XAPP1176 (v1.0) July 26, 2013

Execute-in-Place (XIP) with AXI Quad SPI Using Vivado IP Integrator

Author: Sanjay Kulkarni, Prasad Gutti

Summary

This application note describes the eXecute-in-place (XIP) feature introduced in the AXI Quad SPI v3.0 IP core, released in the Vivado® Design Suite v2013.2. It provides information about the required connections to configure the FPGA from an SPI serial flash device, as well as the configuration flow for the SPI mode. It is designed for use with the Kintex®-7 (KC705) board with Numonyx SPI flash memory, but modifications in the software example file can be implemented for use on any Xilinx board.

Many factors are involved in selecting the code execution method for a given embedded system, including materials cost, processor features, operating system capability and more. Most of the embedded application designs do not need a memory management unit, as the primary requirements might not be performance-oriented. XIP is a method of executing programs directly from long-term storage rather than copying it into block RAM. It is an extension of using shared memory to reduce the total amount of memory required.

There are several criteria for using this mechanism. From the processor point of view, the storage should mimic regular memory that is operated at a faster rate. The program should be independent of location, and should not modify the image stored in the memory. If measured across all available memory sources, flash memory is preferred over Double Data Rate (DDR) or SRAM memories in embedded systems. Low power and a smaller pin interface count are additional benefits of flash memory.

The aforementioned criteria are suited for parallel/serial flash memory. For 'boot from flash' operation, SPI-based flash memory is preferred due to advances in SPI flash technology, improved density, improved operating speed, and other vendor-based support. The executable code residing in the SPI flash is loaded into DDR through an XIP configured Quad SPI IP core to demonstrate the store and load feature of the XIP mode implemented. Along with this, two modes of XIP (Dual and Quad) are demonstrated to show the improvement in the data transfer in Quad mode.

Structure

This application note is designed to demonstrate the XIP mode capability of the AXI Quad SPI core. The XIP mode systems are built using Xilinx Vivado IP Integrator, version 2013.2, which is part of Vivado® Design Suite. Vivado IP Integrator is a tool through which you can create systems by instantiating and interfacing the processor, interconnect, and interrupt controller, peripheral IPs, memory controller, and UARTs. See *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 1] for more information about the Vivado IP Integrator.

The design also includes software built using the Xilinx Software Development Kit (SDK). The software runs on a MicroBlaze™ processor subsystem and implements control, status, and monitoring functions. Complete Vivado and SDK project files are provided with the associated design files and can be downloaded using the following links:

- Dual XIP Mode
- Quad XIP Mode

These design files allow you to examine and rebuild the design or to use it as a template for starting a new design.

Recommended Design Experience

This application note assumes that the user has some general knowledge of the Xilinx Vivado Design Suite. See the *Vivado Design Suite Quick Reference Guide* (UG975) [Ref 2] for more information about the Vivado tools. For information on EDK see *EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design* (UG683) [Ref 3].

Introduction

The AXI Quad SPI IP core has upgraded functionality to support Legacy, Enhanced, and XIP modes. These three modes are further subcategorized into the three SPI modes, Standard, Dual and Quad modes. Standard mode commands use a single line (IO1) to exchange data, and the dual mode uses two lines (IO0, IO1). Quad mode uses a four line interface to exchange data (IO0, IO1, IO2, IO3). Legacy mode supports applications that are based on the earlier (v2.0) version of the core. Enhanced mode supports the AXI4 Memory Mapped Interface which, in turn, supports the fixed burst capability at the transmit and receive FIFO. The Enhanced mode reduces the AXI interface time required to fill or read the DTR or DRR FIFO. These FIFOs are configurable at compile time and can be either 16 or 256 deep.

XIP mode is the new feature which is used in the 'boot from flash' type of application. XIP mode enables execution of code from serial SPI flash instead of the more traditional way of executing the code from memories such as DDR/SRAM. This application note demonstrates the XIP feature using one of the operating modes of the Xilinx AXI Quad SPI IP core.

Table 1 shows the AXI4 interfaces used for each supported mode.

Table 1: **AXI Quad SPI Configuration Mode – AXI4 Interfaces**

Mode	AXI4 Lite Interface	AXI4 Full Interface
Legacy Mode	Yes	–
Enhanced Mode	–	Yes
XIP Mode	Yes	Yes

Based on the type of SPI slave used, the core is further categorized into three SPI modes. Table 2 shows the operating modes as well as the supported SPI clock frequency and the I/O interfaces.

Table 2: **SPI Mode, SCK Ratio and I/O Interfaces**

SPI Modes	SPI Clock Division Ratio	I/O Interface (CS and SCK Always Present)
Standard	2, 4, 8, 16, 16xn Where n = 1 … 128	IO0, IO1
Dual	2	IO0, IO1
Quad	2	IO0, IO1, IO2, IO3

For the best possible bandwidth on the SPI side, it is recommended that the core be used in Quad mode where the data transactions are occurring on all four lines. The SPI bandwidth is best utilized in this mode because the commands supported in this mode is Fast Read Quad I/O (0xEB h) which support reading of the SPI flash on all I/O lines.

XIP Basics

In the XIP mode, the executable code is stored in the SPI flash memory. In FPGA-based systems, the configurable bitstream is stored in a different region of the SPI flash in a continuous memory location. The FPGA secures a configurable bitstream after power-on-reset. When the complete bitstream is downloaded in the FPGA and the system moves into an active state, the boot loop program starts executing. A small boot loop code is

stored in the local block RAM memory, which guides the processor to jump to the different region of SPI flash memory for further execution.

During this initialization, writable memory is not available, and all computations must be performed within the processor registers. For this reason, first stage boot loaders tend to be written in assembler language and only do the minimum to provide a normal execution environment for the next program. The SPI flash memories are byte addressable. The Executable Link Format (ELF) file is stored in consecutive memory locations so that the processor continuously reads the SPI flash. The boot loop program facilitates reading the complete ELF file from the SPI flash and stores it in external memory (such as DDR or local on-chip memory) before the processor begins execution. Most of the embedded systems RTOS do not include demand paging, and therefore are limited to Fully Shadowed or the XIP mechanism. The code execution methods are further classified into XIP, Fully Shadowed, Demand Paging and Balanced XIP modes. The program can also be executed directly from flash.

For the purposes of this application note, this particular mode comes under *Store and Download*. This is a memory system where the executable code is stored in Non-Volatile Memory (NVM) and copied to RAM at boot-up. At boot-up a small piece of NVM, usually within the micro-processor, runs code to copy the contents of the NVM memory (NAND or NOR) to the RAM and jumps to the RAM location. From that point the code and data are executed out of RAM.

Using the AXI Quad SPI Core in XIP Mode

The XIP mode of the core operates purely in a read-only mode. There are two AXI interfaces on the core; AXI4-Lite for local register configuration, and AXI4 memory mapped for memory access. For the processor, the AXI Quad SPI memory mapped address range is like any other memory, so the processor can simply provide the address, length and size of the data to be read from memory. The SPI flash must be loaded with executable code before the FPGA is configured with a bit stream.

There are two ways to use this mode:

- In the first case, both the configuration bitstream as well as the executable file are stored in SPI flash.
- In the second case, only the executable file is stored in SPI flash while the core is configured through the iMPACT tool.

In the first case, the SPI flash memory address range should be divided in two sections, one each for bitstream and executable file. Usually the FPGA configuration file is stored starting from location 0x000000 (considering SPI flash is 24-bit addressable memory) as this would be default address provided by the FPGA while booting from SPI flash. The software must be configured so that when the FPGA is configured, the processor jumps to the SPI memory location where the executable file is stored. The boot code should be designed to be part of the configuration bitstream that is stored on the flash. After the FPGA is loaded with the configuration memory from flash, the processor executes boot code which is stored in the on-chip block RAM. The boot code is also a part of configuration bitstream. This boot code has information about the ELF file location in the flash. The processor then jumps to the SPI flash location to retrieve the next command(s).

There are two ways the processor can execute the code. One way is that the processor boot code reads the entire SPI ELF location and copies data in the local DDR memory and then starts executing. In other method, the processor uses the on-chip block RAM as scratch pad memory while executing the main code directly from the flash. As the flash access is slower (less than 50 MHz), this approach is best when the system operating expectations are not high.

In the second case, when only the executable file is stored in SPI flash, the iMPACT tool is used to configure the FPGA. The MicroBlaze™ boot loop application should be written in such a way that the processor jumps to the memory location where executable it stored. The executable

will be stored in contiguous format or page formats. As the SPI flash is byte accessible, it is preferable to store the executable code in contiguous format. The sample structure of SPI flash memory is shown in Figure 1. The addresses can be changed as required.

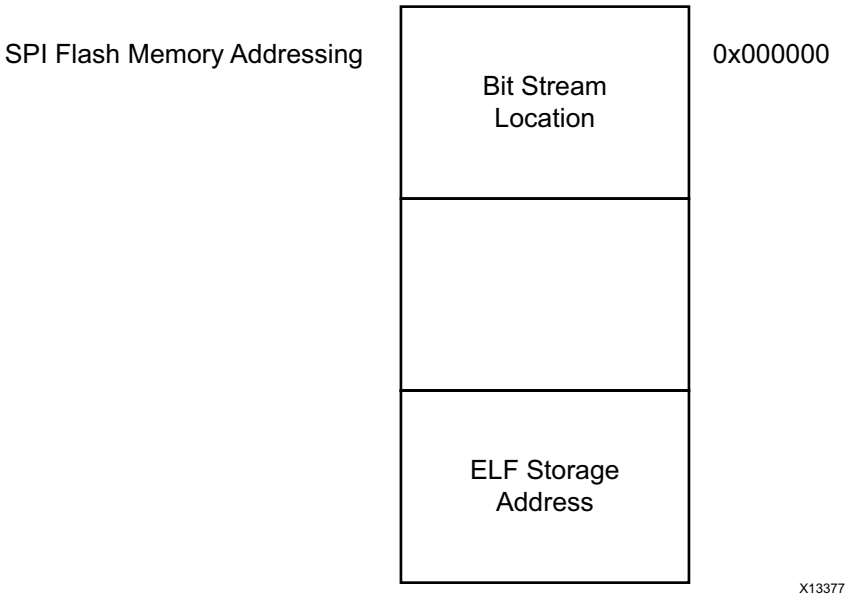


Figure 1: **SPI Flash Memory Address Space**

Table 3 lists the AXI Quad SPI core variables that are used in different configurations.

Table 3: **Core Parameters**

Feature/ Description	Core Configuration		Required In XIP Test
	Core Parameters	XGUI variables	
AXI4 memory mapped base address	C_S_AXI4_BASEADDR	These variables are system configurable	Cacheable Address
AXI4 memory mapped high address	C_S_AXI4_HIGHADDR		Cacheable Address
AXI4-Lite base address	C_BASEADDR		AXI Lite Address
AXI4-Lite high address	C_HIGHADDR		AXI Lite Address
Choice of AXI interface	C_TYPE_OF_AXI4_ INTERFACE	Enhanced Performance Mode	1
Choice of Non-XIP and XIP mode	C_XIP_MODE	Enable XIP Mode	1
SPI clock frequency ratio	C_SCK_RATIO	SPI Frequency Ratio	Default 2
SPI modes	C_SPI_MODE	SPI Mode 0 = Standard 1 = Dual 2 = Quad	2
SPI memory device used as SPI slave	C_SPI_MEMORY 1= Winbond 2 = Numonyx	SPI Slave Device	2

Hardware Requirements

The hardware board required for this system is:

- Xilinx KC705 evaluation board (Rev. C, D, 1.0 or 1.1)

The software design tool requirements for building and downloading the reference system are:

- Vivado 2013.2
- SDK 2013.2

System Diagram

The system for the XIP mode of the AXI Quad SPI core is shown in Figure 2.

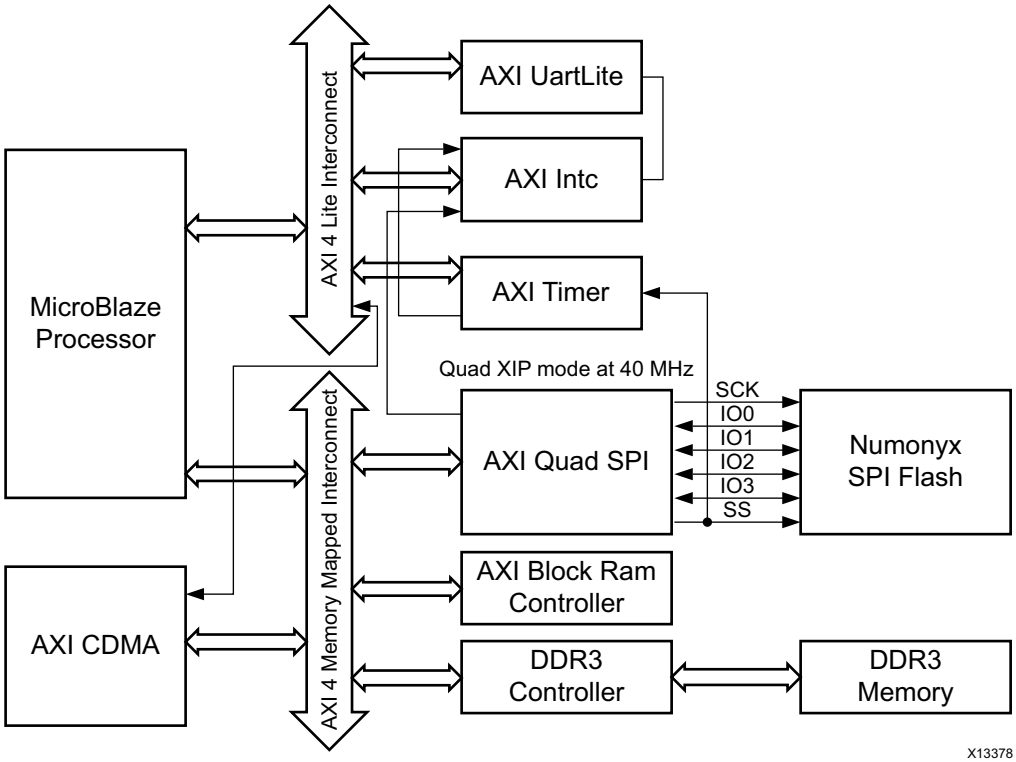


Figure 2: **XIP Mode – Typical System**

System Details

This system is based upon AXI interface, which is the standardized IP interface protocol based on the Advanced Micro-controller Bus Architecture (AMBA®) specification. The AXI interfaces used in the reference design consist of AXI4 memory mapped and AXI4-Lite interfaces. A clock generator and processor system reset block supplies clocks and resets throughout the system. High-level control of the system is provided by an embedded MicroBlaze processor subsystem containing I/O peripherals and processor support IP.

To optimize the system to balance performance and area, multiple AXI interconnect blocks are used to implement segmented/hierarchical AXI interconnect networks with each AXI interconnect block individually tuned and optimized. Table 4 shows the cores, versions and addresses for the system used for the XIP mode demonstration.

Table 4: **Demonstration System Cores and Addresses**

IP Core	Version	Base Address	High Address
MicroBlaze	9.0	N/A	N/A
MIG 7 Series	2.0	0x80000000	0xBFFFFFFF
AXI Quad SPI (AXI4 Full)	3.0	0xC4000000	0xC4FFFFFF

Table 4: **Demonstration System Cores and Addresses *(Cont'd)***

IP Core	Version	Base Address	High Address
AXI Quad SPI (AXI4 Lite)	3.0	0x44A10000	0x44A1FFFF
AXI BRAM Controller	3.0	0xC0000000	0xC001FFFF
AXI UARTLITE	2.0	0x40600000	0x4060FFFF
LMB BRAM IF Controller	4.0	0x00000000	0x0000FFFF
AXI Interrupt Controller (INTC)	3.0	0x41200000	0x4120FFFF
AXI BRAM Controller	3.0	0xC2000000	0xC201FFFF
AXI Timer	2.0	0x41C00000	0x41C0FFFF
Processor System Reset	5.0	N/A	N/A
Clock Generator	5.0	N/A	N/A
AXI Interconnect - Lite	2.0	N/A	N/A
AXI Interconnect - Full	2.0	N/A	N/A
AXI CDMA	4.0	0x44A00000	0x44A0FFFF
UTIL Reduced Logic	1.0	N/A	N/A

Configuring the IP Core in Vivado IP Integrator

The Vivado IP Integrator provides configuration options through XGUI after the desired core is added in the system. The options are shown in the following screen captures and explained below.

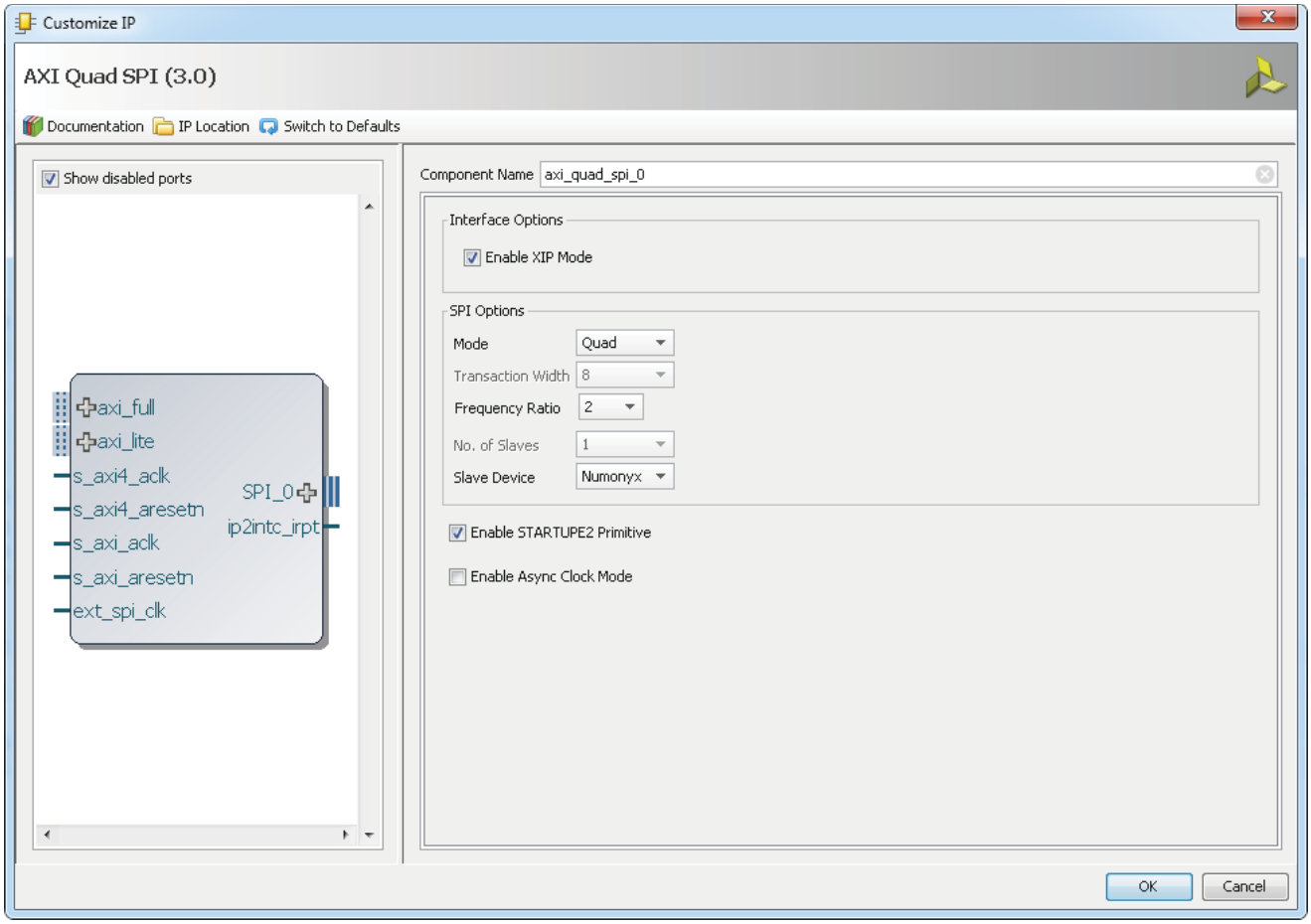


Figure 3: **Vivado IP Integrator Configuration**

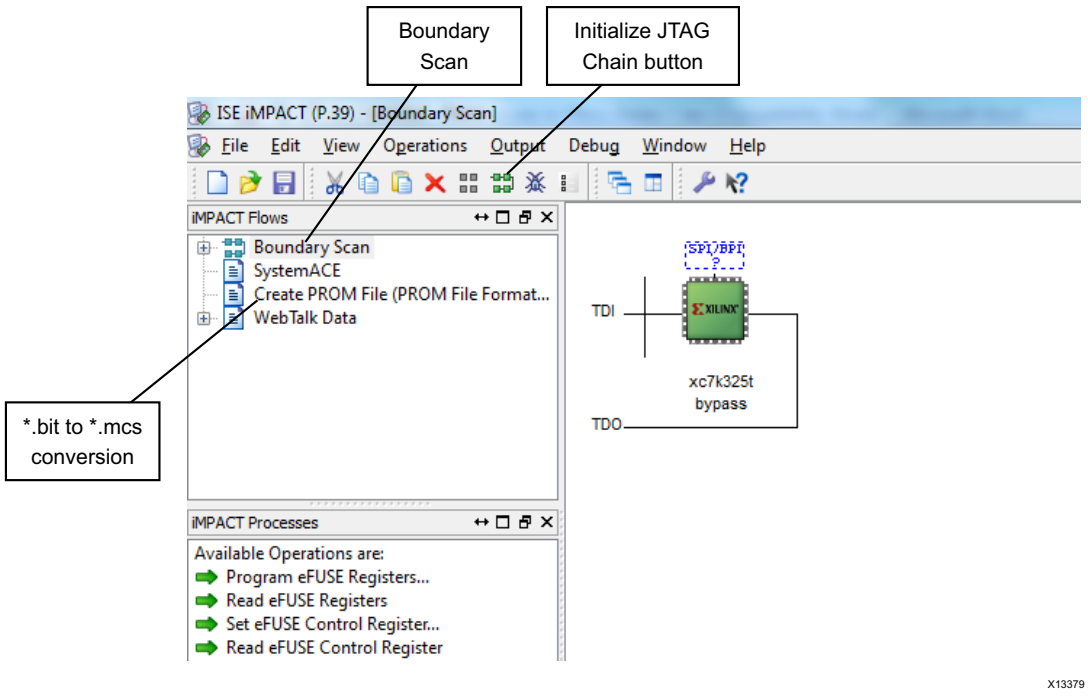
iMPACT and SPI Flash Programming

Xilinx Devices can boot from SPI flash. Before booting, the SPI flash must be configured with the bitstream and the boot-loop code. For this, it is necessary to merge the `top.bit` file (hardware information) and the boot-loop (software information) file into a single `download.bit` file. The iMPACT tool can convert the bit file into the MCS format file. bitstream to MCS Conversion explains how to covert the bit file into an MCS file which is used by iMPACT to program the downstream SPI flash device.

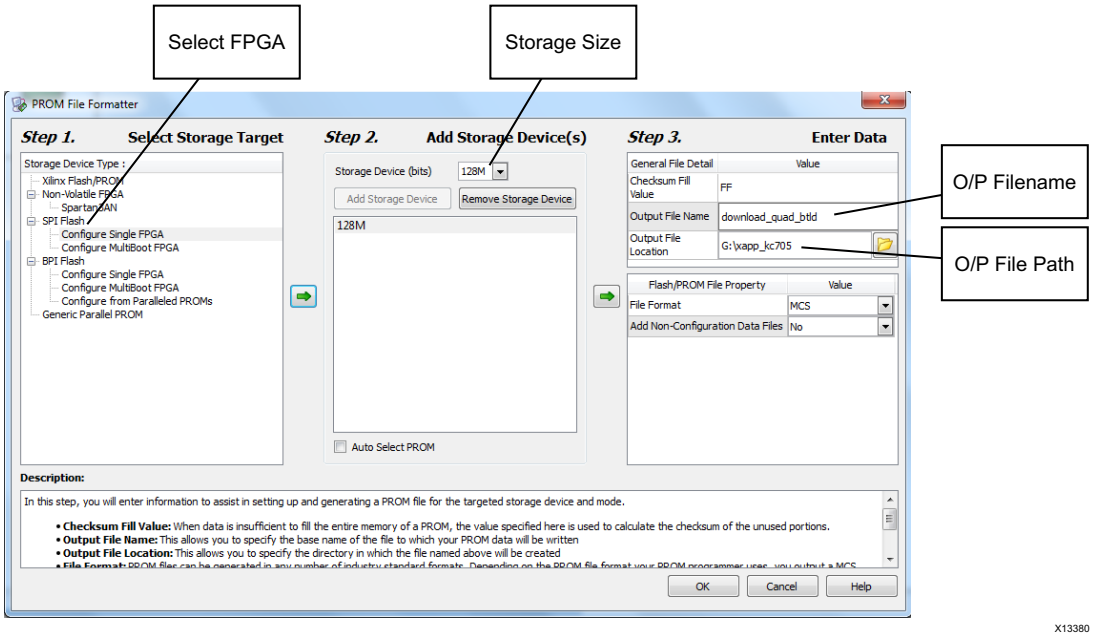
Bitstream to MCS Conversion

The following steps show how to convert the bitstream into an MCS file, merge the two MCS files and download to SPI flash.

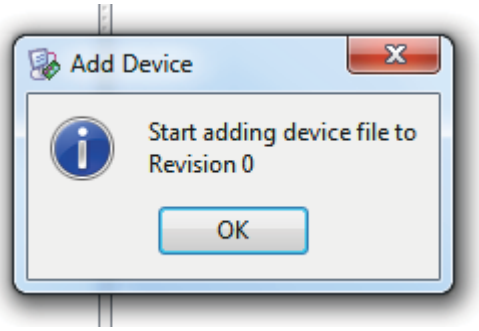
1. Open the iMPACT tool, perform a boundary-scan, initialize the JTAG chain, and select your device.



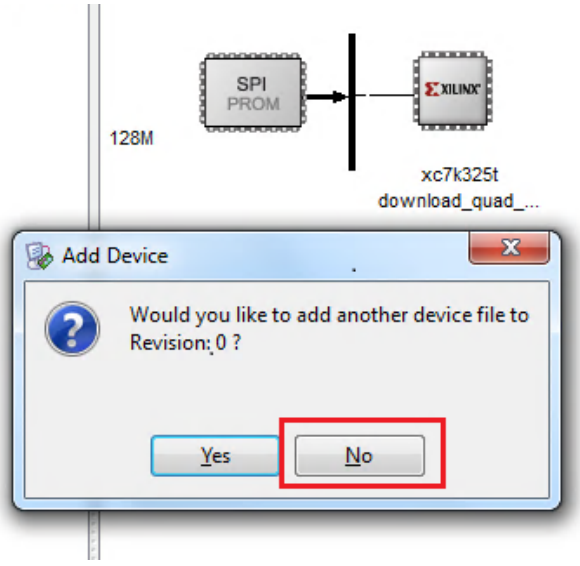
2. Double-click **Create PROM File**.
   a. Select **Configure Single FPGA** and click the arrow between Step 1 and Step 2.
   b. In **Add Storage Device** set the size of the storage device, and click the arrow between Step 2 and Step 3.
   c. The size of storage device can be found in the storage device data sheet.
   d. Enter the **Output File Name** and the **Output File Location** (directory location of your `application.elf` file) and select OK.



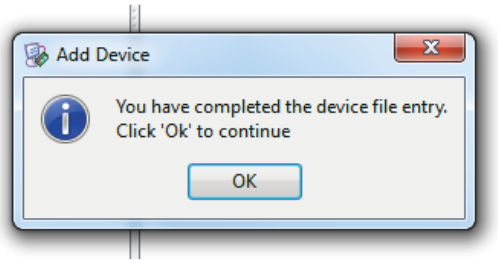
3. In the following window, select **OK** and select the bitfile that you want to convert.



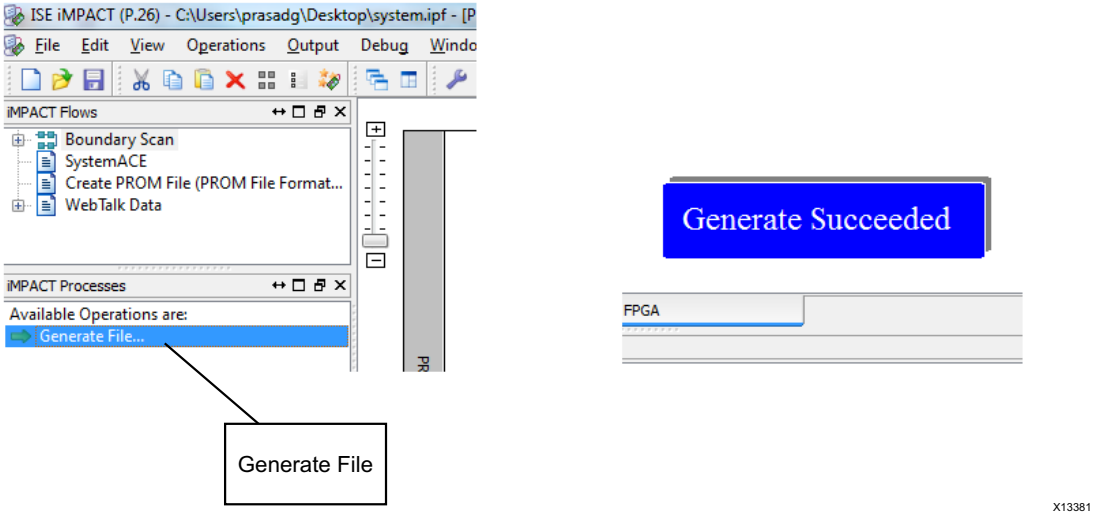
4. iMPACT adds the bitfile and in the following window, select **No** to finish.



5. The message is displayed as shown:



6. Select **OK** to continue, and double-click **Generate File** to generate the MCS file.



7. After generation of the MCS file, a **Generate Succeeded** message appears.

Merging Two MCS Files

SPI Flash programming through iMPACT can only be done using the MCS file format. You can use utilities to convert ELF files into MCS files. One of the MCS utilities can be downloaded from Xilinx by visiting this web page and downloading `xapp1053.zip`. Extract the zip file and go to the FLASH_BURN folder. Get the flash programming windows executable utilities, XIP and XMCSUTIL. Copy these files into the same folder that contains the `application.elf` file.

In addition to this utility, there is a batch file `bit_and_appl_to_mcs.bat` provided in this application zip file. Copy this file to the same folder that contains the `application.elf` file. Use the following steps to convert ELF files to MCS format.

1. Ensure that `bitfile.mcs`, `application.elf`, batch file, XIP and XMCSUTIL are located in the same folder.
2. To convert the `application.elf` file to binary use the following command at the XMD prompt:

```
mb-objcopy -O binary -R .vectors.reset -R .vectors.sw_exception -R
.vectors.interrupt -R .vectors.debug_sw_break -R .vectors.hw_exception
<application_name.elf> <application_name.b>
```

For example:

```
XMD%
XMD% mb-objcopy -O binary -R .vectors.reset -R .vectors.sw_exception -R
.vectors.interrupt -R .vectors.debug_sw_break -R .vectors.hw_exception
xip_led_app.elf xip_led_app.b
XMD%
```

Note: In this command, if you do not use *-R .vectors.reset -R .vectors.sw_exception -R .vectors.interrupt -R .vectors.debug_sw_break -R .vectors.hw_exception*, then the binary file size will be too big.

3. This command generates the binary file. Use this binary file to convert it to an MCS file using the following command:

   ```
   XMCSUTIL -accept_notice -i <application_name.b> -o <application_name.mcs> -29
   ```

   An example of the use of this command follows:

```
XMD%
XMD% XMCSUTL -accept_notice -i xip_led_app.b -o xip_led_app.mcs -29

=> Checking filenames
     - input/output/log filenames valid
xmcsutil<tm> Version 1.24
Copyright (c) 2001-2007 Xilinx, Inc. All rights reserved
Xilinx MCS/HEX Data Processing Utility
**********************************************************************
**//==================================================================\\**
**|| NOTICE: FOR XILINX PROTOTYPE USE ONLY                            ||**
**||                                                                  ||**
**||  SOFTWARE PROVIDED “AS IS”. ALL WARRANTIES, EXPRESS OR IMPLIED,  ||**
**||  ARE HEREBY DISCLAIMED. SOFTWARE NOT AUTHORIZED FOR USE IN       ||**
**||  PRODUCTION ENVIRONMENTS OR FOR USE IN OR WITH LIFE-SUPPORT OR   ||**
**||  MISSION-CRITICAL APPLIANCES, SYSTEMS, OR DEVICES.               ||**
**||                                                                  ||**
**||  This software is for use with Xilinx devices only. Please send  ||**
**||  all technical questions and comments to:                        ||**
**||                                                                  ||**
**||  xspi@xilinx.com                                                 ||**
**||                                                                  ||**
**\\==================================================================//**
**********************************************************************
===[Program notice/license accepted via -accept_notice command option]==
**********************************************************************
Mon May 20 17:10:53 2013
Converting bin file [xip_led_app.b] to MCS format [xip_led_app.mcs]
Converted [7900] bytes

- max MCS byte addr [0x00001EDB] in file [xip_led_app.mcs]

Elapsed clock time = 0 seconds
XMD%
```

4. The `application.mcs` and `bitfile.mcs` files have now been created. Use the following command to merge both MCS files into a single MCS file.

   ```
   XMCSUTIL -accept_notice -i <system_bitstream.mcs> <application_name.mcs>
   -o <combined.mcs> -16 -segaddr 0x00 <start of SPI flash address where elf
   to be stored> -usedataaddr -padff
   ```

   For example:

   ```
   XMCSUTIL -accept_notice -i download_btld_quad.mcs xip_led_app.mcs -o
   combined.mcs -16 -segaddr 0x00 0xC00000 -usedataaddr -padff
   ```

In this example, the `xip_led_app.mcs` file is stored at 0xC00000 SPI flash address. You can choose any other address as required. See the *Flash Memory Bootloading Using SPI with Spartan®-3A DSP 1800A Starter Platform Application Note* [Ref 5] for more information on how to configure the SPI addresses.

5. This procedure can also be executed using the batch file provided in this application zip file to avoid any errors while executing the commands.

   Edit the batch file with the correct bitfile name, application name and 'start of SPI flash address' and execute the following command at the XMD prompt:

   ```
   bit_and_appl_to_mcs.bat
   ```

   This gives the `combined.mcs` file.

```
XMD%
XMD% XMCSUTL -accept_notice -i download_btld_quad.mcs xip_led_app.mcs -o
combined_quad.mcs -16 -segaddr 0x00 0xC00000 -usedataaddr -padff

=> Checking filenames
     - input/output/log filenames valid
xmcsutil<tm> Version 1.24
Copyright (c) 2001-2007 Xilinx, Inc. All rights reserved
Xilinx MCS/HEX Data Processing Utility
**********************************************************************
**//==================================================================\\**
**|| NOTICE: FOR XILINX PROTOTYPE USE ONLY                            ||**
**||                                                                  ||**
**||  SOFTWARE PROVIDED "AS IS". ALL WARRANTIES, EXPRESS OR IMPLIED,  ||**
**||  ARE HEREBY DISCLAIMED. SOFTWARE NOT AUTHORIZED FOR USE IN       ||**
**||  PRODUCTION ENVIRONMENTS OR FOR USE IN OR WITH LIFE-SUPPORT OR   ||**
**||  MISSION-CRITICAL APPLIANCES, SYSTEMS, OR DEVICES.               ||**
**||                                                                  ||**
**||  This software is for use with Xilinx devices only. Please send  ||**
**||  all technical questions and comments to:                        ||**
**||                                                                  ||**
**||  xspi@xilinx.com                                                 ||**
**||                                                                  ||**
**\\==================================================================//**
**********************************************************************
===[Program notice/license accepted via -accept_notice command option]===
**********************************************************************
Mon May 20 17:11:06 2013

Combining [2] MCS files
-[2] MCS files will be combined into a single MCS file [combined_quad.mcs]
-> adding bytes from MCS files [download_btld_quad.mcs]; relocating to addr
[0x0]

    =>added [download_btld_quad.mcs] = 11443612 bytes | 91548896 bits
-> adding bytes from MCS file [xip_led_app.mcs]; relocating to addr [0xC00000]
   * added [1139300] 0xFF bytes at addr range [0x00AE9D9C : 0x00BFFFFF]
   => added [    xip_led_app.mcs] = 1147200 bytes | 9177600 bits

=> Total bytes written = [12590812 | 100726496 bits]
   - to output file [combined_quad.mcs]
   - max hex byte addr [0x00C01EDC]

Elapsed clock time = 183 seconds
XMD%
```

6. The following command provides the number of bytes in the MCS file:

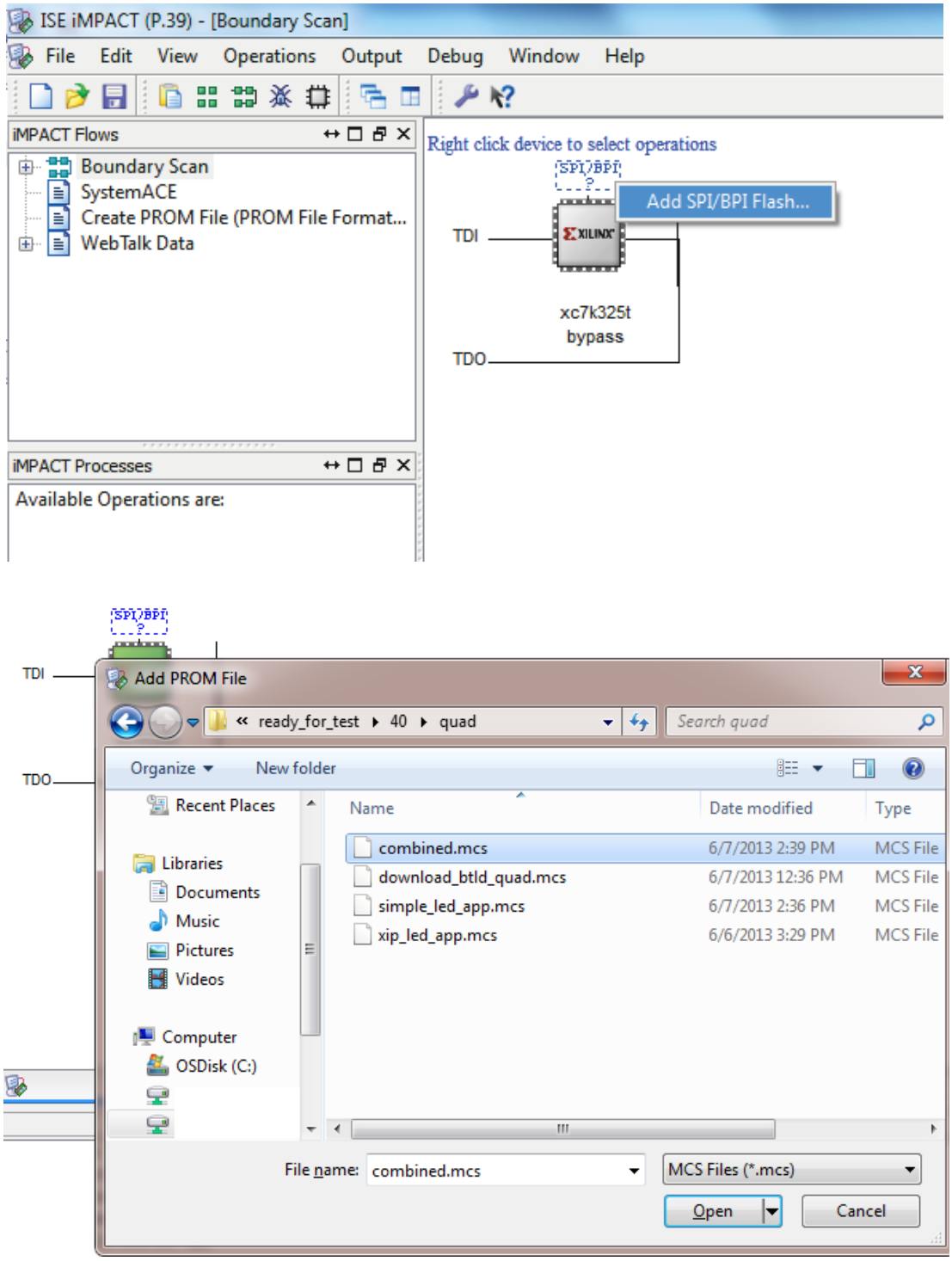
```
XMCSUTIL -bytecount -i <application_name.mcs> -o count.log
```

`count.log` can be used to get the number of bytes present in this file. This command should be executable from the directory where the `application.elf (xip_led_app.elf)` and the other two utilities are stored.

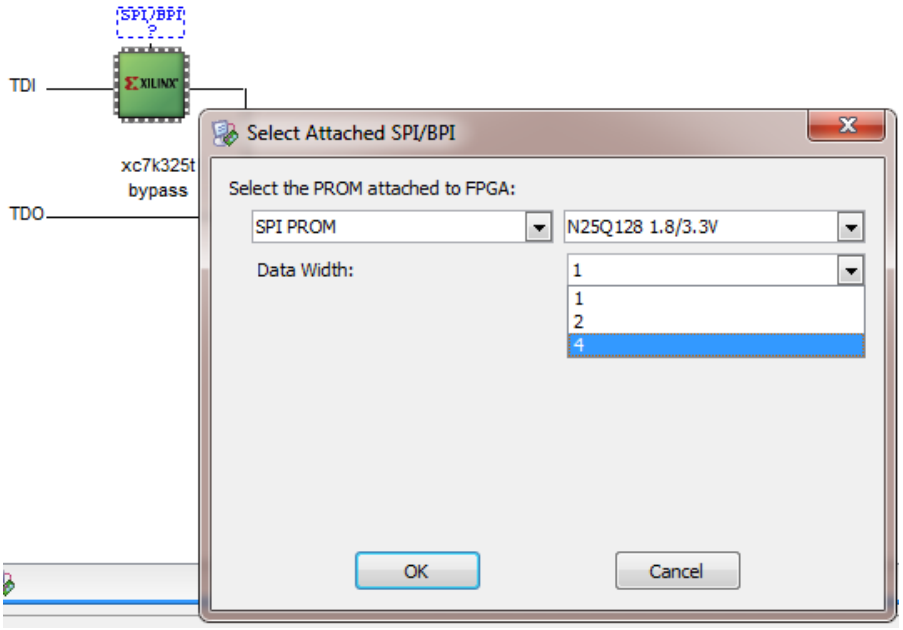
```
XMD%
XMD% XMCSUTL -byte_count -i xip_led_app.mcs -o count.log

=> Checking filenames
     - input/output/log filenames valid
xmcsutil<tm> Version 1.24
Copyright (c) 2001-2007 Xilinx, Inc. All rights reserved
Xilinx MCS/HEX Data Processing Utility
**********************************************************************
**//==================================================================\\**
**|| NOTICE: FOR XILINX PROTOTYPE USE ONLY                            ||**
**||                                                                  ||**
**||  SOFTWARE PROVIDED "AS IS". ALL WARRANTIES, EXPRESS OR IMPLIED,  ||**
**||  ARE HEREBY DISCLAIMED. SOFTWARE NOT AUTHORIZED FOR USE IN       ||**
**||  PRODUCTION ENVIRONMENTS OR FOR USE IN OR WITH LIFE-SUPPORT OR   ||**
**||  MISSION-CRITICAL APPLIANCES, SYSTEMS, OR DEVICES.               ||**
**||                                                                  ||**
**||  This software is for use with Xilinx devices only. Please send  ||**
**||  all technical questions and comments to:                        ||**
**||                                                                  ||**
**||  xspi@xilinx.com                                                 ||**
**||                                                                  ||**
**\\==================================================================//**
**********************************************************************
==> Use "-accept_notice" option to accept notice automatically
-++< Press ENTER to accecpt notice and continue >==-
**********************************************************************
Mon May 20 17:30:18 2013
==> MCS file 1 [xip_led_app.mcs] has [7900] bytes <63200 bits.

Elapsed clock time = 0 seconds
XMD%
```

Programming Flash Device with MCS File and Execution

1. Select the Boundary-scan window on the iMPACT tool. Right-click the SPI/BPI button. Click the Add SPI/BPI Flash button, and select the file to be programmed.

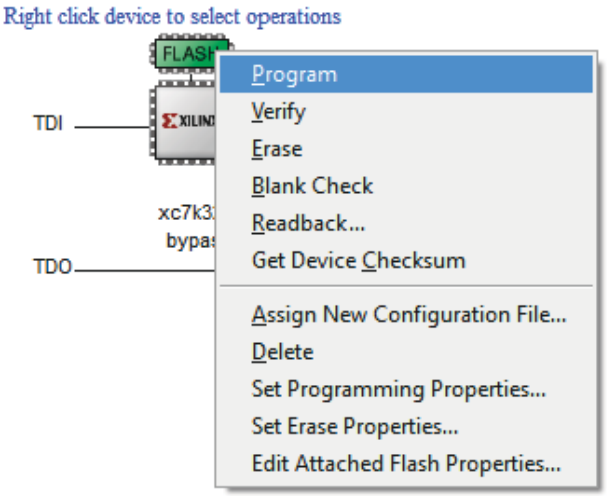


2. In the next window, select the correct flash device, and set the **Data Width** to 4 for Quad mode, if the flash device supports this. Select OK to select the flash. (In XIP mode, in the **Data Width** field, select 4, 2 or 1 for Quad, Dual or Standard mode respectively).



3. Right-click the **FLASH** button. Select **Program**. It shows **Generate Succeeded** when complete.

Note: Programming SPI flash through iMPACT can take up to 20 minutes.



4. After programming, switch off the board and then switch it back on. The FPGA is configured by the SPI Flash and the boot loader is executed. It copies `application.elf` from the SPI flash to the DDR memory and the program counter is set with the address of the DDR memory.
5. Make sure that hyperTerminal is ready before the FPGA is configured, otherwise no output is displayed on hyperTerminal.

6. The application is now executed. A representation of a sample output is shown:

```
Booting from SPI Flash in XIP DUAL IO Mode on Jun 7 2013 at 13:41:55

Axi Quad SPI Version: V3.0
Board used          : KC705 Rev 1.0
FPGA Device used    : xc7k325tffg900-2
Tool Used           : Vivado IPI
Tool Version        : 2013.2

Clearing the DDR ... @ 0x80000000
Loading ELF from SPI Flash (0xC4C00000) to DDR (0x80000000)

************************************************************************
* Hello World ... Using DUAL IO Mode (',')                             *
* Booted from DDR ...                                                  *
* Time taken for                                                       *
* Appl loading from SPI Flash to DDR ->                  837 us *
* Processor starts execution from DDR ->               463404 us *
* Throughput Measured (Appl Size)/(Appl load time) :     9.9 MB/s *
************************************************************************

Starting LED Test
LEDs glow for 5 times

Memory RD/WR test @ 0x80001F6C
Memory Test PASSED

Execution is Finished
```

7. To execute the booting application without resetting the board, press the RESET/ CPU RST push button on the board. The same application is executed again and you should see the same result.

Note: In the sample test output **Appl loading from SPI Flash to DDR** is the actual read time of the application from SPI flash to the DDR memory, when the chip select is active. **Processor starts execution from DDR** is the actual time taken by boot loader to load the application into DDR memory and start the application to be executed. **Throughput Measured** gives the actual throughput of the QSPI to read the application from flash in MB/s. The application size in this application note is ~8 KB.

Hardware Systems Provided

This section provides details about the hardware systems that are supported by this reference design and application note.

XIP Mode Test System - Quad Mode

1. In the XIP mode test system, the application provides the references for the procedure outlined in iMPACT and SPI Flash Programming, already completed for XIP mode. Download the MCS file in the SPI Numonyx flash from the folder and allow the FPGA to boot from SPI flash.

```
Booting from SPI Flash in XIP QUAD IO Mode on Jun 7 2013 at 12:32:28

Axi Quad SPI Version: V3.0
Board used          : KC705 Rev 1.0
FPGA Device used    : xc7k325tffg900-2
Tool Used           : Vivado IPI
Tool Version        : 2013.2

Clearing the DDR ... @ 0x80000000
Loading ELF from SPI Flash (0xC4C00000) to DDR (0x80000000)

*****************************************************************
* Hello World ... Using QUAD IO Mode (',')                      *
* Booted from DDR ...                                           *
* Time taken for                                                *
* Appl loading from SPI Flash to DDR ->              421 us     *
* Processor starts execution from DDR ->          462988 us     *
* Throughput Measured (Appl Size)/(Appl load time) :  19.7 MB/s *
*****************************************************************

Starting LED Test
LEDs glow for 5 times

Memory RD/WR test @ 0x80001F6C
Memory Test PASSED

Execution is Finished
```

2. The zip file, `kc705_xip_quad_mode`, contains the following folders:
   - HW – Contains the basic system files.
   - ready_for_download – contains the MCS file, and the `bit_and_appl_to_mcs.bat` file. The `bit_and_appl_to_mcs.bat` file is used to get the combined MCS file for both the bitstream and application files.

     To obtain the combined MCS file, place the `.bat` file in the same location as the `download.mcs` and `<application_name>.elf` files.

     For convenience, Xilinx has provided a combined MCS file (`boot_from_flash_quad_mode.mcs`) that can be used directly for testing. This file should be programmed in SPI flash through iMPACT.
   - SW – this folder contains the individual files for applications.
     - `bootloader` and `xip_quad_app` which is executed after booting.

XIP Mode Test System - Dual Mode

1. In the XIP mode test system, the application provides the references for the procedure outlined in iMPACT and SPI Flash Programming, already completed for XIP mode. Download the MCS file in the SPI Numonyx flash from the folder and allow the FPGA to boot from SPI flash.

```
Booting from SPI Flash in XIP DUAL IO Mode on Jun 7 2013 at 13:41:55

Axi Quad SPI Version: V3.0
Board used          : KC705 Rev 1.0
FPGA Device used    : xc7k325tffg900-2
Tool Used           : Vivado IPI
Tool Version        : 2013.2

Clearing the DDR ... @ 0x80000000
Loading ELF from SPI Flash (0xC4C00000) to DDR (0x80000000)

*********************************************************************
* Hello World ... Using DUAL IO Mode (',')                          *
* Booted from DDR ...                                               *
* Time taken for                                                    *
* Appl loading from SPI Flash to DDR ->                 837 us      *
* Processor starts execution from DDR ->              463404 us     *
* Throughput Measured (Appl Size)/(Appl load time) :   9.9 MB/s     *
*********************************************************************

Starting LED Test
LEDs glow for 5 times

Memory RD/WR test @ 0x80001F6C
Memory Test PASSED

Execution is Finished
```

2. The zip file, `kc705_xip_dual_mode`, contains the following folders:
   - HW – Contains the basic system files.
   - ready_for_download – contains the MCS file, and the `bit_and_appl_to_mcs.bat` file. The `bit_and_appl_to_mcs.bat` file is used to get the combined MCS file for both the bitstream and application files.

     To obtain the combined MCS file, place the `.bat` file in the same location as the `download.mcs` and `<application_name>.elf` files.

     For convenience, Xilinx has provided a combined MCS file (`boot_from_flash_dual_mode.mcs`) that can be used directly for testing. This file should be programmed in SPI flash through iMPACT.
   - SW – this folder contains the individual files for applications such as:
     - `bootloader` and `xip_dual_app` which is executed after booting.

Conclusion

This application note demonstrates the XIP mode where executable data is loaded from SPI flash to DDR memory before the processor starts execution. The AXI Quad SPI core provides this capability when configured in XIP mode. Quad mode always provides better performance than any other SPI mode. This application note is intended to demonstrate the use of the AXI Quad SPI IP and the mechanism used to write bootable applications.

Reference Design Details

Table 5 describes the contents of the reference design that accompanies this application note.

Table 5: **Reference Design Contents**

Parameters	Description
General	
Developer Name	Sanjay Kulkarni, Prasad Gutti
Target devices (stepping level, ES, production, speed grades)	Kintex7
Source code provided	Yes
Source code format	VHDL/Verilog (Some cores are encrypted)
Design uses code/IP from existing Xilinx application note/reference designs, CORE Generator™ software, or third party	Reference designs provided for Vivado and video cores generated from the CORE Generator tool
Simulation	
Design uses code/IP from existing Xilinx application note/reference designs, CORE Generator software, or third party	Reference designs provided for Vivado and video cores generated from the CORE Generator tool 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	
Implementation software tools/versions	Vivado Design Suit 2013.2
Implementation software tools/versions used	Vivado Design Suit 2013.2
Static timing analysis performed	Yes (Passing timing in PAR/TRCE)
Hardware Verification	
Hardware verified?	Yes
Hardware Platform used for verification	KC705 board

Device Utilization and Performance

This section provides the device resource usage estimates for both setups described in this application note.Table 6 describes the supported device.

Table 6: **Device Utilization**

Device	Speed Grade	Package	Slice Registers	Occupied Slices	Slice LUTs	I/Os	RAMB36/ FIFO36	RAMB18/ FIFO18
xc7l235t	–2	ffg900	15965	7190	19065	130	90	0

XIP System Utilization

Table 7 shows the module level utilization for the IP cores listed.

Table 7: **Module-level Utilization**

IP Core	Instance Name	Slices	Slice Registers	LUTs	LUT RAM	Block RAM	DSP Slice	BUFG CTRL	BUFR
axi_quad_spi	axi_quad_spi_1	1533	577	508	44	N/A	N/A	N/A	N/A
axi_intc	axi_intc_1	352	112	148	N/A	N/A	N/A	N/A	N/A
microblaze	microblaze_1	4467	1407	1700	198	11	3	N/A	N/A
mdm	mdm_1	125	76	31	04	N/A	N/A	1	N/A
mig_7series	mig_1	34380	10931	13183	2325	N/A	N/A	2	N/A
axi_bram_ctrl	axi_bram_ctrl_1	922	316	326	2	N/A	N/A	N/A	N/A
axi_bram_ctrl	axi_bram_ctrl_2	907	316	234	2	N/A	N/A	N/A	N/A
axi_cdma	axi_cdma_1	2361	1040	878	81	N/A	N/A	N/A	N/A
axi_timer	axi_timer_1	659	216	266	N/A	N/A	N/A	N/A	N/A
axi_gpio	axi_gpio_1	201	92	76	N/A	N/A	N/A	N/A	N/A
axi_uartlite	axi_uartlite_1	232	86	96	10	N/A	N/A	N/A	N/A
axi_interconnect_lite	axi_interconnect_1	567	120	231	N/A	N/A	N/A	N/A	N/A
axi_interconnect_full	axi_interconnect_2	2521	639	976	14	N/A	N/A	N/A	N/A
proc_sys_reset	proc_sys_reset_1	52	32	90	1	N/A	N/A	N/A	N/A

References

1. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator,* (UG994)
2. *Vivado Design Suite Quick Reference Guide*, (UG975)
3. *EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design* (UG683)
4. *LogiCORE IP AXI Quad Serial Peripheral Interface Product Guide*, (PG153)
5. Flash Memory Bootloading Using SPI with Spartan-3A DSP 1800A Starter Platform (XAPP1053)

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
07/26/2013	1.0	Initial Xilinx release.

Notice of Disclaimer