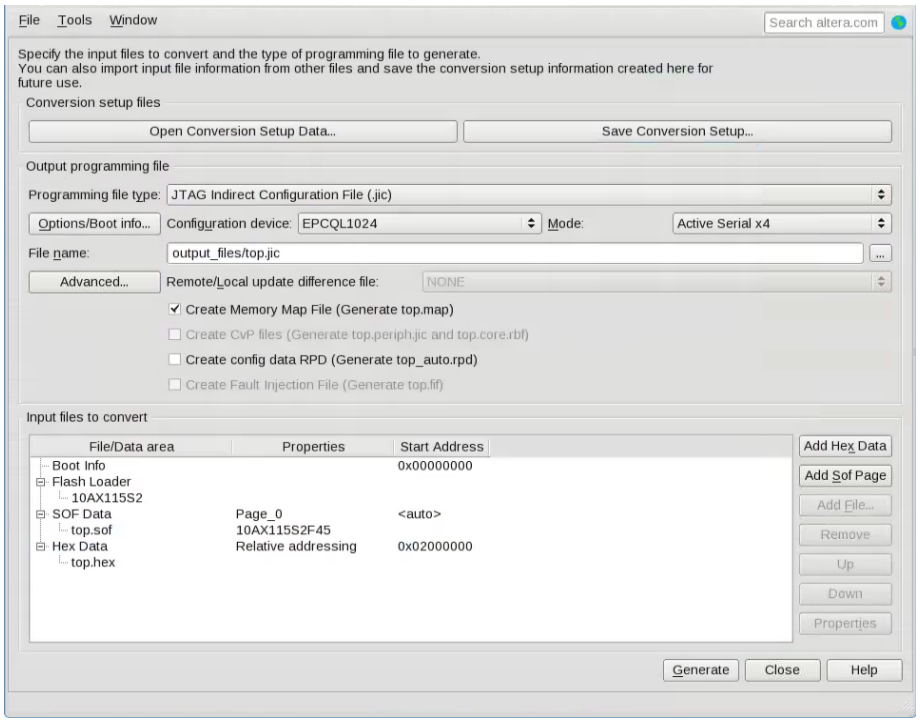
   Note: The `.smh` file is in Intel HEX standard format, i.e. bytes addressing little endian. You may need to convert the `.smh` file to match the endianness of your system

2. Launch **Convert Programming File** tool from **File** menu.
3. At **Output programming file** section, select **JTAG Indirect Configuration File (.jic)** from the **Programming file type** list.
4. Select **EPCQL1024** from **Configuration device** list.
5. Select **Active Serial x4** from **Mode** list.

6. Give the **File name** as `output_files/top.jic`.

   Optional to check **Create Memory Map File (Generate top.map)** and **Create config data RPD (Generate top_auto.rpd)**.
7. At **Input files to convert** section, select **Flash Loader** at the column of **File/Data area**.
8. Click **Add Device** button and select **Arria 10**, **10AX115S2** and click **OK**.
9. Select **SOF Data** at **File/Data area** column, click **Add File** button and select the `top.sof` inside the `output_files` folder.
10. Select `top.sof` under **SOF Data**, click the **Properties** button, enable **Compression** and click **OK** to close the **SOF File Properties** dialog.
11. Click **Add Hex Data** at **Input files to convert**section.
12. Select **Relative addressing** and set the **start address** to `0x2000000`. Leave **Big endian** as the default selection for **Endianness**. Select `top.hex` from your `output_files` folder, and click **OK**.

    The figure below shows the final setting for the `.jic` file generation. Verify and click **Generate** button.



13. Click **Close** button to close the **Convert Programming File** after `.jic` is generated successfully.

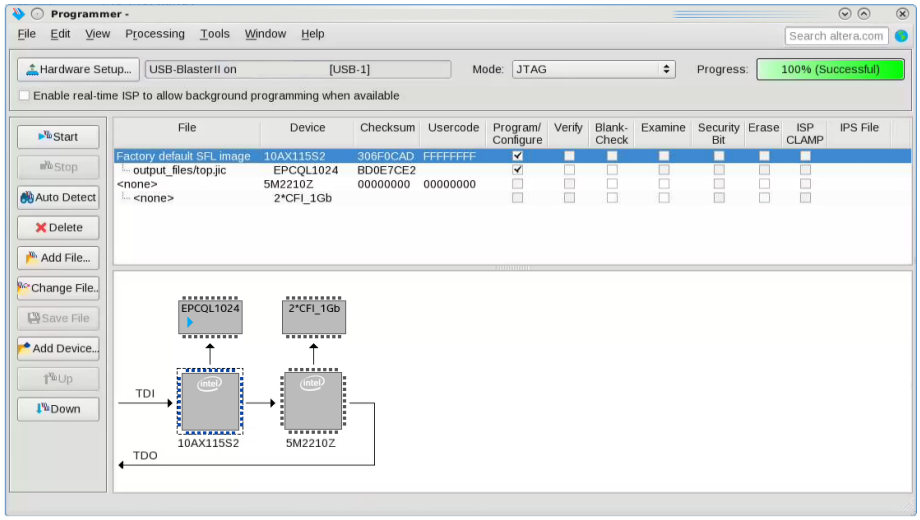
6.3.2. Programing .jic File into EPCQ-L

Before performing this task, ensure that your board configuration scheme is set to Active Serial by setting the MSEL[2:0] pins to `b'0101` or `b'011`. Refer to the Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices for more information.

To program the generated `.jic` file into the EPCQ-L, perform the following steps:

1. Launch **Programmer** at **Tools** menu.
2. Ensure that the valid programming cable is selected at **Hardware Setup**.
3. Click **Auto Detect** button and you should see the detect JTAG chain displayed in the programmer window.
4. Select **Arria 10 FPGA**, click the **Change File** button and select `top.jic` file in your `output_files` folder.
5. Check the **output_files/top.jic Program/Configure**, the **Factory default SFL image Program/Configure** will be checked automatically.

   The diagram below shows the final setting of the programmer.



6. Click **Start** to program `top.jic` file, this operation may take several minutes to complete.

Related Information

Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices

6.3.3. Launching Signal Tap Logic Analyzer

To observe the signals monitored by the Signal Tap, you must launch the Signal Tap Logic Analyzer and start the Signal Tap operation before the fault injection operation. To launch the Signal Tap Logic Analyzer, perform the following steps:

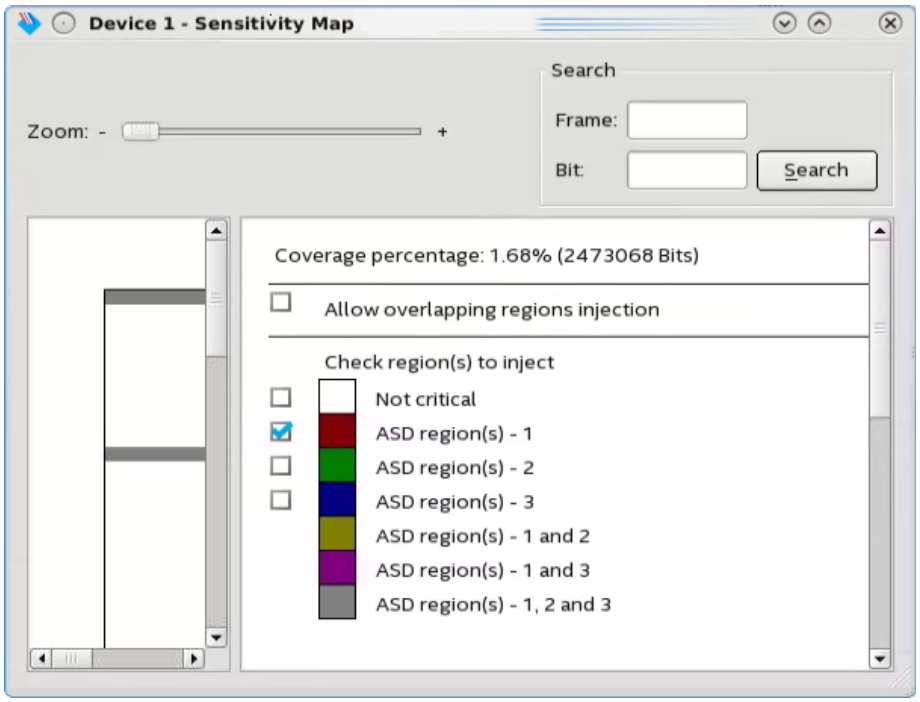
1. Launch **Signal Tap Logic Analyzer** from **Tools** menu.
2. Make sure the **Hardware** and **Device** is selected.

   Your Signal Tap operation cannot be started at this point until the FPGA is configured.

6.3.4. Configuring Arria 10 and Reading .smh File with Fault Injection Debugger

To configure the Arria 10 with Fault Injection Debugger, perform the following steps:

1. Launch **Fault Injection Debugger** from **Tools** menu.
2. Make sure a valid programming cable is selected in **Hardware Setup**.
3. Click **Auto Detect**, the windows should display the detected Arria 10 in the JTAG chain.
4. Select **Arria 10 device**, click **Select File**, select the `top.sof` from the `output_files` folder and click **Open**.
5. Check the **Program/Configure**.
6. Click **Start** to start the configuration operation.
7. Right click the **Arria 10 device**, click **Select SMH file**.
8. Select the `top.smh` from `output_files` folder and click **Open**.
9. Right click the **Arria 10 device**, click **Show Device Sensitivity Map**.
10. Select**ASD region(s) - 1** in the **Sensitivity Map** window as shown in the figure below.

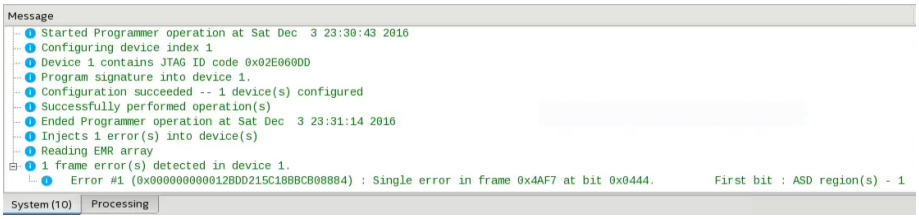


11. Close the **Sensitivity Map** window.

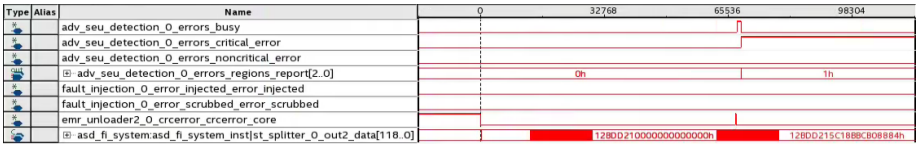
6.3.5. Injecting Error with Fault Injection Debugger

You can now inject the error to the CRAM with the Fault Injection Debugger. Prior to error injection, you must start the Signal Tap to monitor the targeted signals. Perform the following steps:

1. In Signal Tap **Logic Analyzer** window, select the **Signal Tap** instance and click **Run Analysis** in **Processing** menu, or hit F5.
2. Back to the **Fault Injection Debugger** window, check **Inject Fault** and click **Start**.You may see the Quartus Prime System message shows `Injects 1 error(s) into device(s)`.
3. Click **Read EMR**, the System message shows the injected error location as in the figure below.



The Signal Tap Logic Analyzer will read the error as the critical error and reports the affected region as 0x1, this should match to the System message that reports the error located at ASD region 1.



7. Implementing ECC Feature in Arria 10 ROM Design

The ROM IP core does not have ECC selection in the user interface. However, you can enable the ECC feature for ROM design by using the RAM: 2-PORT IP core.

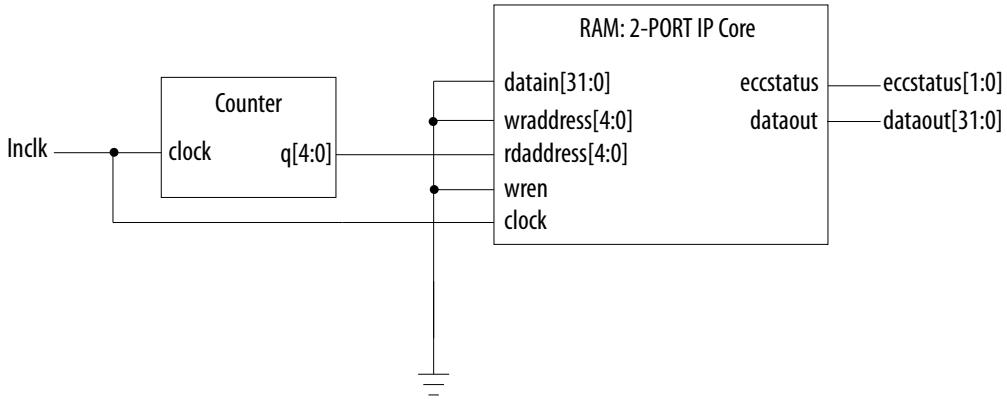
Steps to implement the ECC feature in Arria 10 ROM Design.

1. Instantiate the RAM: 2-PORT IP with the following settings:

Parameters	Settings
Operation Mode	Select **With one read port and one write port**.
Use different data width on different ports	Disable
RAM Block Type	Select **M20K**.
Create byte enable for port A and **Create byte enable for port A**	Disable
Enable Error Correction Checking	Enable
Do you want to specify the initial content of the memory?	Select **Yes, use this file for the memory content data** and specify the location of the file.

2. Connect the signals of the IP according to the following figure.

Figure 13. ROM with ECC Feature Using RAM: 2-PORT IP



7.1. Examples of Error Detection and Correction

The following examples initiate the ROM content using the `.mif` file with the associated address shown in the following table.

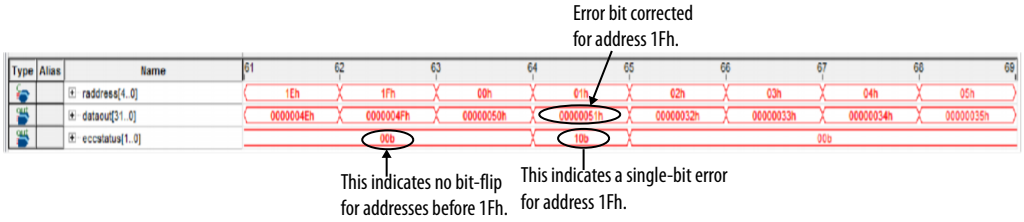
Table 6. Example of ROM Content Initialization

Address	ROM content
00h	32h
01h	33h
02h	34h
: :	: :
1Dh	4Fh
1Eh	50h
1Fh	51h

Single-bit Error

The following figure shows an example of a single-bit error waveform following an SEU event impact on ROM content of address 1Fh. The waveform indicates that there is a two-clock cycle latency on the output with respect to the associated read address. When the ROM content is free from bit-flip, the `eccstatus` signal shows 2b'00. The ROM content of address 1Fh was initialized with data 51h using the `.mif` file as shown in the *Example of ROM Content Initialization* table. The ECC status signal shows 2b'10 indicating a single error bit is detected at the ROM content of address 1Fh. The IP corrects the error at the output.

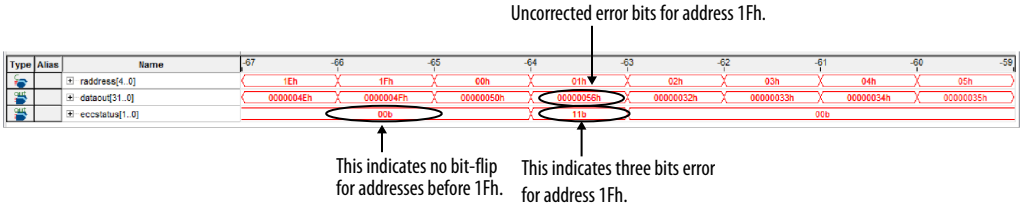
Figure 14. Single Bit Error Waveform



Three Adjacent Bits Error

The following figure shows an example of three adjacent bits error waveform following a multi-bit upset (MBU) event on the ROM content of address 1Fh. The waveform indicates that there is a two-clock cycle latency on the output with respect to the associated read address. The ROM content of address 1Fh was initialized with data 51h using the `.mif` file as shown in the *Example of ROM Content Initialization* table. The ECC status signal shows 2b'11 which indicates 3 adjacent bits error detected at the ROM content of the address 1Fh and uncorrectable data appears at the output.

Figure 15. 3 Adjacent Bits Error Waveform



8. Modifying Single-Device .jam Files for Use in a Multi-Device JTAG Chain

The .jam file codes in this document are meant for a single-device JTAG chain. To use these codes in a multi-device JTAG chain, add instruction register (IR) and data register (DR) lengths of the devices in the chain other than the .jam file's target device.

1. Check the instruction register lengths of all the other devices in the JTAG chain.
   - IR length:
     - Intel FPGA and CPLD devices: 10
     - Hardware processor system (HPS) in Intel SoC FPGA devices: 4
   - DR length in any device: 1
2. Locate the PROCEDURE EXECUTE line in the .jam file codes and add codes in the following steps to new lines after it.
3. If there are devices in the chain before the target device, add the following codes:

```
POSTIR <total IR length before the target device>;
POSTDR <total DR length before the target device>;
```

4. If there are devices in the chain after the target device, add the following codes:

```
PREIR <total IR length after the target device>;
PREDR <total DR length after the target device>;
```

Example 2. Other Devices Exist in JTAG Chain Before or After Target Device

For each example chain, add the codes after the PROCEDURE EXECUTE line:

- Download cable TDI → other device 1 (IR=10) → target device → download cable TDO:

```
POSTIR 10;
POSTDR 1;
```

- Download cable TDI → target device → other device 1 (IR=10) → download cable TDO:

```
PREIR 10;
PREDR 1;
```

- Download cable TDI → target device → other device 1 (IR=10) → other device 2 (IR=10) → download cable TDO:

```
PREIR 20;
PREDR 2;
```

- Download cable TDI → other device 1 (IR=4) → target device → other device 2 (IR=10) → download cable TDO:

```
POSTIR 4;
POSTDR 1;
PREIR 10;
PREDR 1;
```

Related Information

Reading EMR using JTAG Interface on page 16

9. Document Revision History for AN 737: SEU Detection and Recovery in Arria 10 Devices

Document Version	Changes
2024.07.08	Added note in *Error Message Register Width and Description* and *Error Type in EMR*.
2021.10.21	Added section about the Quartus Prime software SEU FIT reports.
2020.04.13	• Updated the *Implementing ECC Feature in Arria 10 ROM Design* chapter to show steps to implement ECC feature using the RAM: 2-PORT Intel FPGA IP. • Remove the *Arria 10 ROM with ECC Reference Design Files* link.
2019.08.09	Added steps to modify `.jam` file for use in a multi-device JTAG chain.
2018.09.04	• Added a note in *System Requirements* stating that a licensed version of Quartus Prime software is required to generate SMH files. • Updated hyperlinks.

Date	Version	Changes
March 2017	2017.03.15	Rebranded as Intel.
February 2017	2017.02.13	• Updated *Timing Diagram for Column-Based Check-Bits* diagram description. • Added note to Case A and B in *Correctable and Uncorrectable Error Cases* table. • Updated device development kit ordering part number. • Added note to *Creating Arria 10 SEU Fault Injection and Hierarchy Tagging Design with Qsys* to state the availability of `a10-seu-complete.zip` design and skipping pregenerated steps. • Updated device selection in *Converting .sof File and .smh File to .jic File*.
October 2016	2016.10.31	• Added *ROM with ECC Reference Design*. • Updated EDCRC reference design target device and reference design file.
March 2016	2016.03.03	Updated CRC_ERROR pin behavior when uncorrectable error cannot be located.
March 2016	2016.03.02	Initial release.