# Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol

Authors: Steve Elzinga and Chris Martin

XAPP521 (v1.0) February 1, 2012

Summary

The ARM® core AMBA® specification (version 4.0) AXI interconnect standard includes three Advanced eXtensible Interface version 4 (AXI4) interconnect protocols—AXI4 interconnect, AXI4-Lite protocol, and AXI4-Stream interconnect. The Xilinx AXI video direct memory access (AXI VDMA) core is offered with an AXI4-Stream interface for video data. Video applications that require the video data to be buffered in memory need an AXI4-Stream interface to connect with the AXI VDMA. Previous versions of Xilinx video IP cores used a protocol called Xilinx streaming video interface (XSVI) for video data. Refer to Table 2 for a description and list of the XSVI signals. This application note details bridging an XSVI interface to an AXI4-Stream interface, enabling video designs with Xilinx video IP cores and XSVI interfaces to use the AXI VDMA. Figure 1 illustrates the basic structure of the bridging between XSVI and AXI4-Stream.
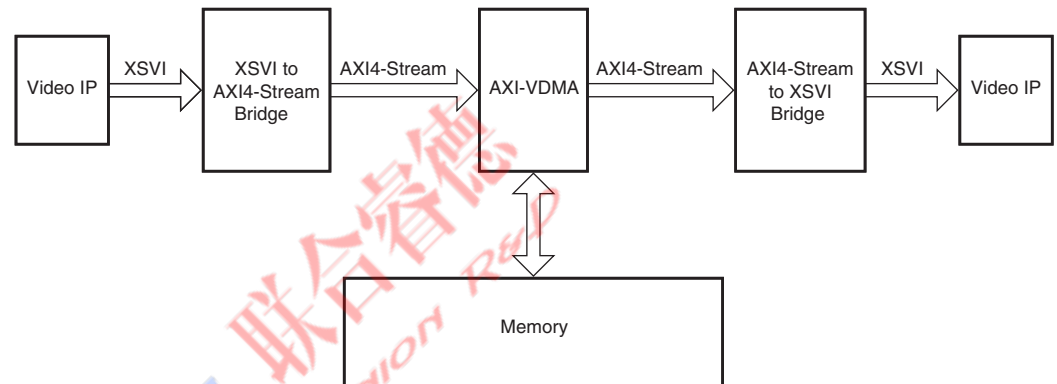
Figure 1: **Basic XSVI to AXI4-Stream Structure**

Design Overview

An embedded reference design, described in Table 1, is included to illustrate the functionality of the bridges described in this application note.

Table 1: **Reference Design Specifics**

Reference Design	Description
Targeted Devices	Spartan®-6 and Virtex®-6 FPGAs
Platform Tested	ML605 board
Modules Used	Avnet DVI I/O FMC card
Input Video	720p60
Output Video	720p60
Software tool used to verify reference design	Xilinx ISE® 13.3 software

XSVI and AXI4-Stream protocols are non-addressable, point-to-point interfaces with minimal overhead, allowing throughput to be the main priority. The XSVI protocol streams images and

video along with a minimal set of control signals from one video peripheral to another. The AXI4-Stream protocol streams non-specific data to and from peripherals.

Many video systems need to buffer the data in external memory. The VDMA core provides video cores with direct access to memory for buffering.

The latest version of the VDMA core (http://www.xilinx.com/support/documentation/ipaudiovideoimageprocess_processing_axi-vdma.htm) comes with AXI4-Lite and AXI4-Stream interfaces. Some Xilinx Video IP cores use an XSVI interface for video data. If the video processing cores with XSVI need to buffer the images or video, then a bridge is required to bridge between XSVI and AXI4-Stream to send video data to the AXI VDMA.

XSVI to AXI4-Stream

To convert from the XSVI Interface to the AXI4-Stream interface requires the XSVI timing control signals to be translated into native AXI4-Stream control signals. A FIFO is used to buffer the data to simplify the translation from the XSVI protocol to the AXI4-Stream protocol. The XSVI interface consists of the signals listed in Table 2.

Table 2: **XSVI Signals**

Signal	Description
active_chroma	Valid chroma samples in the case of chroma sub-sampling
active_video	Data on the video_data signal is valid
field_id	For interlaced video: 0 = Even, 1 = Odd
hblank	Horizontal blanking interval
hsync	Horizontal synchronization
vblank	Vertical blanking interval
video_clk	Pixel clock
video_data	The video stream
vsync	Vertical synchronization

AXI4 and AXI4-Lite protocols are typically used in a processor system, whereas the AXI4-Stream interface is a non-addressable point-to-point connection. Detailed discussion on the AXI4-Stream protocol is beyond the scope of this application note. More information about AXI4 can be found in the *AXI4 Reference Guide* [Ref 1]. The AXI4-Stream signals of interest are shown in Table 3.

Table 3: **AXI4-Stream Signal Subset**

Signal	Description
aresetn	Global reset, active-low input
tdata	Data passed across the interface
tlast	Indicates the last data word of a packet or frame
tready	Slave is ready to accept data
tvalid	Master is ready to transmit data

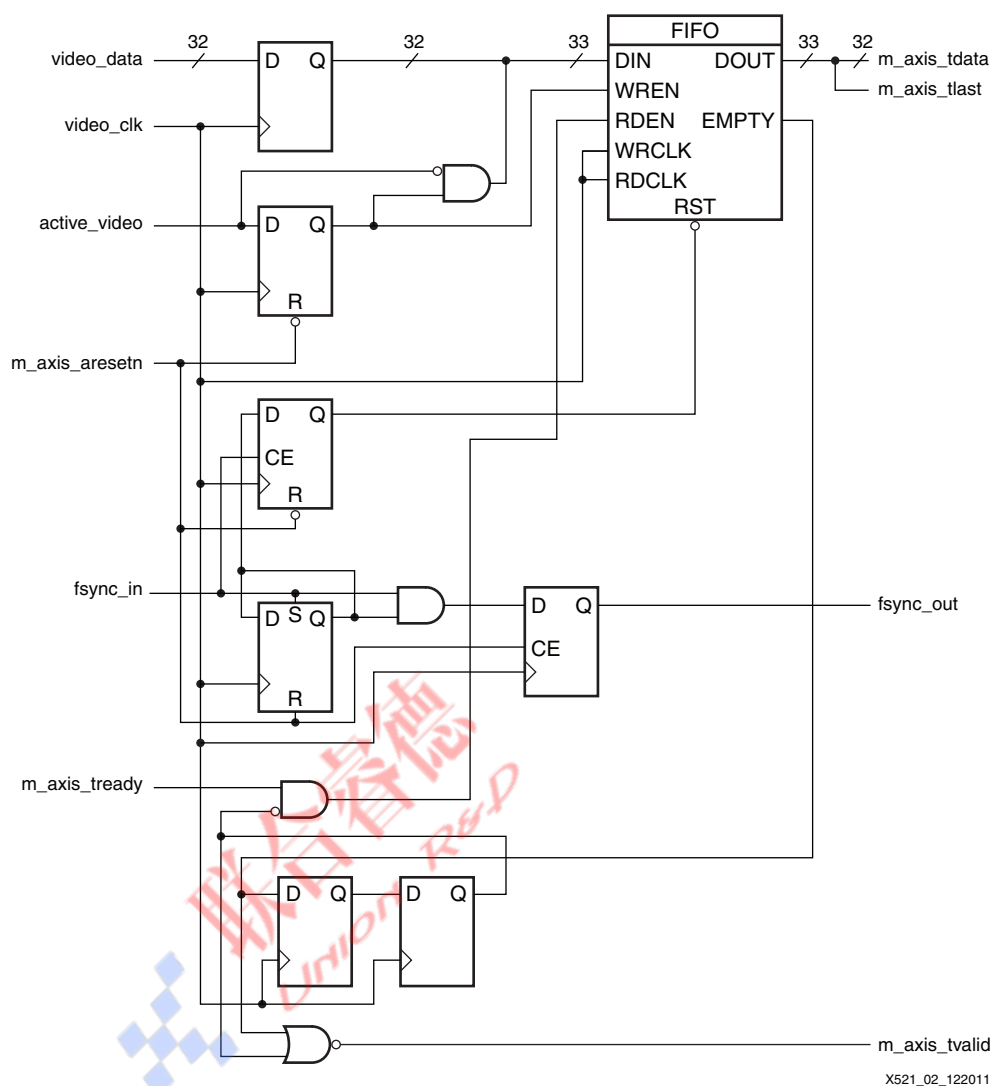The conversion from XSVI to AXI4-Stream is illustrated in Figure 2.

X521_02_122011

Figure 2: **XSVI to AXI4-Stream Bridge**

Because the AXI VDMA core does not have an XSVI interface, the bridge is used. The only XSVI-specific signals used by the AXI VDMA are the frame synchronization and active video signals.

In compliance with the AXI4-Stream specification and to successfully bridge the data between XSVI and AXI4-Stream, the data coming from the bridge (master side of the AXI4-Stream) has to be stable when m_axis_tvalid is asserted. The receiving (slave) peripheral asserts the m_axis_tready signal, telling the bridge to start sending the data that is stored in the FIFO. Figure 3 is a representation of this transaction. Figure 3 shows the m_axis_tdata values are stable before m_axis_tready is asserted, followed by m_axis_tvalid, which is in compliance with the AXI4-Stream specification.
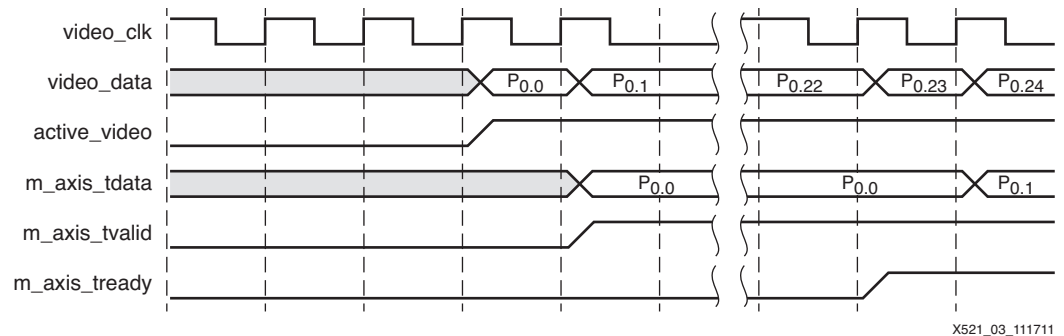
X521_03_111711

Figure 3: **m_axis_tvalid Asserted Before m_axis_tready**

AXI4-Stream to XSVI

The process to convert from an AXI4-Stream to XSVI is similar to the process to convert from XSVI to AXI4-Stream. As shown in Figure 4, a FIFO buffers video data before streaming the data out to be processed. As timing synchronization signals are not passed through the AXI4-Stream interface, the video timing controller core generates the synchronization signals needed for the video data. The axi2xsvi bridge allows both XSVI and AXI4-Stream signals as inputs to accommodate the needed synchronization signals. The video clock and the AXI4-Stream run at the same rate to produce a continuous video stream.
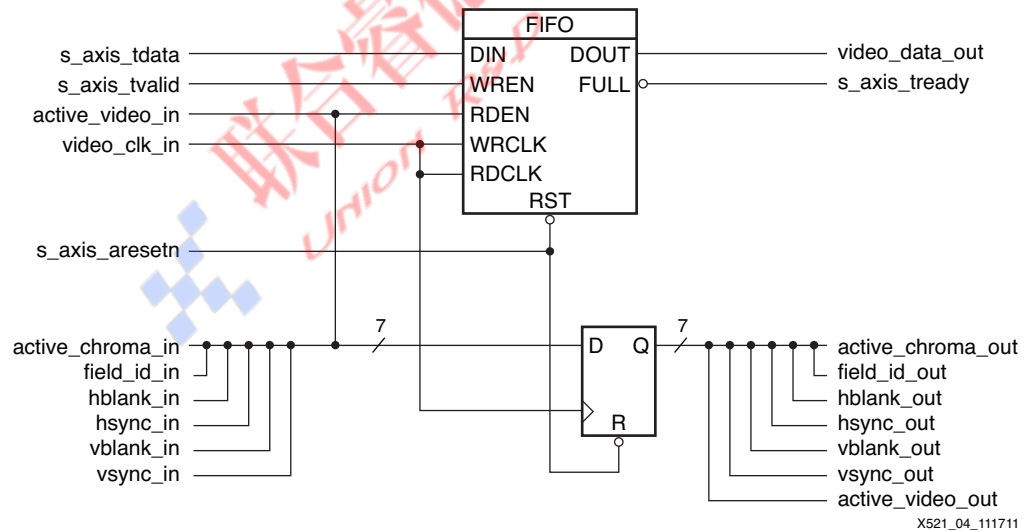
X521_04_111711

Figure 4: **AXI4-Stream to XSVI Bridge**

Configuring the Bridges

The xsvi2axi bridge pcore parameters are listed in Table 4.

Table 4: **Parameters for xsvi2axi Bridge**

Parameter	Values	Description
C_M_AXIS_S2MM_TDATA_WIDTH	8, 16 (default), 24, 32, 40	Data width on the AXI4-Stream side of the bridge
C_MEMORY_TYPE	Block (default), distributed	Either Block RAM or LUT RAM (distributed)
C_VIDEO_DATA	8, 10, 12,16, 24 (default), 30, 36	Data width on the XSVI side of the bridge. Must be ≤C_M_AXIS_S2MM_TDATA_WIDTH
C_WRITE_FIFO_DEPTH	N^2 (1,024 default)	Memory depth of the FIFO

The axi2xsvi pcore parameters are listed in Table 5.

Table 5: **Parameters for axi2xsvi Bridge**

Parameter	Values	Description
C_READ_FIFO_DEPTH	N^2 (default 4)	Memory depth of the FIFO
C_S_AXIS_DATA_WIDTH	8, 16 (default), 24, 32, 40	Data width on the AXI4-Stream side of the bridge
C_VIDEO_DATA	8, 10, 12,16, 24 (default), 30, 36	Data width on the XSVI side of the bridge. Must be ≤C_S_AXIS_DATA_WIDTH

Bridge Device Utilization and Performance

Table 6 details the worst case utilization and performance numbers of the xsvi2axi bridge targeting a data width of 40 in the slowest speed grade parts. For comparison to the results listed in Table 6, Table 7 lists the required pixel clock speeds for various resolutions.

Table 6: **Bridge Performance Numbers**

Device	Memory Type	LUT / FF Pairs	Block RAM	Speed
Spartan-6 FPGA	Distributed	1635	0	127 MHz
Spartan-6 FPGA	Block RAM	110	2 RAMB16 1 RAMB8	237 MHz
Virtex-6 FPGA	Distributed	1635	0	180 MHz
Virtex-6 FPGA	Block RAM	110	1 RAMB36 1 RAMB18	233 MHz

Table 7: **Resolutions and Pixel Clock**

Resolution	Pixel Clock
720p60	74.25 MHz
1080p/60	148.5 MHz
Max DVI Specification	165 MHz

Reference Design

The reference design is delivered as a MicroBlaze™ embedded processor design running on an ML605 board, which highlights some basic features of the Scaler core. The bridges are delivered as embedded pcores and reside in the EDK project directory's `pcore` folder. The video system targets a 720p60 DVI connection, and requires a 74.25 MHz clock rate. The processor system is AXI-based running at 200 MHz.

The reference design files can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=181114

Table 8 lists the memory addresses of the peripherals used in the embedded system.

Table 8: **Memory Map of Embedded System**

Instance	Peripheral	Base Address	High Address
microblaze_0_d_bram_ctrl	lmb_v10	0x00000000	0x0000FFFF
microblaze_0_i_bram_ctrl	lmb_v10	0x00000000	0x0000FFFF
RS232_Uart_1	axi_uartlite	0x40600000	0x4060FFFF
axi_iic_1	axi_iic	0x40800000	0x4080FFFF
axi_iic_0	axi_iic	0x40820000	0x4082FFFF

Table 8: **Memory Map of Embedded System** *(Cont'd)*

Instance	Peripheral	Base Address	High Address
microblaze_0_intc	axi_intc	`0x41200000`	`0x4120FFFF`
debug_module	mdm	`0x74800000`	`0x7480FFFF`
axi_scaler_0	axi_scaler	`0x7C800000`	`0x7C80FFFF`
axi_vdma_1	axi_vdma	`0x7E200000`	`0x7E20FFFF`
axi_vdma_0	axi_vdma	`0x7E220000`	`0x7E22FFFF`
axi_vtc_1	axi_vtc	`0x7EE00000`	`0x7EE0FFFF`
axi_vtc_0	axi_vtc	`0x7EE20000`	`0x7EE2FFFF`
DDR3_SDRAM	axi_v6_ddrx	`0xC0000000`	`0xCFFFFFFF`

The video cores that are not addressable by the processor are listed in Table 9.

The cores listed in Table 9 are non-addressable cores, and they are also custom cores. The two FMC cores are made by Avnet to interface with the FMC card. The xsvi2axi and axi2xsvi bridges are the bridges highlighted in this application note.

All of the custom cores along with the Scaler core are delivered as local pcores residing in the EDK project `pcore` directory. The Scaler core was generated through the CORE Generator™ GUI as a pcore with a hardware evaluation license.

Table 9: **Non-addressable Cores in Embedded System**

Instance	Peripheral
csc_rgb_to_ycrcb422_0	csc_rgb_to_ycrcb422
csc_ycrcb422_to_rgb_0	csc_ycrcb422_to_rgb
fmc_dvidp_dvi_out_0	fmc_dvidp_dvi_out
fmc_dvidp_dvi_in_0	fmc_dvidp_dvi_in
xsvi2axi_0	xsvi2axi
axi2xsvi_0	axi2xsvi

Figure 5 shows only the connection between the peripherals and the processor (control path) which allows the processor to control the video IP via software. The software control can change settings in the video IP cores while the video stream is being processed.
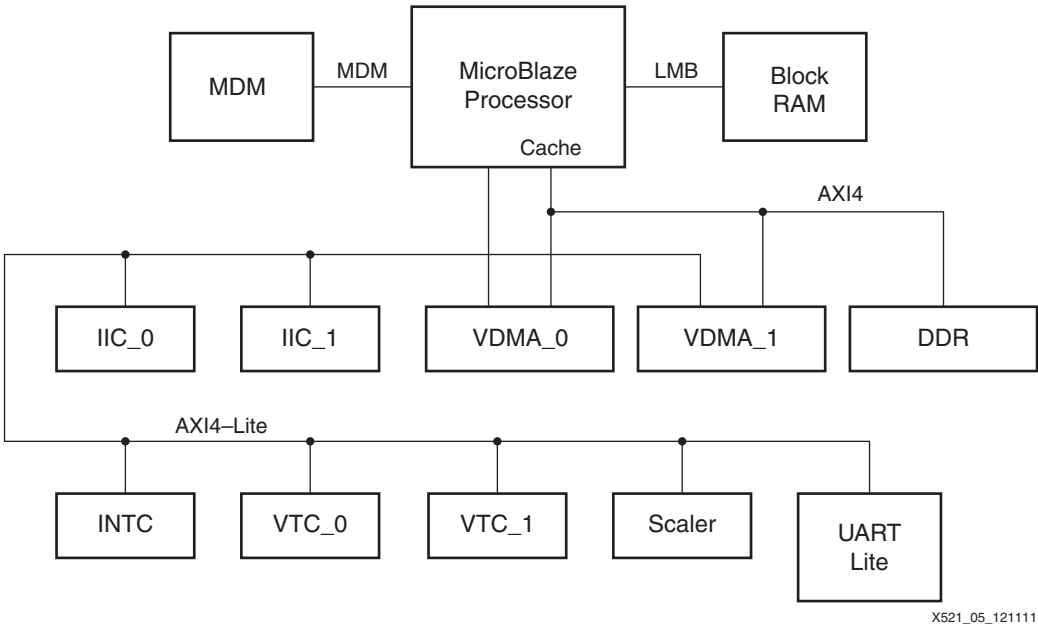
Figure 5: **Sample Video AXI4 Based Embedded Design (Control Path)**

Figure 6 shows the same embedded design, with the streaming interface (data path) connections. Table 10 describes the labels in Figure 6. Some of the peripherals only process video data on the streaming interface and have no connection to the processor. The video stream is completely independent of the processor and the only interaction between the processor and the video cores is through the processor bus.
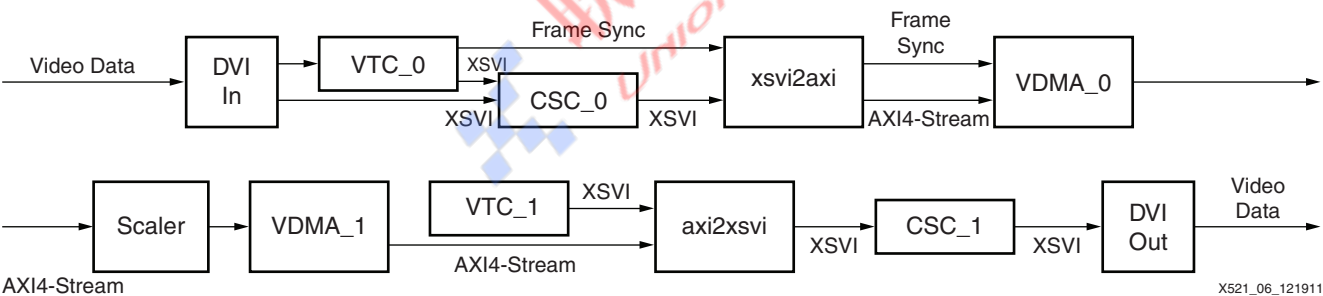

Figure 6: **Embedded Video Design Streaming Connection (Data Path)**

Table 10: **Label Descriptions in Figure 6**

Label	Description
axi2xsvi	AXI4-Stream to XSVI bridge
CSC	Color space converter: Instance 0 converts from RGB to YCrCb 422. Instance 1 reverses the conversion.
DVI Out	Digital visual interface output: Processed digital video streaming to a monitor.
VTC	Video timing controller: Video processing core that can detect and generate timing signals for use by the rest of the cores
xsvi2axi	XSVI to AXI4-Stream bridge

Running the Reference Design

To run the reference design:

1. Plug the Avnet FMC card in to the ML605 board as shown Figure 7.
2. Follow the *ML605 Hardware Setup Guide* [Ref 2].
3. Connect a 720p60 video source to the FMC DVI input connection.
4. Connect a DVI monitor to the video output of the FMC DVI card.
5. Open a terminal to communicate with the ML605 board.
6. Unzip the reference design.
7. Open an `ISE Design Suite` Command Prompt.
8. Change directories into the `ready_for_download` directory in the unzipped reference design project directory.
9. Launch XMD at the command prompt.
10. Run these XMD commands:

```
XMD% fpga -f download.bit
XMD% connect mb mdm
XMD% dow demo.elf
XMD% run
```
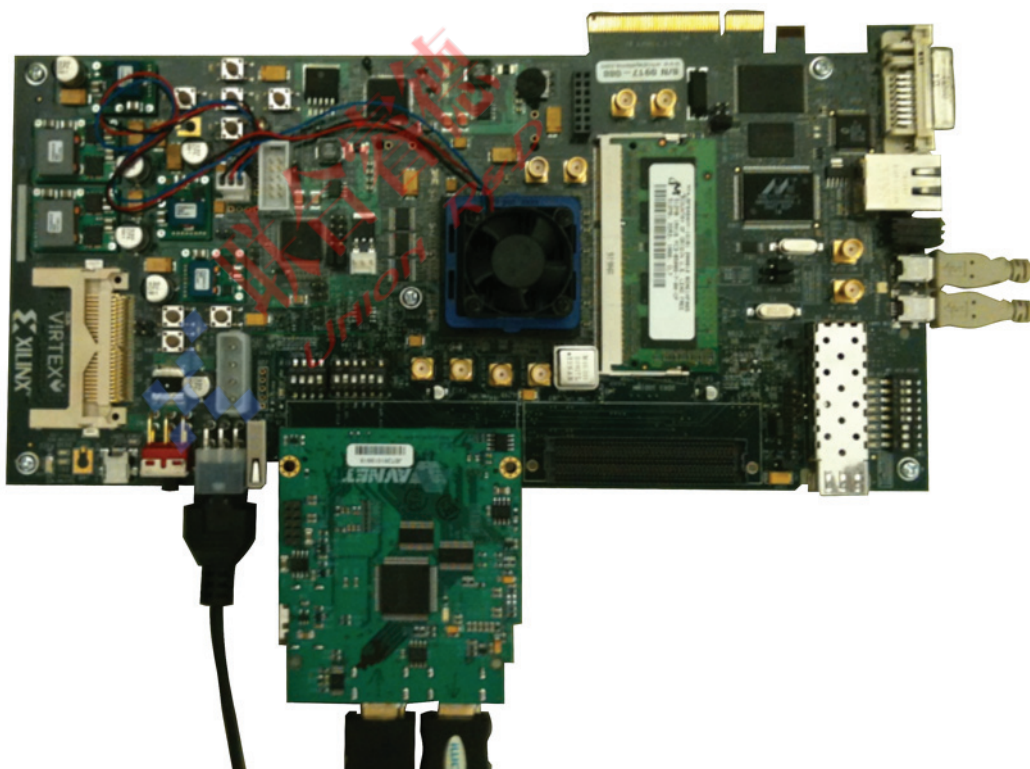

X521_07_111711

Figure 7: **ML605 Board with Avnet FMC Card**

The following information runs and outputs to a terminal.

```
FMC Module Validation
Board Information:
        Manufacturer    = Avnet
        Product Name    = FMC-DVI/DP
        Serial Number   = JBT261019918
        Part Number     = AES-FMCDVI-G
SUCCESS : Detected FMC-DVI/DP module!
Reset and Initialize the FMC devices ...
Detect TFP403 ...
Detected DVI input ...
Waiting for detector lock......Done.
HTotal = 1649
HActiveStart = 260
HFrontPorchStart 1540
HBackPorchStart 40
HSyncStart 0
V0FrontPorchStart 745
V0Total 749
V0BackPorchStart 5
V0SyncStart 0
V0ChromaStart 0
V0ActiveStart 25
Detected Video Dimensions = 1280 x 720
Detected Video Resolution = 720P
resolution = 4
frame_width = 1280;
frame_height = 720.
frame_width_out = 1280;, frame_height_out = 720.
osd_stride = 1280
osd_frame_width = 1280
osd_frame_height = 720
```

Hitting the space bar on the keyboard cycles through the various scaling features listed.

```
-------------------------------------------------------------
--  Video Scaler Test Menu using Scaler/VDMA/TimeBase drivers   --
-------------------------------------------------------------

  Select scaling option

   Use Spacebar to rotate between the following conversions:
       From 1280Hx720V to 1280Hx720V
       From 1066Hx720V to 1280Hx720V
       From 1064Hx600V to 1280Hx720V
       From 852Hx540V to 1280Hx720V
       From 512Hx288V to 1280Hx720V
       From 256Hx144V to 1280Hx720V
 . = Rotate between coefficient sets (forwards).
 , = Rotate between coefficient sets (backwards).
 / = Reload all 16 sets of coefficients.
 r = Register printout
 R = Reset scaler

 ?  = help
-------------------------------------------------------------
scaler_out_width = 1280
scaler_in_width = 1280
scaler_out_height = 720
scaler_in_height = 720
>
```

Rebuilding the Design

To rebuild the hardware portion of the reference design:

1. Unzip the reference design into a working directory.

2. Launch XPS and select the EDK project contained in the zip file.

3. In XPS, select **Device Configuration > Update Bitstream**.

The third option causes XPS to go through the entire implementation process to provide a downloadable bitstream.

After the entire implementation process is completed, the software portion of the reference design can be rebuilt using this procedure:

1. Select **Project > Export Hardware Design to SDK**.

2. Select **Export & Launch SDK**.

3. Add the repository necessary for this application note:
   - In SDK, select **Xilinx Tools > Repository**.
   - Expand **Xilinx SDK**.
   - Highlight **Repositories**.
   - In the Local Repositories window, select the **New** button.
   - Browse to **<project_directory>\Repository** and select **Okay**.

4. Create a new Xilinx C Project and Board Support Package:
   - Select **File > New > Xilinx C Project**.
   - Select **Empty Application**.
   - Keep all defaults and finish the new project creation wizard.

5. Add the custom software services and verify correct drivers:
   - **Xilinx Tools > Board Support Package Settings.**
   - Select **empty_application_bsp_0** if asked.
   - Highlight **Overview** on the left side of the window.
   - Select **fmc_dvip_sw, fmc_iic_sw, fmc_ipmi_sw**.
   - Highlight drivers on the left side of the window.
   - Verify that the axi_vtc and the axi_scaler have their drivers.

6. Add the software application to the project:
   - In SDK, expand the **empty_application_0 project**.
   - Right-click on the `src` directory.
   - Select **Import**.
   - Expand **General**.
   - Select **Archive File** and press **Next**.
   - Browse into the EDK project directory and select the `Demo.zip` file.
   - Press the **Finish** button.

With the project now newly rebuilt, download to a properly connected ML605 board via the SDK Xilinx Tools menu, and launch the software onto the board from SDK:

1. Make sure board is powered on

2. Select **Xilinx Tools > Program FPGA**.

3. Press the **Program** button.

After the FPGA is configured:

1. Open a terminal with the proper COM port selected for the system's USB port.
2. Highlight the **empty_application_0** software project.
3. Press the green **Play** button in the SDK tool bar.

The software output should be onscreen.

References

This document uses the following references:

1. UG761, *AXI4 Reference Guide*
2. XTP084, *ML605 Hardware Setup Guide*
3. *ML605 Documentation* http://www.xilinx.com/support/#nav=sd-nav-link-140997&tab=tab-bk
4. UG683, *EDK Concepts, Tools, and Techniques*
5. *AMBA 4 AXI4-Stream Protocol Specification* http://www.arm.com

Conclusion

The xsvi2axi and axi2xsvi bridges effectively translate between XSVI and AXI4-Stream and allow video designs using XSVI to use the AXI VDMA for memory access. This solution is used to build video systems when the Xilinx video IP does not contain an AXI4-Stream interface.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
02/01/2012	1.0	Initial Xilinx release.

Notice of Disclaimer