AN 866: Mitigating and Debugging Single Event Upsets in Intel® Quartus® Prime Standard Edition

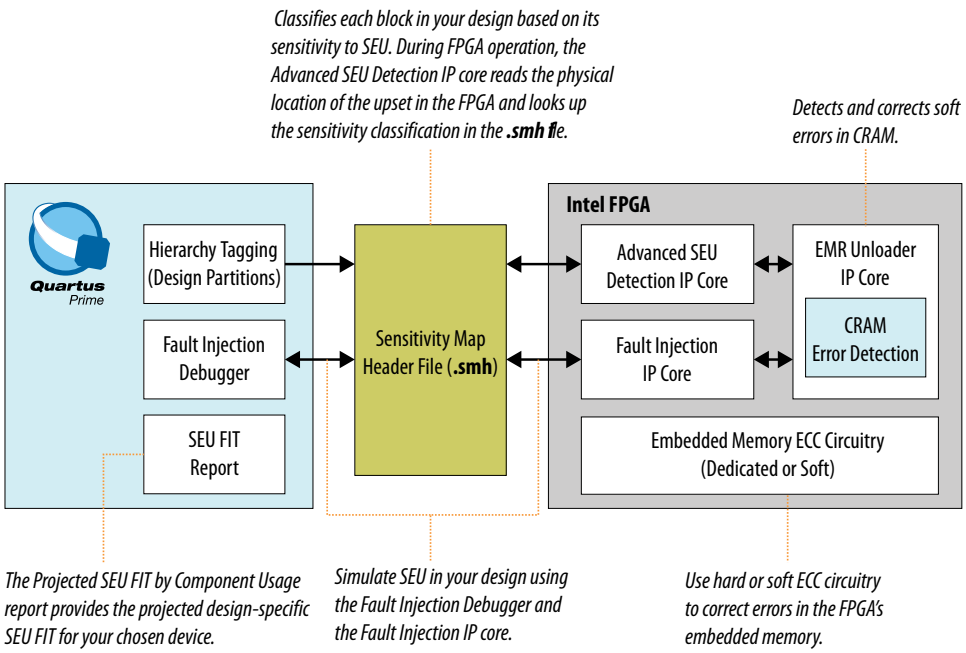
Contents

1. Mitigating Single Event Upset

Single event upsets (SEUs) are rare, unintended changes in the state of an FPGA's internal memory elements caused by cosmic radiation effects. The change in state is a soft error and the FPGA incurs no permanent damage. Because of the unintended memory state, the FPGA may operate erroneously until background scrubbing fixes the upset.

The Intel® Quartus® Prime Standard Edition software offers several features to detect and correct the effects of SEU, or soft errors, as well as to characterize the effects of SEU on your designs. Additionally, some Intel FPGAs contain dedicated circuitry to help detect and correct errors.

Figure 1. Tools, IP, and Circuitry for Detecting and Correcting SEU



Intel FPGAs have memory in user logic (block memory and registers) and in Configuration Random Access Memory (CRAM). The Intel Quartus Prime Programmer loads the CRAM with a `.sof` file. Then, the CRAM configures all FPGA logic and routing. If an SEU strikes a CRAM bit, the effect can be harmless if the device does not use the CRAM bit. However, the effect can be severe if the SEU affects critical logic or internal signal routing.

Often, a design does not require SEU mitigation because of the low chance of occurrence. However, for highly complex systems, such as systems with multiple high-density components, the error rate may be a significant system design factor. If your

system includes multiple FPGAs and requires very high reliability and availability, you should consider the implications of soft errors. Use the techniques in this document to detect and recover from these types of errors.

Related Information

- Introduction to Single Event Upsets
- Intel Stratix 10 SEU Mitigation User Guide
  For information about Intel Stratix® 10 device SEU mitigation in the Intel Quartus Prime Pro Edition software.

1.1. Failure Rates

The Soft Error Rate (SER) or SEU reliability is expressed in Failure in Time (FIT) units. One FIT unit is one soft error occurrence per billion hours of operation.

- For example, a design with 5,000 FIT experiences a mean of 5,000 SEU events in one billion hours (or 114,155.25 years). Because SEU events are statistically independent, FIT is additive. If a single FPGA has 5,000 FIT, then ten FPGAs have 50,000 FIT (or 50K failures in 114,155.25 years).

Another reliability measurement is the mean time to failure (MTTF), which is the reciprocal of the FIT or 1/FIT.

- For a FIT of 5,000 in standard units of failures per billion hours, MTTF is:

  1 ÷ (5,000 ÷ 1 Bh)=1 billion ÷ 5,000 = 200,000 hours = 22.83 years

SEU events follow a Poisson distribution and the cumulative distribution function (CDF) for mean time between failures (MTBF) is an exponential distribution. For more information about failure rate calculation, refer to the *Intel FPGA Reliability Report.*

Neutron SEU incidence varies by altitude, latitude, and other environmental factors. The Intel Quartus Prime software provides SEU FIT reports based on compiles for sea level in Manhattan, New York. The JESD89A specification defines the test parameters.

Tip: You can convert the data to other locations and altitudes using calculators, such as those at www.seutest.com. Additionally, you can adjust the SEU rates in your project by including the relative neutron flux (calculated at www.seutest.com) in your project's `.qsf` file.

Related Information

- SEU FIT Parameters Report on page 11
- JEDEC Standard 89A
- http://seutest.com/
  Soft-error Testing Resources and Calculator
- FPGA Functional Analysis, Quality & Reliability Support

1.2. Mitigating SEU Effects in Embedded User RAM

Stratix® V, Stratix IV, and Arria® V GZ FPGAs offer dedicated error correcting code (ECC) circuitry for embedded memory blocks. FPGA families that do not have dedicated ECC circuitry support ECC by implementing a soft IP core.

You can reduce the FIT rate for these memories to near zero by enabling the ECC encode/decode blocks. On ingress, the ECC encoder adds 8 bits of redundancy to a 32 bit word. On egress, the decoder converts the 40 bit word back to 32 bits. You use the redundant bits to detect and correct errors in the data resulting from SEU.

The existence of hard ECC and the strength of the ECC code (number of corrected and detected bits) varies by device family. Refer to the device handbook for details. If a device does not have a hard ECC block you can add ECC parity or use an ECC IP core.

The SRAM memories associated with processor subsystems, such as for SoC devices, contain dedicated hard ECC. You do not need to take action to protect these memories.

For more information about embedded memories and ECC, refer to the device documentation.

Related Information

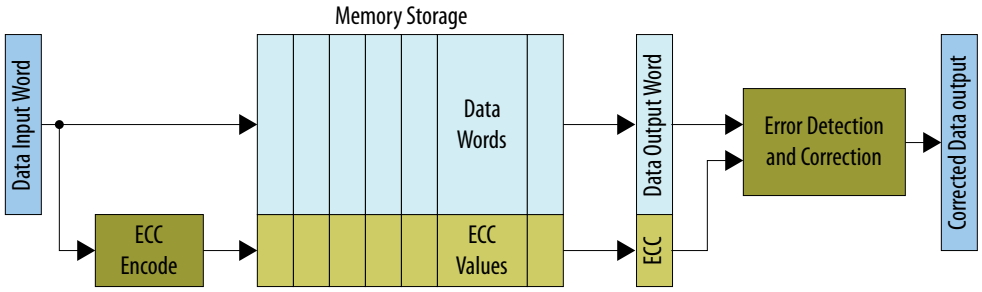
- ECC in Stratix V Memory Blocks
  In *Stratix V Device Handbook Volume 1*
- Stratix IV Embedded Memory Blocks
  In *Stratix IV Device Handbook Volume 1*
- ECC in Arria V Memory Blocks
  In *Arria V Device Handbook Volume 1*

1.2.1. Configuring RAM to Enable ECC

To enable ECC, configure the RAM as a 2-port RAM with independent read and write addresses. Using this feature does not reduce the available logic.

Although the ECC checking function results in some additional output delay, the hard ECC has a much higher $\mathbf{f_{MAX}}$ compared with an equivalent soft ECC block implemented in general logic. Additionally, you can pipeline the hard IP in the M20K block by configuring the ECC-enabled RAM to use an output register at the corrected data output port. This implementation increases performance and adds latency.
For devices without dedicated circuitry, you can implement ECC by instantiating the ALTECC IP core, which performs ECC generation and checking functions.

Figure 2. Memory Storage and ECC



Related Information

ALTECC (Error Correction Code: Encoder/Decoder)

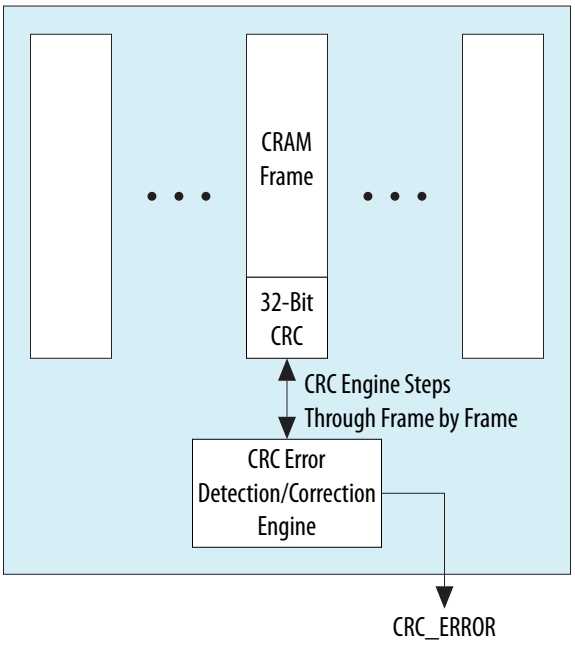
1.3. Mitigating SEU Effects in Configuration RAM

devices contain error detect CRC (EDCRC) hard blocks. These blocks detect and correct soft errors in CRAM, and are similar to those that protect internal user memory.

Intel FPGAs contain frames of CRAM. The size and number of frames is device specific. The device continually checks the CRAM frames for errors by loading each frame into a data register. The EDCRC block checks the frame for errors.

When the FPGA finds a soft error, the FPGA asserts its CRC_ERROR pin. You can monitor this pin in your system. When your system detects that the FPGA asserted this pin during operation, indicating the FPGA detected a soft error in the configuration RAM, the system can take action to recover from the error. For example, the system can perform a soft reset (after waiting for background scrubbing), reprogram the FPGA, or classify the error as benign and ignore it.

Figure 3. CRAM Frame



To enable error detection, point to **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC**, and turn on error detection settings.

Related Information

- Single Level Upsets
- CRAM Error Detection Settings Reference on page 16

1.4. Internal Scrubbing

Intel Arria 10, select Cyclone® V (including SoC devices), and Stratix V FPGAs support automatic CRAM error correction without reloading the original CRAM contents from an external copy of the original `.sof`.

The internal scrubbing feature corrects single-bit and double-adjacent errors automatically.

If the FPGA finds a CRC error in a CRAM frame, the FPGA reconstructs the frame from the error correcting code calculated for that frame. Then the FPGA writes the correct frame into the CRAM.

If you enable internal scrubbing, you must still plan a recovery sequence. Although scrubbing can restore the CRAM array to intended configuration, latency occurs between the soft error detection and correction. During this latency period, the FPGA may operate with errors. If the FPGA must scan a large number of configuration bits, this latency can be up to 100 milliseconds. For more information about latency refer to the device datasheet.

To enable internal scrubbing, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC** and turn on the **Enable internal scrubbing** option.

Related Information

- Error Detection CRC Page
  In *Intel Quartus Prime Help*
- SEU Recovery on page 7
- CRAM Error Detection Settings Reference on page 16

1.5. SEU Recovery

After correcting a CRAM bit flip, the FPGA is in its original configuration with respect to logic and routing. However, the FPGA may have an illegal internal state, for example, if the SEU error affects the function or operation of the circuit, resulting in erroneous output.

Errors due to faulty operation can propagate elsewhere within the FPGA or to the system outside the FPGA. During your design process, determine the possible SEU outcomes and design a recovery response that considers resetting the FPGA to a known state.

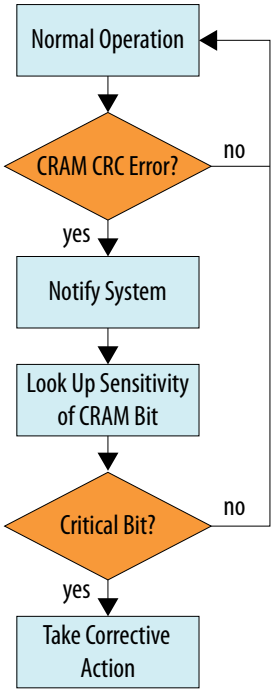
1.5.1. Planning for SEU Recovery

Reconfiguring a running FPGA typically has a significant system impact. When planning for SEU recovery, you must account for the time required to bring the FPGA to a state consistent with the current state of the system. For example, an internal state machine that is in an illegal state may require reset. Also, the surrounding logic may need to account for this unexpected operation.

Often, an SEU impacts CRAM bits that the implemented design does not use (for example, CRAM bits that control unused logic and routing wires). Depending on the implementation, FPGAs with high utilization only use about 40% of available CRAM bits. Therefore, only 40% of potential SEU events in the entire FPGA require intervention, and you can ignore the remaining 60%. Designs that do not completely fill the FPGA use even fewer available CRAM bits.

You can determine which portions of the implemented design are not critical to the FPGA's function. Examples include test circuitry that is not important to the FPGA operation, or other non-critical functions that the system can log but does not need to reprogram or reset.

Figure 4. Sensitivity Processing Flow



Related Information

- AN 737: SEU Detection and Recovery in Intel Arria 10 Devices
- Understanding Single Event Functional Interrupts in FPGA Designs White Paper

1.5.2. Designating the Sensitivity of the Design Hierarchy

In the Intel Quartus Prime software, you indicate the criticality of each logic block by generating partitions, and assigning a sensitivity ID tag to each partition. The Intel Quartus Prime software stores this information in a Sensitivity Map Header File (`.smh`).

When an error occurs during system operation, the system determines the impact of the error by looking up the classification in the `.smh` file. The system can then take corrective action based on the classification.

Note: You must have a licensed version of Intel Quartus Prime software to generate `.smh` files.

To access the `.smh` file, you must add an instance of the IP core to your design.

Related Information

Advanced SEU Detection IP Core User Guide

1.5.2.1. Hierarchy Tagging

The Intel Quartus Prime hierarchy tagging feature allows you to improve design-effective FIT rate by tagging only the critical logic for device operation.

You can also define the system recovery procedure based on knowledge of logic impaired by SEU. This technique reduces downtime for the FPGA and the system in which the FPGA resides. Other advantages of hierarchy tagging are:

- Increases system stability by avoiding disruptive recovery procedures for inconsequential errors.
- Allows diverse corrective action for different design logic.

The `.smh` file contains a mask for design sensitive bits in a compressed format. The Intel Quartus Prime software generates the sensitivity mask for the entire design. The Intel Arria 10, Cyclone V, and Stratix V device families support hierarchy tagging.

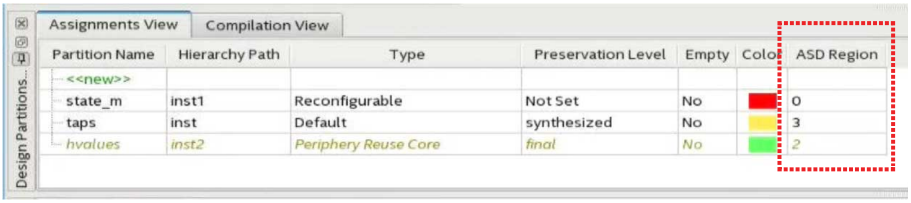
Related Information

SEU Mitigation in Intel FPGA Devices: Hierarchy Tagging
Online Course

1.5.2.2. Using Partitions to Specify Logic Sensitivity ID

1. In the Intel Quartus Prime software, designate a design block as a design partition.
2. Specify the sensitivity ID assigned to the partition in the **ASD Region** column in the **Design Partitions** window.

Figure 5. ASD Region Column in the Design Partitions Window



Assign the partition a numeric sensitivity value from 0 to 16. The value represents the sensitivity tag associated with the partition.

- A sensitivity tag of 1 is the same as no assignment, and indicates a basic sensitivity level, which is "region used in design". If a soft error occurs in this partition, the IP core reports the error as a critical error in the sensitivity region 1.
- A sensitivity tag of 0 is reserved, and indicates unused CRAM bits. You can explicitly set a partition to 0 to indicate that the partition is not critical. This setting excludes the partition from sensitivity mapping.

Note: You can use the same sensitivity tag for multiple design partitions.

Alternatively, use the following assignment:

```
set_global_assignment -name PARTITION_ASD_REGION_ID <asd_id> -section_id
<partition_name>
```

1.5.3. Advanced SEU Detection IP Core

To correct and detect SEU in the FPGA CRAM, you must instantiate the Advanced SEU Detection IP core. When the FPGA's EDCRC detects an SEU, the Advanced SEU Detection IP core looks up the sensitivity of the affected bit in the `.smh` file.

- During system operation, the Advanced SEU Detection IP core reads the FPGA's error message register (EMR) to determine the location of the error.
- The IP core finds the upset location in the `.smh` file.
- The IP core returns whether or not the bit is critical for the design.

You can implement either an on-chip or external sensitivity processor:

- *On-chip sensitivity processor*: the IP core looks up the bit sensitivity in the `.smh` with a user-supplied memory interface.
- *External sensitivity processor*: the IP core notifies external logic (typically via a system CPU interrupt request), and provides cached event message register values to the off-chip sensitivity processor. The external sensitivity processor's memory system stores the `.smh` information.

The *Advanced SEU Detection IP Core User Guide* provides instructions for incorporating the IP core into your design, and describes how to access the `.smh` file.

Related Information

Advanced SEU Detection IP Core User Guide

1.5.3.1. On-Chip Sensitivity Processor

When you implement an on-chip sensitivity processor, the Advanced SEU Detection IP core interacts with user-supplied external memory access logic to read the `.smh` stored in external memory. Once it determines the sensitivity of the affected CRAM bit, the IP core can assert a critical error signal so that the system provides an appropriate response. If the SEU is not critical, the critical error signal may be left un-asserted.

On-chip sensitivity processing is autonomous: the FPGA determines whether an SEU affected it without using external logic. On-chip sensitivity processing requires some FPGA logic resources for the external memory interface.

1.5.3.2. External Sensitivity Processor

When you implement an external sensitivity processor, a CPU (such as the ARM processor in Intel SoC devices) receives an interrupt request when the FPGA detects an SEU. The CPU then reads the FPGA's error message register and looks up the bit sensitivity in the `.smh` stored in the CPU's memory space.

With external sensitivity processing, the FPGA does not need to implement an external memory interface or store the `.smh`. If the system already has a CPU, external sensitivity processing may be more hardware efficient than on-chip processing.

1.6. Intel Quartus Prime Software SEU FIT Reports

The Intel 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.

Note: Intel Arria 10, Arria V, Arria V SoC, Cyclone IV, Cyclone V, Cyclone V SoC, Intel MAX® 10, and Stratix V FPGAs support the Projected SEU FIT by Component Usage report.

1.6.1. SEU FIT Parameters Report

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

Figure 6. SEU FIT Parameters

SEU FIT Parameters

	Parameter	value
1	Device	5SGXEA7N2F45I2
2	Altitude	0.00
3	Neutron Flux	JESD - 89A assuming sea - level(> 10 MeV) 13.00 n / hr / cm2
4	Neutron Flux Multiplier	1.000
5	Alpha Flux	0.001 CPH/cm^2

Change the Neutron Flux Multiplier using the assignment:
set_global_assignment -name RELATIVE_NEUTRON_FLUX <relative_flux>

- **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 Intel Quartus Prime software only scales the neutron component of FIT. Location does not affect the Alpha flux.

Related Information

http://seutest.com/
Soft-error Testing Resources and Calculator

1.6.2. 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 7. Projected SEU FIT by Component Usage Report

Projected SEU FIT by Component Usage

	Component	Raw	Utilized	w/ECC	AVF 0.5	AVF 0.25
1	Configuration (CRAM)	8486	3817	3817	1909	955
1	--Logic	2125	1071	1071	536	268
2	--Routing	6314	2716	2716	1358	679
3	--I/O config	47	30	30	15	8
2	RAM	41696	8593	1218	609	304
1	--HARD-IP (E.G. PCIe)	1446	692	0	0	0
2	--Embedded RAM	40250	7901	1218	609	304
3	--MLAB (LUTRAM)	0	0	0	0	0
3	Registers	2563	794	794	396	198
1	--Hard-IP FF	474	177	177	88	44
2	--DSP/M20K FF	298	61	61	30	15
3	--Design FF	1791	556	556	278	139
4						
5	TOTAL	52745	13204	5829	2914	1457

1.6.2.1. 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)

1.6.2.2. Raw FIT

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

The Intel 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 Intel Quartus Prime software counts device bits for target devices using different parameter information than the Reliability Report. Therefore, the Intel 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

1.6.2.3. 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 on page 8

1.6.2.3.1. 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.

1.6.2.3.2. 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.

1.6.2.4. 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 Intel 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 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.

Note: In Cyclone V devices, the M10K block does not have ECC, therefore, the **w/ECC** column does not apply.

An MLAB is simply a LAB configured with writable CRAM. However, when the Intel Quartus Prime software configures the RAM as write enabled (MLAB), the MLAB has a slightly different FIT/Mb. The Projected SEU FIT by Component Usage report displays a FIT rate in the MLAB row when the design uses MLABs, otherwise the report accounts for the block's FIT in the CRAM row. During compilation, if the Intel Quartus Prime software changes a LAB to an MLAB, the FIT accounting moves from the LAB row to the MLAB row.

The **w/ECC** column does not account for other forms of FIT protection in the design, such as designer-inserted parity, soft ECC blocks, bounds checking, system monitors, triple-module redundancy, or the impact of higher-level protocols on general fault tolerance. Additionally, it does not account for single event effects that occur in the logic but the design never reads or notices. For example, if you implement a non-ECC FIFO function 512 bits deep and an SEU event occurs outside of the front and back pointers, the application does not observe the SEU event. However, the report accounts for the full 512 bit deep memory and includes it in the **w/ECC** FIT rate. Designers often combine these factors into general deflation factors (called architectural vulnerability factors or AVF) based on knowledge of their design. Designers use AVF factors as low (aggressive) as 5% and as high (conservative) as 50% based on experience, fault-injection or neutron beam testing, or high-level system monitors.

1.6.2.5. 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 White Paper

1.6.3. Enabling the Projected SEU FIT by Component Usage Report

The Intel Quartus Prime Fitter generates the Projected SEU FIT by Component Usage report. The Intel 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
```

1.7. Triple-Module Redundancy

Use Triple-Module Redundancy (TMR) if your system cannot suffer downtime due to SEU. TMR is an established SEU mitigation technique for improving hardware fault tolerance. A TMR design has three identical instances of hardware with voting hardware at the output. If an SEU affects one of the hardware instances, the voting logic notes the majority output. This operation masks malfunctioning hardware.

With TMR, your design does not suffer downtime in the case of a single SEU; if the system detects a faulty module, the system can scrub the error by reprogramming the module. The error detection and correction time is many orders of magnitude less than the MTBF of SEU events. Therefore, the system can repair a soft interrupt before another SEU affects another instance in the TMR application.

The disadvantage of TMR is its hardware resource cost: it requires three times as much hardware in addition to voting logic. You can minimize this hardware cost by implementing TMR for only the most critical parts of your design.

There are several automated ways to generate TMR designs by automatically replicating designated functions and synthesizing the required voting logic. Synopsys offers automated TMR synthesis.

1.8. Evaluating a System's Response to Functional Upsets

SEUs can strike any memory element, so you must test the system to ensure a comprehensive recovery response. The Intel Quartus Prime software includes the Fault Injection Debugger to aid in SEU recovery. You can use the Fault Injection Debugger graphically with the GUI, or you can use command line assignments.

The Fault Injection Debugger works together with the Fault Injection IP core. To use the debugging feature you must instantiate the Fault Injection IP core in the FPGA design. During debugging, the IP core flips a CRAM bit by dynamically reconfiguring the frame containing the CRAM bit.

Related Information

- Intel FPGA Fault Injection IP Core User Guide
- Debugging Single Event Upset Using the Fault Injection Debugger on page 18

1.9. CRAM Error Detection Settings Reference

To define these settings in the Intel Quartus Prime software, point to **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC**.

Figure 8. Device and Pin Options Error Detection CRC Tab

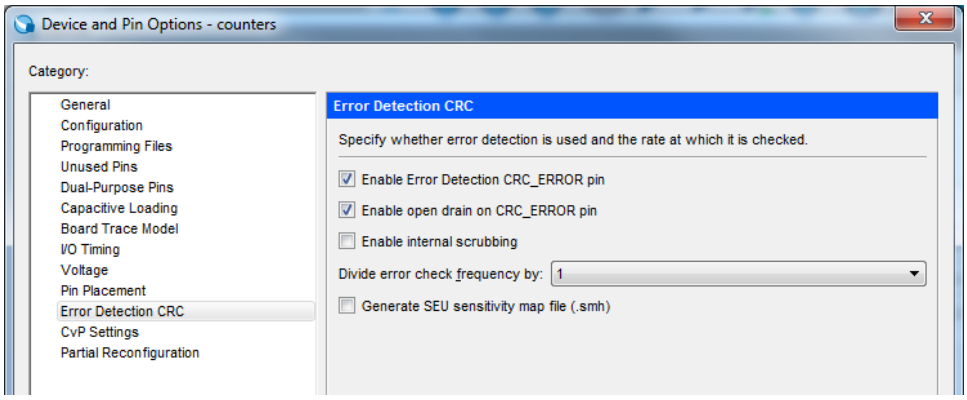


Table 1. CRC Errors Settings

Setting	Description
Enable Error Detection CRC_ERROR pin	Enables CRAM frame scanning
Enable open drain on CRC_ERROR pin	Enables the CRC_ERROR pin as an open-drain output
Divide error check frequency by	To guarantee the availability of a clock, the EDCRC function operates on an independent clock generated internally on the FPGA itself. To enable EDCRC operation on a divided version of the clock, select a value from the list.

1.10. Document Revision History

Table 2. Document Revision History

Date	Version	Changes
2021.09.28	17.1	Revised text and image in SEU Fit Parameters Report for incorrect assignment syntax.
2021.07.07	17.1.0	Revised note regarding EDCRC max FIT rate based on the Reliability Report data.
2021.01.05	17.1.0	• Removed incorrectly included Intel Stratix 10 topics and references. These topics and references do not apply to the Intel Quartus Prime Standard Edition software. • Added link to *Intel Stratix 10 SEU Mitigation User Guide*.
2019.09.10	17.1.0	Updated the topic about failure rates to correct the number of years of one billion hours.
2017.12.15	17.1.0	• Separated description of the SEU FIT Parameters Report and the Projected SEU FIT by Component Usage Report.
2017.11.06	17.1.0	• Added topic: CRAM Error Detection Settings Reference. • Removed topic: Scanning CRAM Frames.

continued...

Date	Version	Changes
2016.10.31	16.1.0	• Removed incorrect link to Reliability Report. • Added MAX 10 and Cyclone IV to list of devices supporting Projected SEU FIT by Component Usage Report.
2016.05.24	16.0.1	• Corrected the steps to enable the SEU FIT reports.
2016.05.03	16.0.0	• Documented the new SEU FIT reports. • Inconsequential wording changes for conformance to style.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	2014.06.30	• Updated formatting. • Added "Mitigating SEU Effects in Embedded User RAM" section. • Added "Altera Advanced SEU Detection IP Core" section.
November 2012	2012.11.01	• Preliminary release.

2. Debugging Single Event Upset Using the Fault Injection Debugger

You can detect and debug single event upset (SEU) using the Fault Injection Debugger in the Intel Quartus Prime software. Use the debugger with the Intel FPGA Fault Injection IP core to inject errors into the configuration RAM (CRAM) of an Intel FPGA device.

The injected error simulates the soft errors that can occur during normal operation due to SEUs. Since SEUs are rare events, and therefore difficult to test, you can use the Fault Injection Debugger to induce intentional errors in the FPGA to test the system's response to these errors.

The Fault Injection Debugger is available for Intel Arria 10 and Stratix V family devices. For assistance with support for Arria V or Cyclone V family devices, file a service request using your My Intel account.

The Fault Injection Debugger provides the following benefits:

- Allows you to evaluate system response for mitigating single event functional interrupts (SEFI).
- Allows you to perform SEFI characterization, eliminating the need for entire system beam testing. Instead, you can limit the beam testing to failures in time (FIT)/Mb measurement at the device level.
- Scale FIT rates according to the SEFI characterization that is relevant to your design architecture. You can randomly distribute fault injections throughout the entire device, or constrain them to specific functional areas to speed up testing.
- Optimize your design to reduce SEU-caused disruption.

Related Information

- myAltera Log In
- Single Event Upsets
- AN 737: SEU Detection and Recovery in Intel Arria 10 Devices
- Understanding Single Event Functional Interrupts in FPGA Designs White Paper

2.1. Single Event Upset Mitigation

Integrated circuits and programmable logic devices such as FPGAs are susceptible to SEUs. SEUs are random, nondestructive events, caused by two major sources: alpha particles and neutrons from cosmic rays. Radiation can cause either the logic register, embedded memory bit, or a configuration RAM (CRAM) bit to flip its state, thus leading to unexpected device operation.

Intel Arria 10, Arria V, Cyclone V, Stratix V and newer devices have the following CRAM capabilities:

- Error Detection Cyclical Redundance Checking (EDCRC)
- Automatic correction of an upset CRAM (scrubbing)
- Ability to create an upset CRAM condition (fault injection)

For more information about SEU mitigation in Intel FPGA devices, refer to the *SEU Mitigation* chapter in the respective device handbook.

Related Information

- SEU Mitigation in Intel Arria 10 Devices: Hierarchy Tagging Online Course
- Mitigating Single Event Upsets in Intel Arria 10 Devices Online Course

2.2. Hardware and Software Requirements

The following hardware and software is required to use the Fault Injection Debugger:

- FEATURE line in your Intel FPGA license that enables the IP core. For more information, contact your local Intel FPGA sales representative.
- Download cable (Intel FPGA Download Cable, Intel FPGA Download Cable II, Intel FPGA Ethernet Cable, or Intel FPGA Ethernet Cable II).
- Intel FPGA development kit or user designed board with a JTAG connection to the device under test.
- (Optional) FEATURE line in your Intel FPGA license that enables the Advanced SEU Detection IP core.

2.3. Using the Fault Injection Debugger and IP Core

The Fault Injection Debugger works together with the IP core. First, you instantiate the IP core in your design, compile, and download the resulting configuration file into your device. Then, you run the Fault Injection Debugger from within the Intel Quartus Prime software or from the command line to simulate soft errors.

- The Fault Injection Debugger allows you to operate fault injection experiments interactively or by batch commands, and allows you to specify the logical areas in your design for fault injections.
- The command-line interface is useful for running the debugger via a script.

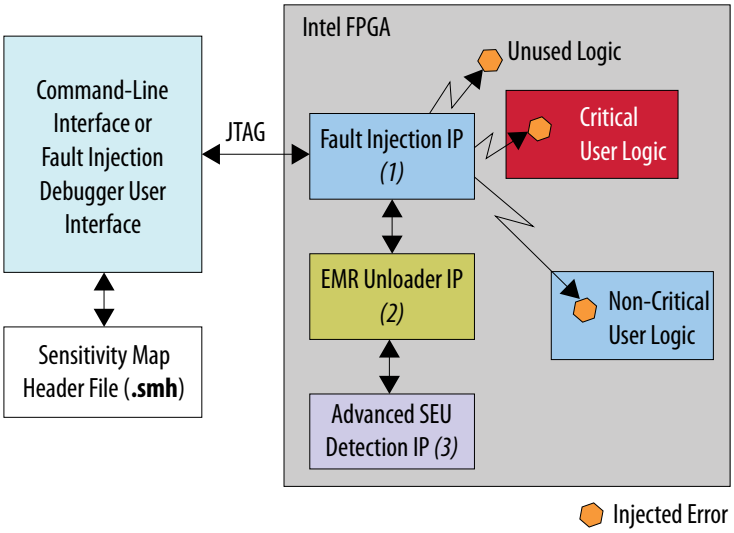
The Fault Injection Debugger communicates with the IP core via the JTAG interface. The IP accepts commands from the JTAG interface and reports status back through the JTAG interface.

Note: The IP core is implemented in soft logic in your device; therefore, you must account for this logic usage in your design. One methodology is to characterize your design's response to SEU in the lab and then omit the IP core from your final deployed design.

You use the IP core with the following IP cores:

- The Error Message Register Unloader IP core, which reads and stores data from the hardened error detection circuitry in Intel FPGA devices.
- (Optional) The Advanced SEU Detection Intel FPGA IP core, which compares single-bit error locations to a sensitivity map during device operation to determine whether a soft error affects it.

Figure 9. Fault Injection Debugger Overview Block Diagram



Notes:
1. The Fault Injection IP flips the bits of the targeted logic.
2. The Fault Injection Debugger and Advanced SEU Detection IP use the same EMR Unloader instance.
3. The Advanced SEU Detection IP core is optional.

Related Information

- About SMH Files on page 23
- AN 539: Test Methodology or Error Detection and Recovery using CRC in Intel FPGA Devices
- Instantiating the IP Core on page 20
- About the EMR Unloader IP Core on page 21
- About the Advanced SEU Detection IP Core on page 22

2.3.1. Instantiating the IP Core

The IP core does not require you to set any parameters. To use the IP core, create a new IP instance, include it in your Platform Designer (Standard) system, and connect the signals as appropriate.

Note: You must use the IP core with the EMR Unloader IP core.

The and the EMR Unloader IP cores are available in Platform Designer and the IP Catalog. Optionally, you can instantiate them directly into your RTL design, using Verilog HDL, SystemVerilog, or VHDL.

Related Information

Intel FPGA Fault Injection IP Core User Guide

2.3.1.1. About the EMR Unloader IP Core

The EMR Unloader IP core provides an interface to the EMR, which is updated continuously by the device's EDCRC that checks the device's CRAM bits CRC for soft errors.

Figure 10. Example Platform Designer System Including the IP Core and EMR Unloader IP Core

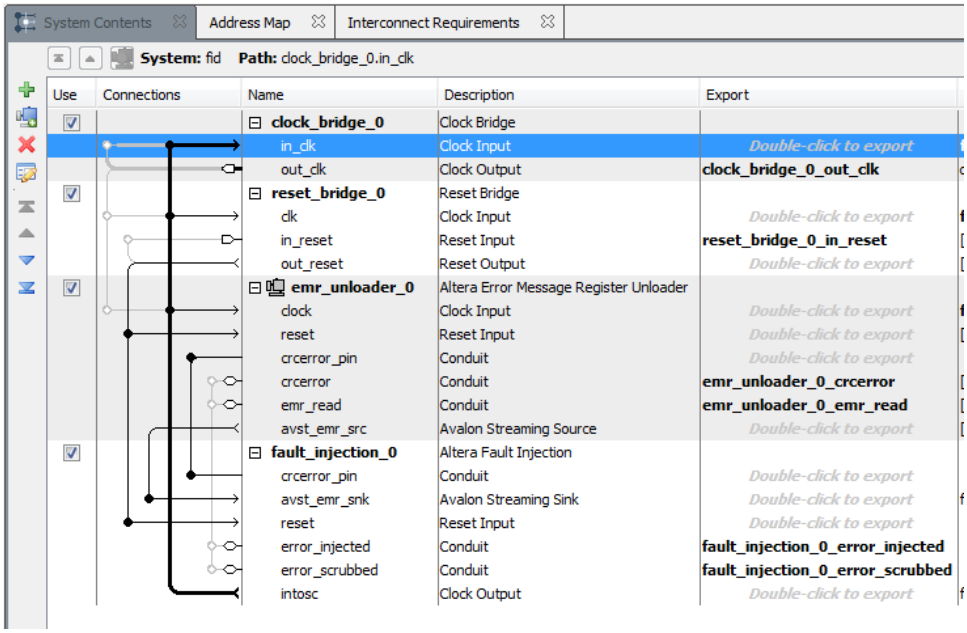
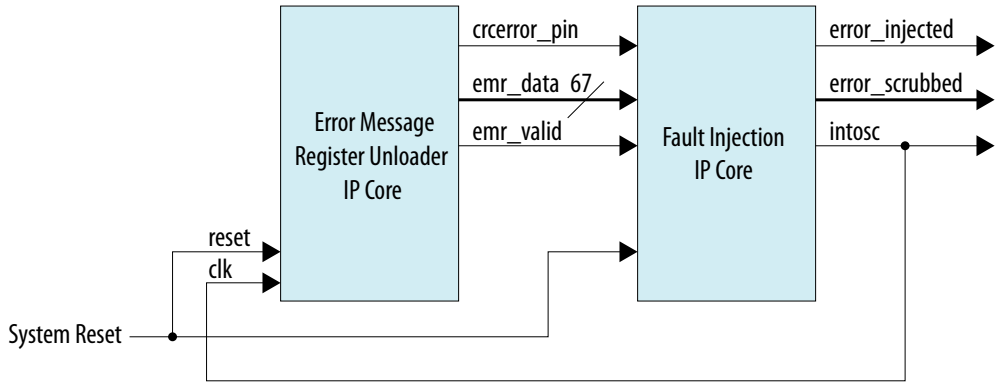


Figure 11. Example IP Core and EMR Unloader IP Core Block Diagram



Related Information

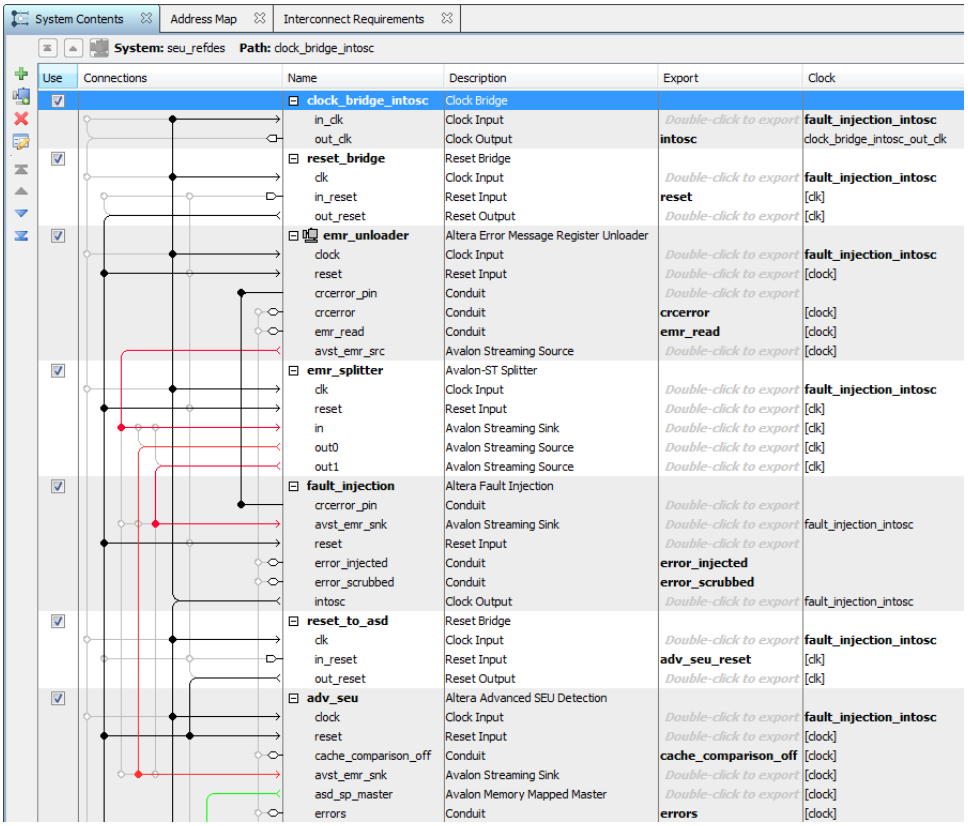
Intel FPGA Error Message Unloader IP Core User Guide

2.3.1.2. About the Advanced SEU Detection IP Core

Use the Advanced SEU Detection (ASD) IP core when SEU tolerance is a design concern.

You must use the EMR Unloader IP core with the ASD IP core. Therefore, if you use the ASD IP and the IP in the same design, they must share the EMR Unloader output via an Avalon®-ST splitter component. The following figure shows a Platform Designer system in which an Avalon-ST splitter distributes the EMR contents to the ASD and IP cores.

Figure 12. Using the ASD and IP in the Same Platform Designer System



Related Information

Intel FPGA Advanced SEU Detection IP Core User Guide

2.3.2. Defining Fault Injection Areas

You can define specific regions of the FPGA for fault injection using a Sensitivity Map Header (**.smh**) file.

The SMH file stores the coordinates of the device CRAM bits, their assigned region (ASD Region), and criticality. During the design process you use hierarchy tagging to create the region. Then, during compilation, the Intel Quartus Prime Assembler generates the SMH file. The Fault Injection Debugger limits error injections to specific device regions you define in the SMH file.

2.3.2.1. Performing Hierarchy Tagging

You define the FPGA regions for testing by assigning an ASD Region to the location. You can specify an ASD Region value for any portion of your design hierarchy using the Design Partitions Window.

1. Choose **Assignments ➤ Design Partitions Window**.
2. Right-click anywhere in the header row and turn on **ASD Region** to display the **ASD Region** column (if it is not already displayed).
3. Enter a value from 0 to 16 for any partition to assign it to a specific ASD Region.
   - — ASD region 0 is reserved to unused portions of the device. You can assign a partition to this region to specify it as non-critical..
   - — ASD region 1 is the default region. All used portions of the device are assigned to this region unless you explicitly change the ASD Region assignment.

2.3.2.2. About SMH Files

The SMH file contains the following information:

- If you are not using hierarchy tagging (i.e., the design has no explicit ASD Region assignments in the design hierarchy), the SMH file lists every CRAM bit and indicates whether it is sensitive for the design.
- If you have performed hierarchy tagging and changed default ASD Region assignments, the SMH file lists every CRAM bit and it's assigned ASD region.

The Fault Injection Debugger can limit injections to one or more specified regions.

Note: To direct the Assembler to generate an SMH file:

- Choose **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC**.
- Turn on the **Generate SEU sensitivity map file (.smh)** option.

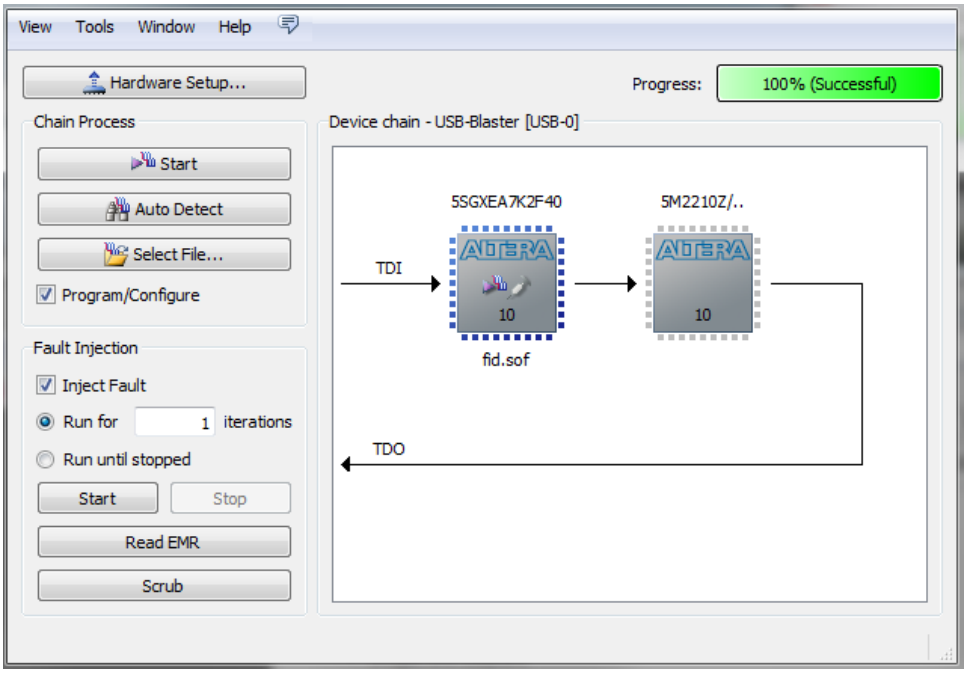
2.3.3. Using the Fault Injection Debugger

To use the Fault Injection Debugger, you connect to your device via the JTAG interface. Then, configure the device and perform fault injection.

To launch the Fault Injection Debugger, choose **Tools ➤ Fault Injection Debugger** in the Intel Quartus Prime software.

Note: Configuring or programming the device is similar to the procedure used for the Programmer or Signal Tap Logic Analyzer.

Figure 13. Fault Injection Debugger



To configure your JTAG chain:

1. Click **Hardware Setup**. The tool displays the programming hardware connected to your computer.
2. Select the programming hardware you wish to use.
3. Click **Close**.
4. Click **Auto Detect**, which populates the device chain with the programmable devices found in the JTAG chain.

Related Information

Targeted Fault Injection Feature on page 30

2.3.3.1. Configuring Your Device and the Fault Injection Debugger

The Fault Injection Debugger uses a **.sof** and (optionally) a Sensitivity Map Header (**.smh**) file.

The Software Object File (**.sof**) configures the FPGA. The **.smh** file defines the sensitivity of the CRAM bits in the device. If you do not provide an **.smh** file, the Fault Injection Debugger injects faults randomly throughout the CRAM bits.

To specify a **.sof**:

1. Select the FPGA you wish to configure in the **Device chain** box.
2. Click **Select File**.
3. Navigate to the **.sof** and click **OK**. The Fault Injection Debugger reads the **.sof**.
4. (Optional) Select the SMH file.

If you do not specify an SMH file, the Fault Injection Debugger injects faults randomly across the entire device. If you specify an SMH file, you can restrict injections to the used areas of your device.

a. Right-click the device in the **Device chain** box and then click **Select SMH File**.

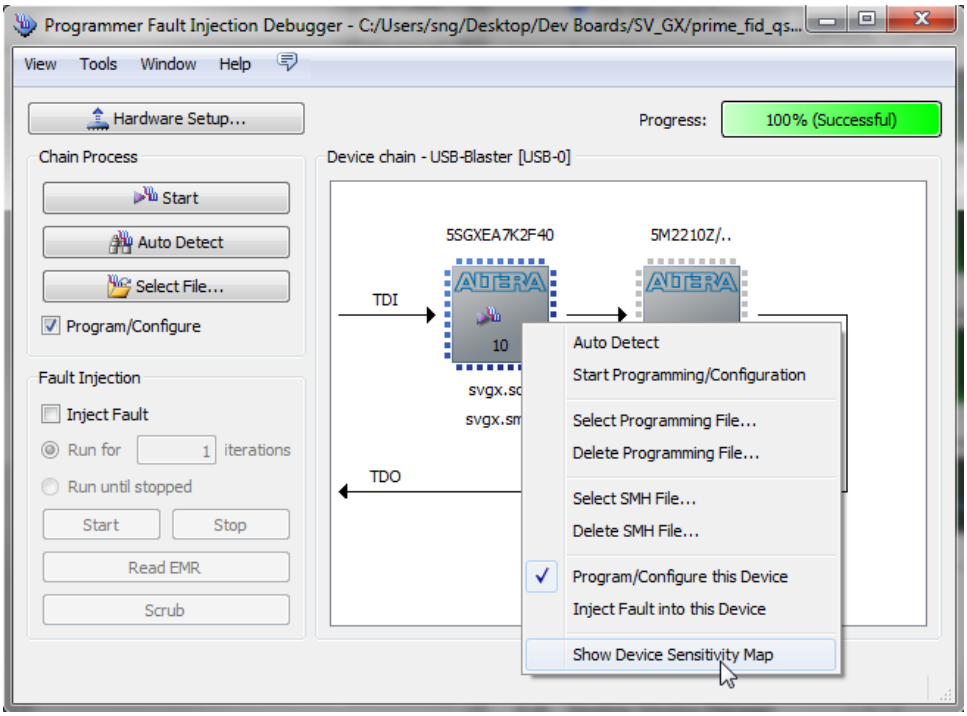
b. Select your SMH file.

c. Click **OK**.

5. Turn on **Program/Configure**.
6. Click **Start**.

The Fault Injection Debugger configures the device using the **.sof**.

Figure 14. Context Menu for Selecting the SMH File



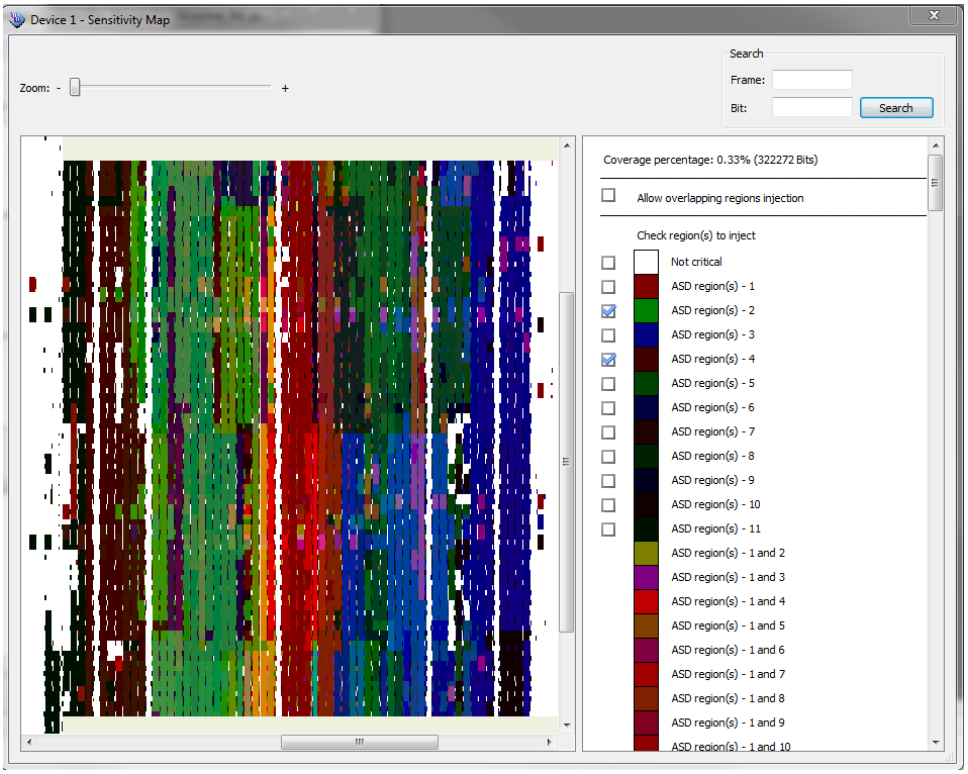
2.3.3.2. Constraining Regions for Fault Injection

After loading an SMH file, you can direct the Fault Injection Debugger to operate on only specific ASD regions.

To specify the ASD region(s) in which to inject faults:

1. Right-click the FPGA in the **Device chain** box, and click **Show Device Sensitivity Map**.
2. Select the ASD region(s) for fault injection.

Figure 15. Device Sensitivity Map Viewer



2.3.3.3. Specifying Error Types

You can specify various types of errors for injection.

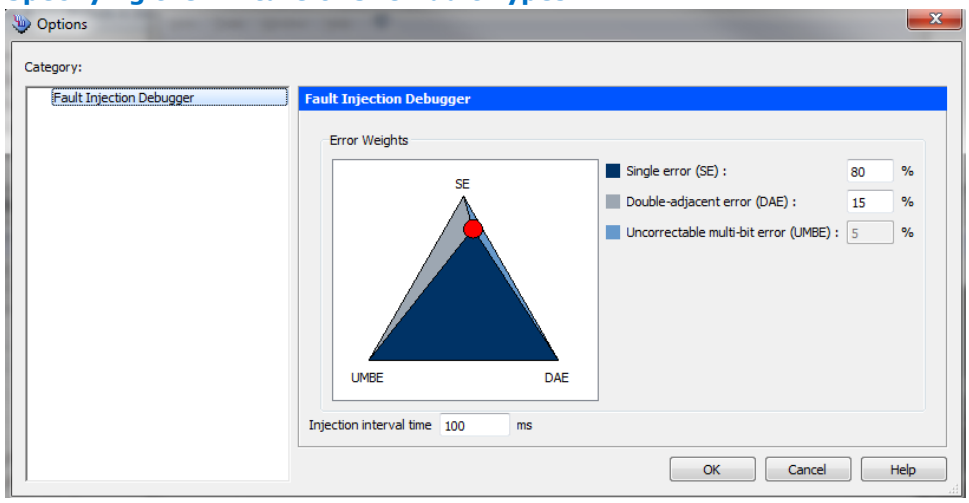
- Single errors (SE)
- Double-adjacent errors (DAE)
- Uncorrectable multi-bit errors (EMBE)

Intel FPGA devices can self-correct single and double-adjacent errors if the scrubbing feature is enabled. Intel FPGA devices cannot correct multi-bit errors. Refer to the chapter on mitigating SEUs for more information about debugging these errors.

You can specify the mixture of faults to inject and the injection time interval. To specify the injection time interval:

1. In the Fault Injection Debugger, choose **Tools ➤ Options**.
2. Drag the red controller to the mix of errors. Alternatively, you can specify the mix numerically.
3. Specify the **Injection interval time**.
4. Click **OK**.

Figure 16. Specifying the Mixture of SEU Fault Types



Related Information

Mitigating Single Event Upset

2.3.3.4. Injecting Errors

You can inject errors in several modes:

- Inject one error on command
- Inject multiple errors on command
- Inject errors until commanded to stop

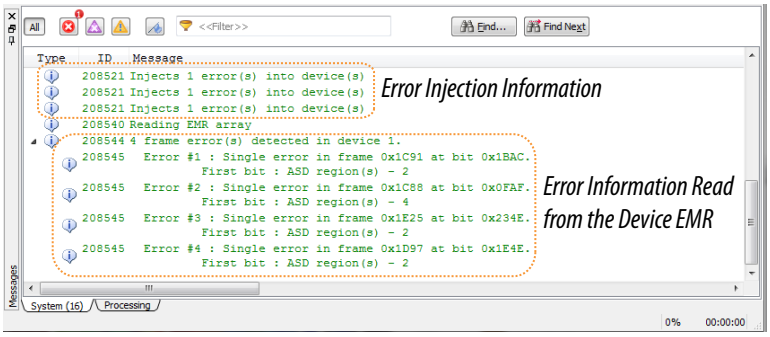
To inject these faults:

1. Turn on the **Inject Fault** option.
2. Choose whether you want to run error injection for a number of iterations or until stopped:
   - If you choose to run until stopped, the Fault Injection Debugger injects errors at the interval specified in the **Tools ➤ Options** dialog box.
   - If you want to run error injection for a specific number of iterations, enter the number.
3. Click **Start**.

   Note: The Fault Injection Debugger runs for the specified number of iterations or until stopped.

The Intel Quartus Prime Messages window shows messages about the errors that are injected. For additional information on the injected faults, click **Read EMR**. The Fault Injection Debugger reads the device's EMR and displays the contents in the Messages window.

Figure 17. Intel Quartus Prime Error Injection and EMR Content Messages



2.3.3.5. Recording Errors

You can record the location of any injected fault by noting the parameters reported in the Intel Quartus Prime Messages window.

If, for example, an injected fault results in behavior you would like to replay, you can target that location for injection. You perform targeted injection using the Fault Injection Debugger command line interface.

2.3.3.6. Clearing Injected Errors

To restore the normal function of the FPGA, click **Scrub**. When you scrub an error, the device's EDCRC functions are used to correct the errors. The scrub mechanism is similar to that used during device operation.

2.3.4. Command-Line Interface

You can run the Fault Injection Debugger at the command line with the **quartus_fid** executable, which is useful if you want to perform fault injection from a script.

Table 3. Command line Arguments for Fault Injection

Short Argument	Long Argument	Description
c	cable	Specify programming hardware or cable. (Required)
i	index	Specify the active device to inject fault. (Required)
n	number	Specify the number of errors to inject. The default value is 1. (Optional)
t	time	Interval time between injections. (Optional)

Note: Use `quartus_fid --help` to view all available options.

The following code provides examples using the Fault Injection Debugger command-line interface.

```
##############################################
#
# Find out which USB cables are available for this instance
# The result shows that one cable is available, named "USB-Blaster"
#
$ quartus_fid --list
    . . .
     Info: Command: quartus_fid --list
```

```
    1) USB-Blaster on sj-sng-z4 [USB-0]
    Info: Intel Quartus Prime 64-Bit Fault Injection Debugger was successful.
     0 errors, 0 warning
##############################################
#
# Find which devices are available on USB-Blaster cable
# The result shows two devices: a Stratix V A7, and a MAX V CPLD.
#
$ quartus_fid --cable USB-Blaster -a
    Info: Command: quartus_fid --cable=USB-Blaster -a
    Info (208809): Using programming cable "USB-Blaster on sj-sng-z4 [USB-0]"
    1) USB-Blaster on sj-sng-z4 [USB-0]
      029030DD   5SGXEA7H(1|2|3)/5SGXEA7K1/..
      020A40DD   5M2210Z/EPM2210
    Info: Intel Quartus Prime 64-Bit Fault Injection Debugger was successful.
    0 errors, 0 warnings

##############################################
#
# Program the Stratix V device
# The --index option specifies operations performed on a connected device.
#   "=svgx.sof" associates a .sof file with the device
#   "#p" means program the device
#
$ quartus_fid --cable USB-Blaster --index "@1=svgx.sof#p"
 . . .
    Info (209016): Configuring device index 1
    Info (209017): Device 1 contains JTAG ID code 0x029030DD
    Info (209007): Configuration succeeded -- 1 device(s) configured
    Info (209011): Successfully performed operation(s)
    Info (208551): Program signature into device 1.
    Info: Intel Quartus Prime 64-Bit Fault Injection Debugger was successful.
    0 errors, 0 warnings

##############################################
#
# Inject a fault into the device.
# The #i operator indicates to inject faults
# -n 3 indicates to inject 3 faults
#
$ quartus_fid --cable USB-Blaster --index "@1=svgx.sof#i" -n 3
    Info: Command: quartus_fid --cable=USB-Blaster --index=@1=svgx.sof#i -n 3
    Info (208809): Using programming cable "USB-Blaster on sj-sng-z4 [USB-0]"
    Info (208521): Injects 3 error(s) into device(s)
    Info: Intel Quartus Prime 64-Bit Fault Injection Debugger was successful.
    0 errors, 0 warnings


##############################################
#
# Interactive Mode.
# Using the #i operation with -n 0 puts the debugger into interactive mode.
# Note that 3 faults were injected in the previous session;
# "E" reads the faults currently in the EMR Unloader IP core.
#
$ quartus_fid --cable USB-Blaster --index "@1=svgx.sof#i" -n 0
    Info: Command: quartus_fid --cable=USB-Blaster --index=@1=svgx.sof#i -n 0
    Info (208809): Using programming cable "USB-Blaster on sj-sng-z4 [USB-0]"
    Enter :
          'F' to inject fault
          'E' to read EMR
          'S' to scrub error(s)
          'Q' to quit
    E
    Info (208540): Reading EMR array
    Info (208544): 3 frame error(s) detected in device 1.
        Info (208545):   Error #1 : Single error in frame 0x1028 at bit 0x21EA.
        Info (10914):   Error #2 : Uncorrectable multi-bit error in frame 0x1116.
        Info (208545):   Error #3 : Single error in frame 0x1848 at bit 0x128C.
    Enter :
          'F' to inject fault
```

```
          'E' to read EMR
          'S' to scrub error(s)
          'Q' to quit
    Q
    Info: Intel Quartus Prime 64-Bit Fault Injection Debugger was successful.
        0 errors, 0 warnings
        Info: Peak virtual memory: 1522 megabytes
        Info: Processing ended: Mon Nov  3 18:50:00 2014
        Info: Elapsed time: 00:00:29
        Info: Total CPU time (on all processors): 00:00:13
```

2.3.4.1. Targeted Fault Injection Feature

The Fault Injection Debugger injects faults into the FPGA randomly. However, the Targeted Fault Injection feature allows you to inject faults into targeted locations in the CRAM. This operation may be useful, for example, if you noted an SEU event and want to test the FPGA or system response to the same event after modifying a recovery strategy.

Note: The Targeted Fault Injection feature is available only from the command line interface.

You can specify that errors are injected from the command line or in prompt mode.

Related Information

AN 539: Test Methodology or Error Detection and Recovery using CRC in Intel FPGA Devices

2.3.4.1.1. Specifying an Error List From the Command Line

The Targeted Fault Injection feature allows you to specify an error list from the command line, as shown in the following example:

```
c:\Users\sng> quartus_fid -c 1 - i "@1= svgx.sof#i " -n 2 -user="@1=
0x2274 0x05EF 0x2264 0x0500"
```

Where:

`c 1` indicates that the fpga is controlled by the first cable on your computer.

`i "@1= svgx.sof#i "` indicates that the first device in the chain is loaded with the object file `svgx.sof` and is injected with faults.

`n 2` indicates that two faults are injected.

`user=”@1= 0x2274 0x05EF 0x2264 0x0500”` is a user-specified list of faults that are injected. In this example, device 1 has two faults: at frame 0x2274, bit 0x05EF and at frame 0x2264, bit 0x0500.

2.3.4.1.2. Specifying an Error List From Prompt Mode

You can operate the Targeted Fault Injection feature interactively by specifying the number of faults to be 0 (-n 0). The Fault Injection Debugger presents prompt mode commands and their descriptions.

Prompt Mode Command	Description
F	Inject a fault
E	Read the EMR
S	Scrub errors
Q	Quit

In prompt mode, you can issue the F command alone to inject a single fault in a random location in the device. In the following examples using the F command in prompt mode, three errors are injected.

```
F #3 0x12 0x34 0x56 0x78 * 0x9A 0xBC +
```

- Error 1 – Single bit error at frame 0x12, bit 0x34
- Error 2 – Uncorrectable error at frame 0x56, bit 0x78 (an * indicates a multi-bit error)
- Error 3 – Double-adjacent error at frame 0x9A, bit 0xBC (a + indicates a double bit error)

```
F 0x12 0x34 0x56 0x78 *
```

One (default) error is injected:

Error 1 – Single bit error at frame 0x12, bit 0x34. Locations after the first frame/bit location are ignored.

```
F #3 0x12 0x34 0x56 0x78 * 0x9A 0xBC + 0xDE 0x00
```

Three errors are injected:

- Error 1 – Single bit error at frame 0x12, bit 0x34
- Error 2 – Uncorrectable error at frame 0x56, bit 0x78
- Error 3 – Double-adjacent error at frame 0x9A, bit 0xBC
- Locations after the first 3 frame/bit pairs are ignored

2.3.4.1.3. Determining CRAM Bit Locations

When the Fault Injection Debugger detects a CRAM EDCRC error, the Error Message Register (EMR) contains the syndrome, frame number, bit location, and error type (single, double, or multi-bit) of the detected CRAM error.

During system testing, save the EMR contents reported by the Fault Injection Debugger when you detect an EDCRC fault.

Note: With the recorded EMR contents, you can supply the frame and bit numbers to the Fault Injection Debugger to replay the errors noted during system testing, to further design, and characterize a system recovery response to that error.

Related Information

AN 539: Test Methodology or Error Detection and Recovery using CRC in Intel FPGA Devices

2.3.4.2. Advanced Command-Line Options: ASD Regions and Error Type Weighting

You can use the Fault Injection Debugger command-line interface to inject errors into ASD regions and weight the error types.

First, you specify the mix of error types (single bit, double adjacent, and multi-bit uncorrectable) using the `--weight <single errors>.<double adjacent errors>.<multi-bit errors>` option. For example, for a mix of 50% single errors, 30% double adjacent errors, and 20% multi-bit uncorrectable errors, use the option `--weight=50.30.20`. Then, to target an ASD region, use the `-smh` option to include the SMH file and indicate the ASD region to target. For example:

```
$ quartus_fid --cable=USB-BlasterII --index "@1=svgx.sof#pi" --
weight=100.0.0 --smh="@1=svgx.smh#2" --number=30
```

This example command:

- Programs the device and injects faults (`pi` string)
- Injects 100% single-bit faults (100.0.0)
- Injects only into `ASD_REGION` 2 (indicated by the `#2`)
- Injects 30 faults

2.4. Document Revision History

Table 4. Document Revision History

Date	Version	Changes
2018.09.24	18.1.0	• First release as part of the stand-alone *Intel Quartus Prime Standard EditionApplication Note.*
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0.0	• Provided more detail on how to use the Fault Injection Debugger throughout the document. • Added more command-line examples.
2014.06.30	14.0.0	• Removed "Modifying the Quartus INI File" section. • Added "Targeted Fault Injection Feature" section. • Updated "Hardware and Software Requirements" section.
December 2012	2012.12.01	Preliminary release.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.