XAPP1247 (v1.0) August 13, 2015

MultiBoot with 7 Series FPGAs and SPI

Authors: Bryan Li and Kiran K Gakhal

Summary

This application note covers the key concepts for building a successful MultiBoot design with 7 series FPGAs in serial peripheral interface (SPI) configuration mode. 7 series MultiBoot features allows the FPGA application to load two or more FPGA bitstreams under the control of the FPGA application. Step-by-step instructions to implement the MultiBoot feature using FPGA bitstream settings, a method to trigger fallback, and details on how to use FPGA status registers for debugging and verifying fallback operation are discussed. This application note includes a reference design that demonstrates MultiBoot and Fallback capabilities of the 7 series FPGAs using SPI configuration mode.

The Kintex®-7 FPGA and N25Q128 Micron Quad SPI serial flash on the KC705 evaluation board are used along with Vivado® Design Suite 2015.1 to demonstrate the design flow. The *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] provides additional information regarding the MultiBoot feature and details on the SPI configuration mode.

Download the Reference Design files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design.

Introduction

The 7 series FPGAs MultiBoot and Fallback features support updating systems in the field. Bitstream images can be upgraded dynamically in the field. The FPGA MultiBoot feature enables switching between images on-the-fly. When an error is detected during the MultiBoot configuration process, the FPGA can trigger a Fallback feature that ensures a known good design can be loaded into the device.

When Fallback occurs, an internally generated pulse resets the entire configuration logic, except for the dedicated MultiBoot logic, the warm boot start address (WBSTAR) register, and the boot status (BOOTSTS) registers. This reset pulse pulls INIT_B and DONE Low, clears the configuration memory, and restarts the configuration process from address 0 of the flash memory device.

This application note is divided into the following key sections:

- MultiBoot Basics
- Internal PROGRAM (IPROG) Command Embedded in the Bitstream
- FPGA SPI Flash Configuration Interface
- Vivado Tools Flow
- Validation in Hardware
- Debug and Checklist
- Appendix A: Advanced Applications
  - Fallback using a Timeout Error

MultiBoot Basics

The FPGA MultiBoot feature enables switching between images on-the-fly for remote updates. When an error is detected during the MultiBoot configuration process, the FPGA can trigger a Fallback that ensures that a known good design can be loaded into the device. The solution involves a flash memory that has a reserved areas to store these components:

- Fallback, or “golden bitstream”
- MultiBoot, or “update bitstream”

A remote programming method is implemented and is used to program new or enhanced bitstreams into the update bitstream area. The FPGA preferably configures with the update bitstream.

If the remote update procedure fails or is interrupted, the FPGA must be able to reliably Fallback and configure from the golden bitstream.

The MultiBoot feature enables the FPGA to selectively load a bitstream from a specified address in flash memory. An internally generated pulse (IPROG) initiates the configuration logic to jump to the update bitstream at the address location specified in the golden bitstream WBSTAR (Warm Boot Start Address) register, and attempts to load the update bitstream. If a configuration error is detected during the update bitstream load, then Fallback is triggered to load the golden bitstream. Figure 1 shows general MultiBoot and Fallback flow demonstrated in this application note.

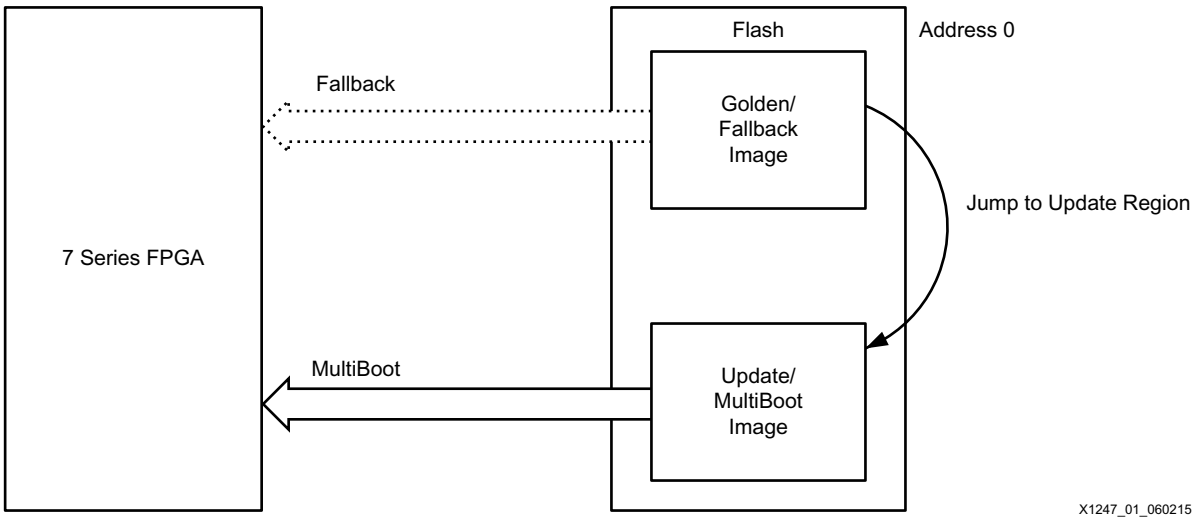

Figure 1: **MultiBoot and Fallback Flow**

In preparation for a MultiBoot event, internal PROGRAM (IPROG) and next image start address (WBSTAR) commands are set in the golden bitstream.

Two approaches to implement MultiBoot are:

1. PROG embedded in the bitstream (focus of this application note). IPROG is enabled using the NEXT_CONFIG_ADDR bitstream option.
2. IPROG using ICAPE2 (not covered in this application note): Apply the register write commands to the ICAPE2 primitive.

IPROG embedded in the bitstream is an automated option – MultiBoot settings are embedded in the bitstream whereas the second option to use ICAPE2 is user-application specific where the design triggers a MultiBoot based on some event that occurs.

Internal PROGRAM (IPROG) Command Embedded in the Bitstream

The warm boot start address (WBSTAR) and internal PROGRAM (IPROG) command can be embedded in the bitstream.

The golden bitstream is stored at flash address 0. The update bitstream is stored at the flash address specified in the WBSTAR (next_config_addr) register in the golden bitstream. The IPROG is automatically embedded in the bitstream when the WBSTAR is set to any address value other than default.

Configuration logic starts executing commands in the golden bitstream stored at flash address 0 as normal. Once it reaches the IPROG command the control jumps to the flash address location specified in the WBSTAR register in the golden bitstream, and configuration logic attempts to load the update bitstream. If configuration logic fails to load the update image due to an error condition, Fallback occurs and configuration logic pulls INIT_B and DONE Low, clears

the configuration memory, and restarts the configuration process by loading the golden bitstream at flash address 0. During Fallback, the FPGA ignores the WBSTAR and IPROG commands. For further details, refer to chapter 7, *Reconfiguration and MultiBoot* in the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1]. See Figure 2 for flash memory components and configuration steps described above.

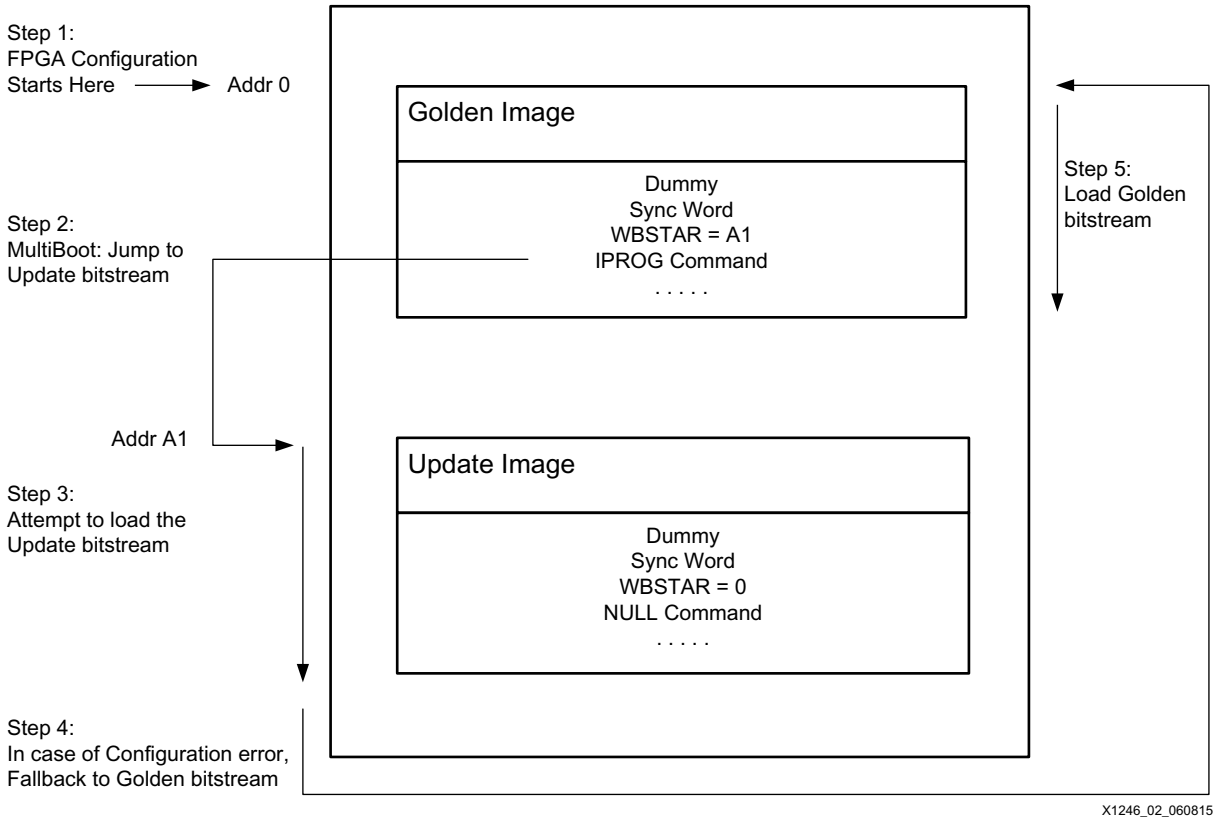

Figure 2: **MultiBoot Fallback Flash Memory Components and Configuration Steps**

FPGA SPI Flash Configuration Interface

Figure 3 shows the basic connectivity between 7 series FPGAs and the SPI flash with a x1 data width. The read and address instructions are sent from the FPGA to the SPI flash via the master-out-slave-in (MOSI) pin. The data is returned from the SPI flash via the master-in-slave-out (MISO) pin. SCK is the clock pin and SS is the active Low slave select pin.

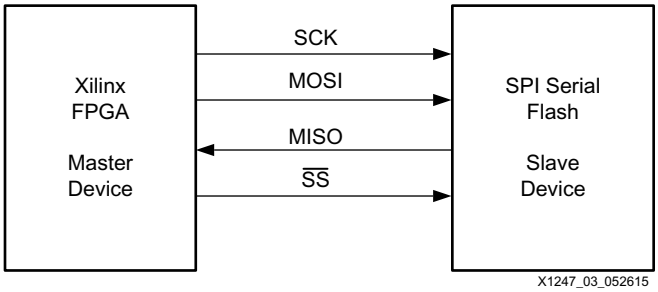

Figure 3: **FPGA Flash Interface**

Refer to the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] or *Using SPI with 7 Series FPGAs* (XAPP586) [Ref 2].

Vivado Tools Flow

Preparing the Bitstreams for MultiBoot Application

This section outlines the bitstream properties required to create golden and update bitstreams for MultiBoot application. For bitstream options not specified, use the default settings.

Table 1 outlines the basic MultiBoot bitstream properties to generate golden and update bitstreams with a description of each property. For detailed description of these properties, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 3].

Table 1: **MultiBoot Bitstream Properties**

Bitstream Properties	Value	Golden	Update	Description
BITSTREAM.CONFIG.CONFIGFALLBACK	Enable	√	√	Enables the loading of default bitstream when configuration attempt fails
BITSTREAM.CONFIG.NEXT_CONFIG_ADDR	Addr A1	√	N/A	Sets the Warm Boot Start Address (WBSTAR[28:0] bits) register with start address for the next configuration image
BITSTREAM.GENERAL.COMPRESS(1)	Enable	√	√	Specifies that FPGA bitstream file compression is enabled

Notes:

1. BITSTREAM.GENERAL.COMPRESS is optional, but it is selected because it helps to reduce the programming and configuration time.

RECOMMENDED: *If using the Vivado Graphical User Interface (GUI) flow, Include MultiBoot properties in the golden and update design XDC file prior to synthesizing and implementing designs in the Vivado Design Suite. This saves you time as you only need to run Synthesis and Implementation once whereas if you add these properties in Vivado bitstream settings GUI window after design implementation, you will need to rerun the complete implementation flow again.*

Open the constraints file (.xdc) for your golden design implementation in Vivado. Copy-paste the following into the constraints file and then save the changes you made to the .xdc file:

```
set_property BITSTREAM.CONFIG.CONFIGFALLBACK ENABLE [current_design]
set_property BITSTREAM.CONFIG.NEXT_CONFIG_ADDR 0x0400000 [current_design]
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 1 [current_design]
```

Note: BITSTREAM.CONFIG.NEXT_CONFIG_ADDR `0x0400000` is an example from the reference design. You can set this address to any other value depending on your SPI flash size.

Note: By default SPI_BUSWIDTH is x1, make sure to set this property if you are not using default x1 mode.

Next, you can open the constraints file (.xdc) in your update design and add the following bitstream properties into the constraints file and then save:

```
set_property BITSTREAM.CONFIG.CONFIGFALLBACK ENABLE [current_design]
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 1 [current_design]
```

Note: By default SPI_BUSWIDTH is x1, make sure to set this property if you are not using default x1 mode.

After adding MultiBoot properties, generate bitstreams for the golden and update designs using the `write_bitstream` Tcl command. To generate a bitstream the project must have an implemented design open. You can use `-verbose` switch with `write_bitstream` to summarize all bitstream options used. For more help use `write_bitstream -help` command:

```
write_bitstream  -verbose <file_name>
```

Generating the SPI Flash Programming File

Use the `write_cfgmem` Tcl command to create the flash programming file (.mcs). `write_cfgmem` takes an FPGA bitstream (.bit) and generates a flash file (.mcs) that can be used to program the SPI flash.

For example, generate a flash programming file (.mcs) file with two FPGA bitstreams (.bit files) as:

```
write_cfgmem -format mcs -interface SPIX1 -size 16 -loadbit "up 0 <path>/golden.bit up
0x0400000 <path>/update.bit"  <path>/filename.mcs
```

Note: Address value `0x0400000` is an example, as used in the reference design. The Addr A1 value set in the golden image (start address of update image) should be used (see Table 1).

Refer to the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 3] or use the `-help` command in Vivado for a detailed description of each `write_cfgmem` command option:

```
write_cfgmem -help
```

Validation in Hardware

The reference design includes flash programming files (.mcs) that can be used for quick verification.

To verify whether the FPGA boots from the golden or update image observe the status of the GPIO (General Purpose Input/Output) LEDs. GPIO LEDs are near the power switch on the KC705 board.

- For the golden Image - GPIO LEDs [7:0] scroll from right to left
- For the update Image - GPIO LEDs [7:0] scroll from left to right

1. Ensure that the appropriate configuration mode (Master SPI) is selected for switch SW13 on the KC705 board as shown in Figure 4. For SPI configuration mode, mode pins M[2:0] are set to `001`.

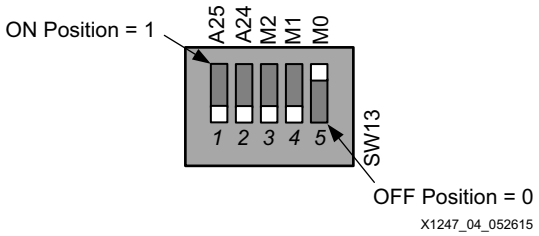

Figure 4: **KC705 SW13: Mode Pin Settings**

2. Connect the JTAG cable and power on the KC705 board.
3. Launch Vivado Hardware Manager and connect to the demo board. Next, program the flash image (.mcs) you just generated in the previous section, or use the `kc705_multiboot_spi.mcs` file from the *ready_to_download* files provided with reference design. For instructions on how to program the Quad SPI flash on the KC705 board, refer to the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 3].
4. After SPI flash programming completes successfully, boot the FPGA from flash by pulsing PROGRAM_B pin (PROG_B button, SW14). Alternatively, you can right-click on the device in Hardware Manager to use the Vivado GUI option **Boot from Configuration Memory Device** (see Figure 5).

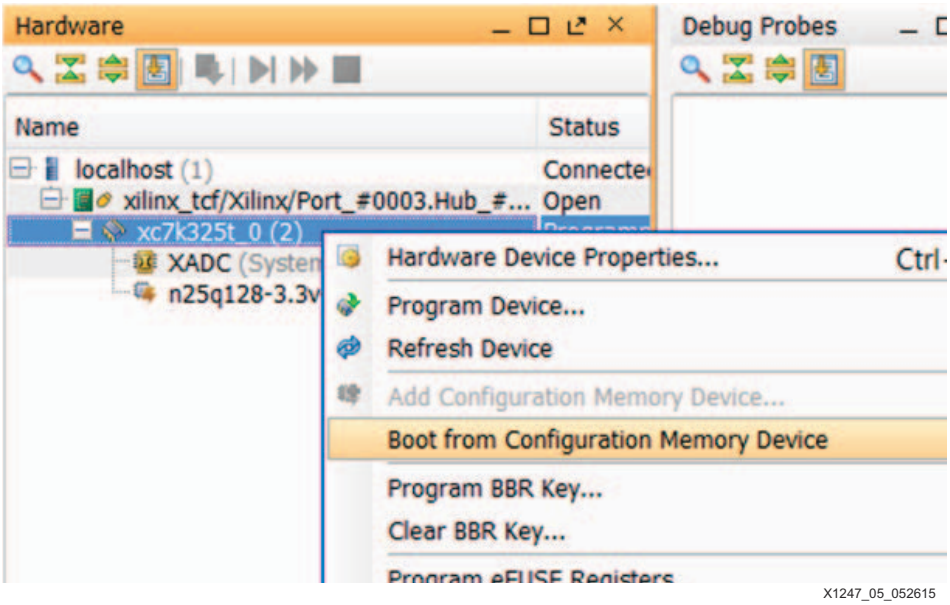

Figure 5: **Vivado Hardware Manager - Booting from Configuration Memory Device**

5. Observe DONE pin status on the board. DONE goes High after successful configuration of the FPGA from SPI Flash.
6. Verify that GPIO LEDs [7:0] on the KC705 board are blinking sequentially from left to right: LED[7] > LED[6] > LED[5] > LED[4] > LED[3] > LED[2] > LED[1] > LED[0]), indicating that the update image has loaded successfully.

   You can also check the BOOT_STATUS register under device properties to verify that the INTERNAL_PROG (IPROG) flag that initiated MultiBoot jumps to the update bitstream (see Figure 6). See the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] for BOOT_STATUS register details.

   Note: Refresh the device by right-clicking the FPGA in the window before checking device Properties.

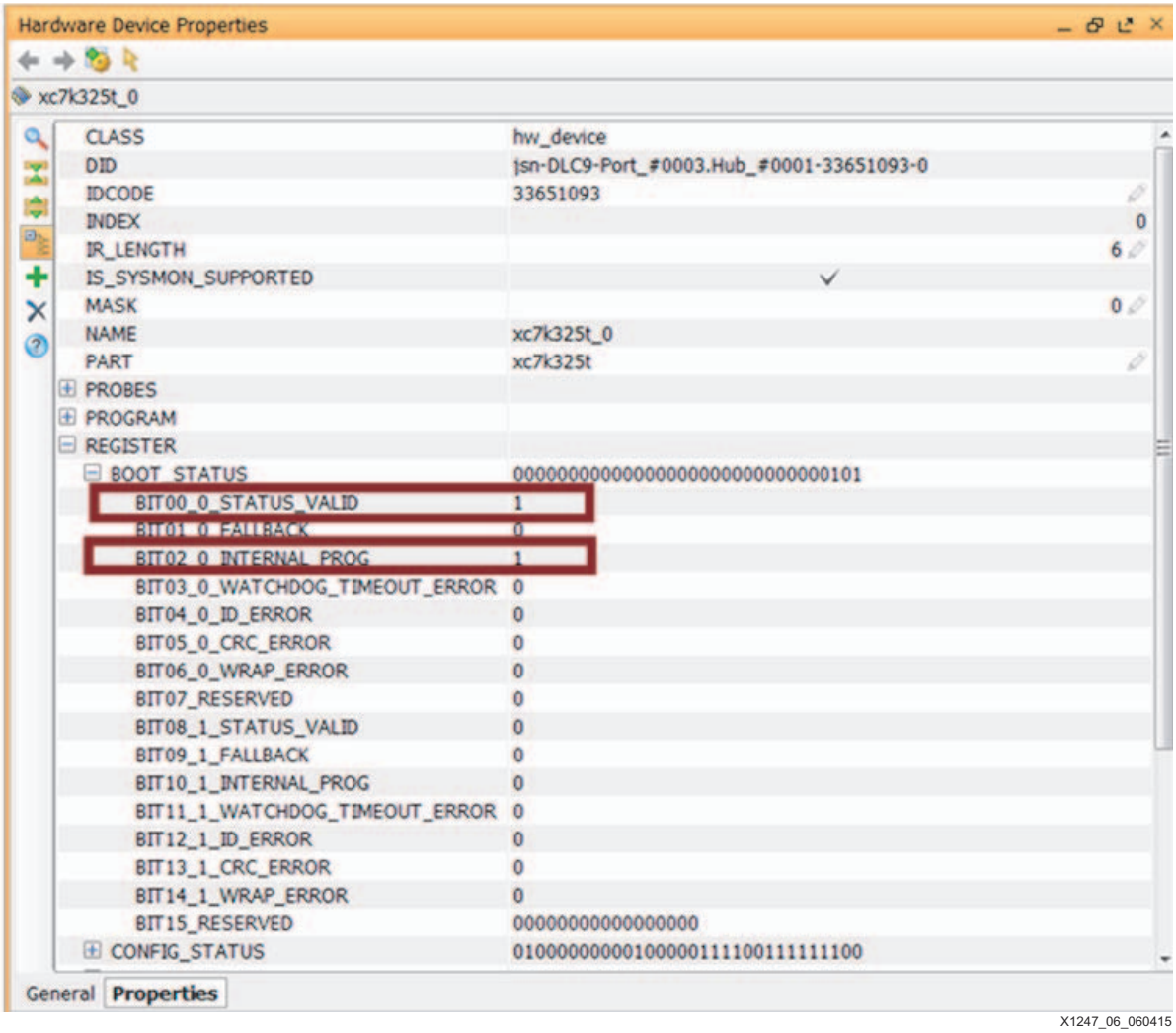

Figure 6: **BOOT_STATUS after Update Image Load Showing STATUS_VALID and INTERNAL_PROG Flags**

Fallback – Triggered by CRC error

Fallback to the golden image can be triggered by different means. See the *Reconfiguration and MultiBoot* chapter in UG470 [Ref 1] for more information.

This application note demonstrates Fallback triggered by a CRC error. You can corrupt the update bitstream manually to induce a CRC error. There are many locations between the RESET CRC command and the CRC command where you can flip bits. Figure 7 and Figure 8 show an example. See the *Bitstream Composition* section of in UG470 [Ref 1] for more information on the bitstream format.

1. Open the update bitstream (.bit) in any HEX editor and towards the middle of the bitstream flip some data bytes, for example from `00` to `11` as highlighted in Figure 7 and Figure 8.

```
Top_MultiBoot_Module_B.bit
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000540  00 00 00 00 00 00 00 00 00 00 00 00 00 30 00 20  .............0.
00000550  01 00 00 00 12 30 01 40 04 00 00 00 00 00 00 00  .....0.@........
00000560  00 00 00 00 00 00 00 00 00 30 00 20 01 00 00 00  .........0. ....
00000570  13 30 01 40 04 00 00 00 00 00 00 00 00 00 00 00  .0.@............
00000580  00 00 00 00 00 30 00 20 01 00 00 00 14 30 01 40  .....0. .....0.@
00000590  04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..........
000005A0  00 30 00 20 01 00 00 00 15 30 01 40 04 00 00 00  .0. .....0.@....
000005B0  00 00 00 00 00 00 00 00 00 00 00 00 00 30 00 20  .............0.
000005C0  01 00 00 00 16 30 01 40 04 00 00 00 00 00 00 00  .....0.@........
```

X1247_07_052615

Figure 7: **Original Update Image (Top_MultiBoot_Module_B.bit)**

```
Top_MultiBoot_Module_B_corrupted.bit
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000540  00 00 00 00 00 00 00 00 00 00 00 00 00 30 00 20  .............0.
00000550  01 00 00 00 12 30 01 40 04 00 00 00 00 00 00 00  .....0.@........
00000560  00 00 00 00 00 00 00 00 00 30 00 20 01 00 00 00  .........0. ....
00000570  13 30 01 40 04 00 00 00 00 00 00 00 00 00 00 00  .0.@............
00000580  00 00 00 00 00 30 00 20 01 00 00 00 14 30 01 40  .....0. .....0.@
00000590  04 00 00 00 00 11 00 00 00 00 00 00 00 00 00 00  ..........
000005A0  00 30 00 20 01 00 00 00 15 30 01 40 04 00 00 00  .0. .....0.@....
000005B0  00 00 00 00 00 00 00 00 00 00 00 00 00 30 00 20  .............0.
000005C0  01 00 00 00 16 30 01 40 04 00 00 00 00 00 00 00  .....0.@........
```

X1247_08_060215

Figure 8: **Corrupted Update Image (Top_MultiBoot_Module_B_corrupted.bit)**

2. Save the corrupted update bitstream and generate a new flash programming file (.mcs) with this corrupted bitstream (see Generating the SPI Flash Programming File, page 6 for information on the `write_cfgmem` command):

   ```
   write_cfgmem -format mcs -interface SPIX1 -size 16 -loadbit "up 0 <path>/golden.bit up
   0x0400000 <path>/corruputed_update.bit" <path>/filename.mcs
   ```

   Note: The CRC checks data and commands in the bitstream between the Reset CRC (RCRC) command and the final CRC check before EOS (end of startup). See the *Bitstream Composition* section in UG470 [Ref 1] for more information. Therefore configuration logic can only catch a CRC error when the data or command is corrupted between the RCRC and CRC commands in the bitstream.

3. Once the new flash image (.mcs) file is created, repeat steps 1-6 in Validation in Hardware, page 7 to program the SPI flash with the corrupted image.

   In step 3, you can use the `kc705_multiboot_corrupted_spi.mcs` file included in *Ready_to_download* directory of the reference design for verification.

4. You can verify that Fallback operation was successful and the golden image was loaded into the FPGA by means of GPIO LEDs [7:0], which should blink sequentially from right to left: LED[0] > LED[1] > LED[2] > LED[3] > LED[4] > LED[5] -> LED[6] -> LED[7].

You can also view the BOOT_STATUS register in the Vivado tool to verify the design completed fallback successfully:

a. The BIT10_1 and BIT13_1 flags show that IPROG was attempted, and a CRC_ERROR was detected.

b. The BIT01_0 flag shows that the fallback bitstream loaded successfully.

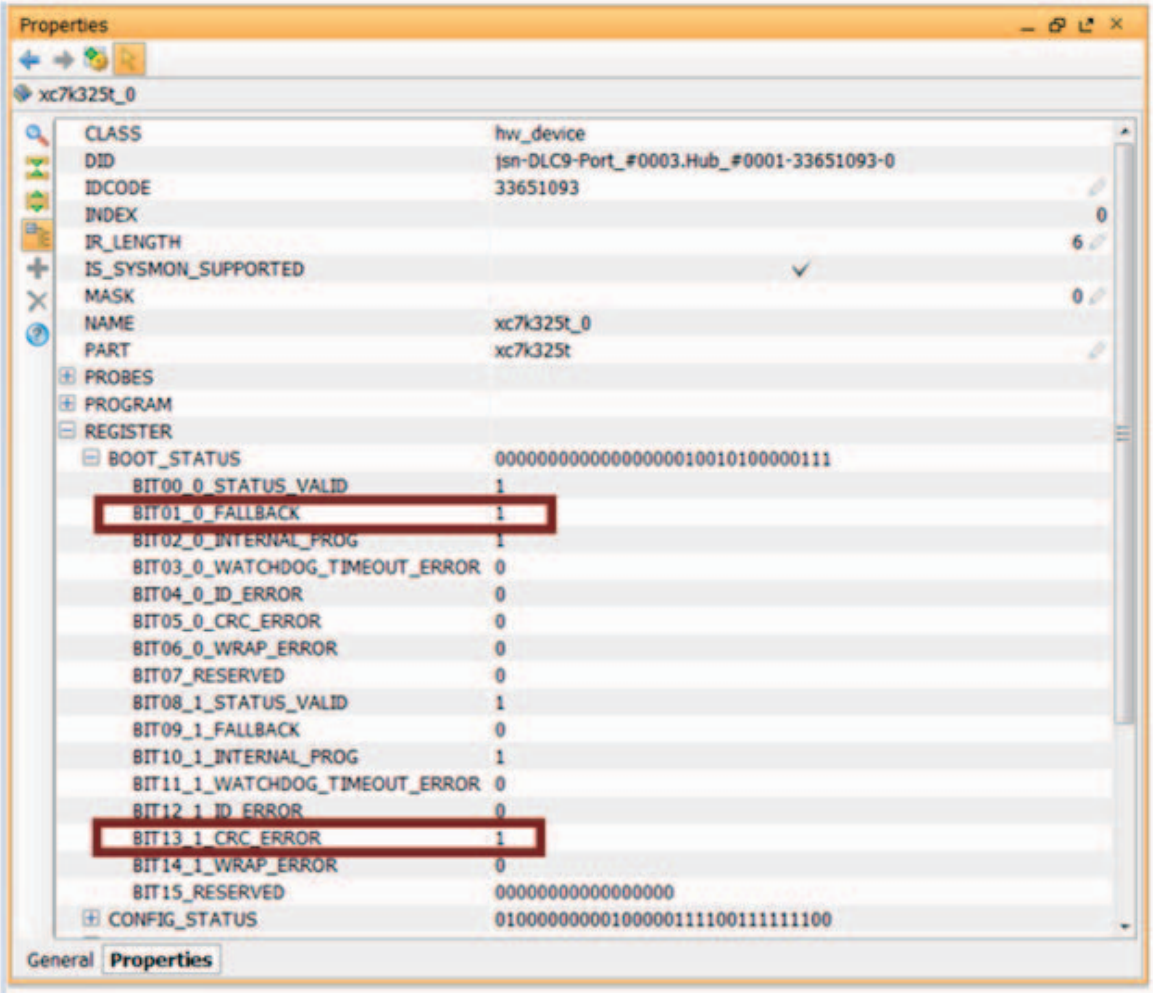

Figure 9: **BOOT_STATUS Registers Showing Fallback and CRC Flags**

Debug and Checklist

File Generation

1. If using the Vivado graphical user interface (GUI) method to set MultiBoot properties, ensure that the golden and update designs are successfully synthesized and implemented before you apply MultiBoot properties.
2. Ensure that the MultiBoot bitstream properties are set correctly for the golden and update images.
3. Ensure that the CONFIGRATE option does not exceed the maximum frequency supported by the target flash, See the *Determining the Maximum Configuration Clock Frequency* section in UG470 [Ref 1].
4. While generating the MCS file, ensure that the data bus width `-interface` option is set correctly.
5. Ensure that the update image start address is same as specified for the NEXT_CONFIG_ADDR property in the golden design. Setting it lower increases FPGA programming time.

Configuration

1. Ensure that your design works as expected without adding MultiBoot properties. This is a good check to debug any possible programming failure issues and helps to determine the root cause, whether the issue is with the design itself or the MultiBoot settings.
2. Erase the flash device completely before programming with the new flash programming (.mcs) file. Use the “Blank Check” operation to verify the erase operation.
3. Capture BOOT_STATUS and CONFIG_STATUS register data for additional debug information. Refresh the device before capturing device properties.
4. Ensure that the flash golden image region is protected at all times and does not change. You can only update and change the update image region.

Reference Design

You can download the Reference Design Files for this application note from the Xilinx® website. Table 2 shows the reference design matrix.

Table 2: **Reference Design Matrix**

Parameter	Description
General	
Developer Name	Xilinx
Target Devices	7 Series FPGAs
Source code provided	Yes
Source code format	VHDL
Design uses code and IP from existing Xilinx Application note and reference designs, CORE Generator software, or third party	N/A
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	Vivado Design Suite version 2015.1
Implementation software tools/versions used	Vivado Design Suite version 2015.1
Static timing analysis performed	No
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KC705 board and 128 Mb Micron Quad-SPI Flash

Conclusion

This application note demonstrates how to enable the MultiBoot feature in 7 series FPGAs to configure the FPGA with different bitstreams stored in the SPI flash. A complete reference design with quick verification files to demonstrate MultiBoot and Fallback features is provided.

Appendix A: Advanced Applications

Fallback using a Timeout Error

The following errors can trigger fallback:

- CRC error
- IDCODE error
- Watchdog timer time-out error

Fallback trigger by a CRC error is covered in Validation in Hardware, page 7. See the 7 *Series FPGAs Configuration User Guide* (UG470) [Ref 1] for further details on these fallback errors.

To implement a robust in-system update solution you need to ensure that the golden image is protected at all times in flash memory, only the update image needs to be changed.

During the MultiBoot operation, the IPROG command embedded in the golden bitstream initiates the jump to the address location specified in the WBSTAR register, and configuration logic starts searching for the next SYNC word to load the bitstream. Once the SYNC word is detected, configuration logic starts listening to commands and data following the SYNC word to configure the FPGA. If there is no update image present at the next_config_addr location or if the SYNC word in the update image is corrupted, configuration logic scans through the entire flash memory searching for a valid SYNC word.

For corrupted update region scenarios in serial flash you need to have a way to set the Watchdog timer to help trigger Fallback to the golden region. See the *Watchdog Timer* section in UG470 [Ref 1] for more information.

Setting the Watchdog Timer

One option is to use the *BITSTREAM.CONFIG.TIMER_CFG* property in both the golden and update images to enable timer value in the bitstream. Set the following bitstream *Timer* property in the golden and update image XDC file along with properties listed in Table 1, page 5.

```
set_property BITSTREAM.CONFIG.TIMER_CFG <Timer Value> [current_design]
```

In 7 series FPGAs, the watchdog timer uses a divided version of the dedicated internal clock, CFGMCLK. Timer value should be adequate to cover the entire FPGA configuration time until startup is complete. Any wait time in startup for DCI match, MMCM lock, or DONE should also be included. Therefore calculating correct timer value requires some effort based on these conditions.

This application note provides a solution to use separate barrier or timer images instead of using the bitstream property to enable the watchdog timer. Using barrier or timer images eliminates the requirement to calculate timer value for the *TIMER_CFG* bitstream property.

A Tcl script, provided with the reference design, can be used to generate barrier images and an address map to place these images along with the golden and update images in flash memory. Flash memory components and configuration steps to implement timer images in a MultiBoot application are shown in Figure 10.

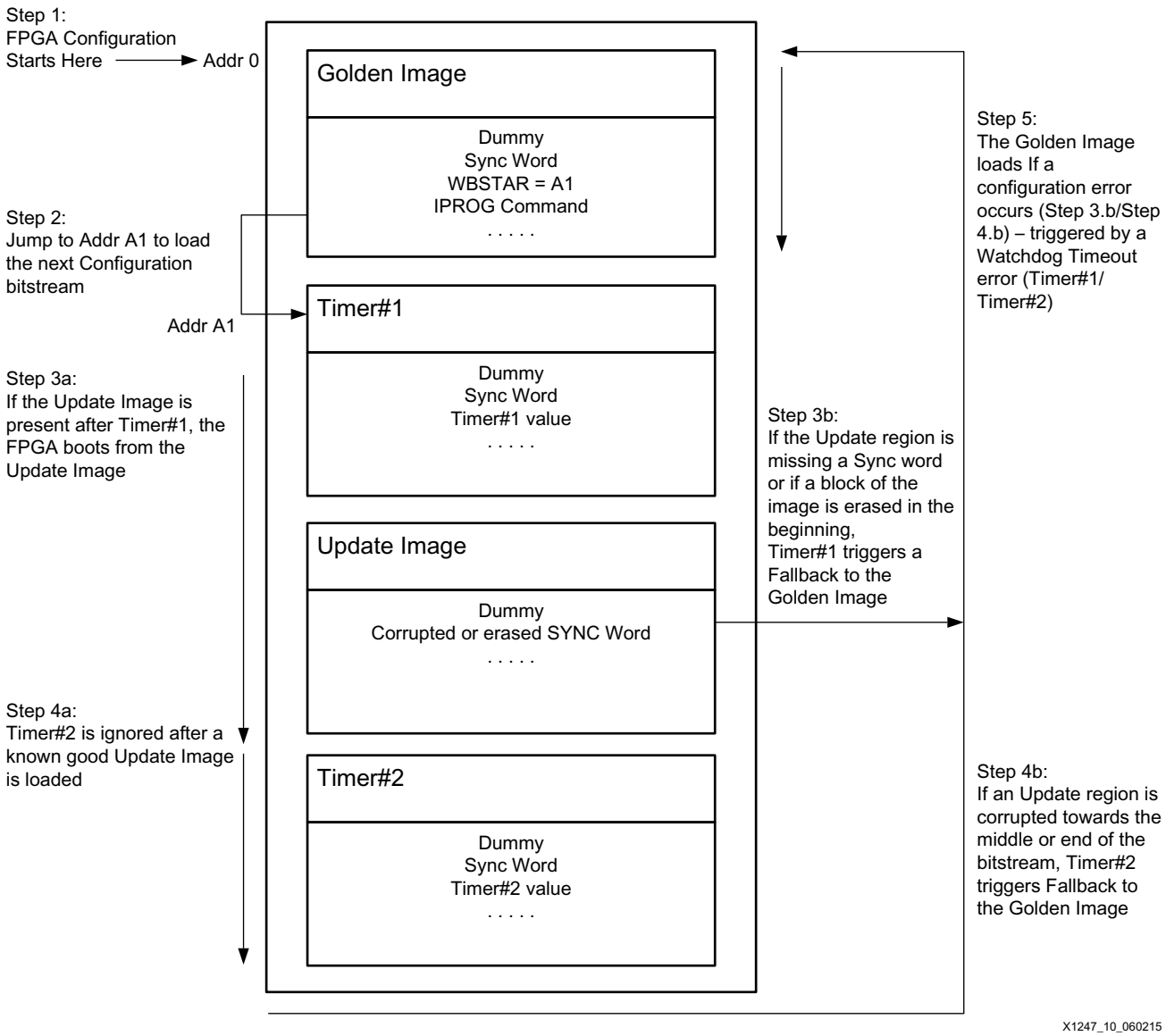

Figure 10: **MultiBoot, Fallback, and Barrier Flash Memory Components and Configuration Steps**

IMPORTANT: *You need to ensure that golden, timer1, and timer2 regions in SPI flash memory are secured at all times, only the update image region can be modified.*

The basic process for design implementation with barrier images as outlined in Figure 10 is as follows:

1. Configuration logic starts executing commands in the golden bitstream stored at flash address location zero. The IPROG command embedded in the golden bitstream initiates the control to jump to the address location stored in the golden WBSTAR register. In this example WBSTAR points to the timer1 start address.

2. The timer1 or the barrier image before the update image has a short timer enabled that helps sync the configuration logic to a known good SYNC word, reducing the chance a corrupted update image can cause a false sync.
3. a. Known good update image present following timer1 – Update image loads as normal.

   b. Corrupted or sector of data erased i.e., no SYNC word (AA995566) in the beginning of the update image – Configuration logic starts observing the bitstream after it sees the SYNC (AA995566) word. In the case of a corrupted or erased SYNC word, data or commands are ignored and configuration logic keeps scanning through the flash for a valid SYNC word. In this scenario the timer value set in timer1 triggers Fallback by means of a watchdog timeout error and failsafe or golden image is loaded (proceed to Step5).
4. a. timer2 is ignored if a known good update image is present following timer1.

   b. Update region corrupted or sector of data erased towards the end of update image – In this scenario configuration logic does not see the end of startup to complete configuration. timer2 comes into play and Fallback to address zero i.e., the golden image location is triggered by a watchdog timeout error.
5. The golden failsafe image loads as a result of Fallback triggered by the timeout error.

Barrier Composition

The barrier or timer images created using the script are a group of commands to synchronize configuration logic and set the configuration timer value. Table 3 identifies the basic commands included in the barrier image that enable the TIMER register.

Table 3: **Barrier Image Composition**

FFFFFFFF	Dummy pad word
000000BB	Bus width auto detect, word 1
11220044	Bus width auto detect, word 2
FFFFFFFF	Dummy pad word
FFFFFFFF	Dummy pad word
AA995566	Sync word
20000000	NOOP
20000000	NOOP
30022001	Packet Type 1 command: Write TIMER register
xxxxxxxx	Timer Register value. This enables TIMER_CFG and sets the TIMER value
20000000	NOOP
20000000	NOOP

Tcl Script to Create Address Table and Barrier Images

The Tcl script `multiboot_address_table.tcl` is provided with the downloadable xapp1247.zip file.

To create the address table and barrier images, unzip the reference design files to a directory. Open a Vivado command prompt and change the directory to point to the location that points to the script location in reference design directory. The reference design has a *multiboot_address_table folder* that includes the `multiboot_address_table.tcl` script.

Use this syntax to run the script:

```
>> tclsh multiboot_address_table.tcl <flash_type> <data_width> <freq_mhz>
< flash_size_mbit> <bitstream_size>
```

Where:

flash_type: flash types tested are spi, bpi

data_width: Flash data width = 1, 2, 4, 16

freq_mhz: Frequency of CCLK = ConfigRate setting

flash_size_mbit: Size of flash device in Mb

bitstream_size: Size of bitstream in bytes

Note: Get the bitstream size from UG470 [Ref 1]. If compression is enabled enter the compressed bitstream size. Be aware that for compressed bitstreams, subsequent builds can vary significantly in size and this script needs to be re-run.

Running the Tcl script gives you two timer images: timer1 and timer2. These files are stored in the same directory where you run the script. The script also outputs address map locations indicating where to place the files in serial flash and the `write_cfgmem` command to concatenate four files together. The following is a sample output displayed after running the script in the Windows command prompt.

```
c:\xapp1247\multiboot_address_table>tclsh multiboot_address_table.tcl spi 1 3 12 8
1132000
Flash type            : SPI
Flash width (bits)    : 1
CCLK frequency (MHz)  : 3
Flash density (Mbits) : 128
Bitstream size (B)    : 1132000

Writing Timer: timer1.bin
Writing Timer: timer2.bin

Golden bitstream address : 0x00000000
Timer1 image address     : 0x0013FC00
Multiboot image address  : 0x00140000
Timer2 image address     : 0x00280000

write_cfgmem command:
write_cfgmem -format mcs -size 16 -interface SPIx1 -loadbit "up 0x00000000 <golden>
up 0x00140000 <multiboot>" -loaddata "up 0x0013FC00 timer1.bin up 0x00280000
timer2.bin" <output_mcs>
```

In the golden design implementation, ensure that the NEXT_CONFIG_ADDR setting i.e., *Addr A1* in Table 1, page 5, points to the timer1 image start address output displayed after running the script. For example, for the sample address map shown on the previous page, the golden image WBSTAR register stores `0x0013FC00` set by bitstream property NEXT_CONFIG_ADDR:

```
set_property BITSTREAM.CONFIG.NEXT_CONFIG_ADDR 0x0013FC00 [current_design]
```

Update design constraints to enable MultiBoot remain unchanged.

In the sample output, note that the *-loaddata* switch is used with the `write_cfgmem` command for timer1 and timer2 .bin files and the *-loadbit* switch is used with the golden and update .bit files. Pay attention to the start address sequence for each file while generating the serial flash programming file.

Use the `-help` command in the Vivado tool for a detailed description of each `write_cfgmem` command option:

```
write_cfgmem -help
```

References

1. *7 Series FPGAs Configuration User Guide* (UG470)
2. *Using SPI with 7 Series FPGAs* (XAPP586)
3. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
4. *Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS182)
5. *KC705 Evaluation Board for the Kintex-7 FPGA User Guide* (UG810)
6. Kintex-7 FPGA KC705 Evaluation Kit website

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/13/2015	1.0	Initial Xilinx release

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

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.

© Copyright 2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.