XAPP1246 (v1.1) February 28, 2017

MultiBoot with 7 Series FPGAs and BPI

Author: Kiran K Gakhal

Summary

Note: Spartan®-7 devices do not support BPI.

This application note covers the key concepts for building a successful MultiBoot design with 7 series FPGAs in byte-wide peripheral interface (BPI) configuration mode. A reference design demonstrating MultiBoot and Fallback capabilities of a 7 Series FPGA using an internally generated configuration clock source (CCLK) is provided.

The Virtex®-7 FPGA and parallel NOR flash (BPI flash memory) on the VC707 evaluation board are used to demonstrate the flow with Vivado® Design Suite 2015.1. The *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] provides additional information regarding the MultiBoot feature and details on the BPI configuration mode.

Note: Fallback MultiBoot is not supported in the Virtex-7 HT FPGAs, as specified in UG470 [Ref 1].

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 MultiBoot feature provides designers with an inexpensive method to perform dynamic full reconfiguration of an FPGA. Dynamic full reconfiguration has a wide range of applications, including robust design management and field upgrade capability. Field update solutions use the FPGA's built-in MultiBoot and Fallback features.

MultiBoot

The MultiBoot feature allows the FPGA to selectively load its bitstream from an attached flash memory containing two or more bitstreams. In this mode, the FPGA application triggers a MultiBoot operation, causing the FPGA to reconfigure from a different bitstream.

Fallback

In case of a configuration error, the Fallback feature resets the FPGA and retries configuration from address zero of flash memory. Typically, in the MultiBoot application, the bitstream at address zero is never modified to ensure its known good condition for all cases. Therefore, the “golden bitstream” at address zero should be factory programmed, verified prior to deploying in the design, and the parallel NOR flash region that includes the golden bitstream should be locked-down with write-protect.

This application note is divided into the following key sections:

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 the general MultiBoot and Fallback flow demonstrated in this application note.

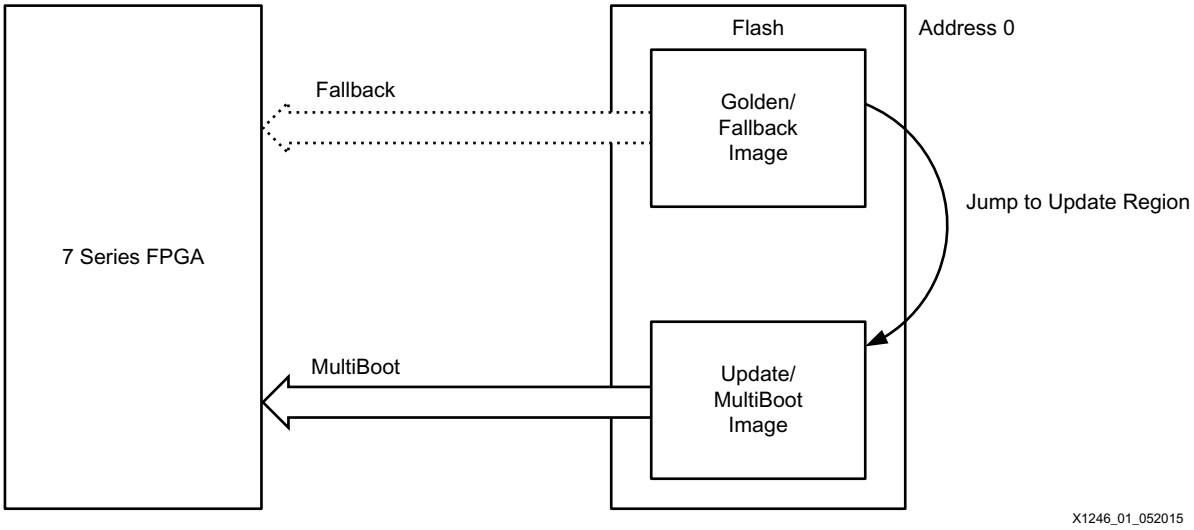

Figure 1: **MultiBoot and Fallback Flow**

In preparation for a MultiBoot event, internal PROGRAM (IPROG) and next image start address (WBSTAR) commands are set in the golden bitstream. See the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] for details on the IPROG command and WBSTAR register.

Three primary approaches to implement MultiBoot are:

1. IPROG is embedded in the bitstream (focus of this application note). IPROG is enabled using the NEXT_CONFIG_ADDR bitstream option.
2. Using RS[1:0] pins with and without IPROG (discussed in Appendix A: Advanced Applications). FPGA RS pins connect to the most significant address pins of flash memory.
3. IPROG using ICAPE2 (not covered in this application note). Apply the register write commands to the ICAPE2 primitive.

Note: The first two options are automated (MultiBoot settings are embedded in the bitstream) whereas the last option 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 inside a 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 (refer to Different Fallback Scenarios, page 22), 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.

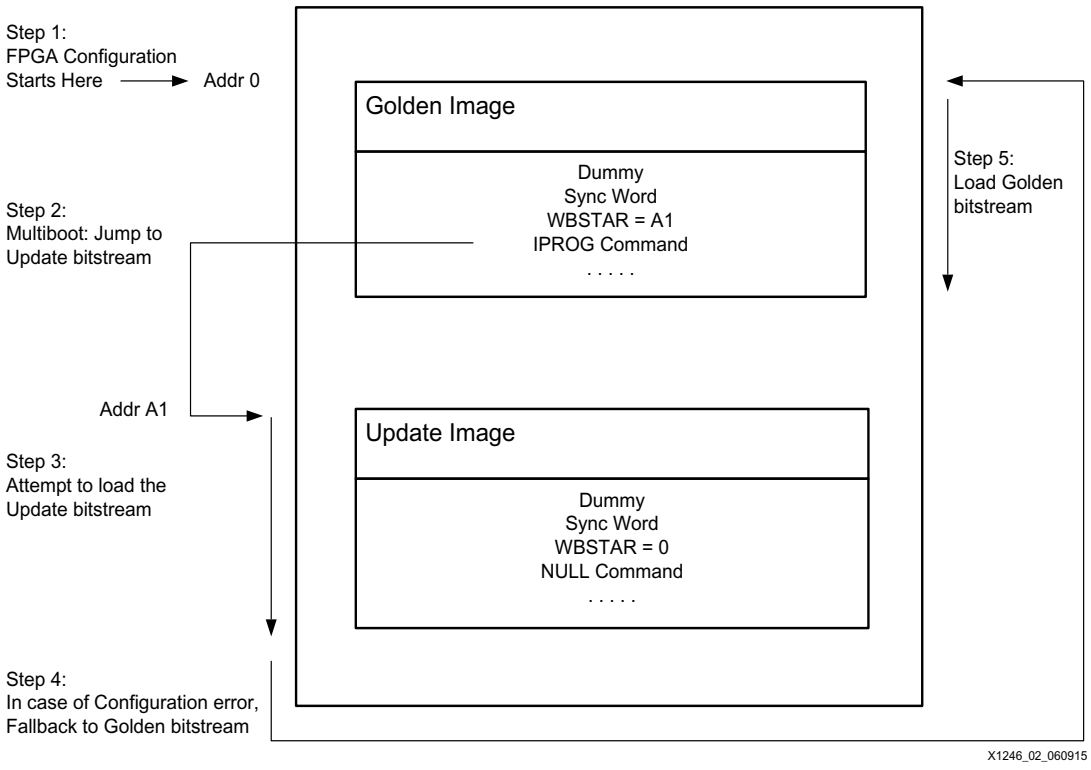

Figure 2: **MultiBoot Fallback Flash Memory Components and Configuration Steps**

FPGA BPI Flash Configuration Interface

Figure 3 shows the associated signals for the BPI configuration interface. Refer to the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] for FPGA signal description and Micron's G18 flash data sheet [Ref 2] for flash signal descriptions and details.

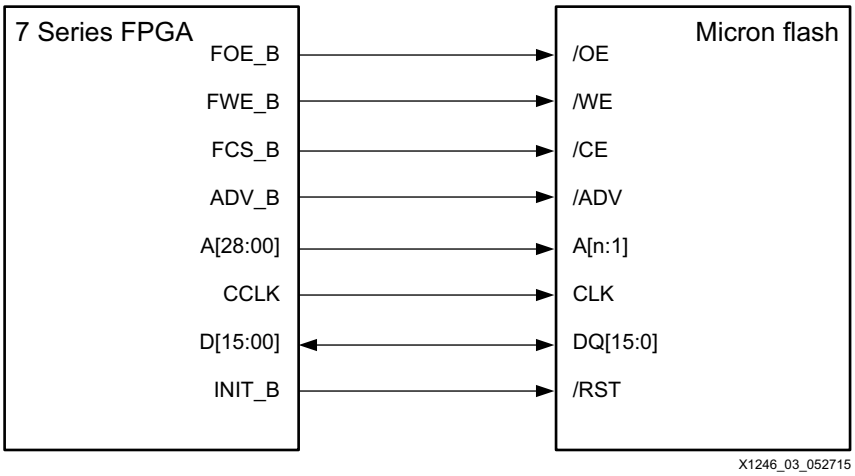

Figure 3: **FPGA Flash Interface**

The RS[1:0] and EMCCLK are optional connections for the BPI configuration mode. EMCCLK is not highlighted in this application note. For BPI configuration mode, Fallback loading of the golden image always uses the asynchronous read mode. Consequently, bitstream settings for synchronous read mode are ignored during Fallback loading of the golden image. As a result, the golden image settings for the CCLK frequency must be set within the system capabilities for asynchronous read operations if EMCCLK is used for MultiBoot images. Appendix A: Advanced Applications, page 17 provides additional information on RS[1:0] pin use.

Preparing Bitstream for MultiBoot Application

This section outlines the bitstream properties required to create golden and update bitstreams for MultiBoot application.

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: *Include MultiBoot properties in the golden and update design XDC file prior to synthesizing and implementing designs in the Vivado Design Suite.*

Add these constraints to the golden XDC file:

```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGFALLBACK ENABLE [current_design]
set_property BITSTREAM.CONFIG.NEXT_CONFIG_ADDR 32'h00800000 [current_design]
```

Note: BITSTREAM.CONFIG.NEXT_CONFIG_ADDR = 32'h00800000 is an example from the reference design.

Add these constraints to the update XDC file:

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

Alternatively, you can add MultiBoot settings using the Vivado graphical user interface:

1. Open the synthesized and implemented design in the “Program and Debug” section and select **Bitstream Settings**. In the Project Settings window with “Bitstream” highlighted, click **Configure additional bitstream settings**.
2. Or, click **Tools** > **Edit Device Properties**.

3. Under **Edit Device Properties/General**, set Enable Bitstream Compression by selecting **TRUE** from the drop-down options, as shown in Figure 4.

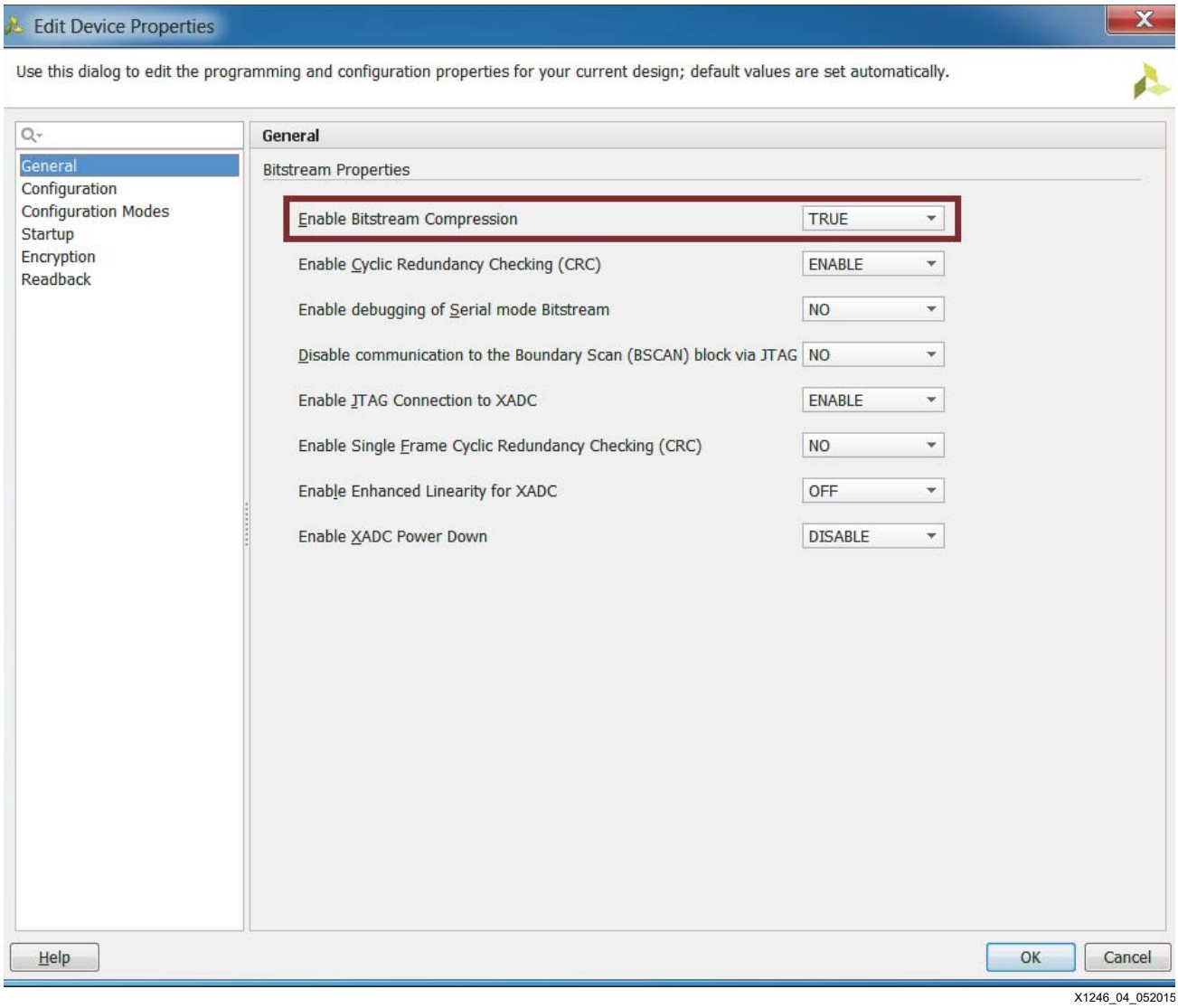

Figure 4: **General Bitstream Properties**

4. Under **Edit Device Properties/Configuration/MultiBoot Settings**, ensure that *Load a Fallback bitstream when a configuration attempt fails* is set to **ENABLE (**for both the golden and update bitstreams), and that *Starting address for the next configuration in a MultiBoot setup* (golden bitstream only) is set, as shown in Figure 5.

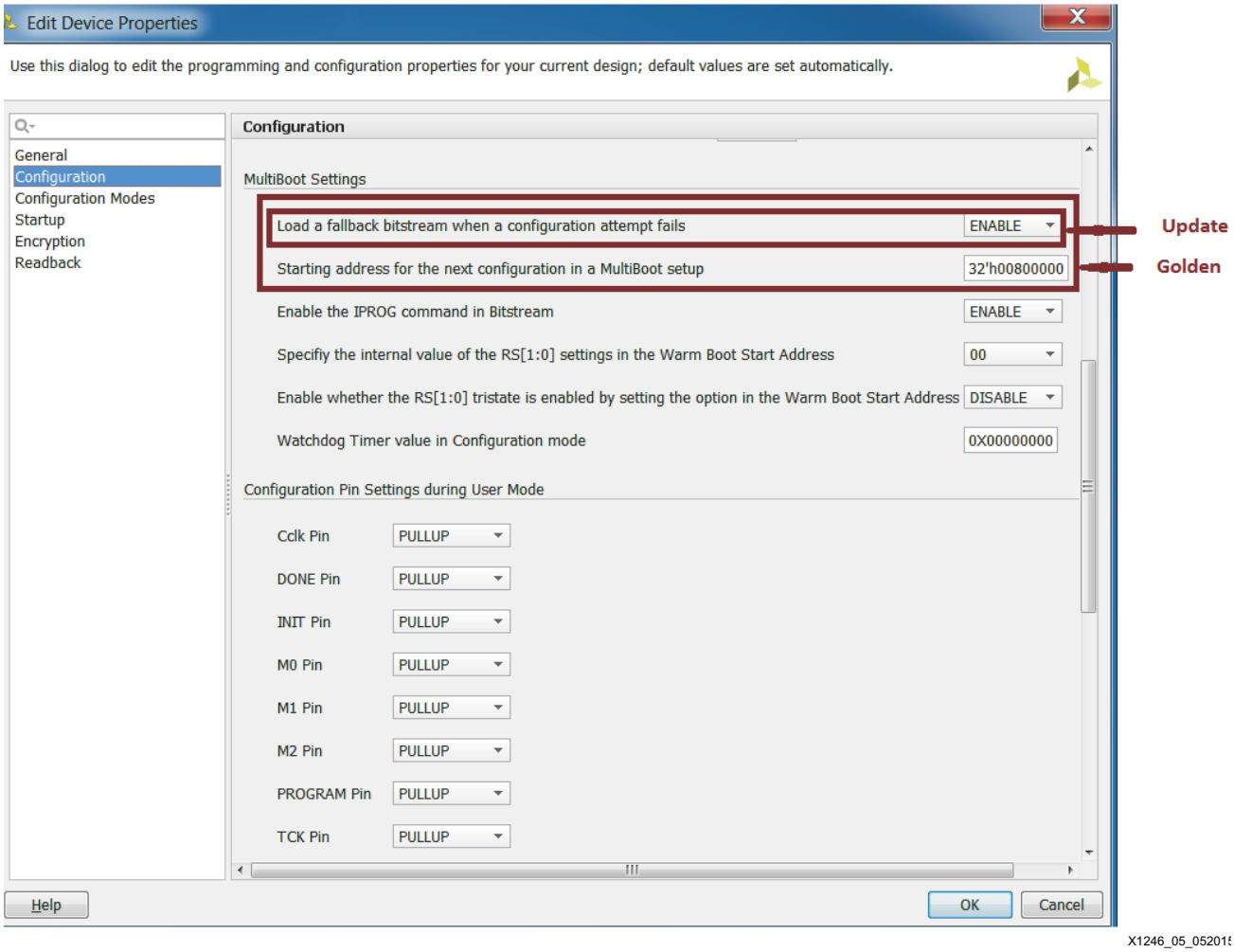

Figure 5: **Configuration Options to Generate Golden and Update Bitstreams**

5. After setting the above GUI options, save the changes. Double-click **Generate Bitstream.** You will need to rerun Synthesis and Implementation to apply the MultiBoot settings.

RECOMMENDED: *Set MultiBoot settings in the constraints file, this saves time and only requires Synthesis and Implementation to be run once.*

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` with `write_bitstream` to summarize all bitstream options used. For additional help use the `write_bitstream -help` command:

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

Flash Programming File Generation

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 parallel NOR flash.

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

```
write_cfgmem -format mcs -interface BPIX16 -size 128 -loadbit "up 0 <path>/golden.bit up
0x00800000 <path>/update.bit"  <path>/filename.mcs
```

Note: Address value `0x0800000` 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 VC707 board.

- For the golden Image - GPIO LEDs [3:0] blink sequentially from right to left
- For the update Image - GPIO LEDs [3:0] blink sequentially from left to right

Basic Board Setup

1. Ensure that the appropriate configuration mode (Master BPI) is selected for switch SW11 on the VC707 board as shown in Figure 6.

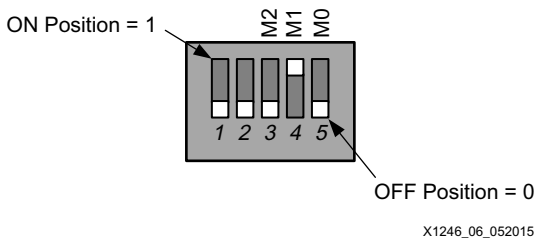

Figure 6: **VC707 SW11: Mode Pin Settings**

2. Open the Vivado Hardware Manager and connect to the VC707 demo board. For step-by-step instructions to program parallel NOR flash using Vivado, refer to the *Programming Configuration Memory Devices* section of the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 3].

   You can use the `Golden_Update.mcs` file included in the *Ready_to_download* directory of the reference design for verification.

3. After parallel NOR flash programming completes successfully, boot the FPGA from flash by pulsing PROGRAM_B.
   - Alternatively, you can use the `boot_hw_device` Tcl command:

```
boot_hw_device  [lindex [get_hw_devices] 0]
```

   - Or, in the Vivado Hardware Manager GUI, right-click on the device and select the **Boot from Configuration Memory Device** option as shown in Figure 7.

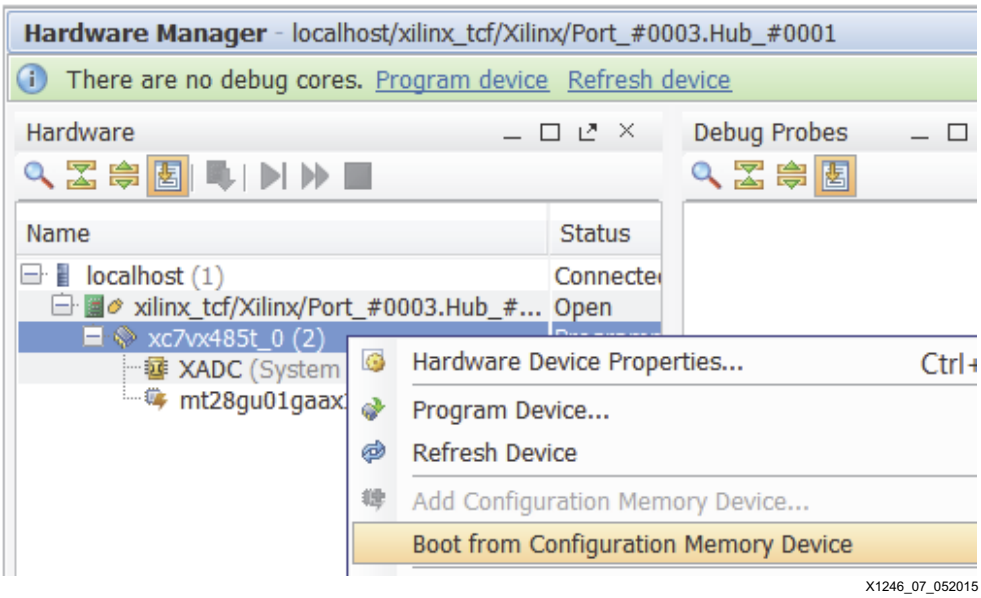

Figure 7: **Boot from Configuration Memory Device**

4. Observe the DONE pin status on the board. DONE goes High after successful configuration of the FPGA from parallel NOR flash.

MultiBoot Operation

1. Check the status of the GPIO LEDs to verify that MultiBoot operation was successful and the FPGA was loaded with the update bitstream: GPIO LEDs [3:0] should blink sequentially from left to right.

   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 8).

   Note: Refresh device before checking device properties.

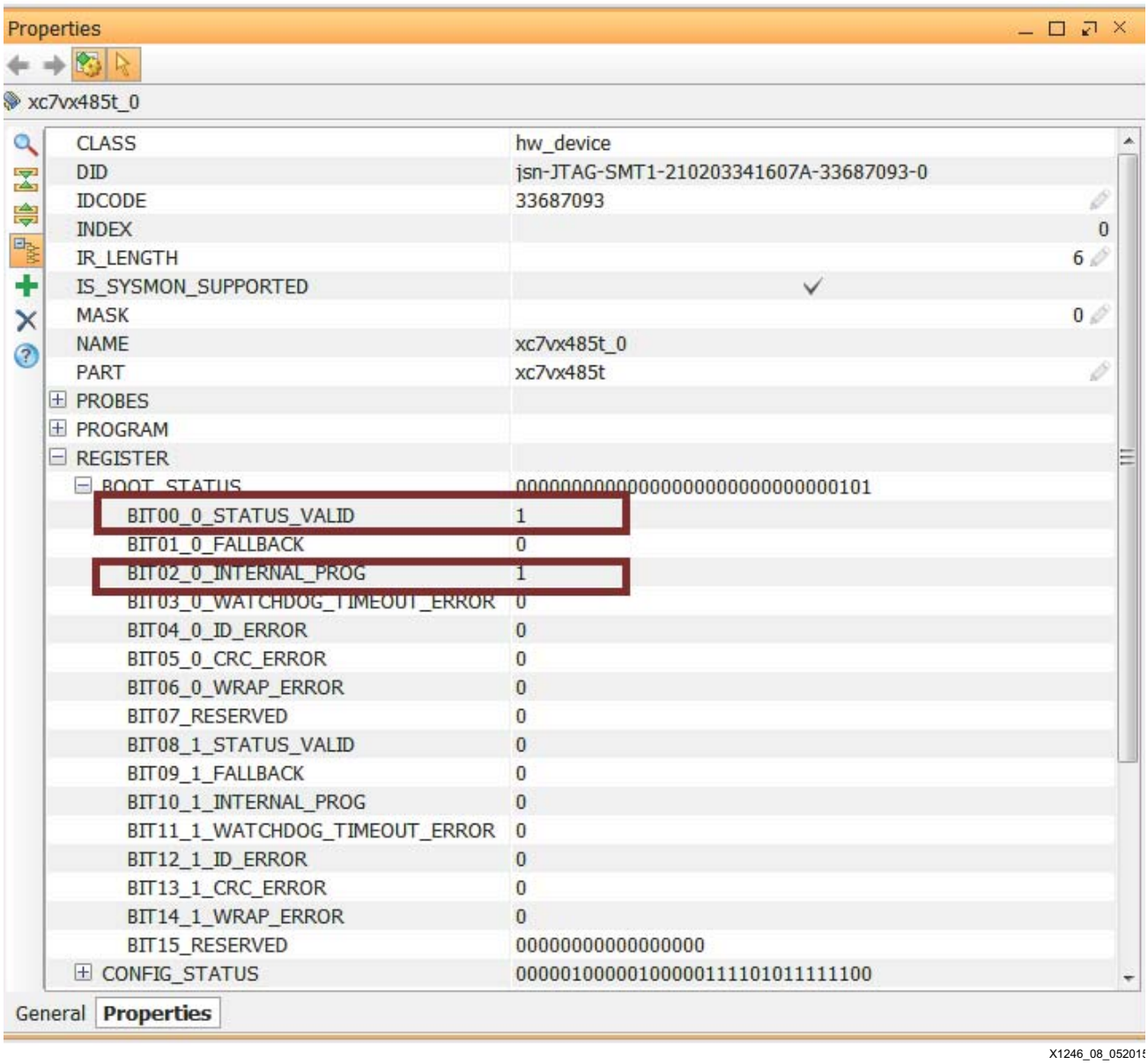

Figure 8: **BOOT_STATUS after Update Image Loads**

Fallback Example – Triggered by CRC Error

1. Fallback to golden image can be triggered by different means, refer to the *Reconfiguration and MultiBoot* chapter in the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 1] for more information.

   This application note demonstrates an example of 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 9 and Figure 10 below shows an example). Refer to UG470 [Ref 1] for more information on the bitstream format.

   Open the update bitstream (.bit) in any HEX editor and towards the middle of bitstream flip some data bytes, say from `00` to `FF`, as highlighted in Figure 9 and Figure 10.

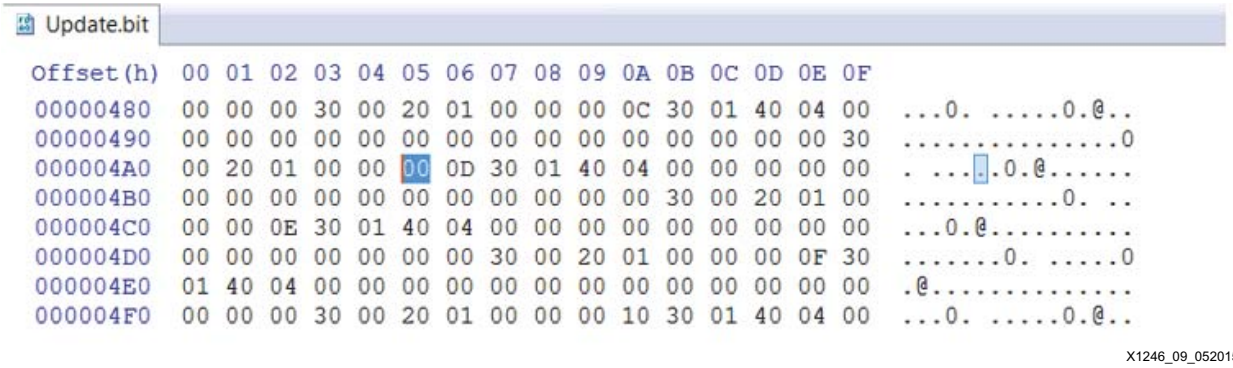

Figure 9: **Original Update Image**

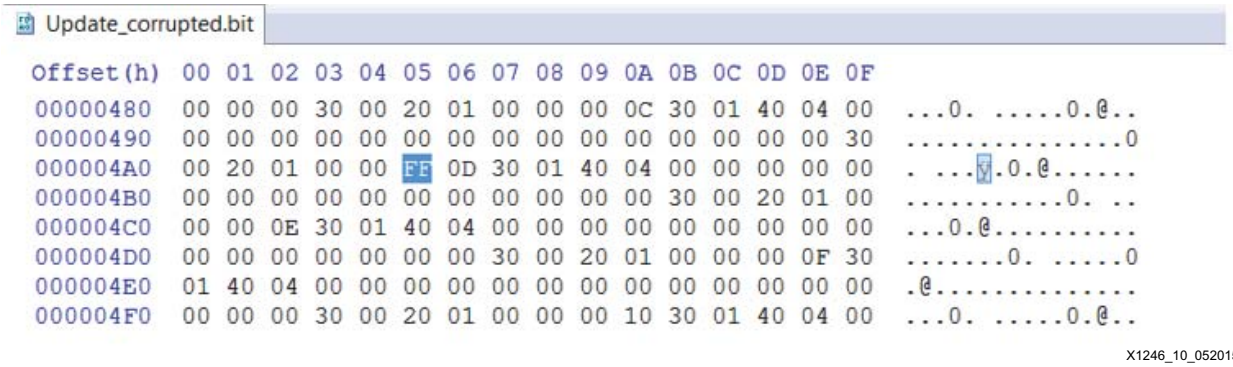

Figure 10: **Corrupted Update Image**

Save the corrupted update bitstream and generate new flash programming file (.mcs) with this corrupted bitstream (see Flash Programming File Generation, page 9 for `write_cfgmem` command information).

Note: The CRC check covers all data and commands in the bitstream between the Reset CRC (RCRC) command and the final CRC check before EOS (end of startup), refer to UG470 [Ref 1] for more information. Therefore, configuration logic can only catch a CRC error when any data or command is corrupted between RCRC and CRC commands in the bitstream.

2. Repeat steps 1 to 4 in the Basic Board Setup, page 9 to program the parallel NOR flash with the .mcs file generated with the corrupted update bitstream.

   In step 2, you can use the `golden_update_corrupted.mcs` file included in *Ready_to_download* directory of the reference design for verification.

3. After the FPGA boots and DONE goes High, check the status of the GPIO LEDs to verify that the Fallback operation was successful and the FPGA with the golden bitstream loaded.
   - GPIO LEDs [3:0] rotate from right to left.

   To verify that Fallback is triggered by the CRC error, check the BOOT_STATUS register under device properties after refreshing the device. As shown in Figure 11, the CRC error and Fallback flags are set in the BOOT_STATUS register.

   - BIT10_1 and BIT13_1 show that IPROG was attempted, and a CRC_ERROR was detected.
   - BIT01_0 shows that the Fallback bitstream loaded successfully. The IPROG bit was also set in this case, because the Fallback bitstream contains an IPROG command. Although the IPROG command is ignored during Fallback, the status still records this occurrence.

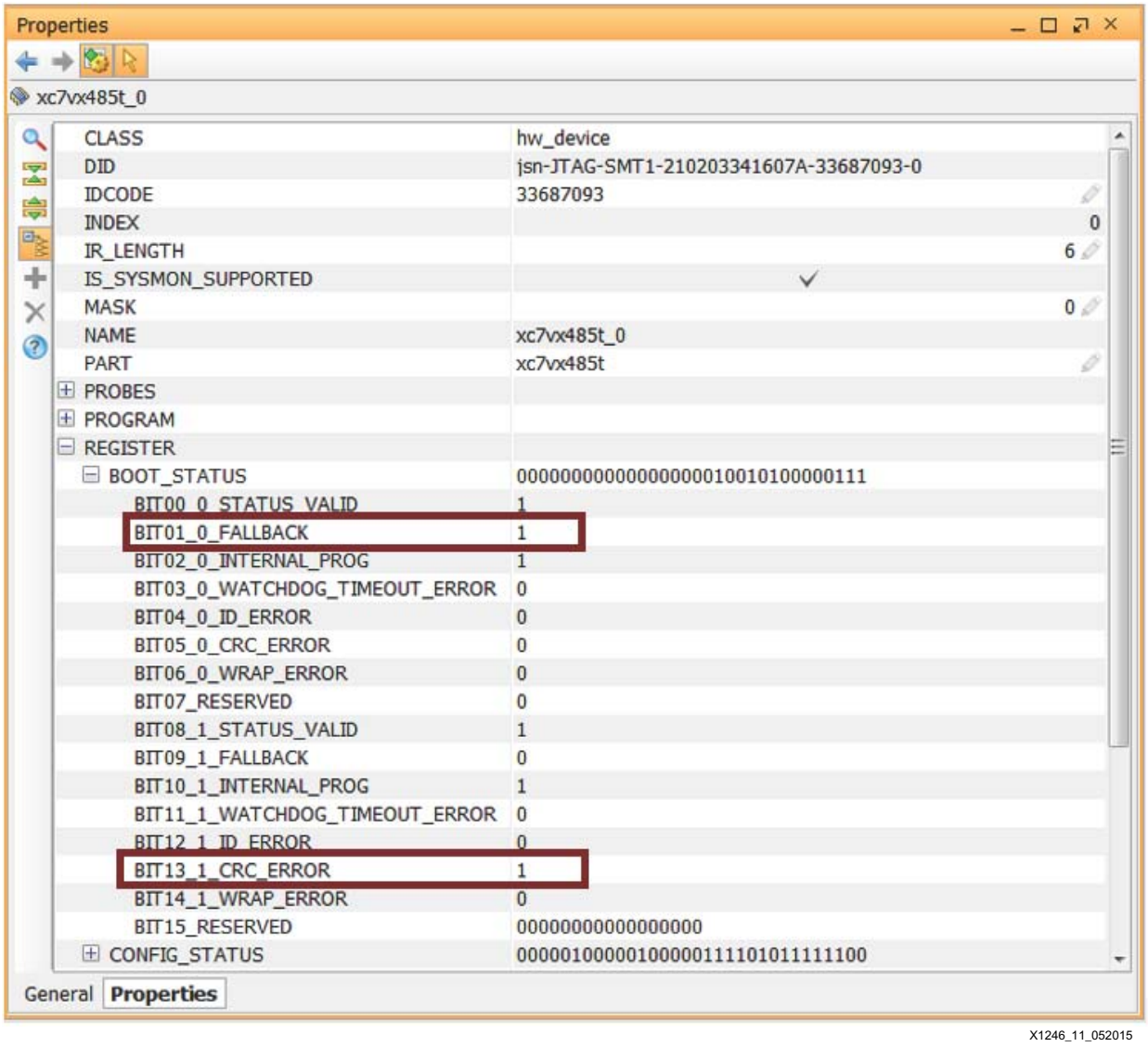

Figure 11: **BOOT_STATUS Showing Fallback and CRC Flags Set**

Debug and Checklist

This section summarized a checklist and common debug steps for MultiBoot operation using parallel NOR Flash.

File Generation

1. If using the Vivado graphical user interface method to set MultiBoot properties, ensure that 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. If RS[1:0] bits are connected to the MSB address lines of flash, ensure that additional REVISIONSELECT properties are set correctly in the golden XDC, as discussed in Appendix A: Advanced Applications, page 17.
3. Ensure that the CONFIGRATE option does not exceed the maximum supported frequency by the target flash (refer to *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 the 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 programming failure issues and helps locate the root cause, whether it is design or incorrect MultiBoot settings.
2. Ensure to erase the flash device completely before programming the new flash programming (.mcs) file.
3. Use the *Blank Check* operation to verify the erase operation.
4. Revision Select RS[1:0] pins support optional features, discussed in Appendix A: Advanced Applications, page 17. If the RS pins are connected to the most significant address pins of the parallel NOR flash instead of the FPGA upper address pins for revision control, ensure that the RS pin option is set correctly in the flash programming options, see MultiBoot using Revision Select (RS[1:0]) pins, page 17.
5. Capture BOOT_STATUS and CONFIG_STATUS register status for additional debug information. Ensure to refresh the device before capturing device properties.
6. Ensure that 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, both designs
Hardware platform used for verification	VC707 board and 1 Gb Micron BPI 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 parallel NOR flash. A complete reference design with quick verification files to demonstrate MultiBoot and Fallback features is provided.

Appendix A: Advanced Applications

MultiBoot using Revision Select (RS[1:0]) pins

FPGA RS pins can divide the 1 Gb parallel NOR flash into four equal banks. If RS[1:0] pins are connected to the most significant address pins A[26:25] of the parallel NOR flash as shown in Figure 12, then you can select one of multiple stored configuration bitstreams by controlling these pins with user logic or external DIP switches.

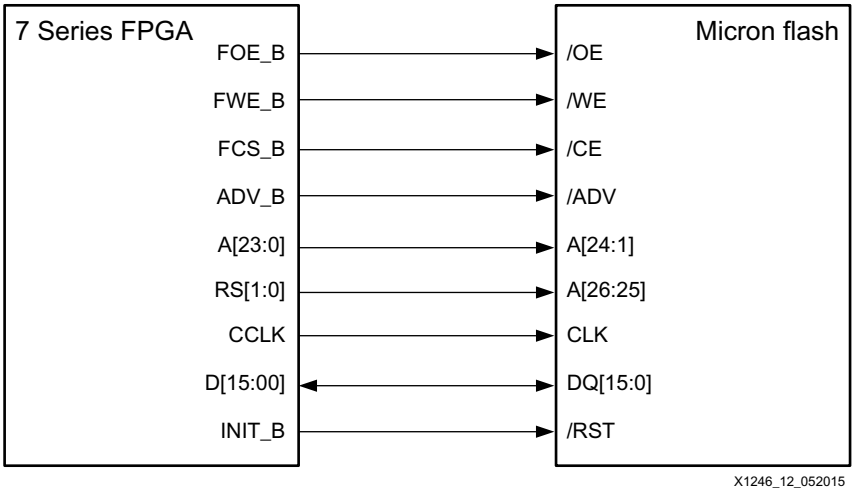

Figure 12: **BPI Configuration Interface with RS[1:0] Connected to the Most Significant Flash Address Pins**

Refer to UG470 [Ref 1] for FPGA signal description and Micron's G18F flash data sheet [Ref 2] for flash signal descriptions and details.

You can demonstrate this use-case using the VC707 demo board which has flash upper address bits connected to FPGA RS[1:0]. Control RS pins with DIP switch SW11 on the VC707 board where:

- SW11[1:2]: FPGA RS pins, RS[1:0] connected to flash most significant address pins
- SW11[3:5]: FPGA mode pins, M[2:0]

Refer to Figure 13 for VC707 switch SW11 settings.

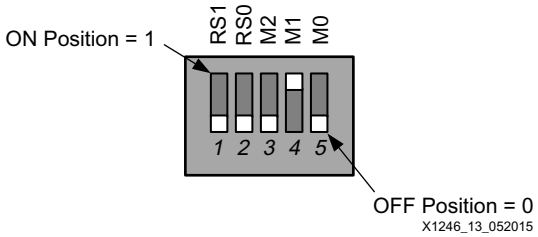

Figure 13: VC707 Board DIP Switch SW11

Table 3 lists RS[1:0] settings for DIP switch SW11 with flash address ranges for four revision applications.

Table 3: **Revision Select Address Boundaries**

Revision Select (SW11[1:2])	MultiBoot to Revision	Start Address
RS[1:0] = 00	Rev#0	0x00000000
RS[1:0] = 01	Rev#1	0x01000000
RS[1:0] = 10	Rev#2	0x02000000
RS[1:0] = 11	Rev#3	0x03000000

To ensure that in case of a configuration failure, FPGA Fallback and the known good image located at start address zero (i.e., Rev0) are loaded, enable the bitstream Fallback option in all revisions (0, 1, 2, and 3) using the CONFIGFALLBACK bitstream property:

```
set_property BITSTREAM.CONFIG.CONFIGFALLBACK ENABLE [current_design]
```

The following is an example of a `write_cfgmem` command to generate MCS with four images with the start addresses as specified in Table 3:

```
write_cfgmem -format mcs -interface BPIX16 -size 128 -loadbit "up 0 <Rev0>.bit up
0x01000000 <Rev1>.bit up 0x02000000 <Rev2>.bit up 0x03000000 <Rev3>.bit"  <output>.mcs
```

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

```
write_cfgmem -help
```

Program the parallel NOR flash with the programming file (.mcs) generated with four revisions (refer to the *Programming Configuration Memory Devices* section of the *Vivado Design Suite User Guide* (UG908) [Ref 3] for flash programming flow). Make sure to select the correct RS pins in the programming options when using the Vivado Hardware Manager to program the flash, as highlighted in Figure 14.

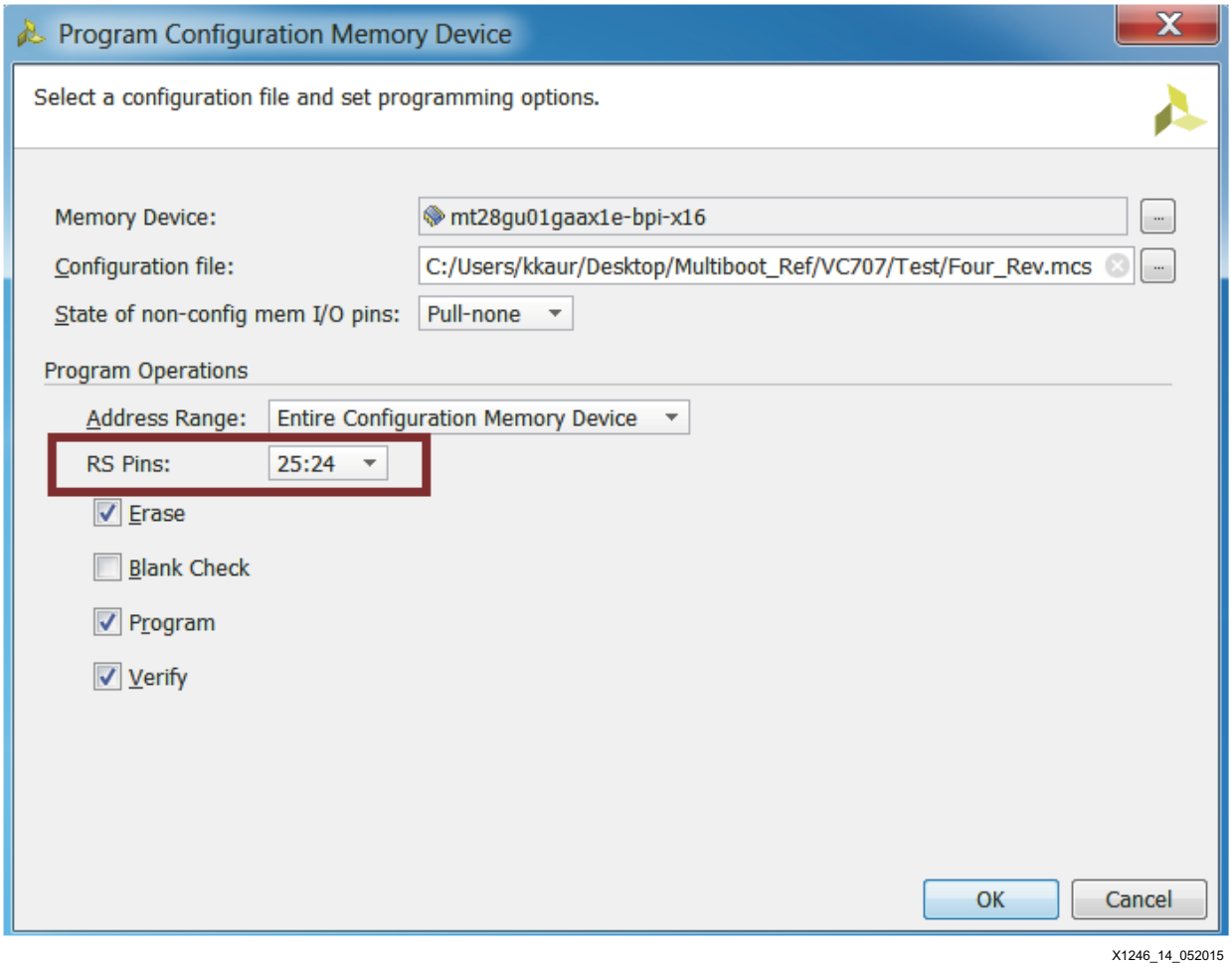

Figure 14: **Flash Properties**

Alternatively if using the Tcl flow you can set the RS pins using the Tcl command:

```
set_property PROGRAM.BPI_RS_PINS {25:24} [ get_property PROGRAM.HW_CFGMEM [lindex 0 ]]
```

After successful flash programming, you can boot the FPGA from the desired revision by toggling SW11[1:2] DIP switches followed by pulsing the PROGRAM_B pin to load the selected revision.

MultiBoot using Revision Select (RS[1:0]) Pins and Internal PROGRAM (IPROG)

FPGA access is limited to one quarter of the flash during MultiBoot operation when the FPGA RS pins are connected to the most significant flash address pins, as shown in Figure 12), because of address incrementing.

Therefore, if WBSTAR contains a next bitstream address that is beyond the address range of default Bank#0 (i.e., RS[1:0] = 00) then for the IPROG command to work you need to ensure that the RS[1:0] (WBSTAR[31:30]) and RS_TS_B (WBSTAR[29]) bits are set correctly in the WBSTAR register (Table 4), and are seen internally by configuration logic to initiate an IPROG jump to a different logical bank region. Table 5 provides the bit description of the WBSTAR register.

Table 4: **WBSTAR Register**

Description	RS[1:0]		RS_TS_B	START_ADDR																											
Bit Index	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5: **WBSTAR Register Description**

Name	Bit Index	Description
RS[1:0]	[31:30]	RS[1:0] pin value on next warm boot. The default is 00.
RS_TS_B	29	RS[1:0] pins 3-state enable: 1: Disabled 0: Enabled (default)
START_ADDR	[28:0]	Next bitstream start address. The default start address is zero.

Ensure to include the additional *Revision Selection* bitstream properties in the golden XDC file, set RS[1:0] (WBSTAR[31:30]) to select a different bank, and set the RS_TS_B (WBSTAR[29]) bit to use RS pins:

```
#Golden Settings
#
#Golden Multiboot properties
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGFALLBACK ENABLE [current_design]
set_property BITSTREAM.CONFIG.NEXT_CONFIG_ADDR 0X01000000 [current_design]

#Revision Selection
set_property BITSTREAM.CONFIG.REVISIONSELECT_TRISTATE ENABLE [current_design]
set_property BITSTREAM.CONFIG.REVISIONSELECT 01 [current_design]
```

No additional settings are required in the update design implementation:

```
#Update Settings
#
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.CONFIGFALLBACK ENABLE [current_design]
```

Figure 15 illustrates the resulting memory map for the flash device associated with this example.

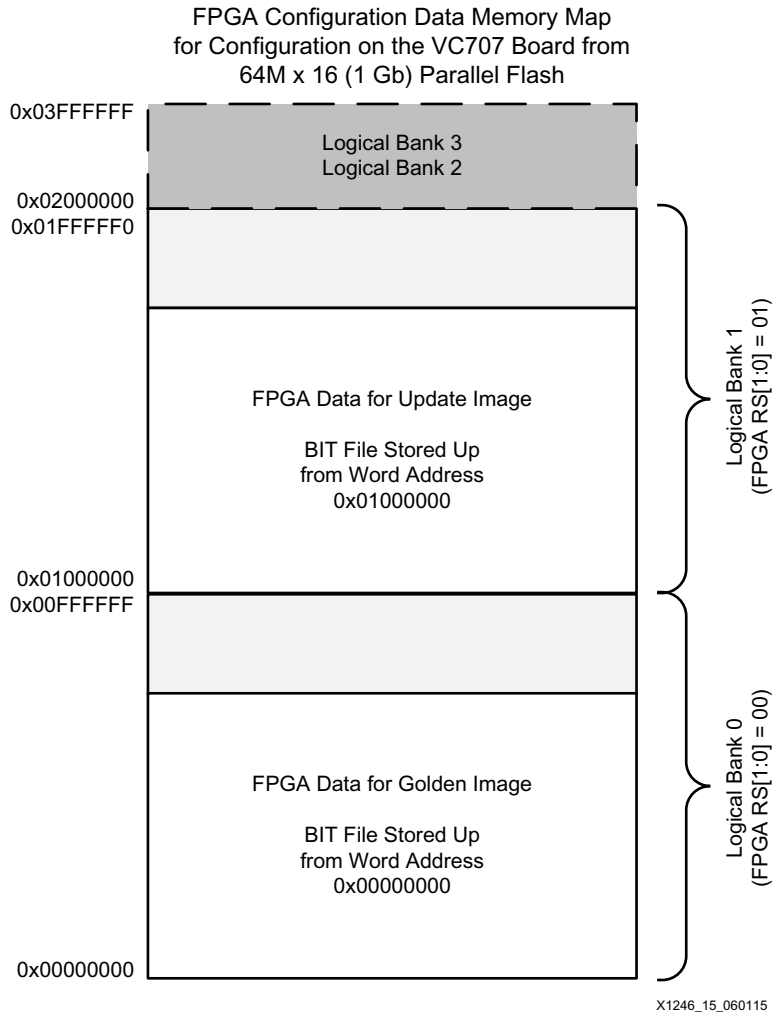

Figure 15: **Data Organization for VC707 MultiBoot when WBSTAR Points to the Address in Bank#1**

While generating the flash programming file using the `write_cfgmem` command, make sure to select the correct update image start address:

```
write_cfgmem -format mcs -interface BPIX16 -size 128 -loadbit "up 0 golden.bit up
0x01000000 update.bit"  <path>/filename.mcs
```

Follow the verification steps in Validation in Hardware, page 9 and refer to the *UltraScale FPGA BPI Configuration and Flash Programming Application Note* (XAPP1220) [Ref 4] for the flash programming flow.

Note: Make sure to select the correct RS pins in the programming options when using the Vivado Hardware Manager to program the flash, as highlighted in Figure 14, page 19.

Different Fallback Scenarios

The 7 *Series Configuration User Guide* (UG470) [Ref 1] outlines the following errors that can trigger Fallback:

- CRC error
- IDCODE error
- Watchdog timer time-out error
- BPI address wraparound error

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. An example of Fallback triggered by a CRC error is provided in Validation in Hardware, page 9. Fallback via IDCODE error can be tested by manually corrupting the IDCODE command in the bitstream similar to CRC error example.

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. If no valid SYNC word is found, configuration logic should wrap around and load the golden image.

When targeting large density BPI flash memory, the device address wrap-around takes a long time to be seen by configuration logic due to an FPGA internal 29-bit counter that needs to be exhausted to trigger the wrap-around error. Therefore, you need to have a way to set the watchdog timer that can help trigger Fallback to the golden region to recover from any corner case corruption scenarios that are not recoverable by the CRC algorithm. Refer to the *Watchdog Timer* section in (UG470) [Ref 1] for more detailed 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 6.

```
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 golden and update image in flash memory. Flash memory components and configuration steps to implement timer images in Multiboot application are shown in Figure 16.

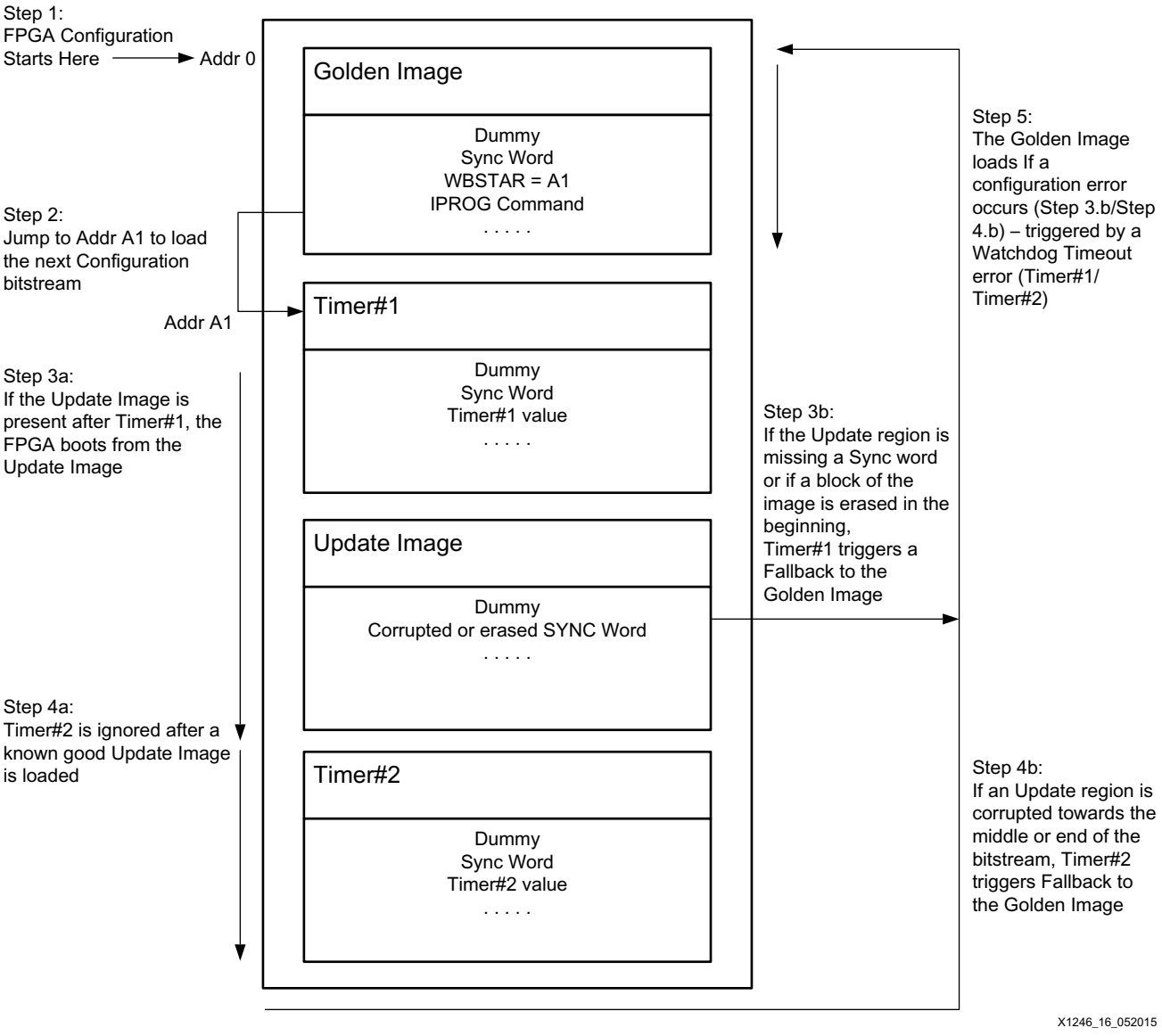

Figure 16: **MultiBoot, Fallback, and Barrier Flash Memory Components and Configuration Steps**

IMPORTANT: *You need to ensure that golden, timer1, and timer2 regions in parallel 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 16 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.

Tcl Script to Create Address Table and Barrier Images

The Tcl script `multiboot_address_table.tcl` is provided with the reference design files.

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 need to be re-run.

Running the Tcl script give 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 Parallel NOR 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:\xapp1246\tclsh multiboot_address_table.tcl bpi 16 3 1024 20273436
Flash type            : BPI
Flash width (bits)    : 16
CCLK frequency (MHz)  : 3
Flash density (Mbits) : 1024
Bitstream size (B)    : 20273436

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

Golden bitstream address : 0x00000000
Timer1 image address     : 0x009BFE00
Multiboot image address  : 0x009C0000
Timer2 image address     : 0x01380000

write_cfgmem command:
write_cfgmem -format mcs -size 128 -interface BPIx16 -loadbit "up 0x00000000 <golden>
up 0x009C0000 <multiboot>" -loaddata "up 0x009BFE00 timer1.bin up 0x01380000 timer2.bin"
<output_mcs>
```

Note: For BPI 16 bit data width mode, timer1.bin and timer2.bin files are bit/byte swapped versions of the commands in Table 6.

In the golden design implementation, ensure that the NEXT_CONFIG_ADDR setting i.e., *Addr A1* in Table 1, page 6, points to the timer1 image start address output displayed after running the script. For example, for the sample address map above, the golden image NEXT_CONFIG_ADDR bitstream property should point to the Timer1 image address (`0X009BFE00`):

```
set_property BITSTREAM.CONFIG.NEXT_CONFIG_ADDR 0x009BFE00 [current_design]
```

Update image design constraints remain unchanged.

In the sample address map, note that *-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 programming file for parallel flash.

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

```
write_cfgmem -help
```

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 6 identifies the basic commands included in the barrier image that enable the TIMER register. For BPI configuration mode, commands in Table 6 are swapped in the output files generated by the script, as described in section Tcl Script to Create Address Table and Barrier Images, page 24.

Table 6: **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

References

1. *7 Series FPGAs Configuration User Guide* (UG470)
2. Micron StrataFlash Embedded Memory MT28GU01GAAA1E (28F00AG18F) Data Sheet
3. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
4. *UltraScale FPGA BPI Configuration and Flash Programming Application Note* (XAPP1220)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/28/2017	1.1	Added note below document title indicating that Spartan-7 devices do not support BPI. Added note to summary indicating that Fallback MultiBoot is not supported in the Virtex-7 HT FPGAs.
08/12/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices