XAPP538 (v1.0) April 4, 2012

Soft Error Mitigation Using Prioritized Essential Bits

Author: Robert Le

Summary

A soft error in configuration memory can corrupt a design. However, the proper operation of a typical design loaded into a Xilinx FPGA only involves a fraction of the total number of configuration memory cells. This application note describes a method for defining the hierarchical regions of interest in a user design and identifying the *prioritized essential bits* associated with the defined user logic using the ISE® design tools, version 13.4 and later. This application note also describes how the LogiCORE™ IP Soft Error Mitigation (SEM) controller can be used with the prioritized essential bits to detect and correct soft errors in the configuration memory of Xilinx® 7 series and Virtex®-6 FPGAs. The described method reduces the effective failures in time (FIT) and increases design availability. The reference design provided with this document illustrates the design flow.

Introduction

Configuration memory is organized as an array of frames, much like a wide static RAM. In many device families, each frame is protected by error correction code (ECC). In all device families, the entire array of frames is protected by cyclic redundancy check (CRC). The two techniques are complementary—CRC is robust for error detection, and ECC provides error location to a high resolution. Every configurable resource in the FPGA is defined by one or more *configuration memory* bits.

A soft error is an unintended change to the state of a configuration memory bit caused by ionizing radiation. If the soft error alters a *critical* bit, the function of the design can be corrupted. (Critical bits are defined here as those bits that cause a functional failure if they change state.) For a typical design loaded into a Xilinx FPGA, only a fraction of the total number of configurable memory cells are used. The memory bits associated with the unused cells are not essential to the design. Therefore, soft error detection and correction should focus only on the configuration bits that are associated with the circuitry of the design. (*Essential bits* are defined here as those bits associated with the circuitry of the design, and are a subset of the device configuration bits.)

Xilinx essential bits technology uses an algorithm to identify which configuration bits are the *essential* bits. If an essential bit is upset, it changes the design circuitry. However, the upset might not affect the function of the design. The Xilinx essential bits technology was developed to provide users of Xilinx FPGAs with an effective method to substantially lower the effective failures in time (FIT) and to increase the availability of the design. The availability of the system is increased, because the system no longer has to treat every upset like a critical failure that takes the system offline. The increased availability helps achieve higher reliability standards.

Xilinx has enhanced the gains offered through essential bits technology by providing a method to priority-filter the essential bits list. This method allows the user to priority-filter the essential bits file based on user-defined hierarchical regions of interest in the design. The bits that are filtered using this method are called *prioritized essential* bits.

Only the designer of the system knows which bits are critical. Generating a complete list of critical bits for a specific design is a time-consuming process that involves validating the correct design behavior while moving an upset through all the configuration memory bits in the design. To realize gains from this methodology, the regions of interest must be either essential or non-essential to the proper operation of the design (that is, there can be no mixing of essential

and non-essential bits in the region). A key state machine in the design is an example of a region of interest that can be priority-filtered as essential to the proper operation of the design, whereas a debug-only module is an example of a region of interest that is non-essential to the proper operation of the design.

The relationship of device configuration bits, essential bits, prioritized essential bits, and critical bits is depicted in Figure 1.

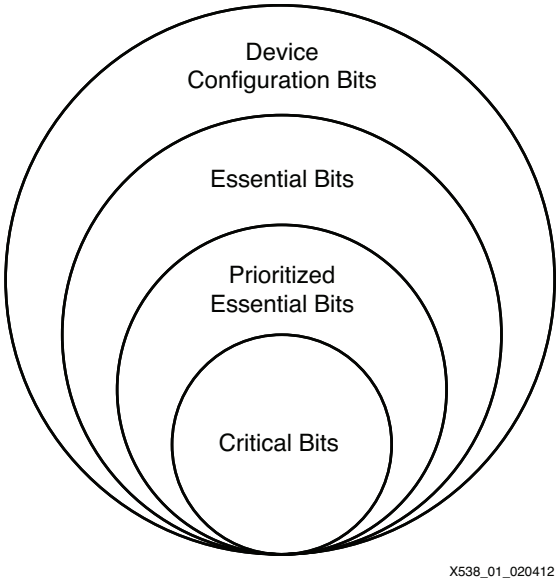

Figure 1: **Relationship of Device Configuration Bits, Essential Bits, Prioritized Essential Bits, and Critical Bits**

The SEM controller is an automatically configured, pre-verified solution that detects and corrects soft errors in the configuration memory of Xilinx FPGAs. The SEM controller does not prevent soft errors. However, it provides a method to better manage system-level effects of soft errors. Proper management of these events can increase design reliability and availability and reduce system maintenance and downtime costs. The SEM controller also supports an optional error classification feature. The error classification feature is a lookup table used to determine if a soft error event has altered essential configuration memory bits. The use of this feature reduces the effective FIT of the design. The cost of enabling this feature is the external storage required to hold the lookup table that contains the essential bits file. This file determines which of the configuration memory bits are essential or non-essential. To use prioritized essential bits, the lookup table content is replaced with the prioritized essential bits file. The design flow is further explained in subsequent sections, which describe how to generate this file using the ISE design tools.

Creating and Using Prioritized Essential Bits

Figure 2 shows a high-level flow diagram for creating and using prioritized essential bits.

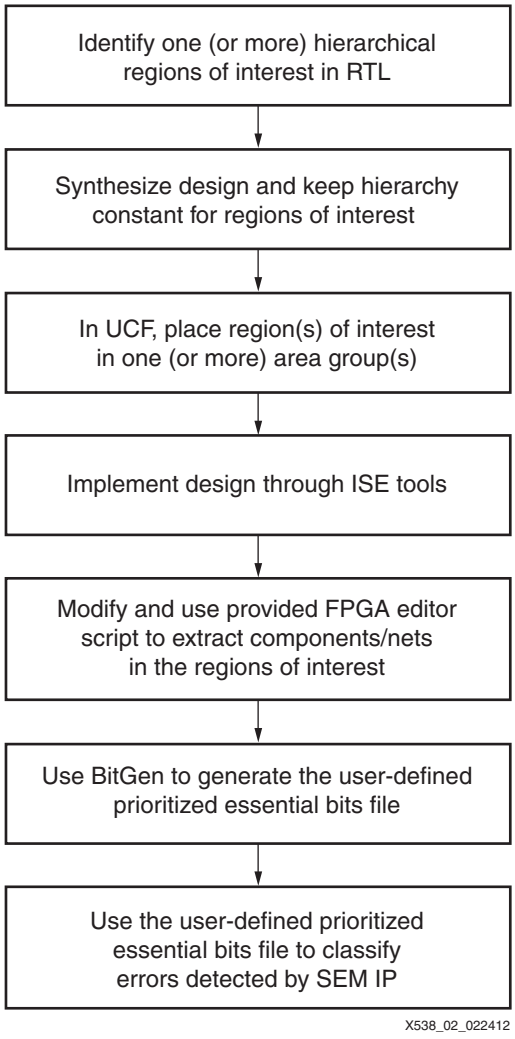

Figure 2: **Creating and Using Prioritized Essential Bits**

The rest of this section describes steps in the design flow in detail.

Identify Hierarchical Regions of Interest in RTL and Synthesize the Design

At the RTL level, the user identfies one or more hierarchical regions of the design that are considered critical to the operation of the design, and partitions the RTL to confine these regions to separate modules. This is the most important and difficult step of the flow. Identification and partitioning depend on the customer design and single-event upset (SEU) mitigation requirements.

To maintain clear hierarchical boundaries between the regions of interest and other non-critical regions, the application of the Keep Hierarchy (KEEP_HIERARCHY) constraint in the RTL is required. The Keep Hierarchy constraint is related to the hierarchical blocks (VHDL entities and Verilog modules) specified in the RTL design. If the hierarchy is maintained during synthesis, the ISE design tools use Keep Hierarchy to preserve the hierarchy throughout implementation.

In the Keep Hierarchy diagram (Figure 3), if Keep Hierarchy is set to the entity or module I2:

- The hierarchy of I2 is kept in the final netlist.
- I4 and I5 can be flattened inside I2.
- I1, I3, I6, and I7 can also be flattened.

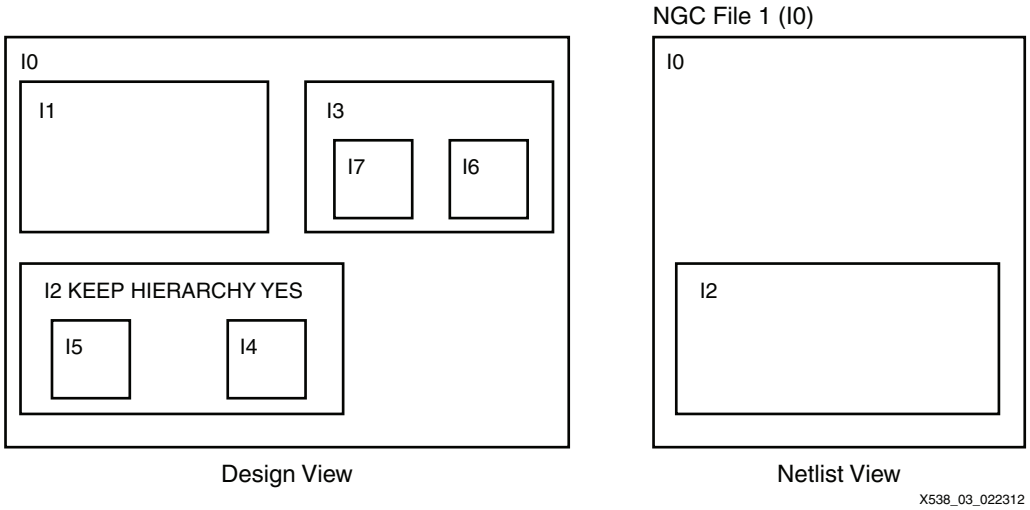

Figure 3: **Keep Hierarchy Diagram**

The Xilinx Synthesis Tool (XST) syntax for defining Keep Hierarchy is:

```
VHDL Syntax:
attribute keep_hierarchy : string ;
attribute keep_hierarchy of instant_name: label is "yes"
Verilog Syntax :
(* keep_hierarchy = "yes" *)
```

The hierarchy of the reference design is shown below:

```
sem_example
 |
 +- sem_core
 |
 +- sem_cfg
 |
 +- sem_mon
 |     |
 |     +- sem_mon_fifo
 |     |
 |     +- sem_mon_piso (region of interest)
 |     |
 |     +- sem_mon_sipo (region of interest)
 |
 +- sem_ext
 |     |
 |     +- sem_ext_byte
 |
 +- sem_hid
 |     |
 |     +- chipscope_icon
 |     |
 |     +- chipscope_vio
 |
 \- BUFG (unisim)
```

In the reference design, the modules sem_mon_piso and sem_mon_sipo are used as an example of regions of interest to illustrate the design flow. The Verilog and VHDL files for these modules are `sem_mon_piso.v/vhd` and `sem_mon_sipo.v/vhd`.

The Keep Hierarchy attributes were applied as follows:

```
Verilog:
module sem_mon (
  …..

  );

  (* keep_hierarchy = "yes" *)
  sem_mon_piso example_mon_piso (
    ……

    );

  (* keep_hierarchy = "yes" *)
  sem_mon_sipo example_mon_sipo (

     ……

    );

 ……
endmodule


VHDL:
entity sem_mon is
port (

  ……

  );
end entity sem_mon;
architecture xilinx of sem_mon is
  ……
attribute keep_hierarchy : string;
attribute keep_hierarchy of example_mon_piso : label is "yes";
attribute keep_hierarchy of example_mon_sipo : label is "yes";
begin
  ……
example_mon_piso : sem_mon_piso
  port map (
     …….

    );
example_mon_sipo : sem_mon_sipo
  port map (
     …….

    );
……
end architecture xilinx;
```

Create Area Constraints for Hierarchical Regions of Interest

The user places identified hierarchical regions of interest into one or more AREA_GROUPs. Area groups are the primary means of placing logic in specific regions of the device, by enabling the partition of the design into physical regions for mapping, packing, placement, and routing. The area groups isolate the regions of interest from other regions of the design.

The user constraints file (UCF) syntax for defining an area group is:

```
INST "X" AREA_GROUP = groupname;
AREA_GROUP "groupname" RANGE = range,
AREA_GROUP "groupname" GROUP = CLOSED;
AREA_GROUP "groupname" PLACE = OPEN;
```

The value “X” is the name of the INST and should match the name of the hierarchical region defined in the RTL. The “groupname” is the name assigned to a group of logical blocks that is to be associated together by the packer and positioned in the area group range by the placer.

The AREA_GROUP with GROUP = CLOSED must be used to prohibit the tools from packing unrelated logic into a single component. This ensures that elements from the logical design can be identified in the physical design by name. The AREA_GROUP with PLACE = OPEN allows placement of unrelated components in the area range.

An example of AREA_GROUP constraints in a UCF is:

```
## Area Constraints on hierarchical region of interest:
INST "example_mon/example_mon_piso/*" AREA_GROUP = "ESSENTIAL_LOGIC" ;
INST "example_mon/example_mon_sipo/*" AREA_GROUP = "ESSENTIAL_LOGIC" ;
AREA_GROUP "ESSENTIAL_LOGIC" RANGE = SLICE_X48Y100:SLICE_X51Y149 ;
## Prohibit addition of unrelated logic into this group...
AREA_GROUP "ESSENTIAL_LOGIC" GROUP = CLOSED ;
## Allow placement of unrelated components in the range...
AREA_GROUP "ESSENTIAL_LOGIC" PLACE = OPEN ;
```

After the UCF has been created, the design can be implemented using the ISE design tools to generate a placed and routed netlist. The provided example implementation script (`implement.sh` or `implement.bat`) shows how to run the design through implementation tools using command lines.

Modify Provided FPGA Editor Script to Extract Components/Nets in the Regions of Interest

After running the design through the standard implementation flow, the user modifies the provided FPGA Editor script (`essential.scr`). The script is used to select the components and nets in the regions of interest and then list these elements before exiting.

The contents of the provided FPGA Editor script delivered with the reference design are:

```
clear
select -k comp *example_mon/example_mon_piso/*
select -k comp *example_mon/example_mon_sipo/*
select -k net *example_mon/example_mon_piso/*
select -k net *example_mon/example_mon_sipo/*
list
clear
exit
```

In this script, the components and nets of the identified regions of interest are selected. In the reference design, the components and nets of interest are:

- `*example_mon/example_mon_piso/*`
- `*example_mon/example_mon_sipo/*`

The user modifies the script to use the hierarchical regions of interest in the user design. The command to run the FPGA Editor script is:

fpga_edline routed.ncd mapped.pcf -p essential.scr

Output from executing the above example generates an fpga_edline log file (`routed_fpga_edline.log`), which can be long. A truncated sample is provided here:

```
#Xilinx FPGA Editor Command Log File
#
#------------------------------
        #Reading routed.ncd...
       #   "sem_example" is an NCD, version 3.2, device xc7k325t, package
fbg900, speed -1
        #Building chip graphics...
        #Loading speed info...
        #1
setattr main edit-mode no-logic-changes
        #2
playback essential.scr
        #clear
        #select -k comp *example_mon/example_mon_piso/*
        #select -k comp *example_mon/example_mon_sipo/*
        #select -k net *example_mon/example_mon_piso/*
        #select -k net *example_mon/example_mon_sipo/*
        #list
        #Comp "example_mon/example_mon_piso/bit_select<1>"
        #Comp "example_mon/example_mon_piso/piso_out"
        #Comp "example_mon/example_mon_piso/tx_start"
        #Comp "example_mon/example_mon_sipo/delay_line<149>"
        #Comp "example_mon/example_mon_sipo/Mshreg_valid_delay_151_0"
        #Comp "example_mon/example_mon_sipo/valid_delay<151>"
        #Net "example_mon/example_mon_piso/hot_delay<15>"
        #Net "example_mon/example_mon_piso/tx_stop
        #Net "example_mon/example_mon_piso/hot_delay<0>"
        #Net "example_mon/example_mon_piso/en_16_x_baud_tx_bit_AND_5_o"
        #Net "example_mon/example_mon_piso/bit_select<2>"
        #Net "example_mon/example_mon_piso/bit_select<1>"
        #clear
        #exit
```

Use BitGen to Generate the Prioritized Essential Bits File

BitGen is invoked with these command line options:

-g essentialbits:yes

-g essentialbitsfilter:{none|mask|enable}

-g essentialbitsfilterfile:<filename>

- If **essentialbitsfilter** is set to **enable**, the operation of BitGen generates essential bits associated with design entities whose names only match those specified in the **essentialbitsfilterfile** option. This option is used in the reference design.
- If **essentialbitsfilter** is set to **mask**, the generated essential bits exclude the bits associated with design entities whose names match those specified in the **essentialbitsfilterfile** option.
- If **essentialbitsfilter** is set to **none**, the generated essential bits contain the bits associated with the whole design.

The **filename** used in **-g essentialbitsfilterfile** is the FPGA edline log file generated by the FPGA Editor script mentioned in Modify Provided FPGA Editor Script to Extract Components/Nets in the Regions of Interest. BitGen accepts the FPGA edline log file

as created (no user modification is required, although manual additions and deletions are possible) and generates the EBD file. The generated EBD file contains the bits that are identified as prioritized essential bits.

Use SEM Controller in Conjunction with Prioritized Essential Bits

The SEM controller supports an optional Error Classification feature. After a soft error is detected and located by the SEM controller, the location of the error is looked up in the generated essential bits data stored in serial peripheral interface (SPI) flash memory. This classification feature enables customers to determine whether a soft error is essential. Xilinx has provided a sample design with the SEM controller including the SPI Flash interface. To use the prioritized essential bits, the lookup table content is replaced with the prioritized essential bits file.

To enable this option, the **enable_classification** parameter of the SEM controller should be set to **true**. The provided reference design uses the core.xco file with these parameters:

```
# BEGIN Parameters
CSET clock_freq=66.0
CSET component_name=sem_core
CSET correction_method=repair
CSET enable_classification=true
CSET enable_correction=true
CSET enable_injection=true
CSET injection_shim=chipscope
CSET retrieval_shim=spi_flash_x1
# END Parameters
```

The SEM controller is generated using the generate_cores.sh or generate_cores.bat scripts provided in the implement directory of the reference design. The command line to generate the SEM controller using the CORE Generator™ tool and core.xco file is:

```
coregen -p coregen.cgp -b sem_core.xco
```

In a typical design, the simplest way to generate the SEM controller is to open the CORE Generator tool and generate the core using the GUI. See *LogiCORE IP Soft Error Mitigation Controller 3.1 User Guide* [Ref 1] for detailed information on core configurations, usage, and the SPI flash interface.

An example design including the SPI flash interface is generated together with the SEM controller. A TCL script (makedata.tcl) is also generated, which processes the BitGen output files and generates three output files:

- An Intel hexadecimal data file (MCS) for programming SPI flash devices
- A raw binary data file (BIN) for programming SPI flash devices
- An initialization file (VMF) for loading SPI flash simulation models

The output files are generated at the command line as follows:

```
xtclsh makedata.tcl -ebd routed.ebd datafile
```

Reference Design

The reference design files for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=183797

Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer Name	Robert Le
Target devices (stepping level, ES, production, speed grades)	Virtex-6 FPGAs and 7 series FPGAs
Source code provided	Yes
Source code format	VHDL, Verilog
Design uses code/IP from existing Xilinx application note/reference designs, CORE Generator tool, or third party	SEM_V3_1 or higher
Simulation	
Functional simulation performed	N/A
Timing simulation performed	N/A
Test bench used for functional and timing simulations	N/A
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version used	XST, ISE® Design Suite 13.4
Implementation software tools/versions used	ISE Design Suite 13.4
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	No

The reference design is based on the generated example design of the IP SEM controller v3.1. The modules in the provided reference design are functionally the same as the modules in the example design of the SEM controller. The Keep Hierarchy attributes are applied to the modules sem_mon_piso and sem_mon_sipo, and the area group constraints on these two modules are added in the UCF to illustrate the design flow. The reference design uses the sem_mon_piso and sem_mon_sipo modules as examples of regions of interest and generates the prioritized essential bit file for these regions.

The reference design can be implemented using the `implement.bat` or `implement.sh` script provided in the `implement` directory of the reference design.

The structure of the reference design is:

- XAPP538/<Kintex-7 or Virtex-6>/<Verilog or VHDL>/
  - `implement`: Implementation scripts
  - `reference_design_source`: Source code and UCF for the reference design
  - `sem_core`: SEM controller core
  - `coregen.cgp`: project file for the CORE Generator tool

The parameters for the SEM controller are:

- Component name: sem_core
- Error injection: Enabled
- Error injection shim: ChipScope
- Error correction: Enabled
- Error correction method: Repair
- Error classification: Enable
- Controller clock frequency: 66 MHz (Kintex-7 device) or 100 MHz (Virtex-6 device)

Conclusion

This application note describes the design flow in the ISE design tools that can be used to lower the effective FIT and increase the availability of the design. This design flow enables customers to reach higher levels of availability by allowing the system to devise what action to take, depending on what part of the design is affected. Using this design flow creates a large number of upsets that be consciously ignored, and because these upsets pose no threat to the operation of the design, the effective FIT rate is significantly lowered.

References

This document uses the following references:

1. UG764, *LogiCORE IP Soft Error Mitigation Controller 3.1 User Guide*
2. UG360, *Virtex-6 FPGA Configuration User Guide*
3. UG470, *7 Series FPGAs Configuration User Guide*

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/04/2012	1.0	Initial Xilinx release.

Notice of Disclaimer

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.