XAPP1234 (v 2.0) August 4, 2016

Throughput Performance Measurement for AXI Quad SPI IP

Author: Prasad Gutti

Summary

This application note demonstrates measurement of the SPI bandwidth by using the SPI flash memory in the Dual and Enhanced Quad modes of the AXI Quad SPI IP core for 1 MB of data. Results were obtained using the Xilinx KCU105 evaluation board for Kintex® UltraScale™ FPGA with Numonyx SPI memory. Measurement can be performed on other boards with a few modifications to the example software files.

The reference design systems are built using the Xilinx® Vivado® IP integrator provided with Vivado Design Suite: System Edition 2014.4. IP integrator is used to create systems by instantiating the processor, interconnect, interrupt controller, peripheral IP cores, memory controller, and UARTs. 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 IP integrator and SDK project files are provided in the reference design zip file to allow examining and rebuilding the design, or as a template for starting a new design.

This application note describes how to implement a Quad SPI system and how to measure throughput and bandwidth of QSPI in all modes using axi_timer, where the core is configured in Enhanced Quad, Dual and Standard SPI modes with a SPI clock rate of 100 MHz. This application note also shows the throughput values obtained in Quad mode is almost 2x times to that obtained in Dual mode.

Introduction

The AXI Quad SPI core supports Legacy, Enhanced and XIP modes. These three modes are further sub-categorized into three SPI modes, Standard, Dual and Quad mode. Standard mode commands use a single line (IO1), Dual mode uses two lines (IO0, IO1) and Quad mode uses four lines (IO0, IO1, IO2, IO3) to exchange data. Legacy mode supports applications that are based on an earlier version of the core (v1_00a). Enhanced mode supports the AXI4 Memory Mapped Interface which provides support for the fixed-burst capability of the transmit and receive FIFOs. 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 elements deep. Table 1 shows the AXI4 interface used for the different modes.

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

Mode	AXI4-Lite Interface	AXI4 Interface
Legacy Mode	Yes	-
Enhanced Mode	-	Yes
XIP Mode	Yes	Yes

One of the three SPI modes is selected based on the type of SPI slave used. Table 2 shows the SPI modes as well as the supported SPI clock frequency and the appropriate I/O interface.

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

SPI Modes	SPI Clock Division Ratio wrt. EXT_SPI_CLK	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 highest possible bandwidth on the SPI side, the core should be used in Quad mode where the data transactions use all four lines. The SPI bandwidth is best utilized in this mode because the supported commands are Fast Read Quad Output (0x6B h), Fast Read Quad I/O (0xEB h), and Quad Input Page Program (0x32 h), all of which support writing or reading of the SPI flash memory on four I/O lines.

Reference Design

To examine the system throughput for a 1 MB read from the SPI flash memory, the core must be configured in Quad mode for optimal performance. Winbond and Numonyx SPI memories now support Dual and Quad mode as the preferred mode over Standard mode. Dual mode commands improve the SPI bandwidth by approximately 2x, while the Quad mode commands improve the SPI bandwidth by 4x compared with SPI Standard mode.

With the AXI Quad SPI v3.2 core, the SPI bandwidth is used as lossless bandwidth. This means that as long as data is present in the Transmit data FIFO, the SPI clock is continuous. Use of efficient software coding techniques such as refilling the DTR FIFO when half empty or polling the DTR Occupancy register for empty space ensures that no idle cycles occur between the SPI transactions. These techniques are applicable to Legacy as well as Enhanced Quad modes. The time taken to write or read 1 MB of data is less in Enhanced Quad mode as compared to Legacy Quad mode due to the addition of burst capability. In all of these tests, an 100 MHz SPI clock is used while targeting the Numonyx memory on the KCU105 board.

Hardware

The AXI Quad SPI throughput measurement system design is shown in Figure 1.

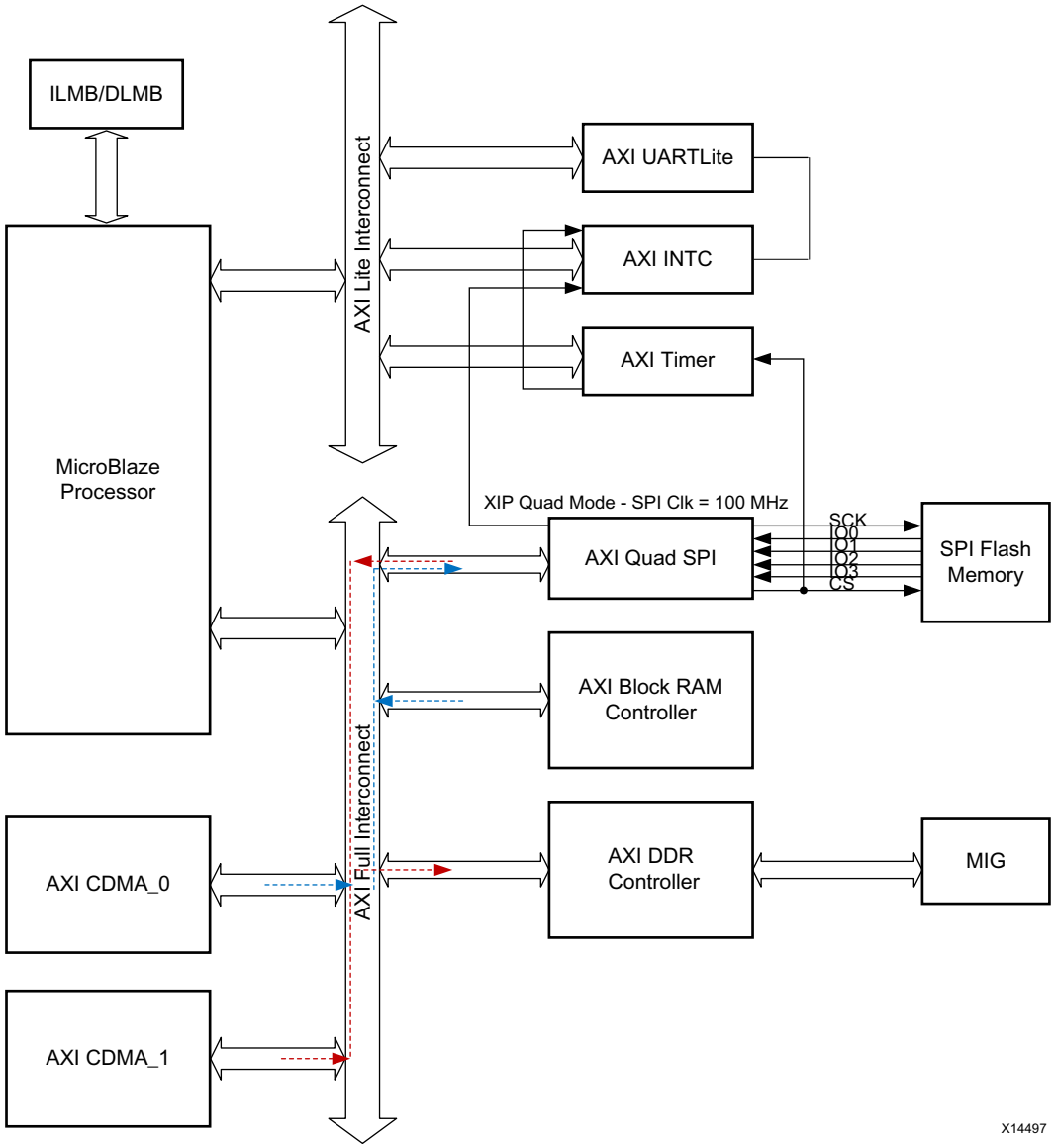

Figure 1: **Throughput Measurement System Design - Enhanced Quad Mode**

Software System

The software design of this application note has both write and read throughput measurements.

Figure 2 shows the logic for writing 1 MB of data into the SPI flash memory for measurement of the system throughput with an AXI4 memory-mapped interface.

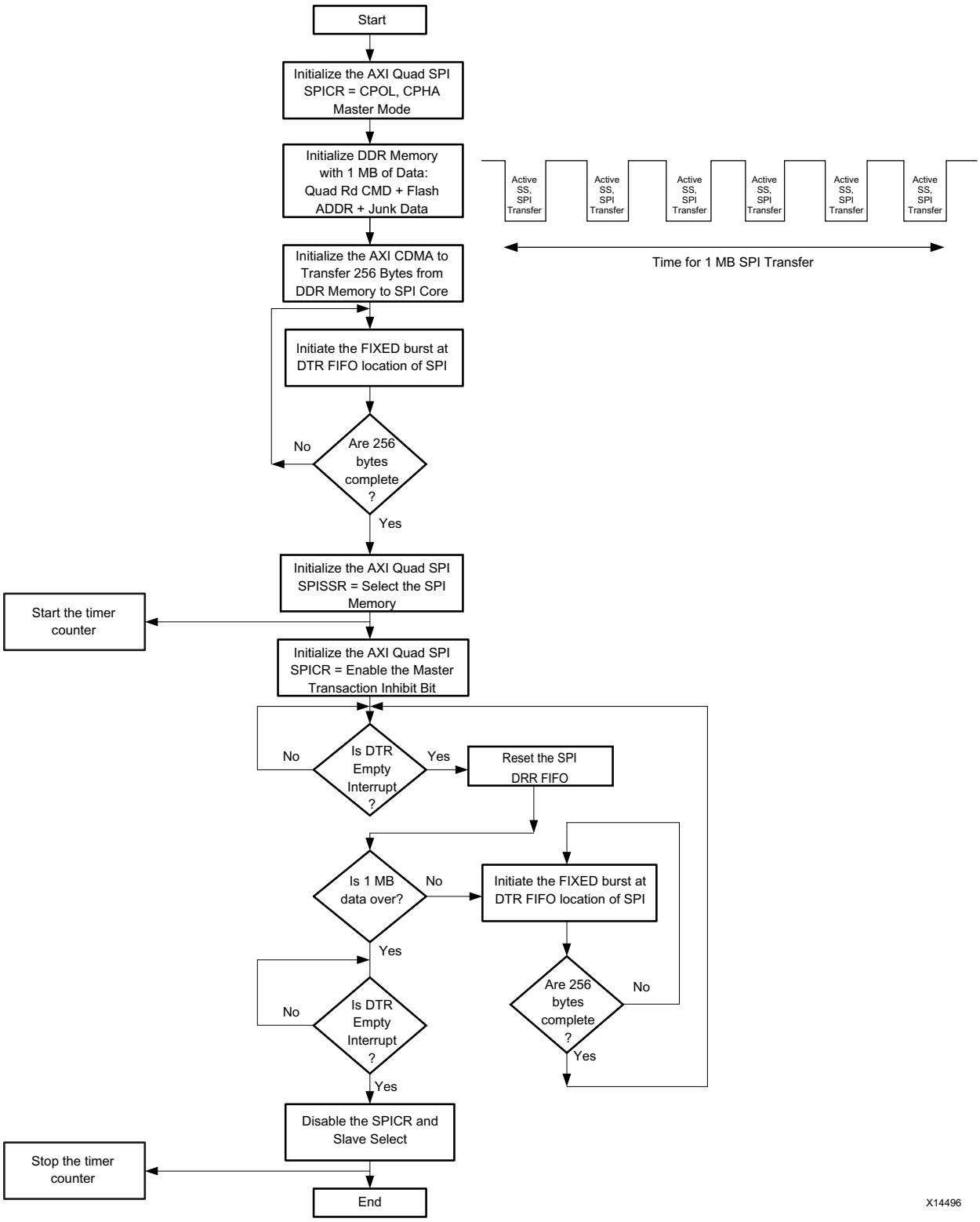

Figure 2: **Software Flowchart - Enhanced Quad Mode Write Transaction**

Figure 3 shows the software flowchart for reading 1 MB of data from the SPI flash memory.

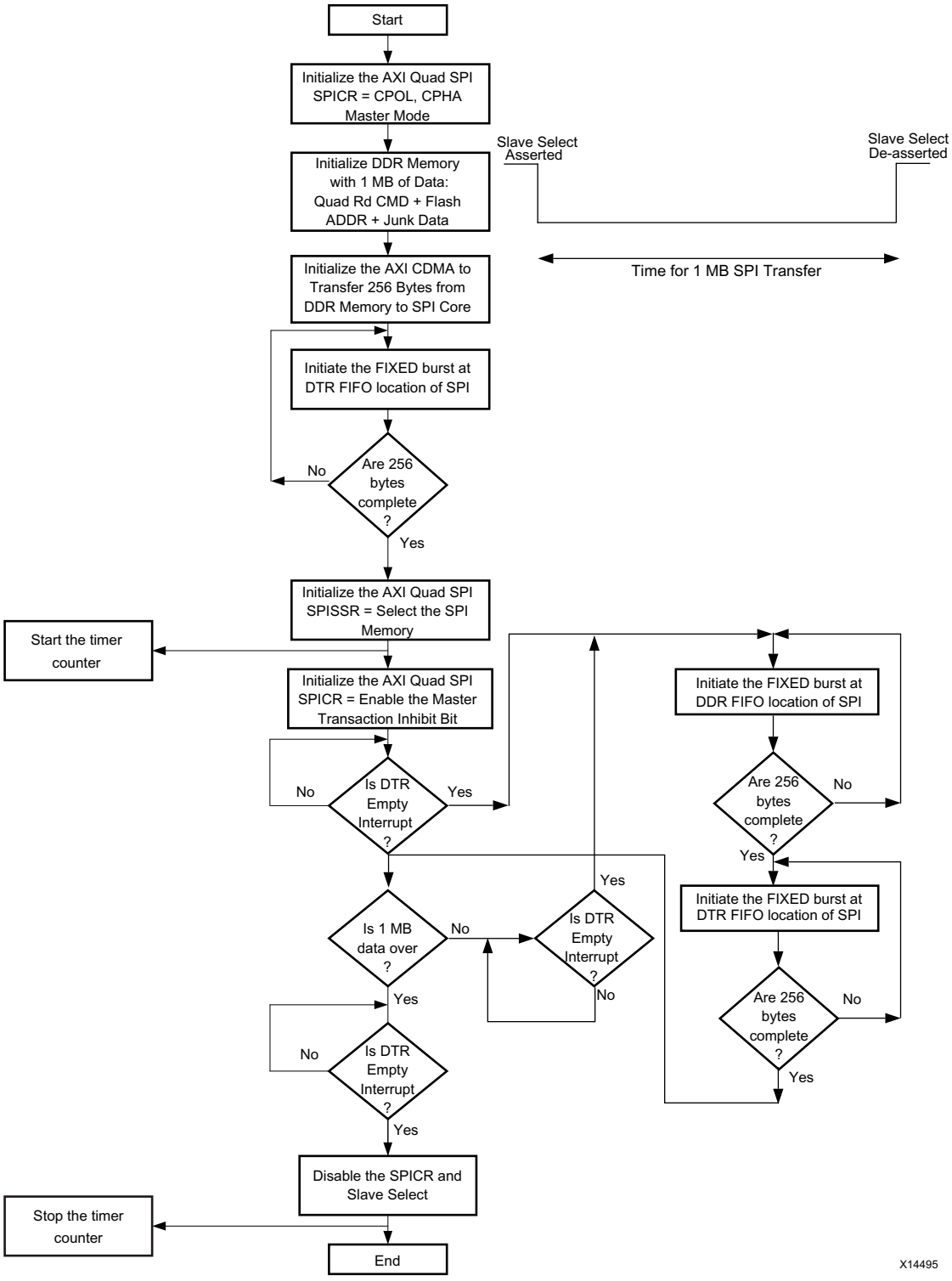

Figure 3: **Software Flowchart - Enhanced Quad Mode Read Transaction**

Write and Read Process Differences

All of the SPI flash devices have an 8-bit interface mode. Only after sending the command, address bytes, and dummy bytes does the actual data transmission start. This data is in 8-bit packet mode. Flash write is restricted to a single page while read has no such restriction. The SPI flash continues to send data to the host while the chip select is asserted and the SPI clock is active. This means that the SPI devices are comparatively slower for the write as opposed to the read process.

Tool Flow and Verification

Table 3 shows the tool flow and verification procedures used for the provided reference design.

Table 3: **Reference Design Checklist**

Parameters	Description
General	
Developer Name	Prasad Gutti
Target devices	Kintex UltraScale FPGA XCKU040-FFVA1156-2-E
Source code provided	Yes
Source code format	VHDL/Verilog (Some cores are encrypted)
Design uses code and IP from existing Xilinx application note and reference designs or third party	Reference designs provided for SDK and cores generated from the Vivado IP catalog
Implementation	
Synthesis software tools/version used	Vivado synthesis
Implementation software tools/versions used	Vivado implementation
Static timing analysis performed	Yes (passing timing in PAR/TRCE)
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KCU105 evaluation kit

Requirements

Hardware

The hardware board(s) and other equipment required for these systems are:

- Xilinx KCU105 evaluation board (Rev B)
- JTAG USB Platform cable or USB cable Type A to micro B
- Micro USB cable for UART data transmission

Software

- Vivado IP integrator 2014.4
- Vivado Design Suite: System Edition 2014.4
- SDK 2014.4

Reference Design Files

The design files for the reference design can be downloaded from:

- Quad Mode Reference Design - https://secure.xilinx.com/webreg/clickthrough.do?cid=378727
- Dual Mode Reference Design - https://secure.xilinx.com/webreg/clickthrough.do?cid=378726

Note: Registration is required to access these reference design files.

Figure 4 shows the directory structure of the reference design files provided with the application note. The application note has two systems corresponding to Enhanced Quad and Dual QSPI modes.

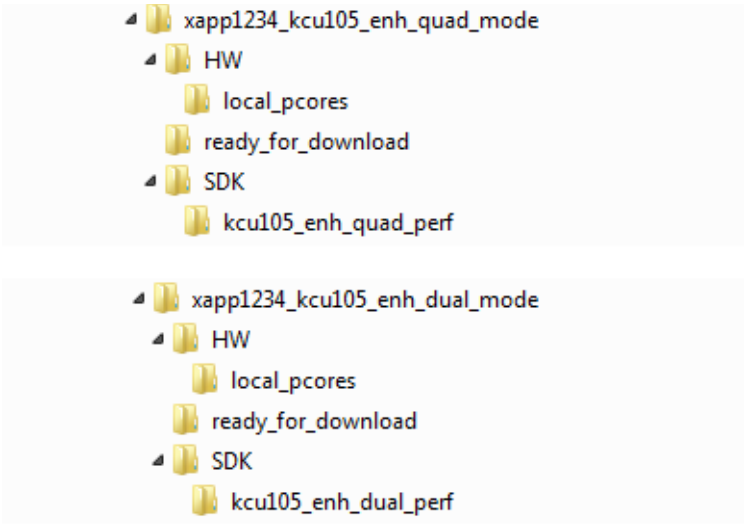

Figure 4: **Reference Design Files Directory Structure**

The `xapp1234_kcu105_enh_quad_mode.zip` file contains the following files and folders:

xapp1234_kcu105_enh_quad_mode: This is the top-level folder.

HW: This folder contains `local_pcores` folder, and the `all.tcl`, `design_1_wrapper.vhd`, `kcu105.xdc` files.

local_pcores: This folder is empty and it can be used to put any local pcores that is being used by the system, if any.

all.tcl: This file contains the Microblaze based complete hardware system for Quad SPI Enhanced Quad Mode using Tcl commands.

kcu105.xdc: This files contains LOC constraints required by System for Quad SPI and UART Lite IPs which are required to connect to external peripherals through pins.

design_1_wrapper.v: This is top-level wrapper file. This file contains top level ports of the system and has STARTUPE3 instantiated in it. STARTUPE3 is used to forward the clock to SPI flash memory.

ready_for_download: This folder contains the `design_1_wrapper_enh_quad_mode.bit` and the `kcu105_enh_quad_perf.elf` files. The `design_1_wrapper_enh_quad_mode.bit` file should be downloaded using the `fpga` XMD command. The *`kcu105_enh_quad_perf.elf`* file should be downloaded using the *`dow`* XMD command.

SDK: This folder contains software file that is being used by the system to measure throughput performance of Quad SPI system in Enhanced Quad Mode.

The `xapp1234_kcu105_enh_dual_mode.zip` file contains the following files and folders:

xapp1234_kcu105_enh_dual_mode: This is the top-level folder.

HW: This folder contains the `local_pcores` folder, and the `all.tcl`, `design_1_wrapper.vhd`, and `kcu105.xdc` files.

local_pcores: This folder is empty and it can be used to to put any local pcores that Is being used by the system if any.

all.tcl: This file contains the Microblaze based complete system for Quad SPI Enhanced DUAL Mode using Tcl commands.

kcu105.xdc: This files contains LOC constraints required by System for Quad SPI and UART Lite IPs which are required to connect to external peripherals through pins.

design_1_wrapper.vhd: This is top level wrapper file. This file contains top level ports of the system and has STARTUPE3 instantiated in it. STARTUPE3 is used to forward the clock to SPI flash memory. Quad SPI Dual mode design uses only two SPI_IO lines by the Quad SPI IP to transfer and receive SPI flash data.

Note: Due to limitation of Kintex UltraScale FPGAs, all four I/O pins are being used by the system: two pins are connected to IO0 and IO1, and the other two pins are used as dummy pins in the system as a workaround.

ready_for_download: This folder contains the `design_1_wrapper_enh_dual_mode.bit` and the `kcu105_enh_dual_perf.elf` files. The `design_1_wrapper_enh_dual_mode.bit` file should be downloaded using the `fpga` XMD command. The *`kcu105_enh_dual_perf.elf`* file should be downloaded using the *`dow`* XMD command.

SDK: This folder contains software file that is being used by the system to measure throughput performance of Quad SPI system in Enhanced Dual Mode.

Reference Design Steps

Setup

Figure 5 shows the Hardware setup for throughput measurement.

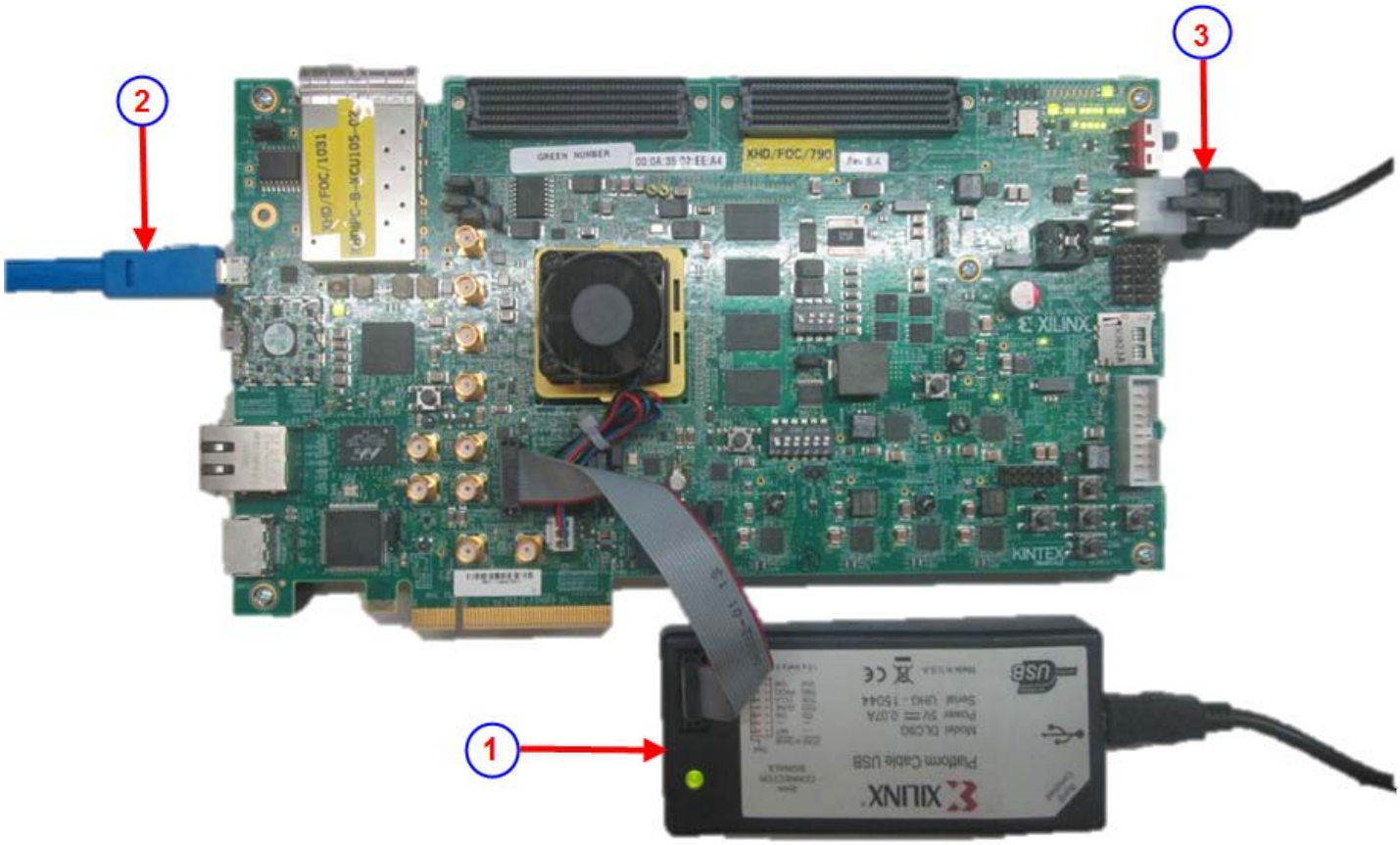

Figure 5: **Hardware Setup**

1. Connect a JTAG USB Platform cable or a USB Type A to Micro B cable from the host PC to the KCU105 board for programming bit and elf files. (In this setup a JTAG Platform cable is used.)
2. Connect a Micro USB cable from the host PC to the USB UART port on the KCU105 for serial communication.
3. Connect the power supply cable and turn on the KCU105 board.
4. Start a Putty program on the host PC with the following settings:
    - Baud rate: **9600**

- Data Bits: **8**
- Parity: **None**
- Stop Bits: **1**
- Flow Control: **None**

Running the Reference Design

This section describes how to build the reference design for both hardware and software.

1. Before beginning, extract the reference design into a local directory. The reference design is referred to as *XAPP1234* in this procedure.
2. You can either:
   - Build the reference design (including generating the BIT and ELF files), and test on the hardware. To do so, follow:
     - Building the Reference Design
     - Running the Design on the Hardware

   or,

   - Use the pre-generated BIT and ELF files available in the XAPP1234 reference design, and test on the hardware. To do so, only follow:
     - Running the Design on the Hardware.

Building the Reference Design

The following steps guide you through building the generating the hardware design (including generating the BIT and ELF files), and the SDK workspace.

Note: **Skip this section** if you plan to use the BIT and ELF files available in with the XAPP1234 reference design, and start at Running the Design on the Hardware.

Creating a Vivado Design Tools Project, and Generating the Bitstream

This section details the steps to start a new Vivado Design Suite 2014.4 project.

1. Launch the Vivado Design Suite.
2. Open the **Tcl Console** in the Vivado Integrated Design Environment (IDE). If you do not see the Tcl Console, select **Window >Tcl Console**.

3. In the Tcl Console, go to the HW directory:

   ```
   cd xapp1234_kcu105_enh_quad_mode/HW
   ```

   or

   ```
   cd xapp1234_kcu105_enh_dual_mode/HW
   ```

4. Source the given `all.tcl` file.

   ```
   source all.tcl
   ```

After the project is created, the Vivado Design Suite generates the output products, synthesizes and implements the design, and generates the bitstream.

Figure 6 shows the address mapping of all the IP cores with the MicroBlaze processor in the system.

Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
microblaze_0					
Instruction (32 address bits : 4G)					
mig_0	C0_DDR4_S_AXI	C0_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF
lmb_bram_if_cntlr_1	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
axi_quad_spi_0	AXI_FULL	MEM0	0xC400_0000	64K	0xC400_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF
Data (32 address bits : 4G)					
mig_0	C0_DDR4_S_AXI	C0_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF
lmb_bram_if_cntlr_0	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
axi_quad_spi_0	AXI_FULL	MEM0	0xC400_0000	64K	0xC400_FFFF
axi_intc_0	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
axi_cdma_1	S_AXI_LITE	Reg	0x44A2_0000	64K	0x44A2_FFFF
axi_cdma_0	S_AXI_LITE	Reg	0x44A1_0000	64K	0x44A1_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF
axi_cdma_1					
Data (32 address bits : 4G)					
mig_0	C0_DDR4_S_AXI	C0_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF
axi_quad_spi_0	AXI_FULL	MEM0	0xC400_0000	64K	0xC400_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF
axi_cdma_0					
Data (32 address bits : 4G)					
mig_0	C0_DDR4_S_AXI	C0_DDR4_AD...	0x8000_0000	1G	0xBFFF_FFFF
axi_quad_spi_0	AXI_FULL	MEM0	0xC400_0000	64K	0xC400_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC200_0000	128K	0xC201_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	128K	0xC001_FFFF

Figure 6: **Base and High Addresses of the IP Core in IP Integrator**

Generating ELF using Xilinx SDK

After bitstream generation is completed:

1. Launch the Xilinx SDK.
2. When prompted, specify `project_1/project_1.sdk` as the workspace path and click **OK**.
3. Select **File > New > Board Support Package** to generate BSP.

4. Select **File > New > Applicaton project** to create a new application project.
5. Name the project as `kcu105_enh_quad_perf`, and select **Use Existing BSP** for this project.
6. Import the software source code that is provided with this package from `SDK` folder to the `kcu105_enh_quad_perf/src` folder.
7. Xilinx SDK generates an ELF file.

Running the Design on the Hardware

The following steps are used to run the bitstream and ELF files on the hardware setup:

1. Connect the JTAG cable and USB-UART cable to the board.
2. Create a `test` folder.

IMPORTANT: *To access the pre-generated BIT and ELF files available in the XAPP1223 reference design for testing:*
- Go to the `ready_for_download` folder (instead of creating a test folder), and
- Skip step 3, step 4 and step 5, and start at step 6.

3. Copy the `design_1_wrapper.bit` file from `project_1/project_1.runs/impl_1/` and paste to the `test` folder.
4. Copy the `kcu105_enh_quad_perf.elf` file from `project_1/project_1.sdk/kcu105_enh_quad_perf/Debug/` and paste to the `test` folder.
5. Go to the `test` folder:

```
cd test
```

6. Start the Xilinx Microprocessor Debugger (XMD) by typing `xmd` in the command prompt.
7. Configure the FPGA with `design_1_wrapper.bit` through a JTAG cable using this command at the XMD prompt:

```
fpga -f design_1_wrapper.bit
```

8. To connect to the processor running on the FPGA type:

```
connect mb mdm
```

9. Reset and stop the FPGA using these commands at the XMD prompt:

```
rst
stop
```

10. To observe the results, open hyperTerminal and configure it to 9600 baud with default configuration. Make sure that the UART cable is connected to the board and the PC.
11. Download the ELF file into memory (block RAM or DDR) and execute the software on the board.

```
stop; rst; dow kcu105_enh_quad_perf.elf
run
```

12. When the test has been completed and displays SUCCESS, PASSED, or FAILED on the hyperTerminal, stop the processor by typing:

```
stop
```

Results

Sample output screens are shown in Figure 7 and Figure 8.

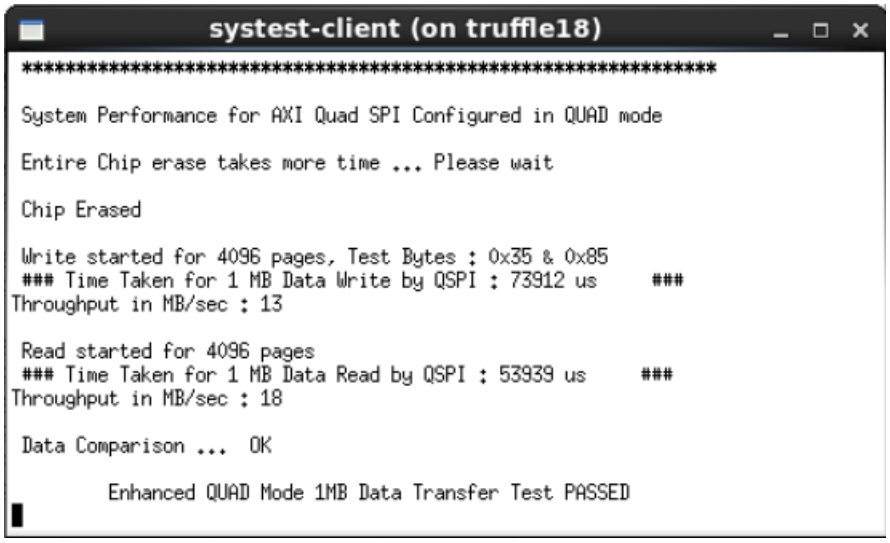

Figure 7: **Sample Output for Enhanced Quad Mode**

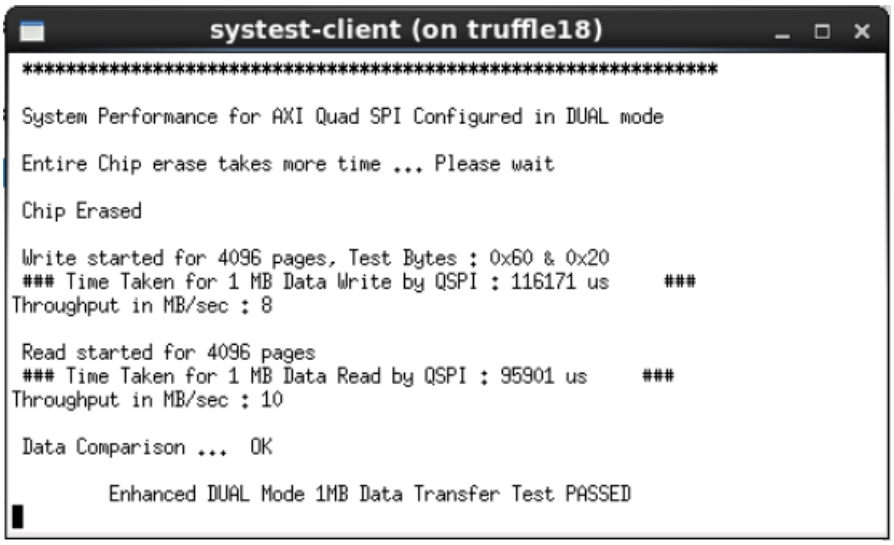

Figure 8: **Sample Output for Enhanced Dual Mode**

References

1. *Throughput Performance Measurement for AXI Quad SPI IP (Kintex-7) Application Note* (XAPP797).
2. *LogiCORE IP AXI Quad SPI Product Guide* (PG153)
3. *Flash Memory Bootloading Using SPI with Spartan-3A DSP 1800A Starter Platform* (XAPP1053)
4. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/04/2016	2.0	• Updated Figure 5: Hardware Setup • Updated minor typographical edits
02/11/2015	1.0	Initial Xilinx release

Please Read: Important Legal Notices