AN 539: Test Methodology of Error Detection and Recovery using CRC in Intel FPGA Devices
1. Test Methodology of Error Detection and Recovery using CRC in Intel® FPGA Devices....3
  1.1. Functional Description..................................................................................................................3
    1.1.1. Configuration Error Detection..................................................................................................5
    1.1.2. User Mode Error Detection.......................................................................................................5
    1.1.3. Error Detection Pin...................................................................................................................5
    1.1.4. Error Message Register.............................................................................................................6
    1.1.5. Error Detection Timing.............................................................................................................8
  1.2. Error Correction...............................................................................................................................9
  1.3. Using the Error Detection CRC Feature............................................................................................9
    1.3.1. Error Detection using User Logic...............................................................................................9
    1.3.2. Error Detection using an External Host......................................................................................12
  1.4. Error Injection...................................................................................................................................15
    1.4.1. Fault Injection Register............................................................................................................16
    1.4.2. Error Injection using the EDERROR_INJECT JTAG Instruction................................................17
    1.4.3. Clearing Fault Injection Register...............................................................................................21
  1.5. Modifying Single-Device .jam Files for Use in a Multi-Device JTAG Chain.................................23
  1.6. Running .jam Files with the Intel Quartus Prime Jam Tools...........................................................25
1. Test Methodology of Error Detection and Recovery using CRC in Intel® FPGA Devices

This application note describes how to use the enhanced error detection cyclic redundancy check (CRC) feature in the Arria® II, Stratix® III, Stratix IV, Arria V, Cyclone® V, and Stratix V devices. It also describes the test methodology you can use when testing the capability of this feature in the supported devices. Arria V, Cyclone V, and Stratix V devices also support error correction feature.

During FPGA configuration, the error detection CRC feature detects configuration bitstream corruption when the bitstream is transferred from an external device into the FPGA. In user mode, the error detection CRC feature detects a single event upset (SEU) and determines the error type and location. In addition, Arria V, Cyclone V, and Stratix V devices support internal scrubbing, an ability to correct errors detected in user mode.

Related Information
- SEU Mitigation in Arria II Devices
- SEU Mitigation in the Cyclone III Device Family
- SEU Mitigation in Stratix III Devices
- SEU Mitigation in Stratix IV Devices
- SEU Mitigation for Arria V Devices
- SEU Mitigation for Cyclone V Devices
- SEU Mitigation for Stratix V Devices
- SEU Mitigation in Intel Cyclone 10 LP Devices

1.1. Functional Description

You can use the error detection CRC feature to detect errors in the FPGA configuration bits during configuration and user modes.

The width of the CRC field is 16 bits in Arria II, Stratix III, and Stratix IV devices; 32 bits in Arria V, Cyclone V, and Stratix V devices.
Figure 1. **Error Detection Block Diagram**

Figure shows the error detection circuitry, syndrome registers, and error injection block.

1. Applicable to the Arria II, Stratix III, and Stratix IV devices only.
2. Applicable to the Arria V, Cyclone V, and Stratix V devices only.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syndrome Register</td>
<td>This register contains the CRC signature of the current frame through the error detection verification cycle. The CRC_ERROR signal is derived from the contents in this register.</td>
</tr>
<tr>
<td>Error Message Register (EMR)</td>
<td>This register contains information about the error type, the location of the error, and the actual syndrome. The types of errors and location reported are single- and double-adjacent bit errors. For Arria V, Cyclone V, and Stratix V devices, the types of errors and location reported are single-, double-, triple-, and quadruple-adjacent bit errors. The location bits for other types of errors are not identified by the EMR. You can shift the contents of this register through the SHIFT_EDERROR_REG JTAG instruction or to the core through the core interface. See related information for more information about error message register.</td>
</tr>
<tr>
<td>JTAG Update Register</td>
<td>This register is automatically updated with the contents of the EMR one cycle after the EMR register content is validated. The JTAG update register includes a clock enable that must be asserted before being sampled into the JTAG shift register. This requirement ensures that the JTAG update register is not being written by the contents of the EMR at exactly the same time as the JTAG shift register is reading its contents.</td>
</tr>
<tr>
<td>User Update Register</td>
<td>This register is automatically updated with the contents of the EMR one cycle after the EMR register content is validated. The user update register includes a clock enable that must be asserted before being sampled into the user shift register. This requirement ensures that the user update register is not being written by the contents of the EMR at exactly the same time as the user shift register is reading its contents.</td>
</tr>
<tr>
<td>JTAG Shift Register</td>
<td>This register is accessible by the JTAG interface and allows the contents of the JTAG update register to be sampled and read out by the SHIFT_EDERROR_REG JTAG instruction.</td>
</tr>
<tr>
<td>User Shift Register</td>
<td>This register is accessible by the core logic and allows the contents of the user update register to be sampled and read by user logic.</td>
</tr>
<tr>
<td>JTAG Fault Injection Register</td>
<td>This register is fully controlled by the EDERROR_INJECT JTAG instruction. This register holds the information of the error injection that you want in the bitstream.</td>
</tr>
<tr>
<td>Fault Injection Register</td>
<td>The contents of the JTAG fault injection register is loaded into this fault injection register when the JTAG fault injection register is being updated.</td>
</tr>
</tbody>
</table>
1.1.1. Configuration Error Detection

When the Intel® Quartus® Prime software generates the configuration bitstream, it also computes a CRC value for each data frame. The number of data frames and the length of the frames in a configuration bitstream vary in each device. Therefore, a configuration bitstream can contain more than one CRC values depending on the number of data frames in the bitstream.

When a data frame is loaded into the FPGA during configuration, the pre-computed CRC value shifts into the CRC circuitry in the FPGA. At the same time, the CRC engine in the FPGA computes the CRC value for the data frame received, and stores the CRC value into the configuration RAM (CRAM). The pre-computed CRC value is compared against the CRC value computed by the CRC engine. If the CRC values do not match, nSTATUS is set low to flag a configuration error.

You can test the capability of the configuration error detection feature by either modifying the configuration bitstream or intentionally corrupting the configuration bitstream during configuration.

1.1.2. User Mode Error Detection

During user mode, the contents of a configured CRAM bit may be affected by soft errors. User mode error detection allows you to determine the type of soft error and to identify the location of the affected bit.

After the configuration cycle is complete, the error detection feature computes the CRC value of the configured CRAM bits and compares the results with the pre-computed CRC value. If both the CRC values match, the resulting CRC signature is set to 0 to indicate that no error was detected. The error detection process continues until the device is reset by setting nCONFIG low.

If CRC errors occur, the resulting signature is non-zero and the CRC_ERROR pin is set high to flag an error. When errors occur, the search engine in the error detection state machine identifies the error type and error location. All types of CRC errors can be detected in a frame. The results of the search engine are stored in the EMR. You can shift out the contents of this register through the JTAG instruction or core interface logics. You can shift out the contents while the error detection block is running.

1.1.3. Error Detection Pin

When you enable the error detection feature, use the CRC_ERROR pin as a dedicated output pin, otherwise use the CRC_ERROR pin as an user I/O pin. When you use this pin as a dedicated output pin, the active high signal on this pin indicates that the error detection circuit has detected errors in the configured CRAM bits. When you enable the WYSIWYG function, the CRC error output is a dedicated path to the CRC_ERROR pin. You can also use this pin as an open-drain output by enabling the options using the Intel Quartus Prime software. Using this pin as an open-drain output provides voltage leveling advantages.

Note: Only use the CRC_ERROR pin during user mode error detection.

Related Information
• SEU Mitigation in Arria II Devices
• Enabling the error detection block in Quartus II software for Cyclone III Device Family
• Enabling the error detection block in Quartus II software for Stratix III Devices
• Enabling the error detection block in Quartus II software for Stratix IV Devices
• Enabling the error detection block in Intel Quartus Prime software for Arria V Devices
• Enabling the error detection block in Intel Quartus Prime software for Cyclone V Devices
• Enabling the error detection block in Intel Quartus Prime software for Stratix V Devices
• SEU Mitigation in Intel Cyclone 10 LP Devices

1.1.4. Error Message Register

The EMR contains information on the error type, the location of the error, and the actual syndrome. This register is 46 bits wide in Arria II, Stratix III, and Stratix IV devices; 67 bits wide in Arria V, Cyclone V, and Stratix V devices. Table 2 on page 7 and Table 3 on page 7 list the types of errors and locations reported. The EMR does not identify the location bits for other types of errors.

The location of the errors consist of the column frame number, byte location within the frame, and bit location in the byte. You can shift out the contents of the register through the `SHIFT_EDERROR_REG` JTAG instruction or to the core through the core interface.

Table 1. \textbf{SHIFT_EDERROR_REG} JTAG Instruction

<table>
<thead>
<tr>
<th>JTAG Instruction</th>
<th>Instruction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT_EDERROR_REG</td>
<td>00 0001 0111</td>
<td>The JTAG instruction connects the EMR to the JTAG pin in the error detection block between the TDI and TDO pins.</td>
</tr>
</tbody>
</table>

The contents of the EMR is updated when an error or errors occur. You must transfer the contents of the EMR out before they are overwritten by the next error message. The minimum interval time between two EMR updates is different for every device. For more information about the minimum update interval of the EMR, see related information below.

\textit{Note:} You can slow down the error detection process to have sufficient time to read out the EMR by controlling the error detection frequency.

\textbf{Related Information}

• Unloading the EMR using an External Host on page 13
• CRC Timing specifications for Arria II Devices
• CRC Timing specifications for Cyclone III Device Family
• CRC Timing specifications for Stratix III Devices
• CRC Timing specifications for Stratix IV Devices
• CRC Timing specifications for Arria V Devices
• CRC Timing specifications for Cyclone V Devices
• CRC Timing specifications for Stratix V Devices
1. Test Methodology of Error Detection and Recovery using CRC in Intel® FPGA Devices

1.1.4.1. EMR for Arria II, Stratix III, and Stratix IV Devices

Figure 2. EMR Contents for Arria II, Stratix III, and Stratix IV Devices

**Table 2. Error Type Represented in the EMR for Arria II, Stratix III, and Stratix IV Devices**

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 1</td>
<td>Bit 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1.1.4.2. EMR for Arria V, Cyclone V, and Stratix V Devices

Figure 3. EMR Contents for Arria V, Cyclone V, and Stratix V Devices

**Table 3. Error Type Represented in the EMR for Arria V, Cyclone V, and Stratix V Devices**

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 4</td>
<td>Bit 3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Others</td>
<td></td>
</tr>
</tbody>
</table>
1.1.5. Error Detection Timing

The CRC_ERROR pin is always set low during CRC signature calculation for a minimum of 32 clock cycles. When errors occur, the CRC_ERROR pin is set high after the EMR has been updated, or after 32 cycles, whichever comes last. Therefore, you can start to unload the contents of the EMR at the rising edge of the CRC_ERROR pin. At the end of this cycle, the CRC_ERROR pin is set low again for a minimum of 32 clock cycles. If the next frame also contains error, the CRC_ERROR pin pulls high again and the EMR is updated with the new error information.

**Figure 4. Example of the CRC_ERROR Pin Behavior**

Example of how the CRC_ERROR pin behaves when an error occurs in a data frame.

<table>
<thead>
<tr>
<th>Frame</th>
<th>N</th>
<th>N+1</th>
<th>N+2</th>
<th>N+3</th>
<th>N+4</th>
<th>N+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Integrity</td>
<td>No CRC Error</td>
<td>CRC Error</td>
<td>CRC Error</td>
<td>No CRC Error</td>
<td>CRC Error</td>
<td>No CRC Error</td>
</tr>
<tr>
<td>Reading Frame Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC_ERROR Pin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allowed time for reading out EMR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You can control the speed of the error detection process by setting the division factor of the clock frequency in the Intel Quartus Prime software. The divisor is a power of two, in which n is between one and eight. The divisor ranges from two through 256.

**Figure 5. Error Detection Frequency Equation**

\[
\text{Error Detection Frequency} = \frac{\text{Internal Oscillator Frequency}}{N}
\]

**Related Information**

- CRC Timing specifications for Arria II Devices
- CRC Timing specifications for Cyclone III Device Family
- CRC Timing specifications for Stratix III Devices
- CRC Timing specifications for Stratix IV Devices
- CRC Timing specifications for Arria V Devices
- CRC Timing specifications for Cyclone V Devices
- CRC Timing specifications for Stratix V Devices
- SEU Mitigation in Intel Cyclone 10 LP Devices
1.2. Error Correction

In addition to error detection capability, Arria V, Cyclone V, and Stratix V devices also support internal scrubbing. Internal scrubbing has the ability to internally correct soft errors. This is done on a per frame basis. Internal scrubbing corrects errors without the need of reconfiguring the device. Internal scrubbing operates in user mode. Enable the internal scrubbing feature in the Intel Quartus Prime software by turning on the Enable internal scrubbing option under the Error Detection CRC page in the Device & Pin Options dialog box.

Figure 6. Internal Scrubbing Feature

1.3. Using the Error Detection CRC Feature

To determine the location of the CRC errors, unload the EMR. You can shift the contents of the EMR through the JTAG instruction or core interface.

1.3.1. Error Detection using User Logic

This section describes how to use the error detection feature through user logic with a design example.

Related Information
- Design Example for Arria II, Stratix III, and Stratix IV Devices
- Design Example for Arria V, Cyclone V, and Stratix V Devices

1.3.1.1. Detecting CRC Errors through the CRC_ERROR Output Signal

To detect CRC errors through the CRC_ERROR output signal, use the CRC_ERROR output signal. Check whether a CRC error has been found in the CRC calculation before unloading the contents of the EMR. To do this, route the crcerror port from the WYSIWYG atom to the dedicated CRC_ERROR pin or any user I/O. To route the crcerror port to a user I/O, you must insert a D flipflop (DFF) in between the crcerror port and the user I/O.

1.3.1.2. Unloading the EMR through User Logic

To unload the EMR through user logic, establish an interface between the user logic and the error detection circuit using the WYSIWYG atom. The WYSIWYG atom provides access to the error detection block. In addition, you must design a user logic control block to control the necessary control signals to access the WYSIWYG atom.
Figure 7. Interface Between the User Logic Control Block and the WYSIWYG Atom

![Diagram of interface between User Logic Control Block and WYSIWYG Atom]

**Note:** WYSIWYG is a technique that performs optimization on the Verilog Quartus Mapping netlist within the Intel Quartus Prime software.

1.3.1.3. Accessing the Error Detection Block through User Logic

Use the `<device>_crcblock` WYSIWYG component to establish the interface from the user logic to the error detection circuit. The `<device>_crcblock` primitive atom contains the input and output ports that you must include in the atom. To access the logic array, insert the `<device>_crcblock` WYSIWYG atom into your design.

You cannot use the information provided by the EMR to detect soft errors. Instead, use the information provided by the `CRC_ERROR` output signal because it is not affected by soft errors.

Figure 8. User Logic Error Detection Block Diagram

![Diagram showing error detection feature and WYSIWYG atom]

**Note:** For the Intel Quartus Prime software version 8.0 SP1 or earlier, you must enable the error detection CRC feature in the Device & Pin Options dialog box in the Intel Quartus Prime software when you include the `<device>_crcblock` WYSIWYG atom in your design.
To enable the `<device>_crcblock` WYSIWYG atom, name the atom for each device accordingly. For example, the WYSIWYG atom for the Stratix III device is `stratixiii_crcblock` and the Arria II device is `arriaii_crcblock`.

### Table 4. CRC Block Input and Output Ports
Table lists the input and output ports that you must include in the WYSIWYG atom.

<table>
<thead>
<tr>
<th>Port</th>
<th>Input/Output</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;crcblock_name&gt;</code></td>
<td>Input</td>
<td>Unique identifier for the CRC block and represents any identifier name that is legal for the given description language, such as the Verilog HDL, VHDL, and Altera Hardware Description Language (AHDL). This is a required field.</td>
</tr>
<tr>
<td><code>.clk(&lt;clock source&gt;)</code></td>
<td>Input</td>
<td>Designates the clock input of this cell. All operations of this cell are with respect to the rising edge of the clock. Whether loading data into the cell or out of the cell, this always occurs on the rising edge. This is a required port.</td>
</tr>
</tbody>
</table>
| `.shiftnld (<shiftnld source>)` | Input        | • An input into the error detection block.  
• If `shiftnld`=1, the user shift register shifts the data to the `regout` port at each rising edge of the `clk` port.  
• If `shiftnld`=0, the user shift register parallel loads the contents of the user update register. This is a required port.  
• This input triggers clock enable for the user update register to de-assert after two `EDCLK` cycles. After driving the `ED_SHIFTNLD` signal low, wait at least two `EDCLK` cycles before clocking the `ED_CLK` signal. |
| `.crcerror (<crcerror out destination>)` | Output      | • Output of the cell that is synchronized to the internal oscillator of the device (100-MHz or 80-MHz internal oscillator) and not to the `clk` port. This output asserts automatically high if the error block detects that a SRAM bit has flipped and the internal CRC computation has shown a difference with respect to the pre-computed value.  
• Connect this signal to an output pin or a bidirectional pin. If you connect this output signal to an output pin, you can only monitor the `CRC_ERROR` pin (the core logic cannot access this output). If the core logic uses the `CRC_ERROR` signal to read the error detection logic, connect this signal to a `BIDIR` pin. The signal is fed to the core indirectly by feeding a `BIDIR` pin that has its output enable port (oe) connected to `VCC`.  
• The signal that is routed to the `CRC_ERROR` pin is also routed to the core. |
| `.regout(<output destination>)` | Output      | • Output of the user shift register synchronized to the `clk` port, to be read by the core logic.  
• This shifts one bit at each cycle and follows the LSB-first format. |

### Example 1. Example of the Input and Output Ports of a WYSIWYG Atom in a Stratix III Device
Example shows the input and output ports of a WYSIWYG atom in a Stratix III device.

```vhdl
stratixiii_crcblock <crcblock_name> 
{ .clk(<clock source>),  
  .shiftnld(<shiftnld source>),  
  .crcerror(<crcerror out destination>),  
  .regout(<output destination>)};
defparam crc_wysiwyg_atom.oscillator_divider = 2
```

**Note:** The `crc_wysiwyg_atom.oscillator_divider` parameter describes the clock divider for the error detection frequency of the `<device>_crcblock` WYSIWYG atom.
1.3.1.4. User Logic Control Block

You define the user logic control block. A design example is provided with this application note. In this design example, the user logic control block controls the input port to the WYSIWYG atom to read out the contents of the user update register. The user update register is updated with the contents of the EMR. To read out the contents of the user update register, follow these steps:

1. Drive the SHIFTnLD signal low.
2. Wait at least two ED_CLK cycles.
3. Clock clk_in one cycle for one rising edge to load the contents of the user update register to the user shift register.
4. Drive the SHIFTnLD signal high.
5. Clock Clk_in 29 cycles to read out 30 bits of the error location information.
6. Clock Clk_in an additional 16 cycles to read out the syndrome of the error.

1.3.1.5. User Logic Control Block Signals

Table 5. User Logic Control Block Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk_in</td>
<td>Input</td>
<td>Clock source to the user logic control block.</td>
</tr>
<tr>
<td>Start_write</td>
<td>Input</td>
<td>This input triggers the user logic control block to start driving the SHIFTnLD and ED_CLK output signals. When this input is high, the user logic control block starts to run the mechanism to unload the user update register. This input is usually tied to the crcerror output port from the WYSIWYG atom because the EMR is unloaded at the rising edge of the crcerror port.</td>
</tr>
<tr>
<td>SHIFTnLD</td>
<td>Output</td>
<td>Output to the WYSIWYG atom. Use this signal to drive the shiftnld port of the WYSIWYG atom.</td>
</tr>
<tr>
<td>ED_CLK</td>
<td>Output</td>
<td>Output clock to the WYSIWYG atom. Use this clock as the clock source for the WYSIWYG atom. This clock is connected to the clk port of the WYSIWYG atom. The source of this clock comes from the clk_in input signal.</td>
</tr>
</tbody>
</table>

1.3.2. Error Detection using an External Host

This section describes how to use the error detection feature through an external host with the design example.

Related Information

- Design Example for Arria II, Stratix III, and Stratix IV Devices
- Design Example for Arria V, Cyclone V, and Stratix V Devices
1.3.2.1. Detecting a CRC Error through the CRC_ERROR Output Signal

Before unloading the contents of the EMR, use the CRC_ERROR output signal to check whether a CRC error has been found throughout the CRC calculation. To do this, enable the error detection CRC feature in the Device & Pin Options dialog box in the Intel Quartus Prime software and observe the dedicated CRC_ERROR pin.

1.3.2.2. Unloading the EMR using an External Host

To unload the contents of the EMR using an external host, such as 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.

The following examples show 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 2. Example of .jam File to Unload the Contents of the EMR for Arria II, Stratix III, and Stratix IV Devices

```
ACTION UNLOAD_EMR = EXECUTE;
DATA EMR_DATA;
BOOLEAN out[46];
BOOLEAN in[46]=03FFFFFFF;
ENDDATA;
PROCEDURE EXECUTE USES EMR_DATA;
  DRSTOP IDLE;
  IRSTOP IDLE;
  STATE IDLE;
  IRSCAN 10, $017;
  WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
  DRSCAN 46, in[45..0], CAPTURE out[45..0];
  WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;
  PRINT " ";
  PRINT "Data read out from the EMR Register: ", 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 "Syndrome:"; 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];
```
Example 3. Example of .jam File to Unload the Contents of the EMR for Arria V, Cyclone V, and Stratix V Devices

ACTION UNLOAD_EMR = EXECUTE;
DATA EMR_DATA;
BOOLEAN out[67];
BOOLEAN in[67] = $7FFFFFFFFFFFFFF;
INTEGER i;
ENDDATA;
PROCEDURE EXECUTE USES EMR_DATA;
DRSTOP IDLE;
IRSTOP IDLE;
STATE IDLE;
IRSCAN 10, $017;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 67, in[66..0], CAPTURE out[66..0];
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;
PRINT " ";
PRINT "Data read out from the ";
PRINT "EMR_Register ::", 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 "Syndrome ::", 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];
'}}
1.4. Error Injection

To test the error detection block, you can intentionally inject errors. This error injection methodology provides design verification and system fault tolerance characterization. Using the EDERROR_INJECT JTAG instruction, you can inject a single error, double errors, or double-adjacent errors to the configuration memory.

The EDERROR_INJECT JTAG instructions is not designed to test internal scrubbing. If you want to validate the internal scrubbing feature, Intel recommends that you use the Intel Quartus Prime Fault Injection Debugger.

### Table 6. EDERROR_INJECT JTAG Instruction

<table>
<thead>
<tr>
<th>JTAG Instruction</th>
<th>Instruction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDERROR_INJECT</td>
<td>00 0001 0101</td>
<td>This instruction controls the 21-bit JTAG fault injection register that is used for error injection.</td>
</tr>
</tbody>
</table>

**Note:** For Stratix V III devices, you can only execute the EDERROR_INJECT JTAG instruction at a 50 MHz error detection frequency. Refer to the related information.

**Related Information**

- [SEU Mitigation in Stratix III Devices](#)
- [Using the Fault Injection Debugger, Fault Injection Intel FPGA IP Core User Guide](#) Describes the Fault Injection Debugger and how to use it to inject, record, and clear injected errors.
1.4.1. Fault Injection Register

The \texttt{EDERROR\_INJECT} JTAG instruction controls the contents of the JTAG fault injection register. The register holds the information of the error that you want to inject into the configuration memory.

You can scan the location of the error or errors into the 21-bit (46-bit for Arria V, Cyclone V, and Stratix V devices) JTAG fault injection register at any time. The contents of the JTAG fault injection register is loaded into the fault injection register when the last and first data frame is being processed. In other words, you can only inject the error into the first frame of the configuration data but you can monitor the error information at any time. The register holds the information of the error that you want to inject into the configuration memory.

\textit{Note:} The contents of the fault injection register are cleared when zero bits are loaded into the fault injection register from the JTAG fault injection register.

1.4.1.1. Fault Injection for Arria II, Stratix III, and Stratix IV Devices

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure9.png}
\caption{Fault Injection Register for Arria II, Stratix III, and Stratix IV Devices}
\end{figure}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Error Type & Description \\
\hline
Bit 20 & Bit 19 & \\
\hline
0 & 0 & No error injection. \\
\hline
0 & 1 & Single byte error injection. \\
\hline
1 & 0 & Double-adjacent error injection. \\
\hline
1 & 1 & Invalid error injection. \\
\hline
\end{tabular}
\caption{Error Type of the Injected Error for the EMR for Arria II, Stratix III, and Stratix IV Devices}
\end{table}
1.4.1.2. Fault Injection for Arria V, Cyclone V, and Stratix V Devices

Figure 10. Fault Injection Register for Arria V, Cyclone V, and Stratix V Devices

<table>
<thead>
<tr>
<th>Error type</th>
<th>Byte Location of the Injected Error</th>
<th>Error Byte Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Bits</td>
<td>10 Bits</td>
<td>32 Bits</td>
</tr>
</tbody>
</table>

Table 8. Error Type of the Injected Error for the EMR for Arria V, Cyclone V, and Stratix V Devices

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>No error injection.</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>Single byte error injection.</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>Double-adjacent error injection.</td>
</tr>
<tr>
<td>Others</td>
<td>Invalid error injection.</td>
</tr>
</tbody>
</table>

1.4.2. Error Injection using the EDERRO error JTAG Instruction

This section describes the testing process. When using a .jam file to automate the testing process, you can test and verify the functionality of the error detection block without having to reconfigure the device. Use the EDERRO error JTAG instruction to control the contents of the JTAG fault injection register to flip the readback bits of a particular location in the first data frame as specified in the JTAG fault injection register.

Note: Intel recommends observing the behavior of the CRC_ERROR pin and unloading the contents of the EMR using the recommended approaches to verify the error has been correctly injected.

1.4.2.1. Injecting Single-Bit Error for Arria II, Stratix III, and Stratix IV

If you want to inject a single-bit error into the 7th bit of the 5th byte in the first data frame, you must shift in the data that contains the information of the injected error into the fault injection register.

The following figure shows the binary bit sequence 0 1000 0000 0100 0100 0000 is shifted into the fault injection register. Because bit 19 and bit 20 of the fault injection register is 01, a single-bit error is injected. The byte location of the injected error is defined as 000 0000 0100, thus the single-bit error is injected into the 5th byte of the data frame. The error byte location value is 0100 0000, which indicates the 7th bit of the 5th byte data is the specific location in which the single-bit error is injected.
Figure 11. Injecting a Single-Bit Error for Arria II, Stratix III, and Stratix IV Devices

Figure shows the error injection mechanism.

FIR register content

<table>
<thead>
<tr>
<th></th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01 000 000</td>
<td>0100 0100</td>
</tr>
</tbody>
</table>

Correct data in first data frame

<table>
<thead>
<tr>
<th></th>
<th>Byte 4</th>
<th>Byte 3</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1100 000</td>
<td>1010 000</td>
<td></td>
</tr>
</tbody>
</table>

Data in first data frame with single-bit error

<table>
<thead>
<tr>
<th></th>
<th>Byte 4</th>
<th>Byte 3</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1000 000</td>
<td>1010 000</td>
<td></td>
</tr>
</tbody>
</table>

The following example shows the .jam file that you must execute in the EDEERROR_INJECT JTAG instruction to inject the single-bit error into the first data frame for Arria II, Stratix III, and Stratix IV devices.

Example 4. Example of .jam File to Inject Single-Bit Error for Arria II, Stratix III, and Stratix IV Devices

```
ACTION ERROR_INJECT = EXECUTE;
DATA DEVICE_DATA;
BOOLEAN out[21];
BOOLEAN in[21] = $080440;
ENDDATA;
PROCEDURE EXECUTE USES DEVICE_DATA;
BOOLEAN X = 0;
DRSTOP IDLE;
IRSTOP IDLE;
STATE IDLE;
IRSCAN 10, $015;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 21, in[20..0], CAPTURE out[20..0];    'shift out the previous content from fault injection register
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 21, in[20..0], CAPTURE out[20..0];    'shift out current content from fault injection register
WAIT IDLE, 10 CYCLES, 25 USEC, IDLE;
```

1. Test Methodology of Error Detection and Recovery using CRC in Intel® FPGA Devices

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

AN-539 | 2019.08.09

Send Feedback
PRINT " ";
PRINT "Data read out from the FIR Register: ", 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 "Error Type: ", out[20], out[19];
PRINT " ";
PRINT "Byte Location: ", out[18], out[17], out[16], out[15], out[14], out[13],
out[12], out[11], out[10], out[9], out[8];
PRINT " ";
PRINT "Error Byte Value: ", out[7], out[6], out[5], out[4], out[3], out[2],
out[1], out[0];
STATE IDLE;
EXIT 0;
ENDPROC;

Related Information

• Running .jam Files with the Intel Quartus Prime Jam Tools on page 25
• Modifying Single-Device .jam Files for Use in a Multi-Device JTAG Chain on page 23

1.4.2.2. Injecting Single-Bit Error for Arria V, Stratix V, and Cyclone V Devices

If you want to inject a single-bit error into the 15th bit of the 7th byte in the first data frame, you must shift in the data that contains the information of the injected error into the fault injection register.

The following figure shows the binary bit sequence 00 0100 0000 0110 0000 0000 0000 0100 0000 0000 0000 is shifted into the fault injection register. Because bits 45, 44, 43, and 42 of the fault injection register is 0001, a single-bit error is injected. The byte location of the injected error is defined as 00 0000 0110, thus the single-bit error is injected into the 7th byte of the data frame. The error byte location value is 0000 0000 0000 000 0100 0000 0000 0000, which indicates the 15th bit of the 7th byte data is the specific location in which the single-bit error is injected.
### Figure 12. Injecting a Single-Bit Error for Arria V, Stratix V, and Cyclone V Devices

Figure shows the error injection mechanism.

<table>
<thead>
<tr>
<th>FIR register content</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00 0100 0000 0110 0000 0000 0000 0100 0000 0000 0000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correct data in first data frame</th>
<th>Byte 6</th>
<th>Byte 5</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1000 0100 0010 0000 1100 0000 0000 0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data in first data frame with single-bit error</th>
<th>Byte 6</th>
<th>Byte 5</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1000 0100 0010 0000 1000 0000 0000 0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following example shows the .jam file that you must execute in the EDEERROR_INJECT JTAG instruction to inject the single-bit error into the first data frame for Arria V, Stratix V, and Cyclone V devices.

#### Example 5. Example of .jam File to Inject Single-Bit Error for Arria V, Stratix V, and Cyclone V Devices

```plaintext
ACTION ERROR_INJECT = EXECUTE;
DATA DEVICE_DATA;
BOOLEAN out[46];
BOOLEAN in[46] = $040600004000;
ENDDATA;
PROCEDURE EXECUTE USES DEVICE_DATA;
BOOLEAN X = 0;
DRSTOP IDLE;
IRSTOP IDLE;
STATE IDLE;
IRSCAN 10, $015;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 46, in[45..0], CAPTURE out[45..0];  'shift out the previous content from fault injection register
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 46, in[45..0], CAPTURE out[45..0];  'shift out current content from fault injection register
WAIT IDLE, 10 CYCLES, 50 USEC, IDLE;
```

1. Test Methodology of Error Detection and Recovery using CRC in Intel® FPGA Devices
PRINT " Data read out from the FIR Register: ", 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 "Error Type:", out[45], out[44], out[43], out[42];

PRINT " ";

PRINT "Byte Location:", out[41], out[40], out[39], out[38], out[37], out[36], out[35], out[34], out[33], out[32];

PRINT " ";

PRINT "Error Byte Value:", 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];

STATE IDLE;

EXIT 0;

ENDPROC;

Related Information

• Running .jam Files with the Intel Quartus Prime Jam Tools on page 25
• Modifying Single-Device .jam Files for Use in a Multi-Device JTAG Chain on page 23

1.4.3. Clearing Fault Injection Register

After completing the test process, clear the contents of the fault injection register or start a reconfiguration by pulling the nCONFIG signal low to disable the error that you injected. To clear the contents of the fault injection register, scan all zero data into the register.

The following examples show .jam files used to clear the contents of the fault injection register.

Example 6. Example of .jam File to Clear the Contents of the Fault Injection Register for Arria II, Stratix III, and Stratix IV Devices

ACTION ERROR_INJECT_DISABLE = EXECUTE;

DATA DEVICE_DATA;

BOOLEAN out[21];

BOOLEAN in[21] = 0000000;

ENDDATA;

PROCEDURE EXECUTE USES DEVICE_DATA;

BOOLEAN x = 0;
Example 7. **Example of .jam File to Clear the Contents of the Fault Injection Register for Arria V, Stratix V, and Cyclone V Devices**

```
ACTION ERROR_INJECT_DISABLE = EXECUTE;

DATA DEVICE_DATA;
BOOLEAN out[46];
BOOLEAN in[46] = $000000000000;
ENDDATA;
PROCEDURE EXECUTE USES DEVICE_DATA;
BOOLEAN X = 0;
DRSTOP IDLE;
IRSTOP IDLE;
STATE IDLE;
```
IRSCAN 10, $015;
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 46, in[45..0], CAPTURE out[45..0];  'shift out the previous content
from fault injection register
WAIT IDLE, 10 CYCLES, 1 USEC, IDLE;
DRSCAN 46, in[45..0], CAPTURE out[45..0];  'shift out current content from
fault injection register
WAIT IDLE, 10 CYCLES, 50 USEC, IDLE;
PRINT ";
PRINT "Data read out from the FIR Register: ", 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 "Error Type: ", out[45], out[44], out[43], out[42];
PRINT ";
PRINT "Byte Location: ", out[41], out[40], out[39], out[38], out[37], out[36],
out[35], out[34], out[33], out[32];
PRINT ";
PRINT "Error Byte Value: ", 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];
STATE IDLE;
EXIT 0;
ENDPROC;

Note: Intel recommends reconfiguring the device after error detection testing and
verification.

Related Information
- Running .jam Files with the Intel Quartus Prime Jam Tools on page 25
- Modifying Single-Device .jam Files for Use in a Multi-Device JTAG Chain on page 23

1.5. 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 8. 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**

- [Unloading the EMR using an External Host](#) on page 13
- [Injecting Single-Bit Error for Arria II, Stratix III, and Stratix IV](#) on page 17
- [Injecting Single-Bit Error for Arria V, Stratix V, and Cyclone V Devices](#) on page 19
- [Clearing Fault Injection Register](#) on page 21
1.6. Running .jam Files with the Intel Quartus Prime Jam Tools

You can execute .jam file using the command-line Intel Quartus Prime Jam Tools, `quartus_jli`.

1. Copy the .jam code from the relevant examples in the topics in the related information.
2. Paste the code into a text file and save the file as `<filename>.jam`.
3. To determine the download cable index number, at the command line, run the following command:
   ```
   quartus_jli -n
   ```
4. To execute the .jam file, at the command line, run the following command:
   ```
   quartus_jli -a <action name> -c <cable index> <filename>.jam
   ```
   For example: `quartus_jli -a error_inject -c 2 errortest.jam`

**Related Information**

- Injecting Single-Bit Error for Arria II, Stratix III, and Stratix IV on page 17
- Injecting Single-Bit Error for Arria V, Stratix V, and Cyclone V Devices on page 19
- Clearing Fault Injection Register on page 21
- Unloading the EMR using an External Host on page 13

1.7. Document Revision History for AN 539: Test Methodology of Error Detection and Recovery using CRC in Intel FPGA Devices

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2019.08.09       | • Updated the topic about error injection to specify that EDERROR_INJECT is not designed to test the internal scrubbing feature. Use the Fault Injection Debugger instead.  
• Updated the topic about clearing the fault injection register to specify that the previous example codes are for Arria II, Stratix III, and Stratix IV devices, and add example codes for Arria V, Stratix V, and Cyclone V devices.  
• Added steps to modify .jam file for use in a multi-device JTAG chain.  
• Added steps to execute .jam file using `quartus_jli` command. |
| 2018.05.07       | • Rebranded to Intel.  
• Updated Examples:  
  — Example of .jam File to Inject Single-Bit Error for Arria II, Stratix III, and Stratix IV Devices  
  — Example of .jam File to Inject Single-Bit Error for Arria V, Cyclone V, and Stratix V Devices  
  — Example of .jam File to Clear the Contents of the Fault Injection Register |

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2017</td>
<td>2017.06.14</td>
<td>Corrected Example 5.</td>
</tr>
<tr>
<td>August 2016</td>
<td>2016.08.03</td>
<td>• Corrected Fault Injection Register sub-section title.</td>
</tr>
</tbody>
</table>
| August 2015 | 2015.08.20 | • Fixed editorial error in Fault Injection Register.  
• Fixed Related links descriptions.  
• Removed link to an application note for mature products, AN357. |

*continued...*
### 1. Test Methodology of Error Detection and Recovery using CRC in Intel® FPGA Devices

#### Version History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 2015</td>
<td>2015.05.04</td>
<td>• Updated Example of .jvml File to Inject Single-Bit Error for Arria V, Cyclone V and Stratix V Devices by correcting double quotation type and added semicolon in line 35.&lt;br&gt;• Template update.</td>
</tr>
<tr>
<td>December 2014</td>
<td>2014.12.15</td>
<td>• Updated Figure 3 on page 5.</td>
</tr>
<tr>
<td>April 2014</td>
<td>3.0</td>
<td>• Updated Table 5 on page 11 to correct the .clk(&lt;clock source&gt;) port operation from falling edge to rising edge of the clock.&lt;br&gt;• Updated document to support Arria V and Cyclone V.&lt;br&gt;• Updated Figure 3 on page 5.&lt;br&gt;• Added Example 3 on page 16.</td>
</tr>
<tr>
<td>June 2011</td>
<td>2.0</td>
<td>• Added Figure 3, Figure 10, and Figure 12.&lt;br&gt;• Added Table 4 and Table 9.&lt;br&gt;• Added Example 5.&lt;br&gt;• Added the &quot;Error Correction&quot; section.&lt;br&gt;• Updated Figure 1, Figure 2, Figure 9, and Figure 11.&lt;br&gt;• Updated Table 1, Table 3, and Table 8.&lt;br&gt;• Updated the &quot;Functional Description&quot;, &quot;Configuration Error Detection&quot;, &quot;User Mode Error Detection&quot;, &quot;Error Message Register&quot;, &quot;Fault Injection Register&quot;, and &quot;Error Injection using the EDERROR_INJECT JTAG Instruction&quot; sections.&lt;br&gt;• Minor text edits.</td>
</tr>
<tr>
<td>April 2009</td>
<td>1.1</td>
<td>• Updated the &quot;Introduction&quot;, &quot;Error Injection using EDERROR_INJECT JTAG Instruction&quot;, &quot;Error Injection&quot;, and &quot;Error Detection Pin&quot; sections.&lt;br&gt;• Updated Table 4 and Table 8.</td>
</tr>
<tr>
<td>December 2008</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>