XAPP1251 (v1.0) April 30, 2015

Xilinx Virtual Cable Running on Zynq-7000 Using the PetaLinux Tools

Authors: Alvin Clark (Avnet Inc.) and Luis Bielich

Summary

It is common for the architecture of a system to have a single interface for communication and debug into the system. Often there is a processor with an Ethernet connection to communicate with the rest of the system. Ethernet is generally chosen because it provides a common interface that can be accessed from a network. Although the processor can communicate with Ethernet, there is not a debug channel for the JTAG signals targeting an FPGA or SoC. To access the JTAG signals of an FPGA or SoC, it normally requires a separate JTAG connector. Eliminating the need for a dedicated JTAG connector satisfies the need of a system architect to require only one interface for communication and debug while maintaining accessibility over a network.

The Vivado® design tools can communicate the same JTAG commands over a TCP/IP connection to a microprocessor implementing the XVC v1.0 Protocol [Ref 1]. This implementation supports all of the existing Vivado hardware debug features.

This application note shows how to get Xilinx Virtual Cable (XVC) running on a Zynq®-7000 device with a Linux operating system generated with the PetaLinux Tools. A reference design is provided for the Avnet MicroZed board. The target FPGA in this application note is on an AC701 board and will be programmed and debugged by the MicroZed board running XVC on Linux.

Reference Design

Starting with Vivado design tools 2014.3.1, the Vivado hardware debug tools include support for XVC. The XVC protocol allows the Vivado design tools to communicate JTAG commands over Ethernet to an embedded system so that a target Xilinx FPGA can be programmed and/or debugged. This enables a vendor agnostic solution for debugging and programming a Xilinx FPGA or SoC. Programming capabilities include the same support as a traditional JTAG connection provides. Debugging capabilities include operability with System Debugger (XSDB) or with Vivado design tools hardware debug IP cores.

Although XVC can be implemented with several different processors, this application note provides a reference design utilizing a Zynq-7000 device. The Zynq-7000 family provides a compelling solution for XVC by accelerating the JTAG commands in the Programmable Logic (PL). This improves system performance by off-loading the processor from bit-banging JTAG commands.

The Zynq-7000 device embedded Cortex™ A9 processor parses the Ethernet packets with a TCP/IP connection and converts the packets into JTAG commands.

The Zynq-7000 device ultimately acts as a bridge from Ethernet back to JTAG with a TCP/IP server running on the embedded processor. Figure 1 illustrates the high-level block diagram of a typical XVC topology.

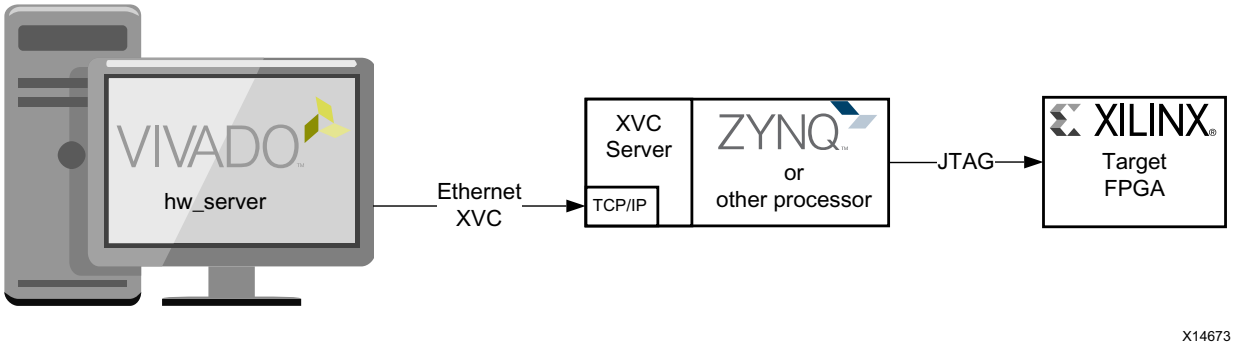

Figure 1: **High Level Block Diagram of XVC Topology**

Running Linux on Zynq-7000 devices enables a simple implementation of the XVC server because the application layer can leverage the TCP/IP stack included in the Linux kernel. The server application implements the XVC protocol and runs the application in the background as a daemon. After the packets are processed, the Zynq-7000 device processing system (PS) communicates with an AXI4-Lite to JTAG Controller in the PL to create the JTAG commands to the target FPGA for programming or debug. The AXI4-Lite to JTAG Controller is packaged using the Vivado design tools as a custom IP so that it can be natively used within IP integrator and so that it can be easily leveraged in a custom design.

The JTAG commands to the FPGA are the same commands that would have been transferred to the FPGA if it were natively communicating with a programming cable or using a Digilent module. This ensures functionality between all the existing Vivado hardware debug design tools.

Hardware

The Processing System and Programmable Logic of Zynq-7000 devices are both utilized in the reference design. The Programmable Logic utilizes a custom AXI4-Lite to JTAG Controller to help balance the load on the processor. All programmable logic components are provided as part of the reference design files. The Processing System utilizes the Ethernet MAC and majority of memory elements of the Processing System to help run Linux. Figure 2 shows a block diagram of the relevant hardware components used in the application note.

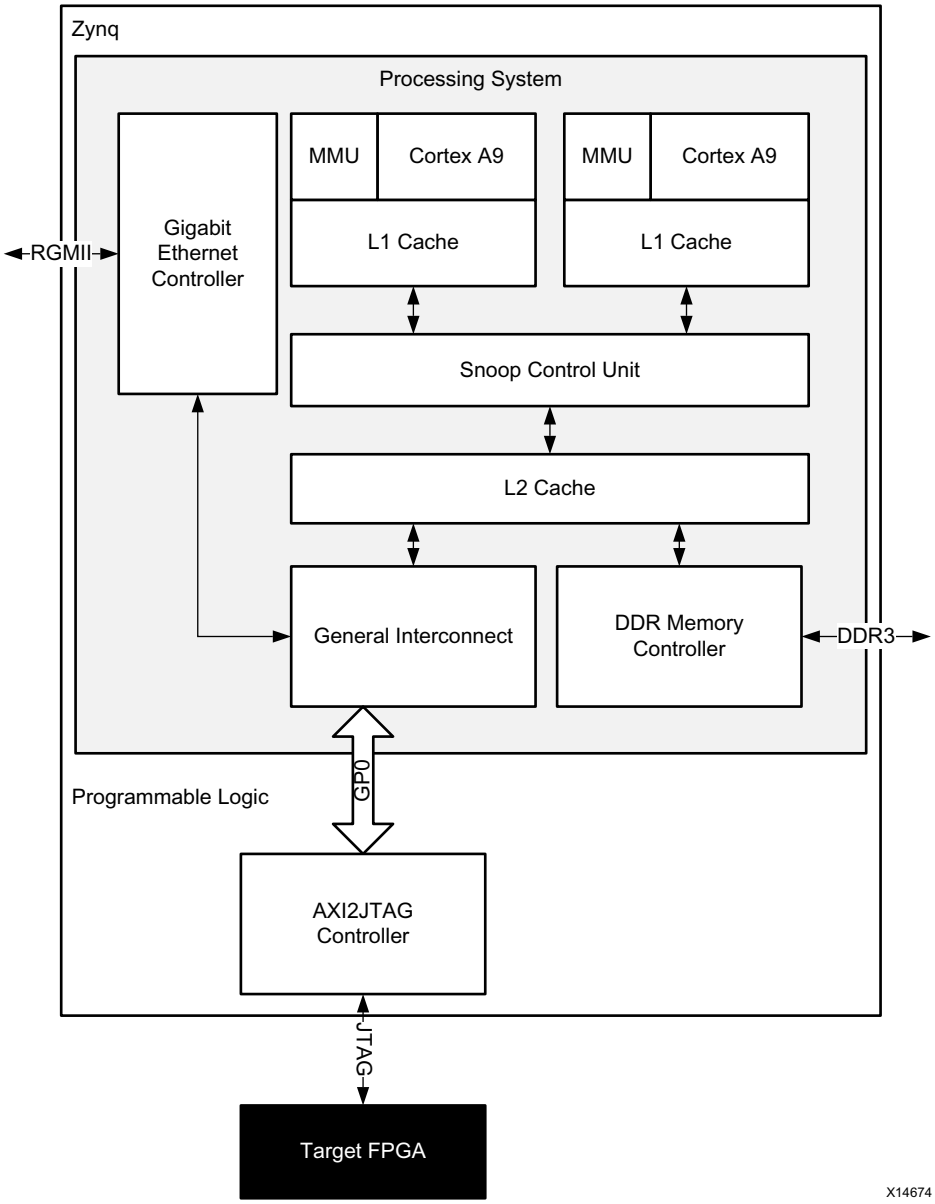

Figure 2: **Relevant Hardware Components**

Gigabit Ethernet Controller

The reference design provided as part of this application note uses the Gigabit Ethernet Controller within the Zynq-7000 device Processing System. It is also possible to use the soft TriMode Ethernet Media Access Controller IP core in the Programmable Logic (PL). The Gigabit Ethernet Controller is configured to work with Marvell 88E1512 PHY using an RGMII interface and the MDIO interface. For more details on this configuration, refer to the *MicroZed Hardware User Guide* [Ref 2].

Cortex A9 Processor APU

The ARM® Cortex A9 processor is running Linux and has a daemon TCP/IP Server Application running that accepts the Ethernet packets and converts the commands to JTAG commands using the *XVC v1.0 protocol* [Ref 1]. The processor is running at 666.67 MHz with the default MMU and caching setup from the Vivado tools. To analyze the Processing System, Xilinx recommends that you open the IP integrator design within the Vivado design tools and to look within the Processing System IP options selected.

AXI4-Lite to JTAG Controller

A custom AXI4-Lite to JTAG Controller is provided with the application note design files. The controller is packaged as a custom IP for the Vivado Integrated Design Environment (IDE) so that it can be used within the IP integrator as shown in Figure 3. The custom controller is designed to shift in up to 32-bit JTAG vectors to interact with the target FPGA. The controller is implemented in the PL to offload the processor by avoiding the need to bit-bang the JTAG signals.

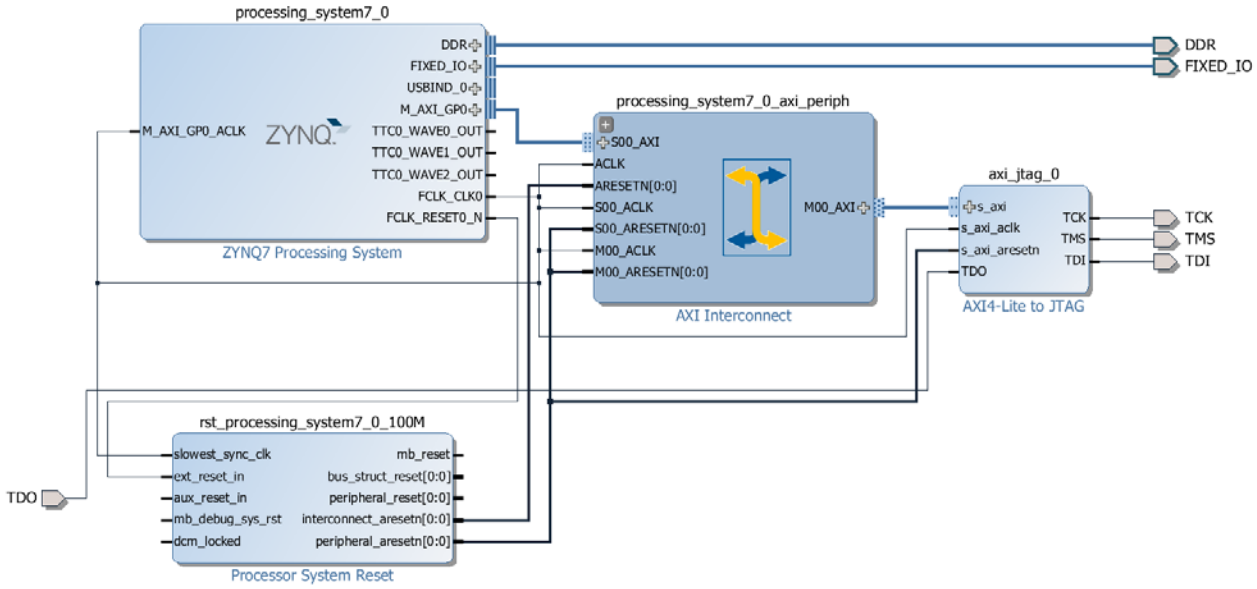

Figure 3: **IP Integrator Design**

The AXI4-Lite to JTAG Controller is an AXI4-Lite peripheral with a register set as shown in Table 1.

Table 1: **Register Set for AXI4-Lite to JTAG Controller**

Address Offset	Name	Access Type	Default Value	Description
0x00	LENGTH	R/W	0x00000000	Length of shift operation in bits
0x04	TMS_VECTOR	R/W	0x00000000	Test Mode Select (TMS) Bit Vector
0x08	TDI_VECTOR	R/W	0x00000000	Test Data In (TDI) Bit Vector
0x0C	TDO_VECTOR	R/W	0x00000000	Test Data Out (TDO) Capture Vector
0x10	CTRL	R/W	0x00000000	Bit 0: Enable Shift Operation

The operation of this module has a simple programming model. The ARM processor writes to the LENGTH, TMS_VECTOR and TDI_VECTOR registers. The order of writing to the LENGTH, TMS_VECTOR, and TDI_VECTOR registers does not matter. Afterward, the Enable Shift Operation bit of the CTRL register must be set to start the transfer. The Enable Shift Operation register is self-clearing, so the software can poll this register to find out when the operation has completed. The TDO_VECTOR register captures TDO bits during a shift operation. The programming sequence for these registers is controlled within the Linux application.

The AXI4-Lite to JTAG Controller contains one user-configurable option to scale down the toggle rate of the JTAG signals. The **TCK Clock Ratio** option allows you to slow down the rate of the JTAG signals to a ratio of the `s_axi_aclk` input clock signal. As an example, in the reference design the `s_axi` clock signals are running at 100 MHz with a **TCK Clock Ratio** option of 16. This yields a toggle rate of 6.25 MHz for Test Clock (TCK.)

IMPORTANT: *You are responsible for calculating the valid rate that TCK should operate at. This depends on the signal integrity of the JTAG signals. Xilinx recommends to start off at a slow toggle rate to verify functionality. Figure 4 shows the **TCK Clock Ratio** option of the custom packaged IP.*

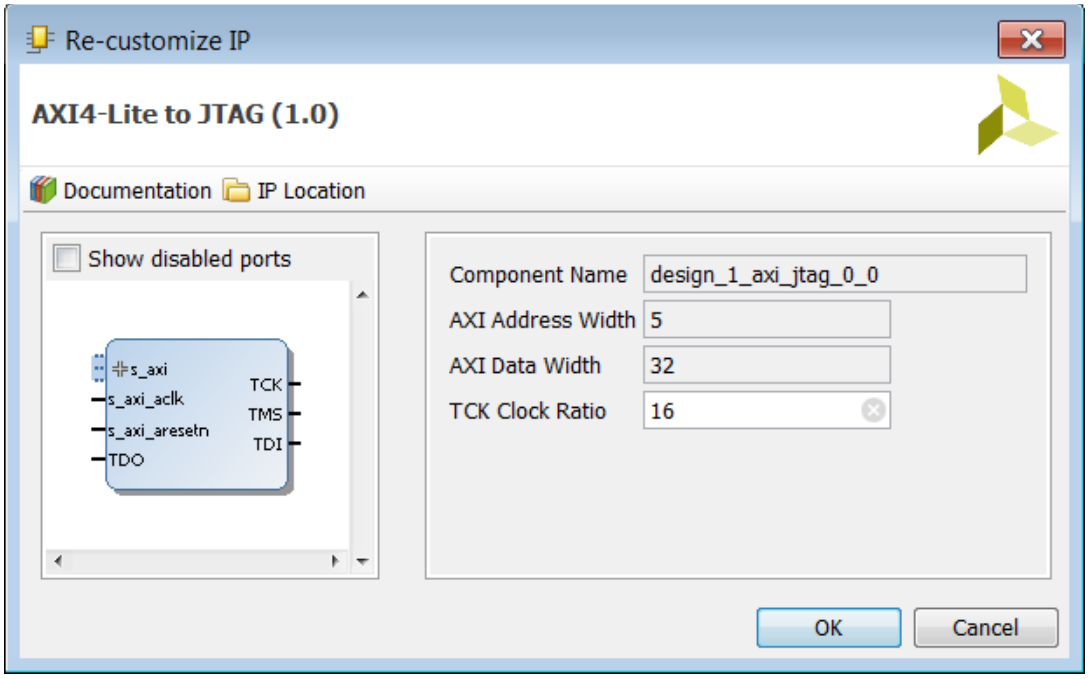

Figure 4: **AXI4-Lite to JTAG Configurable Options**

To use the AXI4-Lite to JTAG Controller in a custom design, add the `hw/source/ip_repo/` directory to the Vivado design tools IP repository. Then the controller will appear as a custom IP in the Vivado IP catalog as shown in Figure 5.

For details on adding a directory as an IP repository, refer to the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

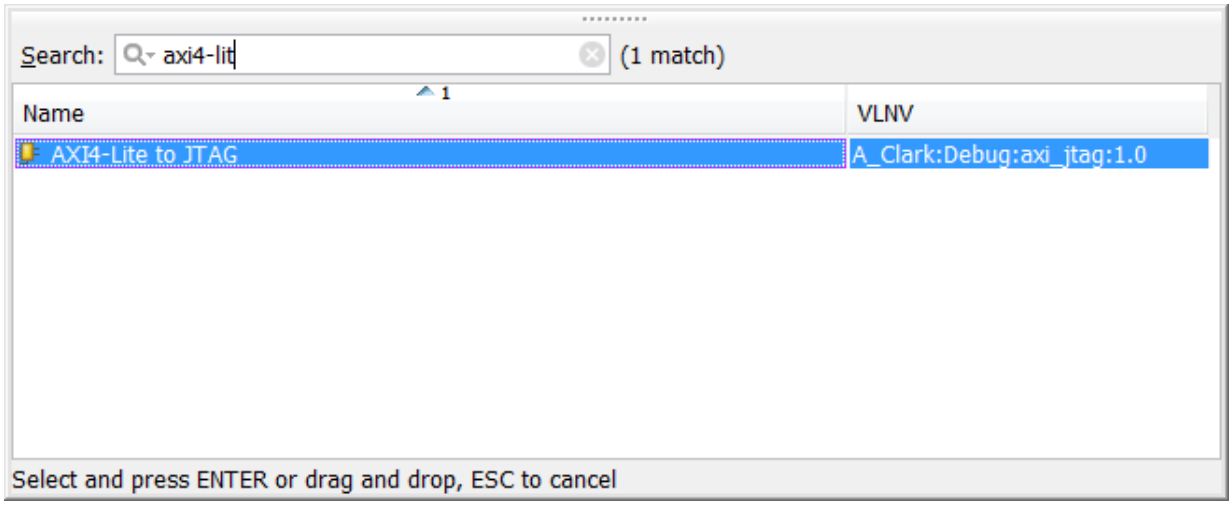

Figure 5: **AXI4-Lite to JTAG within the IP Catalog**

I/O Locations

The JTAG signals on the MicroZed I/O carrier card are pinned out according to Table 2. Also refer to Figure 11 for a physical representation of the connections.

Table 2: **JTAG Signal Pinouts**

Signal Name	Pin on I/O Carrier	I/O Standard	I/O Pin on FPGA
TCK	Pmod JA, Pin 1	LVCMOS33 (PULLUP)	T11
TDI	Pmod JA, Pin 2	LVCMOS33 (PULLUP)	T10
TMS	Pmod JA, Pin 3	LVCMOS33 (PULLUP)	T12
TDO	Pmod JA, Pin 4	LVCMOS33	U12
GND	Pmod JA, Pin 5	N/A	N/A

Software Application

The XVC server is implemented within a Linux application. The Linux application is provided in the `sw/source/app` directory of the reference design. The application software handles the following setup of the TCP/IP socket as well as communication with the AXI4-Lite to JTAG Controller.

The Ethernet payload from the socket is accepted, parsed, and stored in buffers. The first bytes out of the socket indicate the commands. The XVC v1.0 Protocol [Ref 1] only has three commands. The following three XVC v1.0 commands are:

- getinfo:
- shift:[number of bits][TMS vector][TDI vector]
- settck:[period in nanoseconds]

The `getinfo` command retrieves the XVC service version and the maximum vector length of the TMS and TDI bits to be shifted. The current XVC service version is v1.0. The maximum vector length of the TMS and TDI depends on the buffer depths specified in the software application. This application note sets the maximum length to 2,048. So the string returned for the `getinfo` request will be `xvcServer_v1.0:2048\n`.

RECOMMENDED: *Xilinx recommends to tune the buffer size for a custom implementation. Depending on the networking bandwidth and latency, a different buffer length will improve the overall bandwidth of the XVC connection.*

The `shift` command is the primary command received for transmitting data. The `shift` command has three arguments as shown in the packet format in Table 3. The first argument indicates the 'number of bits' in the payload for the TMS and TDI vectors. The 'number of bits' is designated by the following 4 bytes after the `shift` command out of the socket. The maximum value for the 'number of bits' is determined from the response of the `getinfo` command. Following the 'number of bits' is the TMS vector. The length of the TMS vector depends on the 'number of bits' field. Following the TMS vector is the TDI vector which also has a length dependent on the 'number of bits' field. Both the TMS and TDI Vectors are stored in a buffer before being handed off to the AXI4-Lite to JTAG Controller.

Table 3: **Shift Command Format**

Command	Number of Bits	TMS Vector	TDI Vector
“shift:” 6 bytes	4 Bytes	Dependent on ‘Number of Bits’	Dependent on ‘Number of Bits’

The `settck` command attempts to set the `tck` period. The returned value is the actual specified period. The period is a four-byte binary value in little-endian. For example, 100 decimal is 0x64 hex, which is 'D' in ASCII. Therefore 'D' will be the value for a 100 nanosecond period for `tck` for either the request or the response. In the reference design, a constant value of "D" is returned to indicate a 100 ns period for TCK. Table 4 shows the packet format for a `settck` command.

Table 4: **Set TCK Command Format**

Command	Period in Nanoseconds
"settck:" 7 Bytes	"000D" 4 Bytes

After the Ethernet packets are parsed according to the XVC v1.0 protocol [Ref 1], the data is passed along to the AXI4-Lite to JTAG controller in the PL through the uio driver frameworks. The controller works in 4-byte boundaries, which depends on what was parsed from the 'number of bits' field. As each 4-byte boundary of TMS and TDI vectors are transmitted, the TDO vectors are also sampled and stored in another buffer. The software loops until all the bits are flushed through the AXI4-Lite to JTAG controller and all the TDO vectors are stored. When the 'number of bits' field does not align with a 32-bit boundary, the final bits are transmitted and the remaining bits are indicated on the LENGTH register of the AXI4-Lite to JTAG Controller. For example, if the 'number of bits' field requests 73 bits, this is stored in 10 bytes of the buffer. The first two 32-bit boundaries will have a full LENGTH of 32, and the last 32-bit boundary will only have a LENGTH of 9. This is implemented in the example application within the `handle_data` function.

After all the TDO vectors are stored, they are transmitted back to the XVC client and the software listens for the next packet.

Tool Flow and Verification

The following checklist indicates the tool flow and verification procedures used for the provided reference design.

Table 5: **Reference Design Checklist**

Parameter	Description
General	
Developer Name	Alvin Clark and Luis Bielich
Target Devices	MicroZed
Source code provided?	Y
Source code format (if provided)	RTL, Tcl, C
Implementation	
Synthesis software tools/versions used	Vivado synthesis
Implementation software tool(s) and version	Vivado Implementation
Static timing analysis performed?	Y
Hardware Verification	
Hardware verified?	Y
Platform used for verification	MicroZed Board and AC701 (target)

Requirements

This section details any requirements for running the reference design.

Hardware

This application note requires the following hardware to implement the reference design:

- MicroZed Board
- MicroZed I/O Carrier Card
- Ethernet Cable
- Micro USB to USB cable
- Computer with Vivado design tools 2014.4 or newer installed
- IEEE 1149.1 JTAG-Compatible Device (AC701 is used)
- JTAG Flying Wire Adapter (HW-USB-FLYLEADS-G)

Software

- Linux Operating System
- PetaLinux tools
- Vivado Design Tools 2014.3.1 (also validated in 2015.1)
- Software Development Kit (SDK)

Reference Design Files

Figure 6 shows the file directory structure.

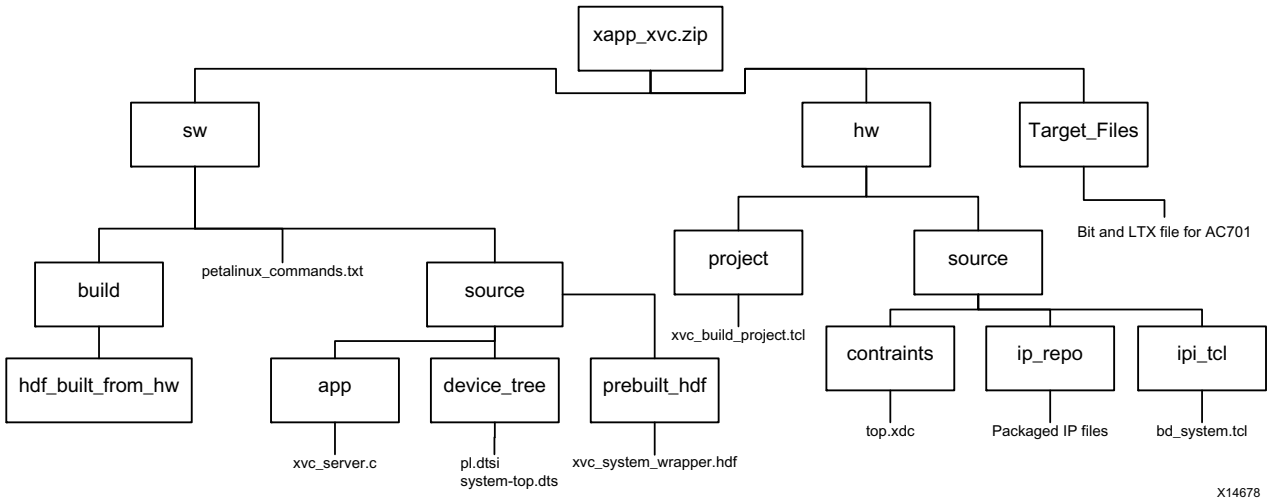

Figure 6: **File Directory Structure**

Licensing

A FLEX license issued by Xilinx is required to run Vivado Design Edition and PetaLinux.

Reference Design Steps

Click here to download the reference design files implemented on the Avnet MicroZed board. The MicroZed I/O carrier card is also needed.

Vivado IDE Steps

1. After downloading the design and unzipping the download, open Vivado IDE and change the working directory to the `hw/project` directory within the Tcl Console. Then source the `xvc_build_project.tcl` within the Tcl Console to build the project, as shown in Figure 7.

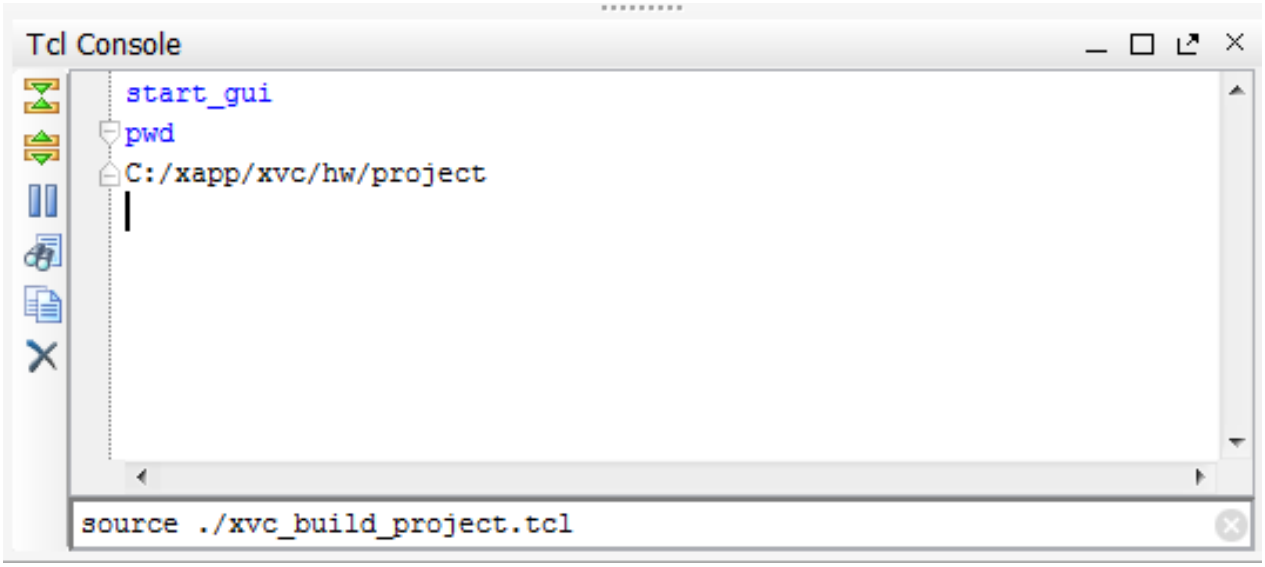

Figure 7: **Sourcing a Tcl Script to Build the Vivado Project**

2. After sourcing the `xvc_build_project` Tcl script, the project is built and you can examine the project sources. Then build the bitstream for the PL by clicking on the **Generate Bitstream** button within the Flow Navigator.
3. After the bitstream is generated, export the hardware design by selecting **File >Export >Export Hardware**. A popup dialog box asks where to export the design. Select to include the bitstream and export the files locally to the project, as shown in Figure 8.

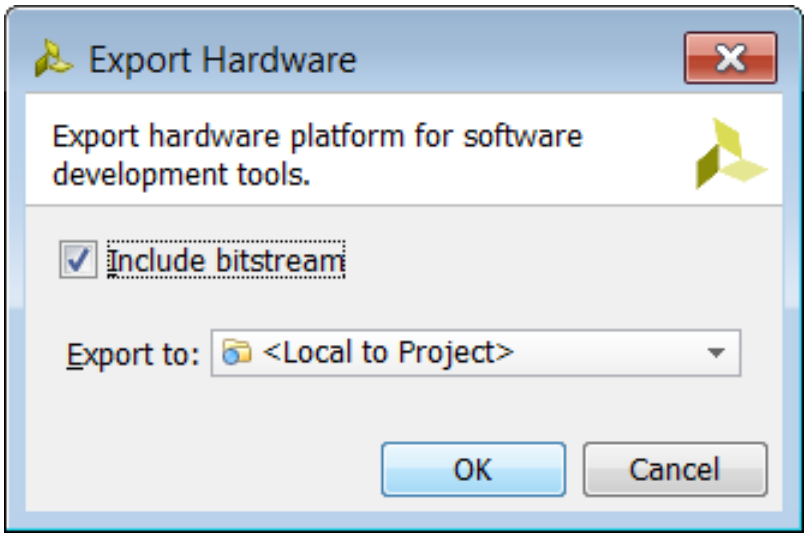

Figure 8: **Export Hardware**

PetaLinux Steps

The rest of the example design is implemented with the PetaLinux tools and requires a Linux operating system. A virtual machine running Linux is acceptable if Windows is the native operating system. Refer to the *PetaLinux Tools Reference Guide* (UG1144) [Ref 4] for help installing the PetaLinux Tools. Xilinx also recommends that you become familiar with the PetaLinux Tools. The PetaLinux tutorials, *PetaLinux Tools Documentation Workflow Tutorial* (UG1156) [Ref 5], serve as a helpful reference to become familiar with the tools quickly.

1. The reference design files include a `sw` directory to include sources for the PetaLinux and also a build area for the PetaLinux Project. Open a command line terminal and change to the `sw/build` directory. Then create a PetaLinux project with the following command:

   ```
   >> petalinux-create -t project -n xvc_linux
   ```

2. After the project is created, you must change into the `xvc_linux` directory created from the `petalinux-create` command. Next a C template application is created and enabled into the Linux build with the following command:

   ```
   >> petalinux-create -t apps -n xvcServer --enable --template c
   ```

3. This creates a template application called `xvcServer`, which will replace the template application with the application software located here: `sw/source/app/xvcServer.c`. From the command prompt, enter the following:

   ```
   >> cp ../../source/app/xvcServer.c ./components/apps/xvcServer/
   ```

4. The following step includes the user hardware selection from the Hardware Definition File (HDF), which was created from the Vivado Project and located in the `xvc_mz.sdk` directory. It might be necessary to copy the HDF file from the `xvc_mz.sdk` directory or use the prebuilt HDF located in the `sw/source/prebuilt_hdf` directory.

   Both HDFs should be identical. Following is the command used if the HDF is copied from the `xvc_mz.sdk` directory to the `sw/build/hdf_built_from_hw` directory:

   ```
   >> petalinux-config --get-hw-description=../hdf_built_from_hw/
   ```

5. The default device tree needs to be modified so that the xvcServer application can use the uio drivers. It also includes modifications for proper operation with the Ethernet PHY on the MicroZed board. Copy the `pl.dtsi` and `system-top.dts` file from the `sw/source/device_tree` directory to the `sw/build/xvc_linux/subsystems/linux/configs/device-tree` directory. Figure 9 and Figure 10 show the modified device tree files.

```
5
6   &gem0  {
7     phy-handle = <&phy0>;
8     ps7_ethernet_0_mdio {
9       #address-cells = <1>;
10      #size-cells = <0>;
11      phy0: phy@0 {
12        compatible = "marvell,88e1510";
13        device_type = "ethernet-phy";
14        reg = <0x0>;
15        marvell,reg-init = <3 16 0xff00 0x1e 3 17 0xfff0 0x00>;
16
17      };
18    };
19  };
20
```

Figure 9: **system-top.dts**

```
6
7     &amba {
8         ranges ;
9         axi_jtag_0: axi-jtag@43c00000 {
10            compatible = "generic-uio";
11            reg = <0x43c00000 0x10000>;
12        };
13    };
14
```

Figure 10: **pl.dtsi**

6. Build Linux with the following PetaLinux command:

```
>> petalinux-build
```

7. After Linux is built, package the Linux build to create the `boot.bin` and the `image.ub` files. The command to package everything needs to have the Vivado design tools sourced. The Vivado design tools need to be sourced because the `Bootgen` tool is needed. Following is the command to package everything:

```
>> petalinux-package --boot --fsbl ./images/linux/zynq_fsbl.elf --fpga
./subsystems/linux/hw-description/xvc_system_wrapper.bit --uboot  --force
```

8. The `boot.bin` file is located in the PetaLinux project directory, while the `image.ub` file is in the `./images/linux` directory. Copy both files on a Micro SD Card and plug it into the MicroZed board.

Connect Hardware

The MicroZed board is connected to the MicroZed I/O carrier card so that programmable logic I/Os can act as the JTAG signals. The I/O board is connected to an AC701 board by a JTAG flying wire adapter as shown in Figure 11.

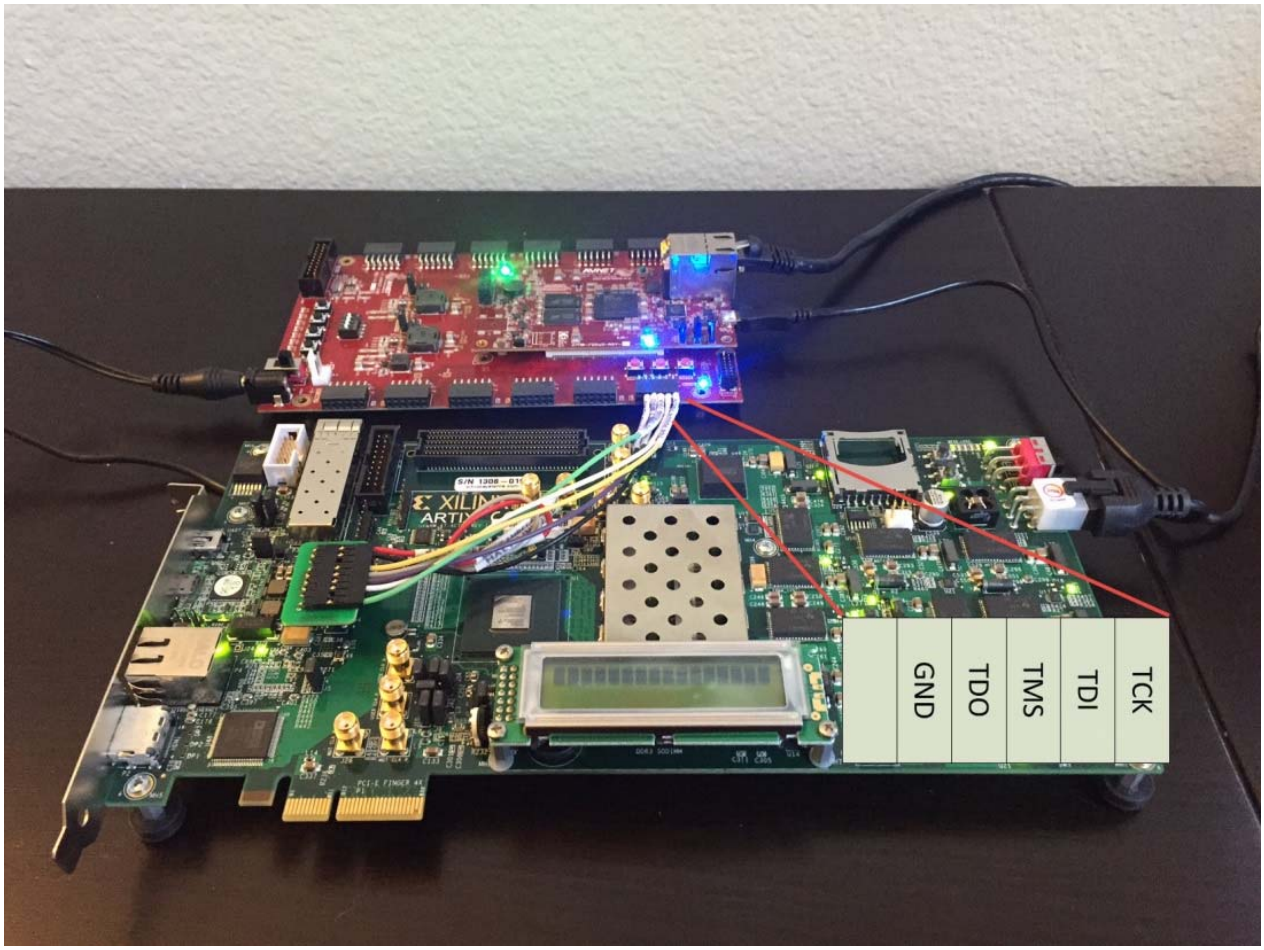

Figure 11: **Hardware Setup**

1. To boot from the Micro SD Card, ensure the mode pins on the MicroZed board are connected as shown in Figure 12.

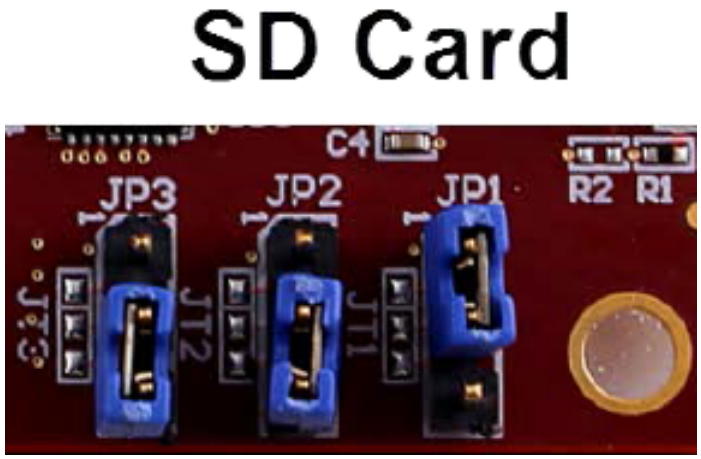

Figure 12: **Boot Mode Jumper Settings for SD Card Boot**

2. Also ensure that the Vcco pins are operating at the correct voltage. For the MicroZed board, the Vcco_34 and Vcco_35 voltages are user-controlled by jumpers on the I/O board. In this reference design, the AC701 board is used and the JTAG interface operates at 3.3V. For this reason, the Vcco on bank 34 will be set to 3.3V.
3. After the MicroZed is connected as shown in Figure 11, connect the Ethernet cable from the MicroZed board to a host machine with Vivado design tools installed, or to a router with a host machine connected to the router. Also connect the USB cable to the MicroZed board and establish a serial connection with the same host machine. Power on the MicroZed board and log into Linux with the user name and password both as 'root'.
4. After you are logged into Linux, launch the xvcServer in the background by typing the following in the console.

```
>> xvcServer &
```

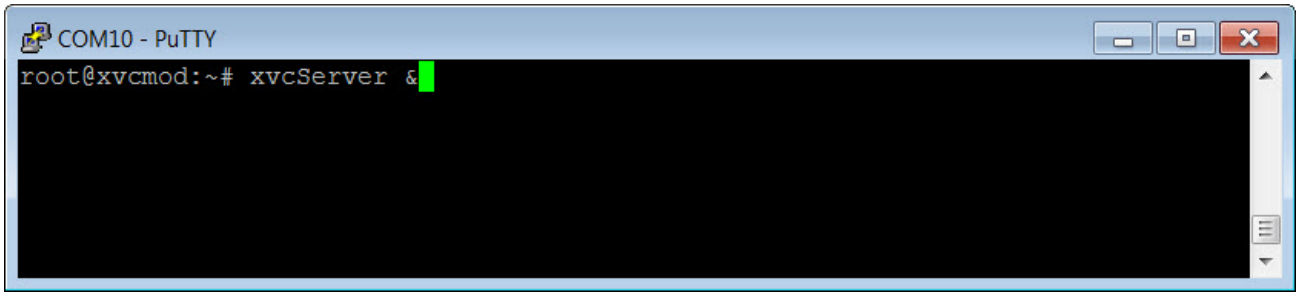

Figure 13: **Running xvcServer as Daemon**

5. Within the Linux console, find the IP address of the eth0 device by typing the following in the console:

```
>> ifconfig
```

The IP address assigned will display as shown in Figure 14. Remember this IP Address as it will be used later.

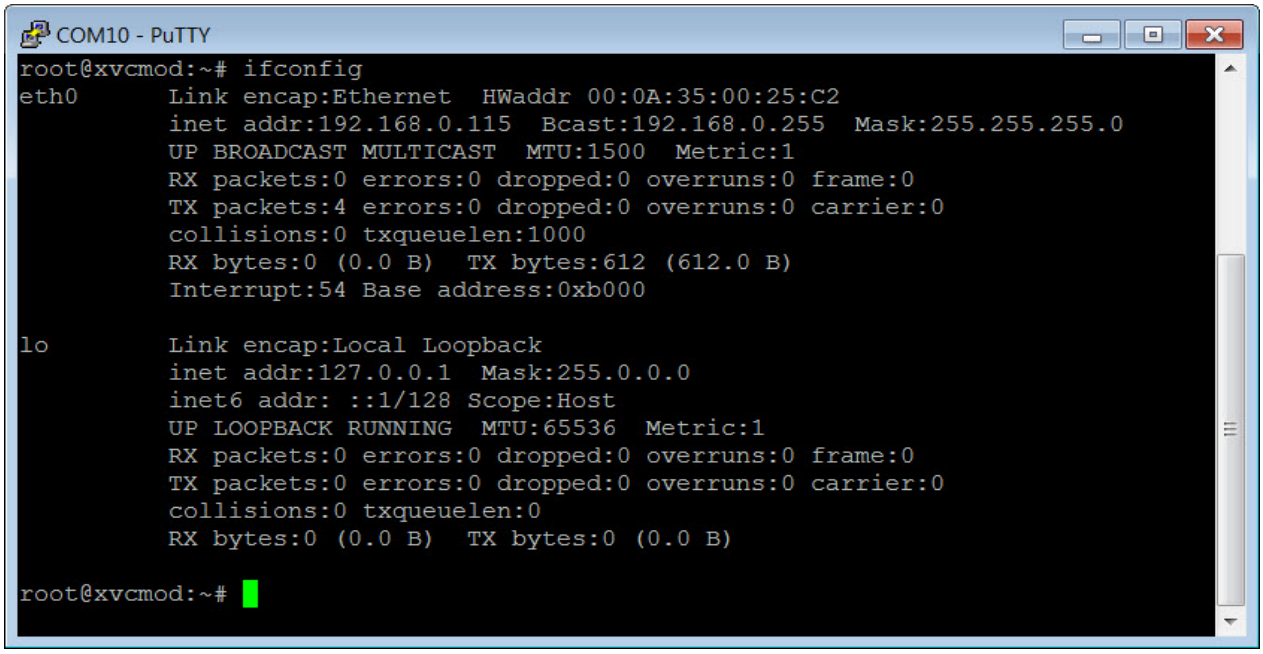

Figure 14: **eth0 Device Showing an Address of 192.168.0.115**

6. If an `inet addr` is not assigned, you can set the address manually with the following command:

```
>> ifconfig eth0 192.168.0.115 netmask 255.255.255.0
```

Vivado Design Tool Flow

After the hardware is correctly connected and XVC is running on the MicroZed board, open a Vivado Hardware Manager session as shown in Figure 15.

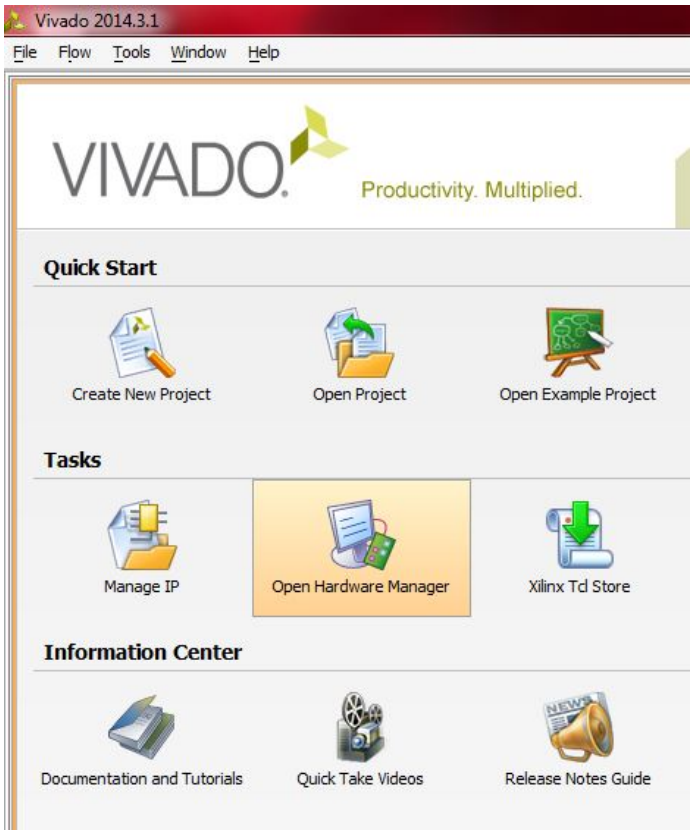

Figure 15: **Opening the Hardware Manager**

1. Start a Hardware Server session within Vivado IDE or with the following command in the Tcl Console:

   ```
   >> connect_hw_server
   ```

2. Open a hardware target with the following command in the Tcl Console. The IP address is based on what was found from Connect Hardware.

   ```
   >> open_hw_target -xvc_url 192.168.0.115:2542
   ```

   The -xvc_url switch allows the hardware server to know the Ethernet address and port to communicate with.

   In this particular setup, the MicroZed board is connected to an AC701 board; however, you can connect to any device. Because the AC701 board is used, the Artix®-7 200T device displays in the Hardware Manager tab as shown in Figure 16.

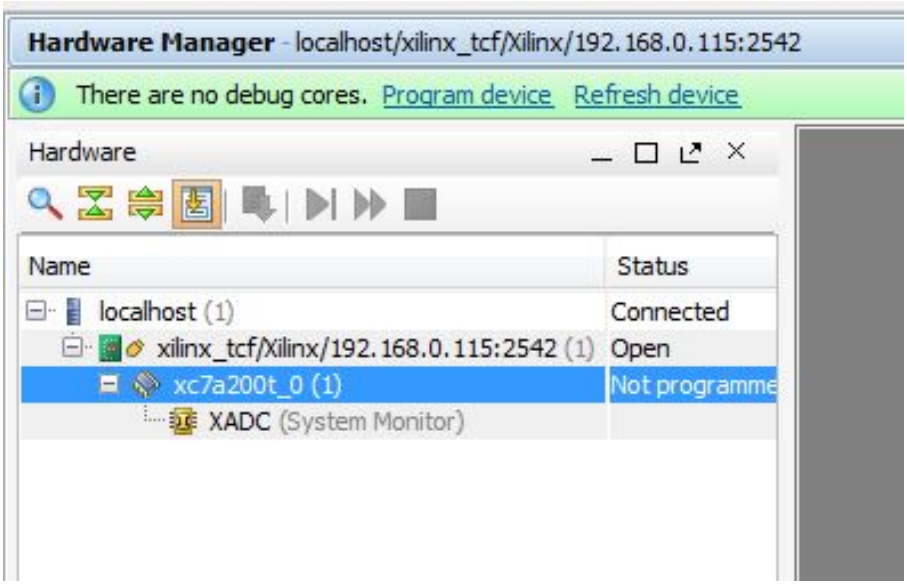

Figure 16: **Connecting to an Artix-7 Device**

You can program the Artix-7 200T device with the provided bit file in the `target_artix_files` directory from the reference design. If a different board is used, you can program the design with a bitstream for that device. For designs with Vivado Hardware Debug IP cores, they will show up after being programmed. The ltx file must be associated with the bit file targeting the FPGA. For information on using bit files and ltx files within the Hardware Manager, refer to the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 6].

Using XVC with XSDB

1. XVC also allows you to remotely debug a Xilinx microprocessor with Xilinx System Debugger (XSDB.) XSDB can be used when hw_server is up and running with an XVC connection. Use the two previously mentioned commands to get the hw_server running:

   ```
   >> connect_hw_server
   >> open_hw_target -xvc_url 192.168.0.115:2542
   ```

2. After hw_server is running, open an XSDB console within SDK or through the command line. Figure 17 shows an example of stopping and starting a MicroBlaze processor connected to the AC701 board.

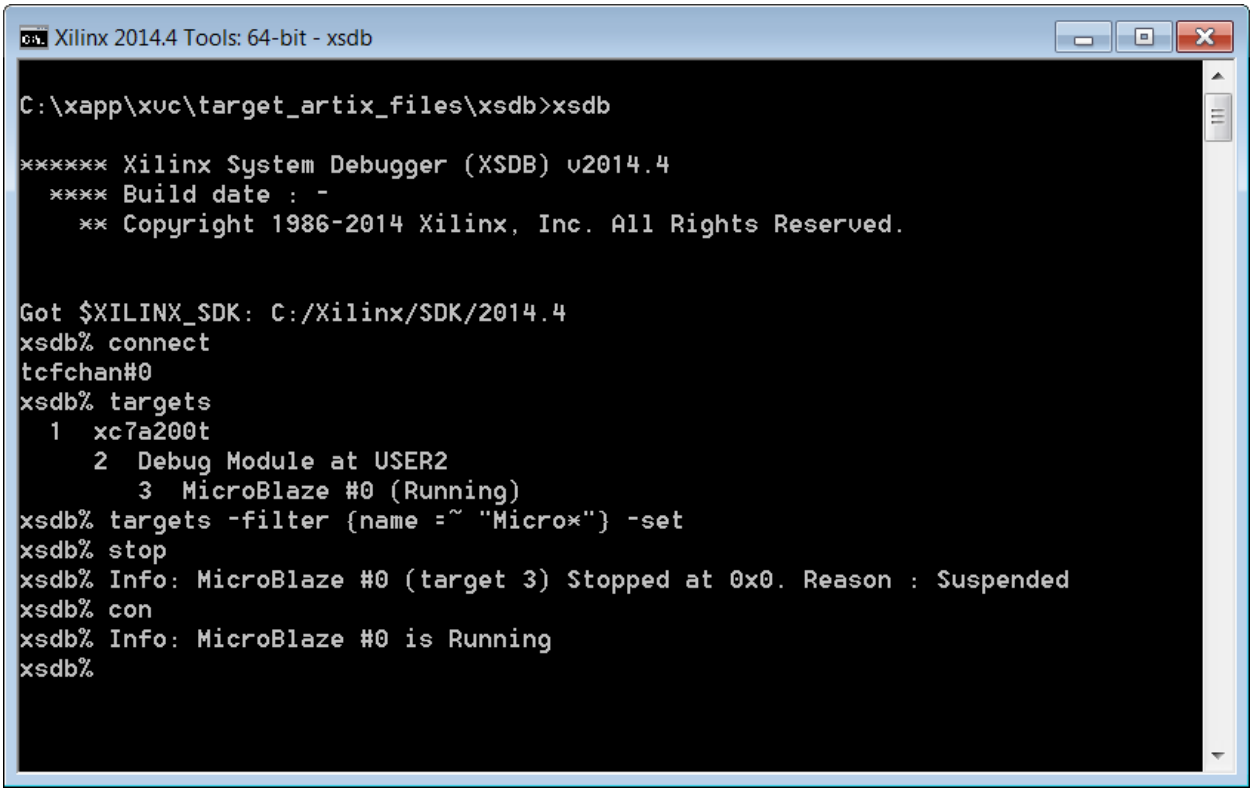

Figure 17: **Stopping and Starting a MicroBlaze Processor Connected to the AC701 Board**

Performance Considerations

The reference design was measured to have a performance of approximately 3.2 Mb/s while programming the AC701. This measurement was made with a TCK frequency of 6.25 MHz and the ARM processor running at 667 MHz.

XVC performance can be optimized by taking the following considerations:

- Reduce TCK ratio.

  The faster the TCK clock operates, the faster XVC operates. As the speed of TCK increases, it is important to ensure that the signal integrity of the JTAG signals will tolerate the increased speed.

- Ensure a 1 Gigabit Ethernet connection.

  The Ethernet controller can operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s. The faster the connection speed, the faster XVC will perform.

- Create a larger software buffer size.
  The larger the buffer size, the faster XVC will operate. The sensitivity of performance improvement depends on the latency on the network. The larger the latency on the network, the more effect a larger software buffer improves performance.

- Increase vector length -- AXI4-Lite to JTAG controller.

  The AXI4-Lite to JTAG Controller can be modified to improve performance. The controller can be modified to accept larger than 32-bit vectors. The larger the vector length, the faster XVC can operate.

- Add interrupts -- AXI4-Lite to JTAG Controller.

  Adding interrupt support helps offload the processor during `shift` XVC commands. Currently the controller needs the CTRL register to poll to see when the Enable Shift Operation bit is cleared. This can improve performance if the processor is already heavily utilized. Adding interrupt support requires a change to the AXI4-Lite to JTAG Controller as well as the application software.

Other Considerations

Following are some other considerations.

- Bit Bang JTAG

  Instead of using the AXI4-Lite to JTAG controller, it is possible to bit bang the JTAG commands over the general purpose I/O. This operation slows down the speed of JTAG, but removes the necessity of programmable logic. This use case is more common for implementations not having programmable logic, like a general purpose processor.

- Use lightweight IP (LwIP) for TCP/IP Stack

  Although this application note uses Linux for the TCP/IP stack, you can also run XVC with LwIP on a bare metal system or on an Real Time Operating System (RTOS.)

- Run XVC on the MicroBlaze™ processor.

  You can run XVC on a MicroBlaze processor. The MicroBlaze processor is supported within the PetaLinux tools, so it is possible to run Linux using this processor. Although implementing the MicroBlaze processor with Linux is a viable option, it is more common to use the MicroBlaze processor with LwIP.

- Security

  XVC does not have built in security levels; it is a simple communication protocol. For example, the following security measures can be implemented to protect access.

  - Secure Socket Shell (SSH) Encrypted Ports
  - Virtual Private Network (VPN)
  - Simple Object Access Protocol (SOAP)

  It is up to you to determine the appropriate levels of security. Implementing security is outside the scope of this document.

Conclusion

The reference design provided as part of this application note has shown how to implement the XVC protocol. XVC allows you to use an existing Ethernet connection to remotely or locally debug a Xilinx FPGA while leveraging all of the existing Vivado Hardware Debug Tools. A product architecture can benefit from using XVC to remove the need for an additional JTAG cable.

References

1. XVC v1.0 Protocol
2. *MicroZed Hardware User Guide* v1.6
3. *Vivado Design Suite User Guide: Designing with IP* (UG896)
4. *PetaLinux Tools Reference Guide* (UG1144)
5. *PetaLinux Tools Documentation Workflow Tutorial* (UG1156)
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
04/30/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices