XAPP1218 (v2.0) February 18, 2015

AXI VDMA Reference Design for the Kintex FPGA KC705 Evaluation Board

Authors: Dinesh Kumar, Prasad Gutti

Summary

This application note demonstrates how to use the LogiCORE™ IP AXI Video DMA (VDMA) core in a typical video application. It describes hardware and software APIs. The hardware system uses two VDMAs with both MM2S and S2MM paths shorted.

You can refer to this design and use the API in video applications that use VDMA. This application note demonstrates VDMA in loopback mode, but the MM2S and S2MM can also each be connected to separate video IP.

The reference design is created and built using Vivado® IP integrator version 2014.4, which is part of Vivado System Edition. The IP integrator is an interactive design and verification environment that you can use to create and verify a hierarchical system by graphically connecting Xilinx, third-party, or proprietary IP, using interface-level connections. It provides a device- and platform-aware, interactive environment that supports intelligent auto-connection of key IP interfaces, one-click IP subsystem generation, real-time DRCs, interface change propagation, and powerful debug capability.

The design also includes software built using the Xilinx Software Development Kit (SDK). The software runs on the MicroBlaze™ processor subsystem and implements control, status, and monitoring functions. Complete Vivado tools and SDK project files are provided with the reference design to allow you to examine and rebuild the design or to use as a template for a new design.

This application note describes how to implement a VDMA system using Vivado IP integrator to read and write buffered data from/to memory continuously and to monitor its callback interrupts after completion of each buffered data transfer.

Reference Design

IP Cores

In addition to a MicroBlaze processor, the reference design includes the following cores:

- MDM
- LMB block RAM
- AXI_INTERCONNECT

- Clock Wizard
- PROC_SYS_RESET
- AXI_UARTLITE
- AXI_INTC
- Memory Interface Generator (MIG)
- Video Direct Memory Access (VDMA)

Hardware

Figure 1 shows an IP integrator block diagram of the reference system.

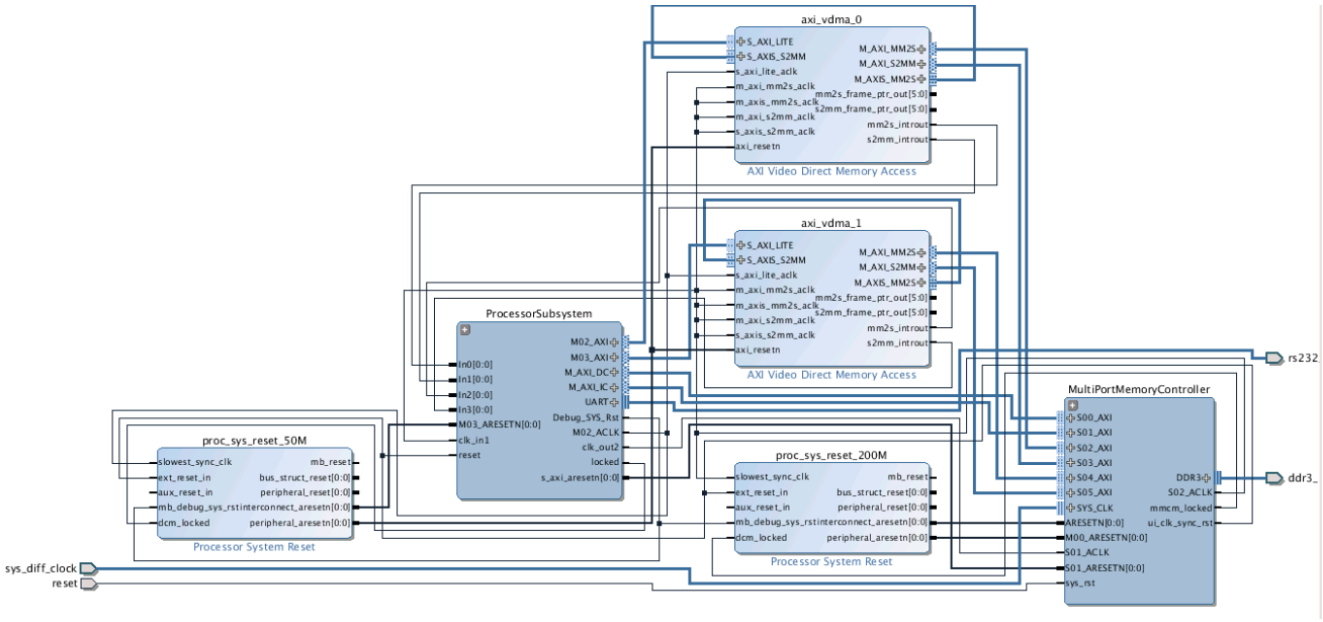

Figure 1: **IP Integrator Block Diagram**

The block diagram shows the system divided into the following modular blocks:

- Multiport memory controller
- Processor subsystem
- Two VDMAs
- Two processor system resets

Multiport Memory Controller

The multiport memory controller consist of an AXI interconnect and a MIG system (Figure 2). The frequency and data width of the MIG and AXI interconnect are set to meet the bandwidth requirement of a typical video application.

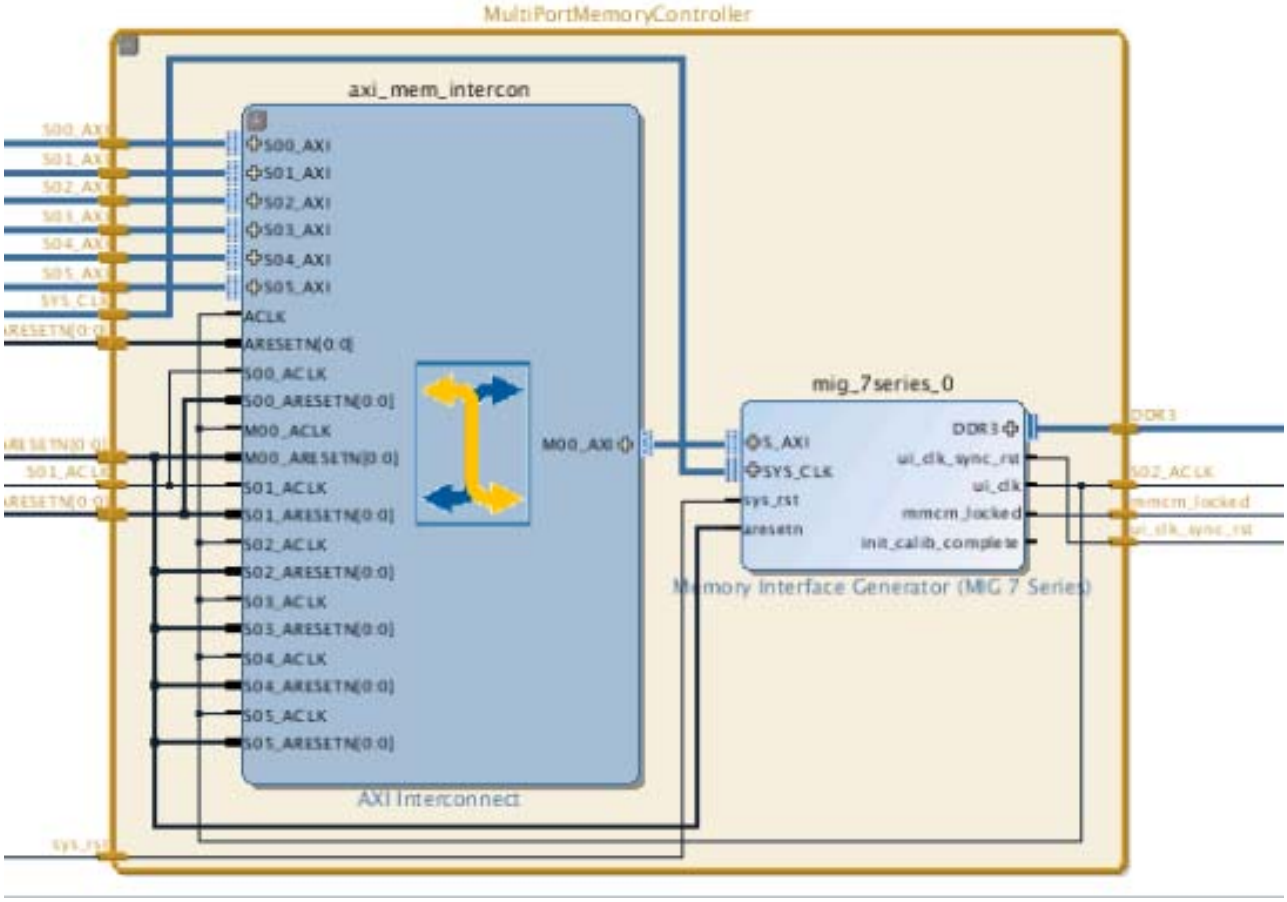

Figure 2: **Multiport Memory Controller**

Processor Subsystem

The processor subsystem consists of a MicroBlaze processor and the IP required for the system to function correctly (Figure 3). An AXI interconnect connects peripherals including local memory, UART Lite, and the interrupt controller to the processor. It also includes a clocking block and a microprocessor debug module (MDM) to enable downloading and debugging through the Xilinx microprocessor debugger (XMD). A reset block manages reset synchronization.

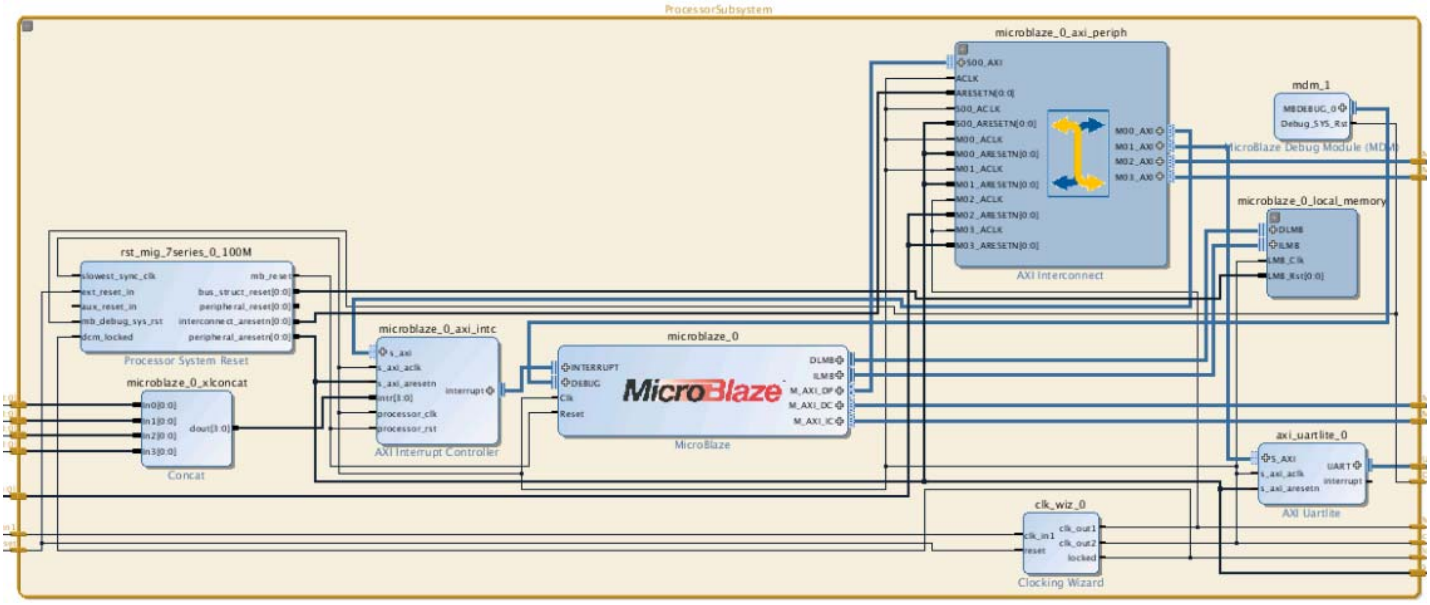

Figure 3: **Processor Subsystem**

VDMAs

The system uses two VDMAs. The MM2S path of one VDMA is connected to its S2MM path. The VDMAs are configured to allow programming of a frame counter and frame counter interrupt by setting C_ENABLE_DEBUG_INFO_7 and C_ENABLE_DEBUG_INFO_15. The VDMA is configured according to the following specifications:

- 200 MHz AXI4-Stream clocks
- 200 MHz AXI clocks
- 50 MHz AXI4-Lite clock
- 1920 by 1080 frame size
- Three frame buffers in circular mode
- S2MM in start-of-file (SOF) on TUSER mode, and MM2S in free-run mode
- Dynamic genlock mode with the S2MM side as the master and MM2S as the slave
- Tdata widths of 24 bits
- MM data width of 32 bits
- DRE disabled; accesses must be properly aligned
- Error interrupts optionally enabled in the software application
- Frame count interrupt enabled to assert every frame in the software application

Using two VDMAs shows that the APIs used in the design are modular and that the same API can be called repeatedly to configure each VDMA.

Processor System Reset

Processor system reset blocks are used to synchronize resets across various clocks used in system.

Software

The software package consists of BSP, `system.xml` files, and the following application files.

vdma_api.c

The main file, consisting of APIs to initialize, configure, and start VDMA.

The API prototype is:

```
int run_triple_frame_buffer(XAxiVdma* InstancePtr, int DeviceId, int hsize, int vsize,
int buf_base_addr, int number_frame_count, int enable_frm_cnt_intr);
```

- `instancePtr` - AXI VDMA driver instance pointer.
- `DeviceId` - Device ID for VDMA to allow the function to work for multiple VDMAs in a system.
- `hsize` - Horizontal size in pixels. The actual data transfer is (`hsize`) x (`tdatawidth`).
- `vsize` - Vertical size in lines.
- `number_frame_count` - The number of frames after which the application expects an interrupt from the VDMA.
- `enable_frm_cnt_intr` - Enable frame count interrupts.

The API returns `XST_SUCCESS` or `XST_FAILURE` based on the configuration of the VDMA.

vdma_api.h

The file containing the prototype of the API; can be added in the application.

vdma.c

Consists of the VDMA application to demonstrate how to use the VDMA triple buffer API.

vdma.h

Includes various files required for the application.

Tool Flow and Verification

The reference design has been fully verified and tested on hardware. The design includes details on the various functions of the different modules and includes all functions required to implement a complete VDMA design.

Table 1 shows the tool flow and verification procedures used for the provided reference design.

Table 1: **Reference Design Checklist**

Parameter	Description
General	
Developer name	Dinesh Kumar, Prasad Gutti
Target devices (stepping level, ES, production, speed grades)	Kintex-7 (KC705) FPGA
Source code provided	Yes
Source code format	VHDL/Verilog (some sources encrypted)
Design uses code and IP from existing Xilinx application note and reference designs or third party	Provided reference designs use core generated in Vivado Design Suite 2014.4
Implementation	
Synthesis software tools/version used	Vivado synthesis
Implementation software tools/versions used	Vivado implementation
Static timing analysis performed	Yes (passing timing in)
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	KC705 board

Requirements

Hardware

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

- Kintex-7 FPGA KC705 Evaluation Kit Base Board (Figure 4)
- JTAG USB Platform cable or USB cable Type-A to micro-B
- Mini USB cable for uart data transmission

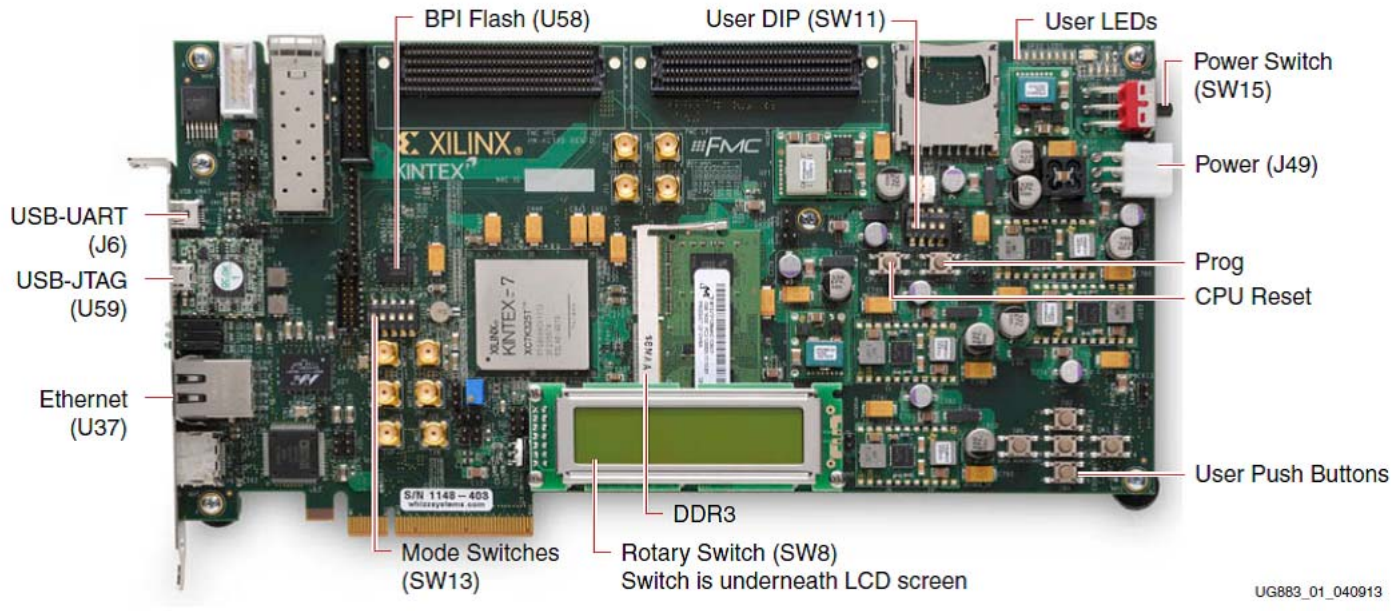

Figure 4: **KC705 Evaluation Kit**

Software

- Vivado Design Suite 2014.4
- Xilinx Software Development Kit 2014.4
- Tera Term/putty terminal emulator for UART serial communication through a COM port

Reference Design Files

You can download the Reference Design Files for this application note from the Xilinx website.

Figure 5 shows the directory structure of the reference design files provided with the application note.

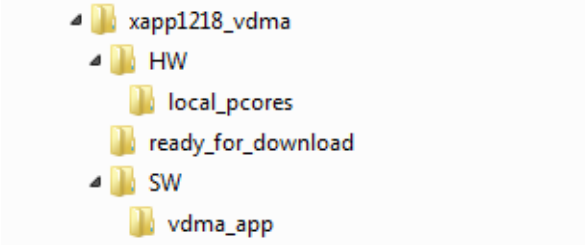

Figure 5: **Directory Structure**

xapp1218_vdma: The top-level folder.

HW: This folder contains `local_pcores` folder and `all.tcl` file.

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 design for the AXI VDMA IP using Tcl commands.

ready_for_download: This folder contains the `design_1_wrapper.bit` and the `vdma_app.elf` files.

design_1_wrapper.bit: This file should be downloaded using the `fpga` XMD command.

vdma_app.elf: This file should be downloaded using the `dow` XMD command.

SDK: This folder contains software files that are being used by the system to transfer buffered packets from/to memory.

Reference Design Steps

Setup

Figure 6 shows the hardware setup for throughput measurement.

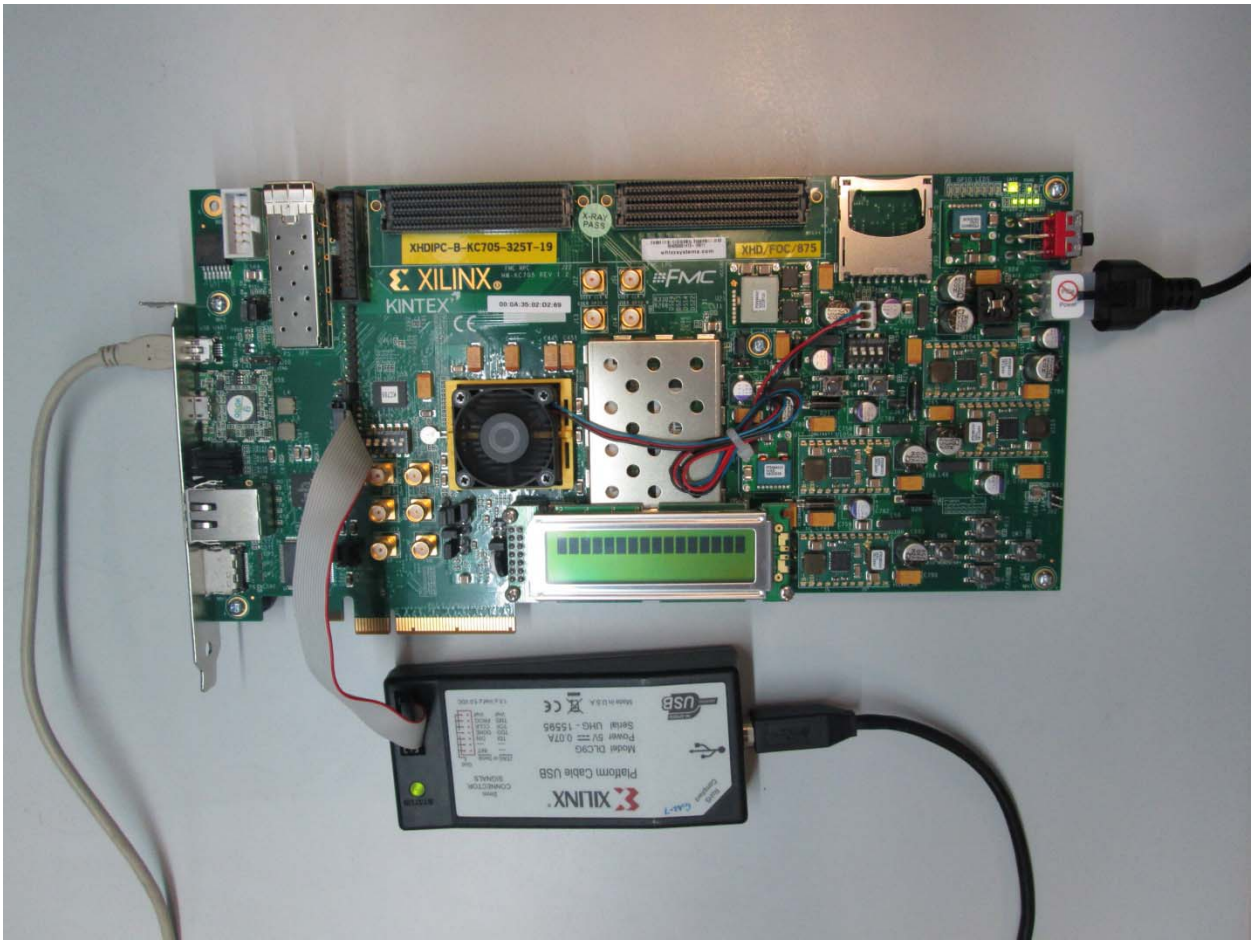

Figure 6: **Hardware Setup**

1. Connect a JTAG USB Platform cable or a USB Type A to Micro B cable from the host PC to the KC705 board for programming BIT and ELF files. (In this setup, a JTAG Platform cable is used.)
2. Connect a Mini USB cable from the host PC to the USB UART port on the KC705 for serial communication.
3. Connect the power supply cable and turn on the KC705 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 use the reference design for both hardware and software.

1. Before beginning, unzip the reference design into a local directory. The references design is referred to as *xapp1218_vdma* 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 XAPP1218 reference design, and test on the hardware. To do so, only follow:
     - Running the Design on the Hardware, and pay particular attention to the **Important Note** in step 2.

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 XAPP1218 reference design, and start at Running the Design on the Hardware.

Creating a Vivado Design Suite Project, and Generating the Bitstream

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

1. Go to the HW directory:

   ```
   cd HW
   ```

2. Launch the Vivado Design Suite.
3. Source `all.tcl` in the Vivado IDE **Tcl Console**. This creates the IP integrator block diagram.

4. Generate the bitstream by clicking **Generate Bitstream** on the left side of the Vivado IDE. This creates the `design_1_wrapper.bit` file at

   `./project_1/project_1.runs/impl_1`

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

Figure 7 shows the address mapping of all the IP cores with the MicroBlaze™ processor in the system.

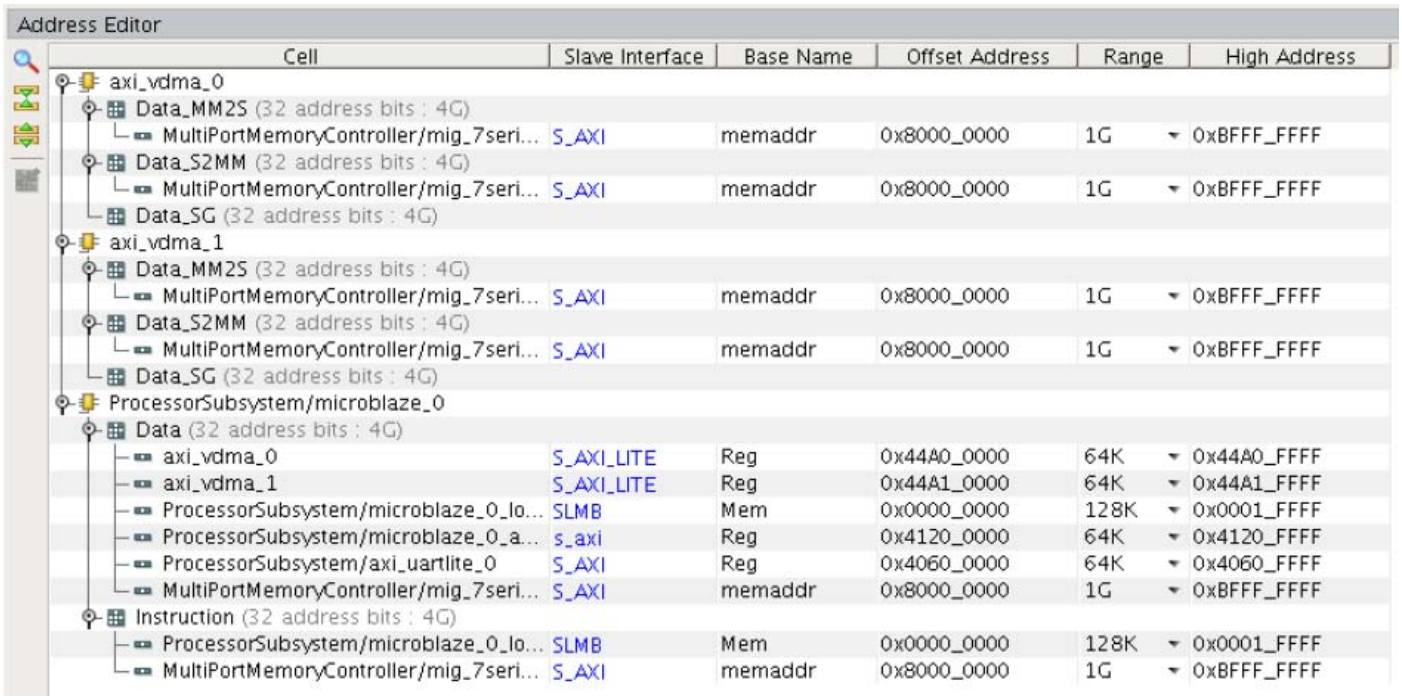

Figure 7: **Base and High Addresses of the IP Cores in IP Integrator**

Exporting the Hardware Workspace in SDK

1. Click **File > Export > Export Hardware for SDK**.
2. Click **File > Launch SDK**.
3. The SDK opens without a created project. If you receive an error or warning from the SDK, click **Yes** or **OK**.
4. In the SDK:
   a. Select **File > New > Board support package** to generate BSP for KC705 board.
   b. Select **File > New > Application project** to create a new blank project. Name the project `vdma_app`, and select **Use existing BSP** for this project.
   c. Click **Next**, select **Empty Application** in the next window, and click **Finish**.
5. Edit the `lscript.ld` file from the `vdma_app/src` folder and increase the stack size from "0x400" to "0x1000" and save it, because the application uses more stack.

6. Right-click the `vdma_app/src` folder and select **Import** to import the vdma source files from the `SW/vdma_app/` folder provided with this package.
7. Double-click **General > File system**, browse through the path to the `SW/vdma_app/` folder and select all source files.

The `vdma_app.elf` file is created in the `project_1/project_1.sdk/vdma_app/Debug` folder.

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 XAPP1218 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 `vdma_app.elf` file from `project_1/project_1.sdk/vdma_app/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` at 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 Hyper Terminal and configure it to 9600 baud with the 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 board.

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

Continuous transfer of buffered frames is taking place and the corresponding callback handlers will handle the interrupts.

Results

The results are shown in Figure 8.

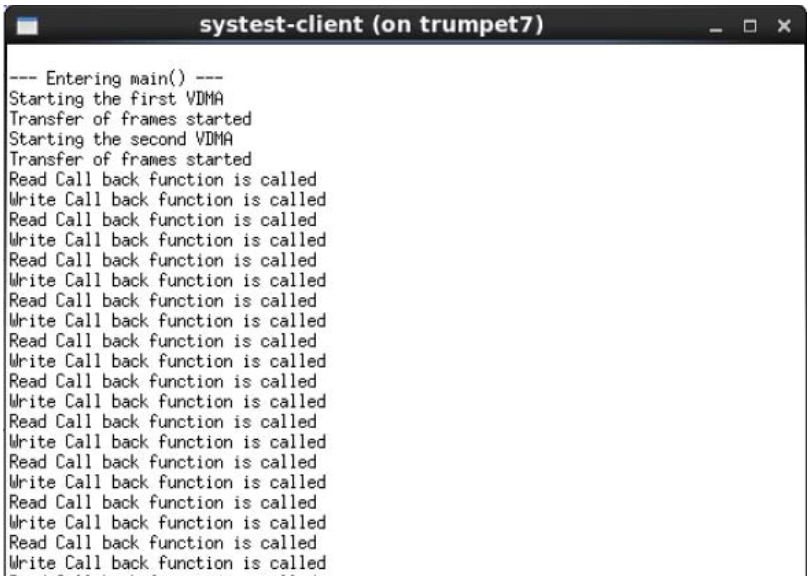

Figure 8: **Application Run Output**

The multiple callbacks indicate that frame counter interrupt occurs for read and write channels. Add code in these callbacks to create a complete system.

References

1. AMBA AXI4 specifications
2. *Kintex-7 FPGA KC705 Evaluation Kit Getting Started Guide* (UG883)
3. *LogiCORE IP AXI Interconnect Product Guide* (PG059)
4. *LogiCORE IP AXI Video Direct Memory Access Product Guide* (PG020)
5. *7 Series FPGAs Memory Interface Solutions* (DS176)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/29/2014	1.0	Initial Xilinx release.
02/18/2015	2.0	Updated for the Vivado Design Suite 2014.4.

Please Read: Important Legal Notices