Bare Metal HDMI for ZedBoard with ADI IP and ADV7511

Version 1.0
June 2014

Table of Contents

Introduction

This document describes a Xilinx Zynq Processor System (PS) integrated with Programmable Logic (PL) peripherals that have been implemented using the Vivado® Design Suite. The target hardware platform is the Avnet ZedBoard.

This reference design provides a Linux-ready FPGA platform and non-OS (bare metal) software projects to exercise and validate the following blocks:

1. Processor System
2. HDMI Video and Audio Output
3. DDR Memory
4. I^2C Peripherals

The design includes additional components in the PS and PL needed to boot and execute a Linux kernel with extensions for Ubuntu (open source). The ADV7511 HDMI Transmitter from Analog Devices is interfaced to the PS via IP blocks provided by ADI, for both Linux and bare metal operation.

For those wishing to build and run Ubuntu software on this platform, consult the Avnet **Ubuntu on Zynq®-7000 All Programmable SoC Tutorial**, which describes in detail all the software steps needed. However, this bare metal design provides a functional starting point for any system that requires HDMI video and audio, with or without an operating system.

Objectives

This reference design will demonstrate how to do the following:

- Validate your Development Kit setup by running the demonstration files booted from an SD card.
- Create and build the Vivado Design Suite project from source, and export the hardware platform to an SDK project.
- Use the SDK to:
  - Import and build software to test the HDMI video and audio
  - Load the Programmable Logic in the Xilinx SoC via JTAG or SD card
  - Execute the software to produce HDMI video and audio using the Analog Devices (ADI) ADV7511 transmitter

Reference Design Requirements

Software

The software requirements for this reference design are:

- Xilinx SDK 2013.4
- Cypress CY7C64225 USB-to-UART Bridge Driver
    - See *Cypress CY7C64225 USB-to-UART Setup Guide* at:
    - http://www.zedboard.org/documentation/1521
- Serial terminal emulation software such as Tera Term
    - http://en.sourceforge.jp/projects/ttssh2
- Optional - Xilinx Vivado® Design Suite 2013.4 (includes SDK)

Hardware

The hardware requirements for this reference design are:

- 64-bit host computer with at least 1.3 GB RAM (typical) and up to 1.9 GB RAM (peak) for Windows-7 or Linux operating systems available to Vivado Design Suite (XC7Z020 device)
    - http://www.xilinx.com/design-tools/vivado/memory.htm#zynq-7000
- Avnet ZedBoard Development Kit
    - 2 USB-A to USB-micro B cables
        - USB UART – *included in the kit*
        - JTAG (*not required for SD demo*)
    - 12v Power supply – *included in the kit*
    - SD card – *included in kit*
- HDMI cable
- HDMI-Capable Monitor
    - HDMI-to-DVI adapter may be necessary if the monitor does not have an HDMI port
    - (Optional) For HDMI audio, sink device must be HDMI-audio capable. Not all monitors equipped with speakers use an HDMI source for audio output.

Supplied Files

The following directory structure is included with this reference design. These "golden" files can be used to validate the operation of your board and external connections. See the section "Running the Demos" for details.

ZedBoard_HDMI_Vivado2013_4.pdf: This document.

boot_source: Contains the files to create the boot.bin images required to boot from the SD card:
- **system_top.bit**: The bitstream for the Vivado hardware design.
- **vHDMI.elf**: Executable file for the bare metal HDMI application for Vivado.
- **zynq_fsbl.elf:** The ARM executable for the Zynq FSBL.

sd_card: Contains the files to run test programs from the SD card:
- **vHDMI_boot.bin**: Executes the bare metal HDMI A/V test.

SDK_project_archive
- **zed_hdmi_sdk_ws.zip**: Software project archive to import to the Xilinx SDK.

vivado:
- **library**: Source files for the library IP used in the hardware platform.
- **projects**: Source files for the HDMI hardware project for Vivado Design Suite.
- **built.tcl**: Tcl script to create and build the hardware platform from the Vivado Tcl Shell.[1]

The archive can be decompressed anywhere on your system, as long as there are no spaces in the pathname, as the Xilinx tools do not parse spaces. For the purposes of this document, it is assumed that the archive has been unzipped to the folder:

C:\ZedBoard_v2013_4\HDMI

Wherever this path is specified, substitute your own installation path.

Caution: When choosing your own path in Windows, use caution in the path length as all files in the project hierarchy must fit within the Windows OS limitation of 260 characters. See http://www.xilinx.com/support/answers/52787.html.

[1] Alternately, the source files may be downloaded from Github and the hardware platform built using the Vivado Design Suite GUI. Instructions for downloading and building in this mode can be found at: https://github.com/analogdevicesinc/hdl

Setting up the ZedBoard Development Kit

Refer to the following figure and perform the steps to set up the board for running the bare metal applications from the SD card, or via JTAG download.

1. Connect 12 V power supply to ZedBoard barrel jack (J20).

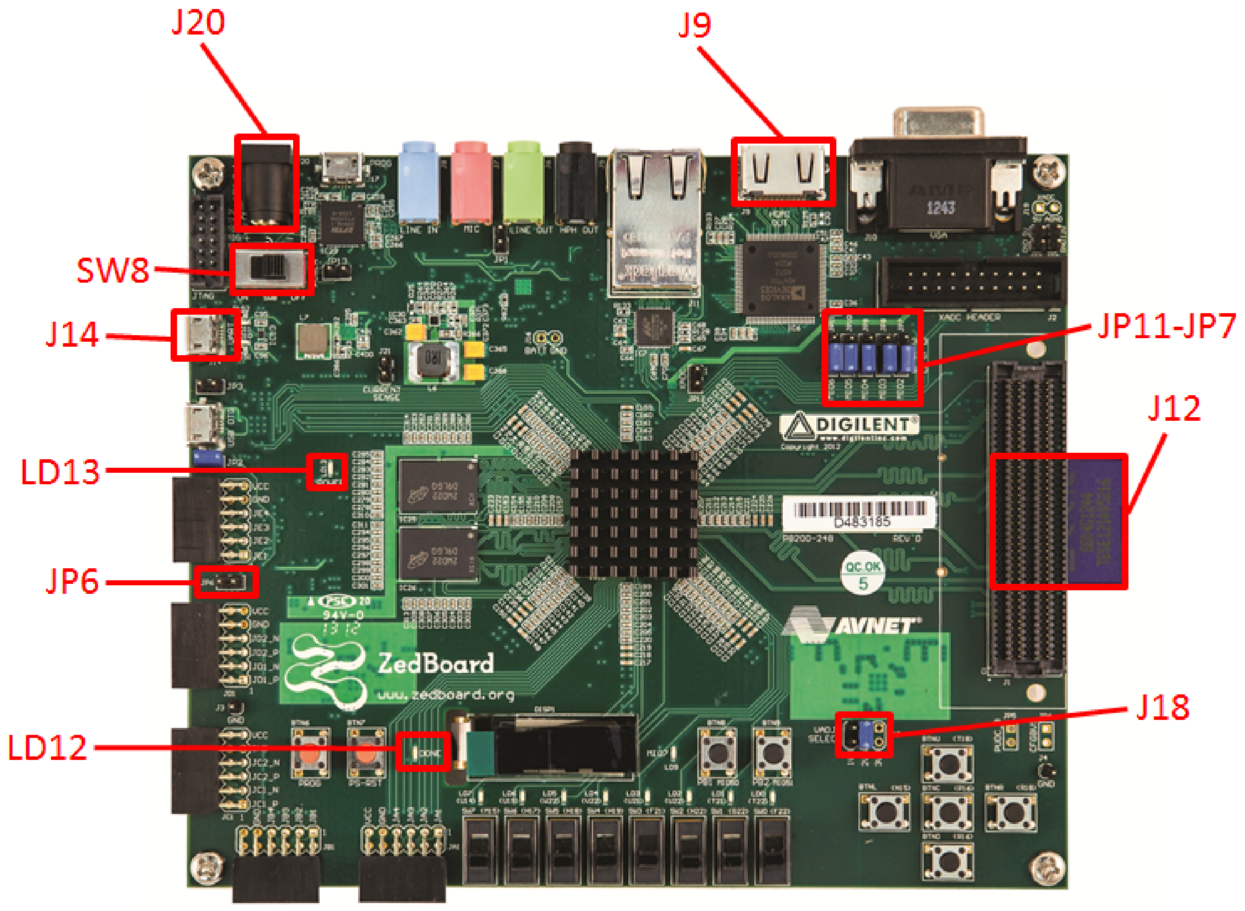


Figure 1 – ZedBoard Jumper Locations

2. Connect the USB-UART port of the ZedBoard (J14), labeled UART, to a PC using the Micro-USB cable.

3. Use Windows Explorer to copy **vHDMI_boot.bin** from the **Supplied Files** to the top level of the SD card. Rename the file to **boot.bin**.

4. Insert the SD card into the ZedBoard SD card slot (J12) located on the underside of the PCB.

5. Insert jumper on J18 2V5 to select 2.5V for Vadj.

6. To boot from the SD card, set the ZedBoard boot mode (JP7-JP11) and MIO0 (JP6) jumpers to SD card mode (shown in the Figure above). To boot from JTAG, set the boot mode jumpers to JTAG mode:

SD Card Boot Mode	JTAG Boot Mode
JP11: SIG - GND	JP11: SIG - GND
JP10: SIG - 3V3	JP10: SIG - GND
JP9: SIG - 3V3	JP9: SIG - GND
JP8: SIG - GND	JP8: SIG - GND
JP7: SIG - GND	JP7: SIG - GND
JP6: Rev B or C: CLOSED Rev D: Don't Care	JP6: Don't care

Figure 2 – Mode Jumper Settings

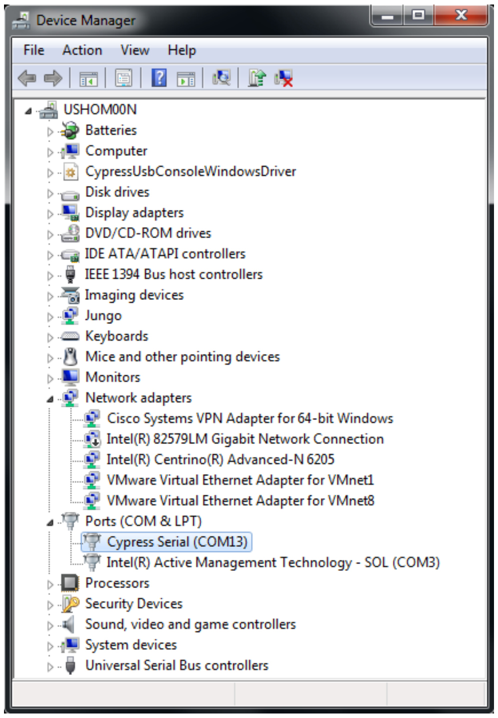
7. Connect an HDMI cable between the HDMI sink device and the J9 connector on the ZedBoard.

8. Turn power switch (SW8) to the ON position. ZedBoard will power on and the Green Power Good LED (LD13) should illuminate.

9. The PC may pop-up a dialog box asking for driver installation.

   ZedBoard has a USB-UART bridge based on the Cypress CY7C64225 chipset. Use of this feature requires that a USB driver be installed on your Host PC. If the host PC requested driver installation, refer to the Cypress CY7C64225 USB-to-UART Setup Guide in the **Reference Design Requirements** section for instructions. When driver installation is complete, continue to the next step.

10. Wait approximately 15 seconds. The blue Done LED (LD12) should illuminate. Use the Windows Device Manager to determine the COM Port.



```
USHOM00N
  Batteries
  Computer
  CypressUsbConsoleWindowsDriver
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  IDE ATA/ATAPI controllers
  IEEE 1394 Bus host controllers
  Imaging devices
  Jungo
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
    Cisco Systems VPN Adapter for 64-bit Windows
    Intel(R) 82579LM Gigabit Network Connection
    Intel(R) Centrino(R) Advanced-N 6205
    VMware Virtual Ethernet Adapter for VMnet1
    VMware Virtual Ethernet Adapter for VMnet8
  Ports (COM & LPT)
    Cypress Serial (COM13)
    Intel(R) Active Management Technology - SOL (COM3)
  Processors
  Security Devices
  Sound, video and game controllers
  System devices
  Universal Serial Bus controllers
```

Figure 3 – Device Manager Showing Enumerated USB-UART

Note: Each unique USB-UART device attached will enumerate under the next available COM port. Here in this example, the Cypress CY7C64225 USB-UART device is enumerated as COM13.

Installing a Serial Console on a Windows 7 Host

Starting with Windows 7, Microsoft no longer includes the HyperTerminal terminal emulator software. However, this example design requires use of terminal emulation software for a serial console connection to the target. A suitable free and open-source replacement for HyperTerminal is TeraTerm. Download and install instructions for TeraTerm can be found at:

http://en.sourceforge.jp/projects/ttssh2.

As an alternative the Terminal applet in the Xilinx SDK may also be used.

Running the Demo Using the SD Card

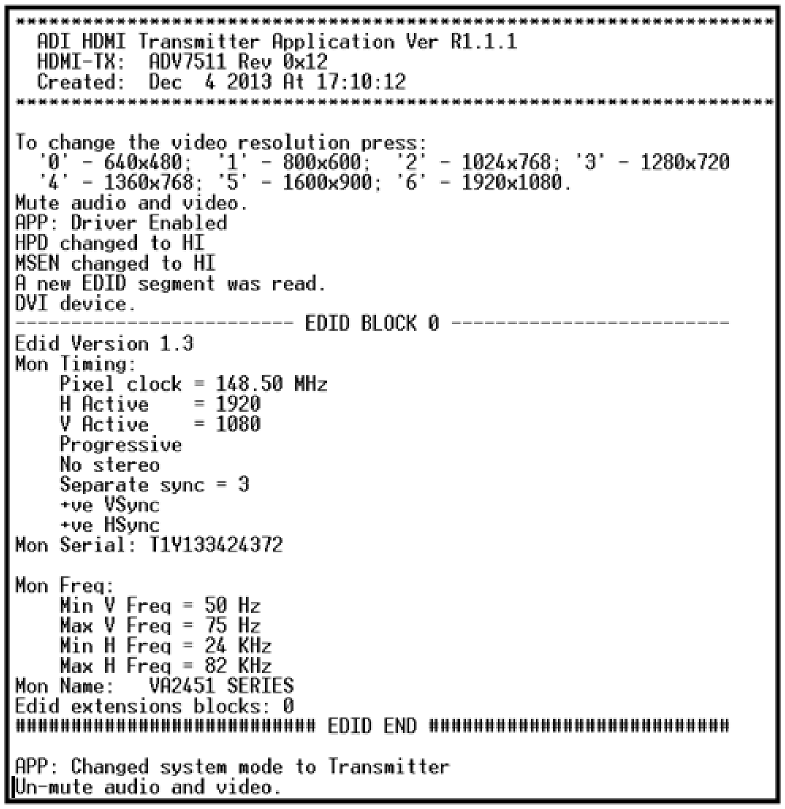
You must have the hardware set up and connected as described in **Setting up the ZedBoard Development Kit**. Be sure to set the boot mode jumpers to SD card mode as shown in **Figure 2 – Mode Jumper Settings**.

If you have not already done so, copy the **vHDMI_boot.bin** from the **Supplied Files** to the top level of the SD card. The card must already have a formatted FAT32 partition with read/write access from Windows to do this.

Rename this file on the SD card to **boot.bin**. This is because boot mechanism used by the Zynq-7000 All Programmable SoC looks for this specific file name on the boot media.

After you have completed this operation, you may insert the SD card into the card cage. Power the board and **AFTER** the blue DONE LED illuminates, start your serial console connected to the COM port selected by the USB-UART driver with communication parameters of 115200,n,8,1. Depending on how long it took you to establish the serial console, you may need to press the **PS-RST**[2] button on the board to repeat the boot procedure and view the serial output.

Expected results on the serial console are shown below:



```
*******************************************************************
  ADI HDMI Transmitter Application Ver R1.1.1
  HDMI-TX:  ADV7511 Rev 0x12
  Created:  Dec  4 2013 At 17:10:12
*******************************************************************

To change the video resolution press:
  '0' - 640x480;  '1' - 800x600;  '2' - 1024x768; '3' - 1280x720
  '4' - 1360x768; '5' - 1600x900; '6' - 1920x1080.
Mute audio and video.
APP: Driver Enabled
HPD changed to HI
MSEN changed to HI
A new EDID segment was read.
DVI device.
------------------------- EDID BLOCK 0 -------------------------
Edid Version 1.3
Mon Timing:
    Pixel clock = 148.50 MHz
    H Active    = 1920
    V Active    = 1080
    Progressive
    No stereo
    Separate sync = 3
    +ve VSync
    +ve HSync
Mon Serial: T1Y133424372

Mon Freq:
    Min V Freq = 50 Hz
    Max V Freq = 75 Hz
    Min H Freq = 24 KHz
    Max H Freq = 82 KHz
Mon Name:   VA2451 SERIES
Edid extensions blocks: 0
########################### EDID END ###########################

APP: Changed system mode to Transmitter
Un-mute audio and video.
```

Figure 4 – HDMI Test Serial Console Output

[2] The PS-RST operation works as described on ZedBoard Rev D and above. For ZedBoard Rev B and Rev C, once the persistent Cypress driver has established its initial COM connection, you can power-cycle the board instead.

The HDMI test will send a video signal to your HDMI sink device at the resolution selected on your serial console. By default, the application will attempt the highest compatible resolution supported by the device, as determined from the EDID information.

If your HDMI display has the ability to process HDMI audio signals, you will also hear a rapid clicking sound from the speakers or headphones attached to the display. Note that at the time of writing there are no bare metal audio drivers for the ADAU1761 codec connected to the on-board jacks. To provide audio signals through the jacks, you must use a Linux distribution that includes the Advanced Linux Audio Support (ALSA) package, such as Ubuntu.

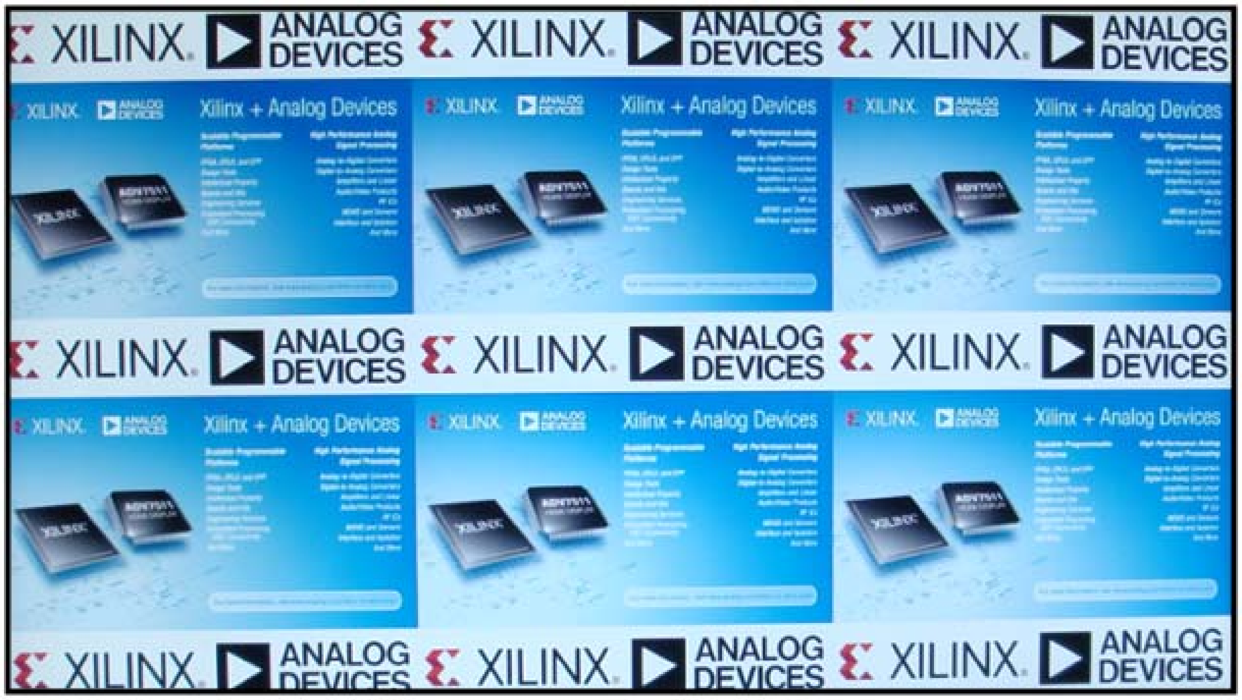


Figure 5 – HDMI Video at Default (High) Resolution

You may change the resolution by entering a single digit 0 through 6 in the serial console. Entering **2** for the connected HDMI sink shown in the example above results in a resolution of 1024x720.

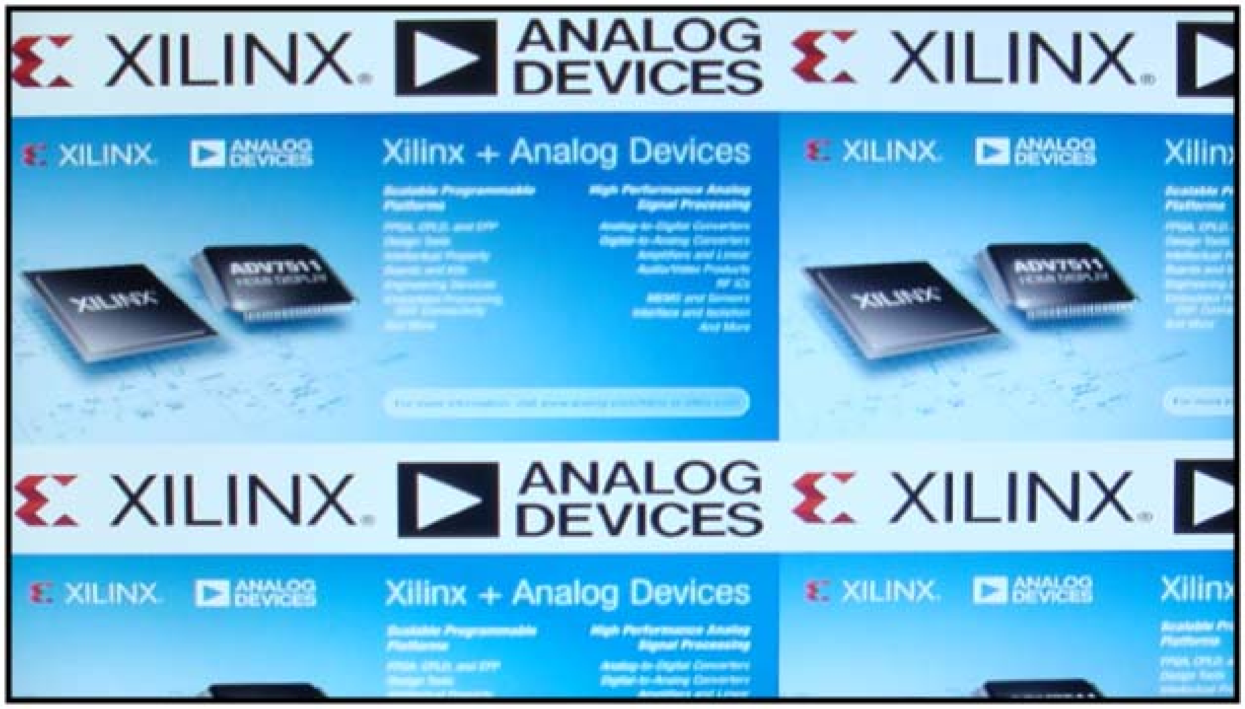


Figure 6 – HDMI Video at Low Resolution

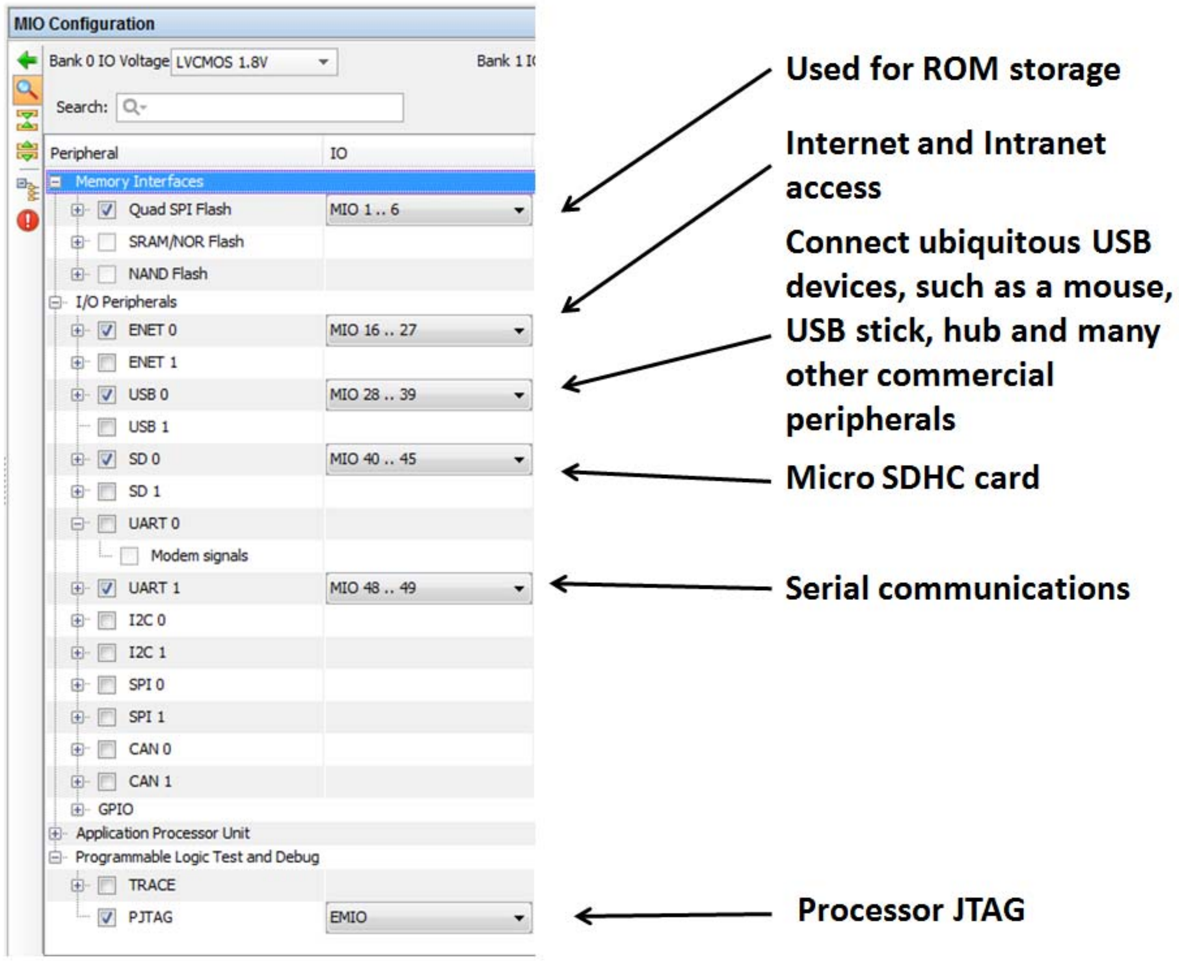
Linux Ready Hardware Platform

In order to design the processor-based hardware system, we must first consider the requirements of the software that will be executing on it. Our ultimate goal, beyond simply using HDMI in a bare metal environment, is to execute a Linux kernel which in turn will load a file system containing all the elements necessary to create an interactive Ubuntu desktop display in High Definition.

This statement implies that our processor system will need to include:

- **DDR Memory**: a full-featured open source embedded Linux kernel capable of executing all the applications and services associated with Linux cannot be accommodated on-chip, so a connection to external memory must be provided.
- **Ethernet**: an Ubuntu desktop environment makes extensive use of the Internet, so we need the Linux kernel to configure a network connection and to obtain an IP address automatically. This means an Ethernet MAC and PHY must be included in our hardware list. For production purposes, a unique MAC address for each device must be defined at the time of manufacture.
- **USB Host**: interaction with the Ubuntu desktop can be performed primarily through a standard mouse and keyboard, both of which require a USB connection. Linux supports these devices with its own USB stack in software, but we must be able to access USB PHY in the design.
- **UART**: to monitor the boot process of the hardware after power up, we will need a console separate from the HDMI display. Access to the console information is made via a serial connection, so our system needs a UART connection.
- **I^{2}C**: our system has requirements for HDMI output and stereo audio input and output. The ZedBoard target has both an HDMI transmitter (Analog Devices ADV7511) and an audio codec (Analog Devices ADAU1761), and both of these chips are controlled via an I^{2}C interface. To properly configure the chips from software, the processor must be able to connect to an I^{2}C interface.
- **SD**: The Ubuntu RFS will be located on an external SD card. This allows flexibility in preparing the file system outside of our embedded platform, and provides solid-state persistent storage that will save the environment between power cycles. In order for the Linux kernel to read the SD card, an interface to the processor must be included.
- **Timer**: Linux operates more efficiently when access to a hardware timer is provided. This is more accurate and portable than a software timer alone.
- **Quad SPI Flash**: This reference design can be modified to utilize QSPI flash in place of the SD to hold the boot files and other elements that may require persistence across power cycles. While not used in this reference design, it is included so this feature can be added later without modifying the underlying platform.
- **Processor JTAG (PJTAG)**: This is not strictly required by the software, but since we are using a Xilinx SoC, PJTAG provides additional capabilities not found in fixed SoC systems, allowing the Zynq PS to reconfigure elements of the Zynq PL at runtime. This facility is not used in this reference design, but the capability is included for future development.

The hard-block processor system at the heart of the Zynq architecture can be configured in Vivado to satisfy the basic software requirements of our system. Consider this block diagram for the dual ARM Cortex-A9 cores and hard peripheral system. A checkmark next to each block indicates that the peripheral is active and will be included in the build. Non-active components are disabled, reducing the overall power consumed by the system during operation.


MIO Configuration
Used for ROM storage
Internet and Intranet access
Connect ubiquitous USB devices, such as a mouse, USB stick, hub and many other commercial peripherals
Micro SDHC card
Serial communications
Processor JTAG

Bank 0 IO Voltage: LVCMOS 1.8V

Peripheral	IO
Memory Interfaces	
✓ Quad SPI Flash	MIO 1 .. 6
☐ SRAM/NOR Flash	
☐ NAND Flash	
I/O Peripherals	
✓ ENET 0	MIO 16 .. 27
☐ ENET 1	
✓ USB 0	MIO 28 .. 39
☐ USB 1	
✓ SD 0	MIO 40 .. 45
☐ SD 1	
☐ UART 0	
☐ Modem signals	
✓ UART 1	MIO 48 .. 49
☐ I2C 0	
☐ I2C 1	
☐ SPI 0	
☐ SPI 1	
☐ CAN 0	
☐ CAN 1	
GPIO	
Application Processor Unit	
Programmable Logic Test and Debug	
☐ TRACE	
✓ PJTAG	EMIO

Figure 7 – MIO for PS Peripherals for Ubuntu

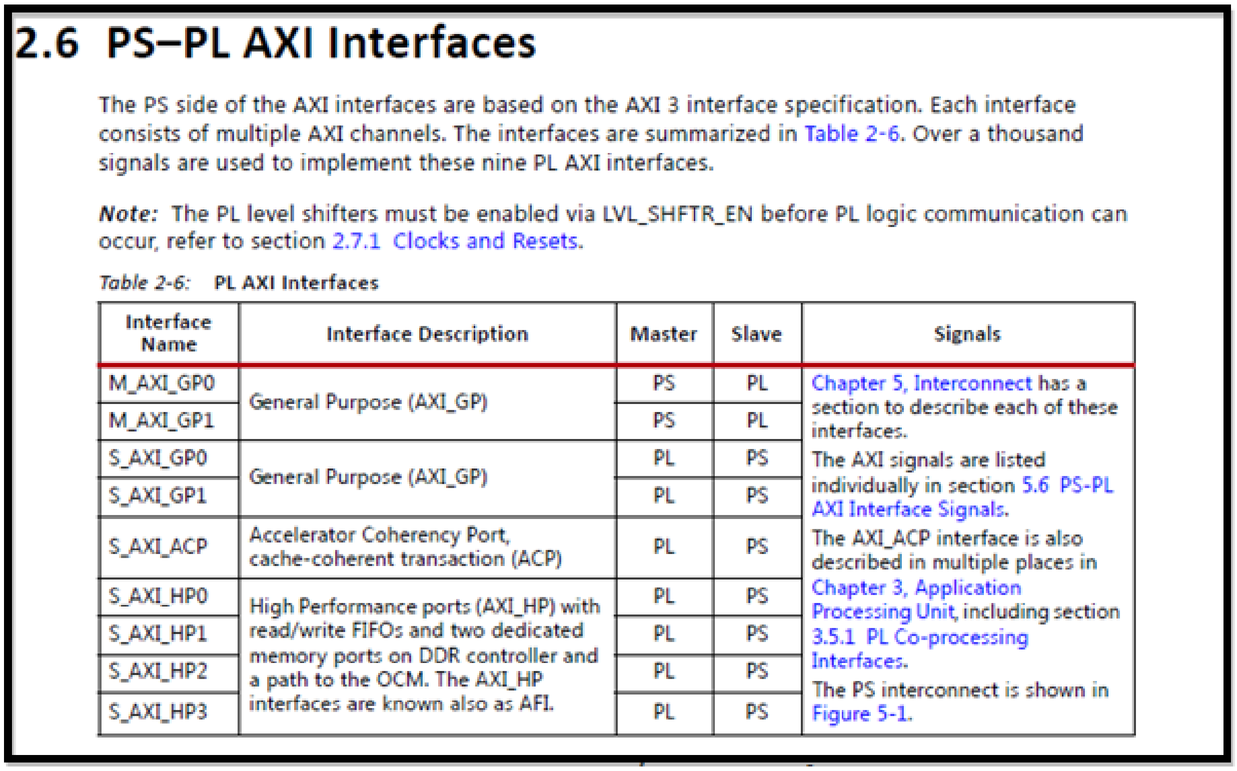
The system elements included above will be common to many processor-based designs. Not shown in the diagram above is the configuration of the Hardware Timer, which is selected in the Clock Configurations panel rather than the MIO Configurations. Although these peripherals are sufficient for many processor designs, in order to communicate with the on-board HDMI transmitter and audio codec, we require further custom peripherals within the FPGA fabric. This is where the flexibility of the FPGA comes to the forefront.

The processor system communicates with PL peripherals using the Advanced eXtensible Interface (AXI) architecture, while third party peripherals may each have their own communications protocols depending on the device and its purpose. In order to convert data formats from AXI to the device specific architectures, we must to instantiate custom IP in the FPGA fabric that will act as a translator between the processor system and each device function.

1. **HDMI:** The processor system has access to the I^{2}C bus via a Xilinx AXI_IIC controller built in the PL. This AXI_IIC communicates with the HDMI transmitter and configures it for operation. However, we have not yet addressed the actual video and audio signals necessary to activate the HDMI monitor. To do this, we need to be able to send streams of data to the transmitter in a format it recognizes.
   a. **axi_hdmi_core**: Custom IP provided by Analog Devices to provide a bridge between the Zynq-7000 All Programmable SoC and the ADV7511 HDMI transmitter chip for 16-bit color data. This IP provides control information to the chip to process the video stream arriving over VDMA.
   b. **axi_spdif_tx_core**: Custom IP provided by Analog Devices to provide a path for transmitting HDMI audio signals from the Zynq-7000 All Programmable SoC to the ADV7511 HDMI transmitter chip. See **Stereo Audio** below for a description.
   c. **axi_hdmi_dma**: Custom IP provided by Analog Devices to provide a path between DDR and the HDMI ADV7511 transmitter chip to move HDMI video data. This DMA engine offloads the details of maintaining a stream of video output from the Zynq processor cores. Access to the processor system is provided through the S_AXI_HP0 interface.
2. **Stereo Audio**: Audio signals, like video, create data streams that are best suited to transmission in the hardware, relieving the processor system from managing the transfers. This means that we need to include additional Direct Memory Access engines in our design, which can be programmed by the processor to move the audio data without constant involvement of the CPUs. We could instantiate the DMA engines in the PL fabric, but in the case of a Zynq All Programmable SoC, we have a hard-block containing 4 PL330 DMA engines that can be individually activated and configured. We will need to use three of these blocks combined with the following IP to route the data to/from their target devices via the S_AXI_HP2 interface.
   a. **axi _spdif_tx_core**: S/PDIF stands for Sony/Philips Digital Interconnect Format. This connects via DMA controller Interface 0 to the HDMI transmitter for transmission of HDMI audio. The transmitter chip handles the details of time-multiplexing the video and audio signals onto the HDMI output lines for transmission to the display device.
   b. **axi_i2s_adi**: Integrated Interchip Sound, an electrical serial bus interface standard used for connecting digital audio devices. This IP connects DMA interfaces 1 and 2 to the Analog Devices ADAU1761 audio codec, one for stereo output to headphones or line-out jacks and the second for stereo input via the microphone or line-in jacks.
3. **CLOCK**
   a. **sys_audio_clkgen**: This block takes the 200 MHz FCLK_CLK1 from the PS as an input and generates a 20 MHz clock output that is used as an input to axi_i2s_adi, axi_spdif_tx_core, and externally to the ADAU1761 MCLK input.
   b. **axi_hdmi_clkgen**: This block takes the 200 MHz FCLK_CLK1 from the PS as an input and generates a 148.6 MHz clock output that is used as an input to the axi_hdmi_core.

4. **I^{2}C**
    a. **axi_iic_main:** Provides the connection between the processor and an IIC multiplexer which then connects to the ADV7511 and ADAU1761 I^{2}C peripherals.
    b. **sys_i2c_mixer:** Since axi_iic_main only has a single IIC port, an external multiplexer is required to connect multiple external IIC peripherals. The sys_i2C_mixer multiplexes the IIC ports of both the ADV7511 and the ADAU1761 back to the axi_iic_main IIC controller in the Zynq PL.
5. **Miscellaneous**
    a. **sys_logic_inv:** This is a simple inverter that inverts the low-asserted Over-Current (OCn) signal coming from the USB VBUS switch (TPS2051B) and connects to the high-asserted Zynq PS USB0 peripheral controller signal USB0_VBUS_PWRFAULT. With this inverter in place, if the TPS2051B detects an over-current situation for a USB device, then the PWRFAULT signal will be asserted.
6. **Interrupts**
    a. **sys_concat_intc:** This block concatenates multiple interrupts from axi_hdmi_core, axi_iic_main, and axi_iic_fmc and then connects to the interrupt input on the Zynq PS.
7. **Interconnects**
    a. **axi_cpu_interconnect**: This is an AXI bridge component built into the PL, with a single AXI Slave connection back to an AXI Master on the PS (M_AXI_GP0), bridging to 7 downstream Master AXI ports. The interconnect peripherals are needed to connect AXI-compatible IP to the PS via one of the High Performance or General Purpose interface channels. The decision on how many peripherals to attach to a single interconnect is determined by the available bandwidth. This interconnect block is attached to the M_AXI_GP0 port, which allows the PS to act as the master and the PL peripherals as slaves. All of the peripherals in the system except for video data are relatively low bandwidth and therefore all connected via this bridge. Those low-bandwidth peripherals include
        i. axi_iic_main
        ii. axi_iic_fmc
        iii. axi_hdmi_clkgen
        iv. axi_hdmi_core (control only)
        v. axi_hdmi_dma (control only)
        vi. axi_spdif_tx_core
        vii. axi_i2s_adi.
    b. **axi_hdmi_interconnect**: The high bandwidth consumer in this design is the HDMI output, so this interconnect is dedicated to the VDMA engine that transports the video data signals to the HDMI transmitter. The connection to the PS is made via the high performance S_AXI_HP0 port.

For information on the channels of communication available between the PS and PL sides of the Zynq SoC, consult the *Xilinx Zynq 7000 All Programmable SoC Technical Reference Manual* (UG585). For quick reference, the figure below provides a high-level synopsis of the available interfaces.



2.6 PS–PL AXI Interfaces

The PS side of the AXI interfaces are based on the AXI 3 interface specification. Each interface consists of multiple AXI channels. The interfaces are summarized in Table 2-6. Over a thousand signals are used to implement these nine PL AXI interfaces.

Note: The PL level shifters must be enabled via LVL_SHFTR_EN before PL logic communication can occur, refer to section 2.7.1 Clocks and Resets.

Table 2-6: **PL AXI Interfaces**

Interface Name	Interface Description	Master	Slave	Signals
M_AXI_GP0	General Purpose (AXI_GP)	PS	PL	Chapter 5, Interconnect has a section to describe each of these interfaces.
M_AXI_GP1		PS	PL	
S_AXI_GP0	General Purpose (AXI_GP)	PL	PS	The AXI signals are listed individually in section 5.6 PS-PL AXI Interface Signals.
S_AXI_GP1		PL	PS	
S_AXI_ACP	Accelerator Coherency Port, cache-coherent transaction (ACP)	PL	PS	The AXI_ACP interface is also described in multiple places in Chapter 3, Application Processing Unit, including section 3.5.1 PL Co-processing Interfaces.
S_AXI_HP0	High Performance ports (AXI_HP) with read/write FIFOs and two dedicated memory ports on DDR controller and a path to the OCM. The AXI_HP interfaces are known also as AFI.	PL	PS	
S_AXI_HP1		PL	PS	
S_AXI_HP2		PL	PS	The PS interconnect is shown in Figure 5-1.
S_AXI_HP3		PL	PS	

Figure 8 – Zynq PS/PS AXI Interfaces

Hardware Design Block Diagram

The following figure shows a high-level block diagram of the hardware design.

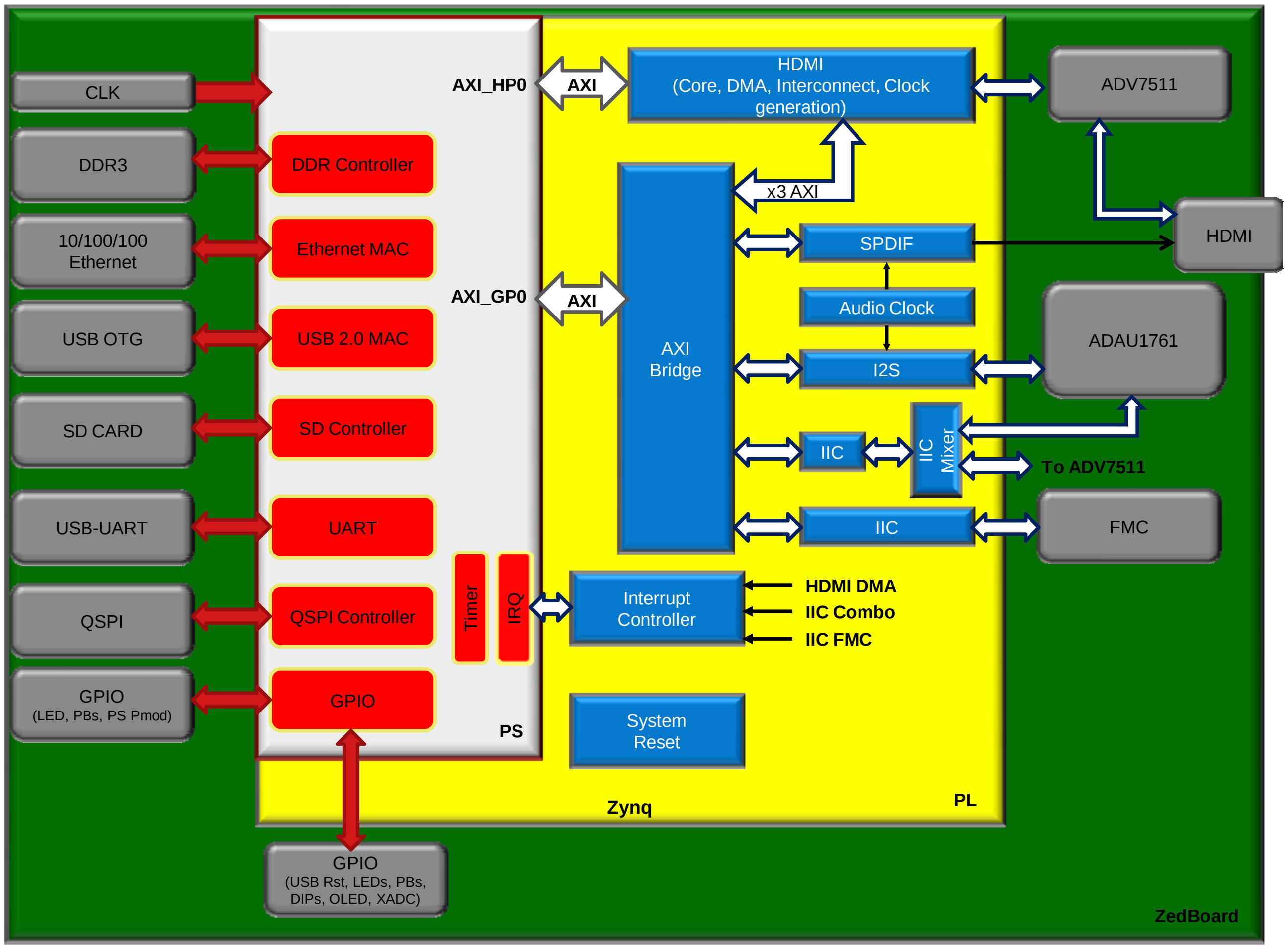


Figure 9 - ZedBoard Hardware Design Block Diagram

Create the Hardware Platform

This section provides the step by step procedure to create a Vivado Design Suite project from source, and subsequently build the design files and export the hardware platform to an SDK project. If you would rather not work with the hardware tools, you may skip to **Create the SDK Workspace** and use the SDK archive from the **Supplied Files**.

Create and Build the Vivado Hardware Project

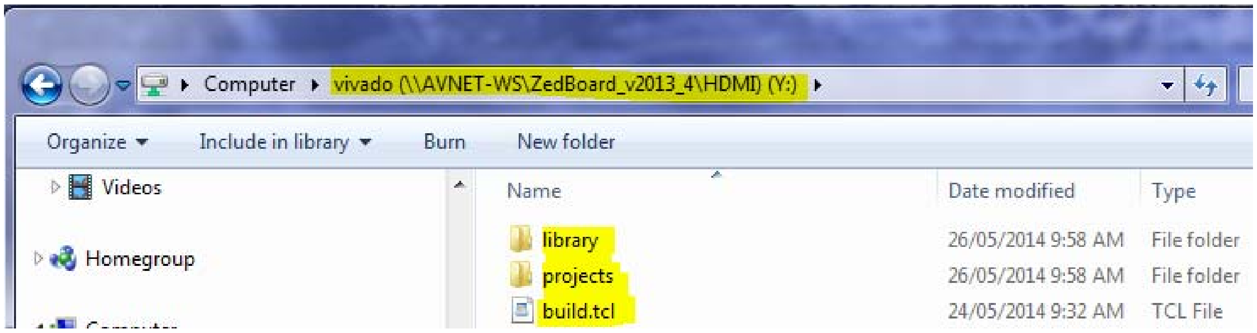
1. Use Windows Explorer to create a new, empty directory named **vivado**. You may create this directory anywhere, but in example it is located at:

   C:\ZedBoard_V2013_4\HDMI\vivado

   Caution: When choosing your own path in Windows, use caution in the path length as all files in the project hierarchy must fit within the Windows OS limitation of 260 characters. See http://www.xilinx.com/support/answers/52787.html.

   In this example, the target directory shown above was mapped to Windows Drive Letter **Y:/**.

2. From the **Supplied Files**, copy the contents of the **vivado** directory to the new empty **vivado** directory you just created.

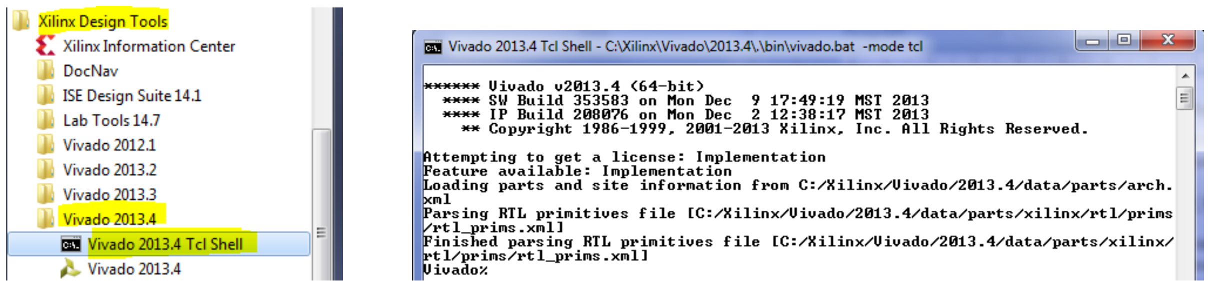


Name	Date modified	Type
library	26/05/2014 9:58 AM	File folder
projects	26/05/2014 9:58 AM	File folder
build.tcl	24/05/2014 9:32 AM	TCL File

Figure 10 - Initial Mapped Vivado Directory

3. From the Windows Start button, launch the *Vivado Tcl Shell*.

   Windows Start | All Programs | Xilinx Design Tools | Vivado 2013.4 | Vivado 2013.4 Tcl Shell



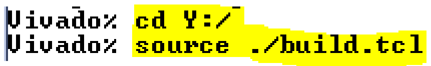
```
****** Vivado v2013.4 (64-bit)
  **** SW Build 353583 on Mon Dec  9 17:49:19 MST 2013
  **** IP Build 208076 on Mon Dec  2 12:38:17 MST 2013
    ** Copyright 1986-1999, 2001-2013 Xilinx, Inc. All Rights Reserved.

Attempting to get a license: Implementation
Feature available: Implementation
Loading parts and site information from C:/Xilinx/Vivado/2013.4/data/parts/arch.
xml
Parsing RTL primitives file [C:/Xilinx/Vivado/2013.4/data/parts/xilinx/rtl/prims
/rtl_prims.xml]
Finished parsing RTL primitives file [C:/Xilinx/Vivado/2013.4/data/parts/xilinx/
rtl/prims/rtl_prims.xml]
Vivado%
```

Figure 11 - Launch the Vivado Tcl Shell

4. In the *Vivado Tcl Shell*, change to the **vivado** directory (or the mapped drive), then execute the Tcl script **build.tcl** to create and build the hardware platform.

   cd Y:/
   source ./build.tcl



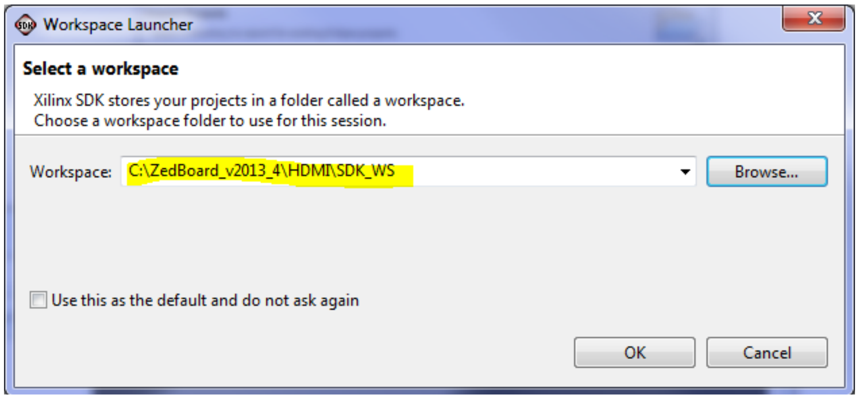
```
Vivado% cd Y:/
Vivado% source ./build.tcl
```

Figure 12 - Run the Tcl Build Script

The Tcl script will generate the Vivado project, synthesize and implement the source files and generate a bitstream that can be imported into an SDK project in the next section.

Import the Hardware Platform into the SDK

1. Launch the Xilinx SDK 2013.4 and open a new, blank workspace. This can be done in any available folder on your host, with the caveat that there can be no spaces in the pathname. For this example, the path selected is **C:\ZedBoard_v2013_4\SDK_WS.** Click the **OK** button.



Workspace Launcher

Select a workspace

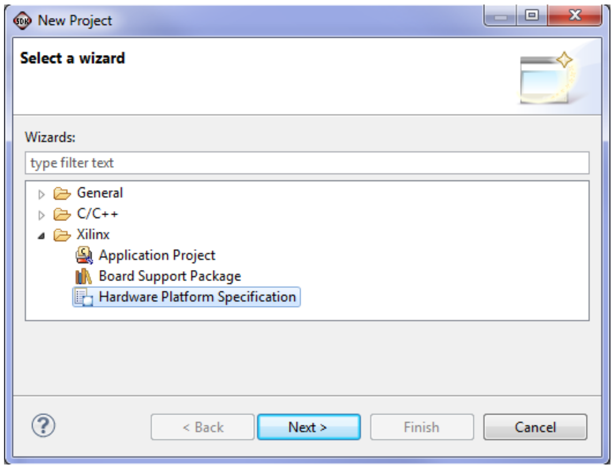
Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\ZedBoard_v2013_4\HDMI\SDK_WS

Use this as the default and do not ask again

Figure 13 – SDK Workspace Path

2. Close the Welcome screen. From the main menu, select **File -> New -> Project**. Expand the *Xilinx* entry, select **Hardware Platform Specification** and click the **Next** button.



New Project

Select a wizard

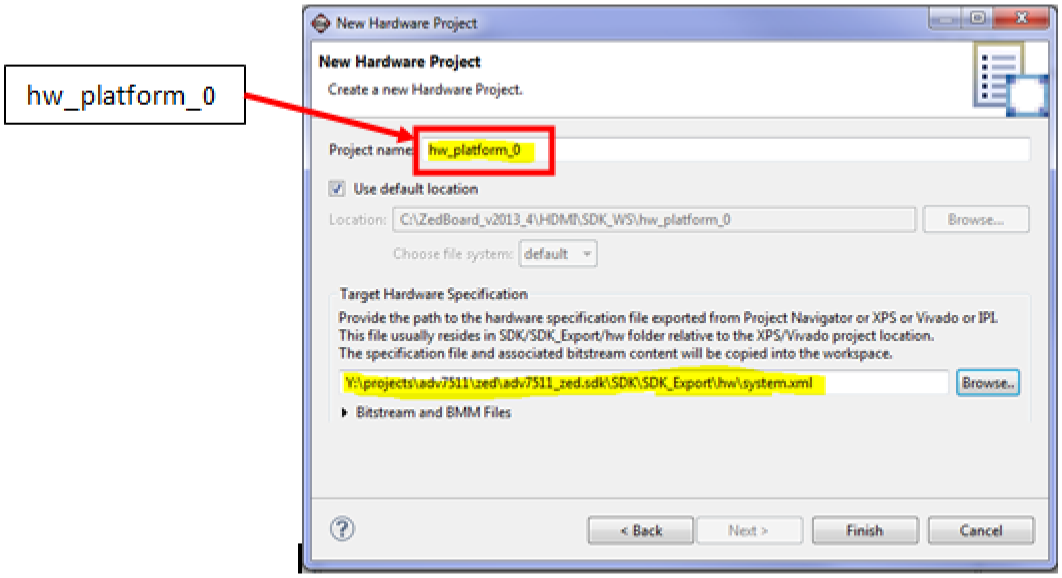
Wizards:
type filter text

Figure 14 – Create New Hardware Project

3. Name the project **hw_platform_0**. This is critical since the BSP and applications to be imported later map to this name. For the Target Hardware Specification, browse to the directory containing the hardware platform source generated by the Tcl script. For the mapped directory Y:/ used in this example, the source files are created in:

   Y:\projects\adv7511\zed\adv7511_zed.sdk\SDK\SDK_Export\hw

   Select the **system.xml** file and click the **Open** button to populate the Target Hardware Specification.



New Hardware Project

Create a new Hardware Project.

Project name: hw_platform_0

Use default location

Location: C:\ZedBoard_v2013_4\HDMI\SDK_WS\hw_platform_0

Choose file system: default

Target Hardware Specification

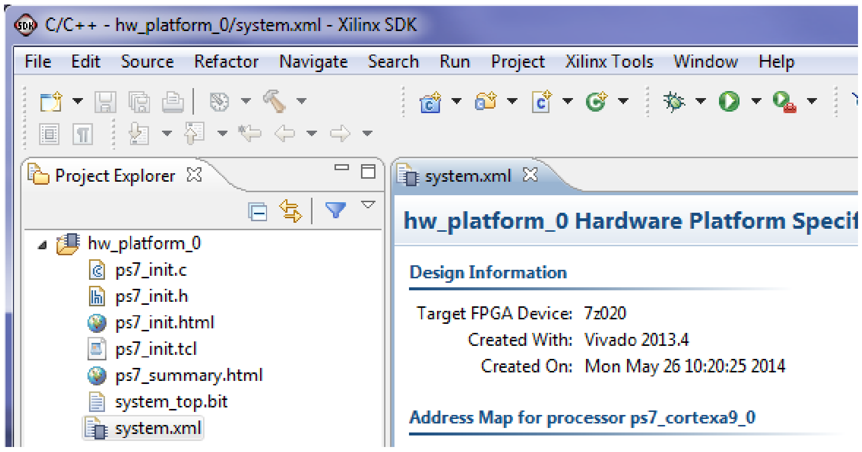
Provide the path to the hardware specification file exported from Project Navigator or XPS or Vivado or IPI.
This file usually resides in SDK/SDK_Export/hw folder relative to the XPS/Vivado project location.
The specification file and associated bitstream content will be copied into the workspace.

Y:\projects\adv7511\zed\adv7511_zed.sdk\SDK\SDK_Export\hw\system.xml

Bitstream and BMM Files

Figure 15 – Select the Hardware Platform

4. Click the **Finish** button to import the hardware platform.



hw_platform_0
- ps7_init.c
- ps7_init.h
- ps7_init.html
- ps7_init.tcl
- ps7_summary.html
- system_top.bit
- system.xml

hw_platform_0 Hardware Platform Specif

Design Information

Target FPGA Device: 7z020
Created With: Vivado 2013.4
Created On: Mon May 26 10:20:25 2014

Address Map for processor ps7_cortexa9_0

Figure 16 – Imported Hardware Platform

The SDK workspace is now ready for application projects. You have already created an SDK workspace, so you may skip to *Step 3* in the next section.

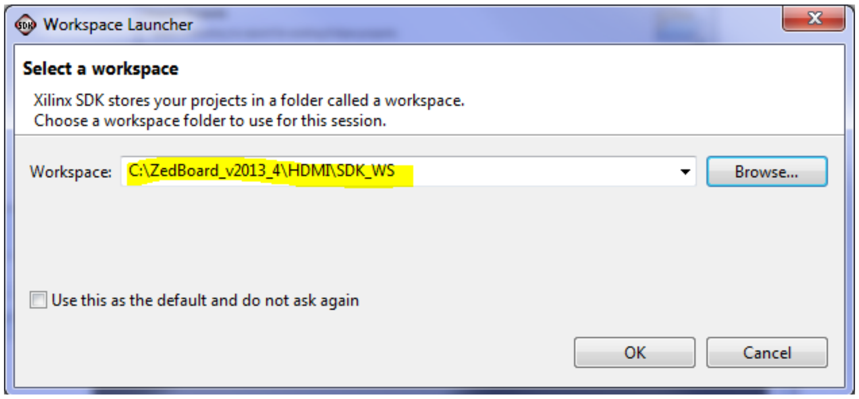
Create the SDK Workspace

This section provides the steps for importing SDK projects from **Supplied Files,** and running the vHDMI application on ZedBoard using a basic download procedure. The vHDMI application project contains custom software to exercise the HDMI transmitter from Analog Devices, which cannot be generated from the standard application templates included in the SDK.

This document assumes a basic familiarity with the operation and features of the Xilinx SDK. If you are unfamiliar with this tool, it is recommended that you consult the Avnet Speedway: *Developing Zynq Software with the Xilinx SDK* (see **Appendix I: Where to Get More Information**).

Import the SDK Projects

1. Launch the Xilinx SDK 2013.4 and open a new, blank workspace. This can be done in any available folder on your host, with the caveat that there can be no spaces in the pathname. For this example, the path selected is **C:\ZedBoard_v2013_4\SDK_WS.** Click the **OK** button.



Select a workspace

Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\ZedBoard_v2013_4\HDMI\SDK_WS

Use this as the default and do not ask again

Figure 17 – SDK Workspace Path

2. Close the *Welcome* screen.

3. From the main menu, select **File -> Import**. Expand the *General* entry, select **Existing Projects into Workspace** and click the **Next** button.

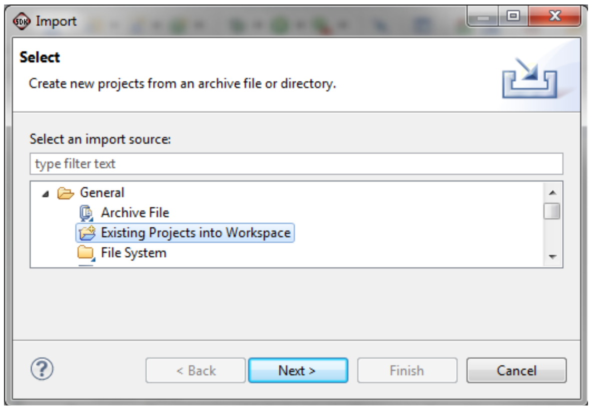
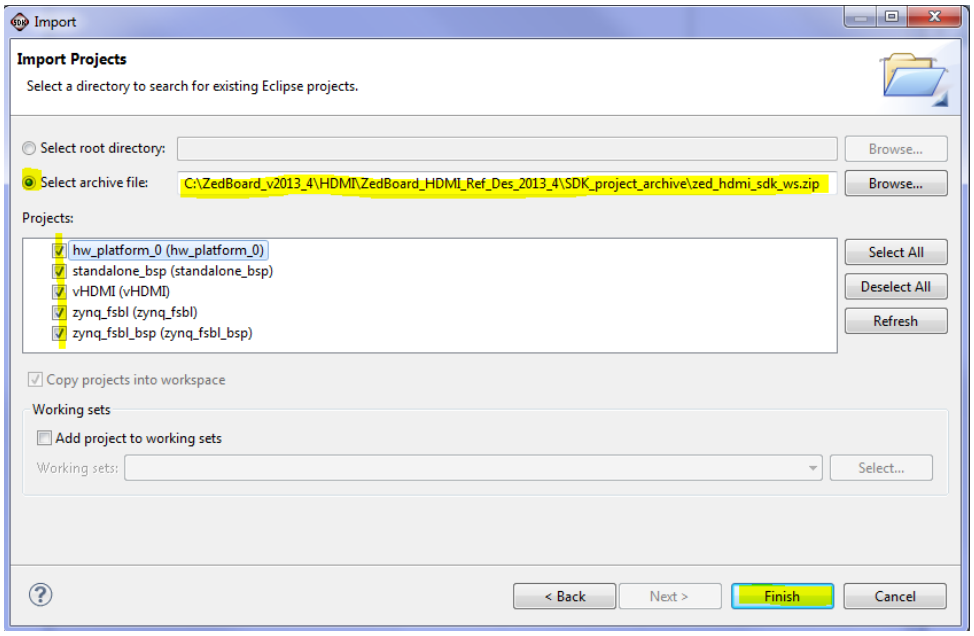


Figure 18 – Import Archive File

4. Click the radio button for **Select archive file**. Browse to the *SDK_project_archive* folder in your **Supplied Files** installation and select the compressed workspace file.

<Installation folder>\SDK_project_archive\zed_hdmi_sdk_ws.zip


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory:

Select archive file: C:\ZedBoard_v2013_4\HDMI\ZedBoard_HDMI_Ref_Des_2013_4\SDK_project_archive\zed_hdmi_sdk_ws.zip

Projects:

- [✓] hw_platform_0 (hw_platform_0)
- [✓] standalone_bsp (standalone_bsp)
- [✓] vHDMI (vHDMI)
- [✓] zynq_fsbl (zynq_fsbl)
- [✓] zynq_fsbl_bsp (zynq_fsbl_bsp)

[✓] Copy projects into workspace

Working sets

[ ] Add project to working sets

Working sets:

Figure 19 – Select SDK Projects from Archive

All projects should be selected by default. If they are not, click the **Select All** button[3]. Click the **Finish** button to import the projects to your new workspace. All projects will compile automatically, and should do so with no errors. Click the *Console* tab to view the compilation progress.

[3] If you built the hardware platform with the Tcl script in *Create the Hardware Platform*, you will already have the *hw_platform_0* project and it will not be selectable here.

5. Your workspace will now contain the following projects:

    a. **hw_platform_0**: The elements of the Vivado hardware platform exported to the original SDK workspace, on which the bare metal (standalone) BSP is based.

    b. **standalone_bsp**: The board support package required by all non-OS applications for the hardware platform included in this workspace. This BSP is automatically generated by the SDK when the first application project is created, following a successful export of the hardware platform from Vivado.

    c. **vHDMI**: The bare metal HDMI demonstration/test application, provided by Analog Devices. See **Appendix I: Where to Get More Information**.

    d. **zynq_fsbl**: The first stage boot loader application for this hardware platform. The first stage bootloader ELF is one of three files (also the bitstream and application ELF) used to create a *boot.bin* file executed from the SD card on power-up. This application can be automatically generated using the application template provided within the SDK.

    e. **zynq_fsbl_bsp**: The board support package for the first stage bootloader. Beginning with the 2013.4 version of the SDK, the FSBL BSP requires the inclusion of a FAT file system when booting from an SD card. This is the only difference between the FSBL BSP and the bare metal BSP. The FSBL BSP is automatically generated when the FSBL application is created.

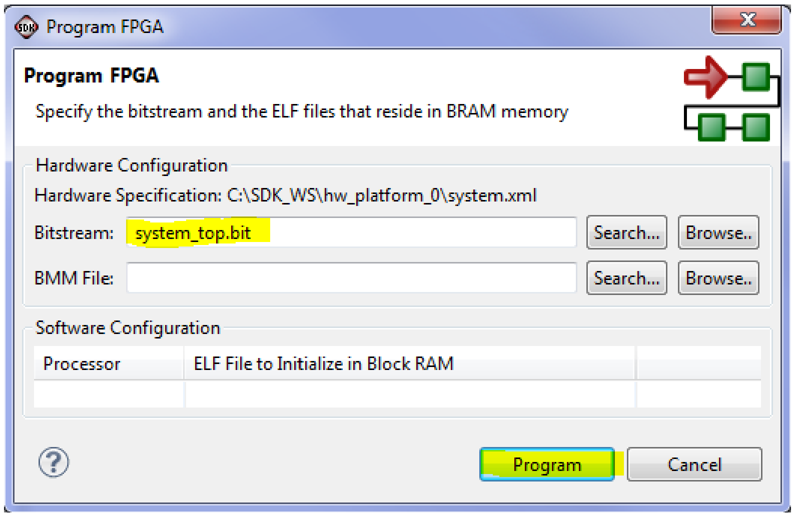
Run/Debug the vHDMI Application via JTAG

The procedure for using JTAG to download applications to the board for the purpose of execution or debugging is described briefly in this section. This example uses the vHDMI application, but the download procedure is similar for all application projects. Ensure you have the HDMI cable connected between the target and your monitor, and that the monitor is switched on.

Your hardware should be connected as described in **Setting up the ZedBoard Development Kit** and the mode jumpers should all be in the **off** position.

Ensure you have a USB cable connected between the host computer and the JTAG port on the target.

1. Apply power to ZedBoard. Program the SoC by selecting the icon from the SDK task bar and accepting the default bit stream.



Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Specification: C:\SDK_WS\hw_platform_0\system.xml

Bitstream: system_top.bit

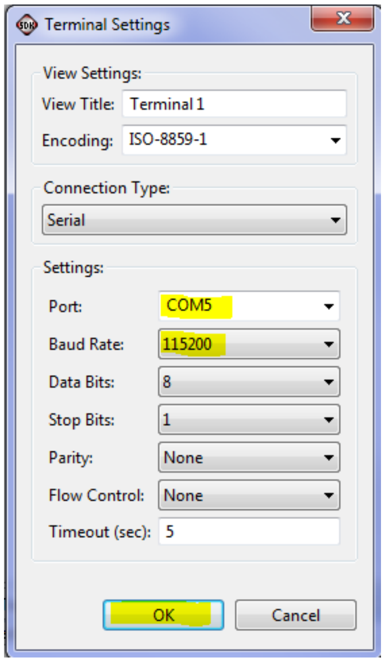
BMM File:

Software Configuration

Processor	ELF File to Initialize in Block RAM

Figure 20 – Bitstream Selection

2. Select the Terminal 1 tab, and access the Settings panel by clicking on the icon. Select the *Connection Type* as **Serial**, the *Baud Rate* as **115200**, and select the *Port* that was automatically used by the Cypress USB-to-UART driver when the FPGA was loaded in the previous step. In this example, the port is **COM5**. Click the **OK** button.



View Settings:

View Title: Terminal 1

Encoding: ISO-8859-1

Connection Type:

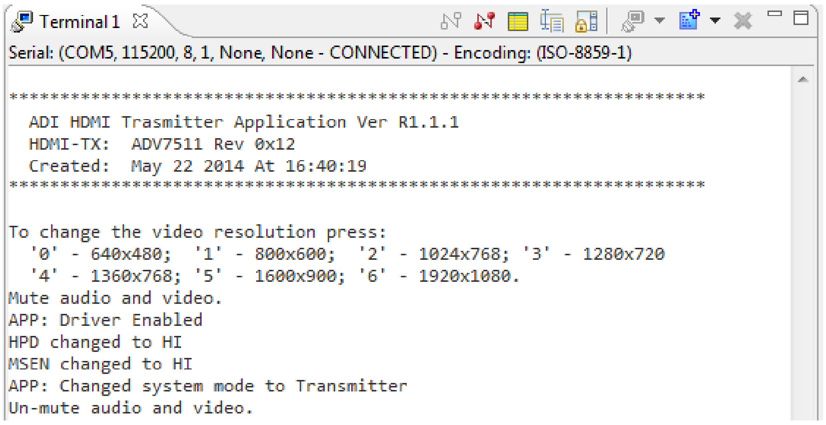
Serial

Settings:

Port:	COM5
Baud Rate:	115200
Data Bits:	8
Stop Bits:	1
Parity:	None
Flow Control:	None
Timeout (sec):	5

Figure 21 – Set the SDK Terminal Parameters

3. In the Project Explorer tab of the SDK, right-click on the **vHDMI** project and **select Run As -> Launch on Hardware (GDB)**. In a few seconds the application will download and the output will appear in the **Terminal 1** tab. You may need to select the **Terminal 1** tab to switch from the Console[4], or you can drag the Terminal tab up to the main source panel in the SDK so that it is always on top.



```
Serial: (COM5, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)

********************************************************************
  ADI HDMI Trasmitter Application Ver R1.1.1
  HDMI-TX:  ADV7511 Rev 0x12
  Created:  May 22 2014 At 16:40:19
********************************************************************

To change the video resolution press:
  '0' - 640x480;  '1' - 800x600;  '2' - 1024x768; '3' - 1280x720
  '4' - 1360x768; '5' - 1600x900; '6' - 1920x1080.
Mute audio and video.
APP: Driver Enabled
HPD changed to HI
MSEN changed to HI
APP: Changed system mode to Transmitter
Un-mute audio and video.
```

Figure 22 – vHDMI Test Output in SDK Terminal

The output should be identical to the vHDMI Test executed from the SD card in the demonstration section[5]. You can create a boot image using the projects included in this workspace and the **Xilinx Tools -> Create Zynq Boot Image** menu item. If you need instructions for this, consult the Avnet *Developing Zynq Software with the Xilinx SDK Speedway*.

4. If you would prefer to debug the application, right-click **vHDMI** and **select Debug As -> Debug Configurations**. In the *Debug Configurations* panel, select the **Xilinx C/C++ application (System Debugger)** and click the icon at the upper left to create a new configuration.

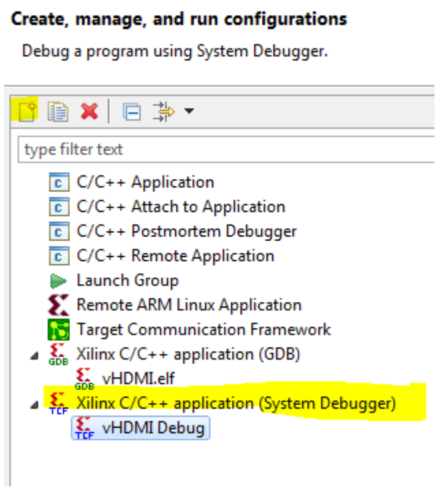
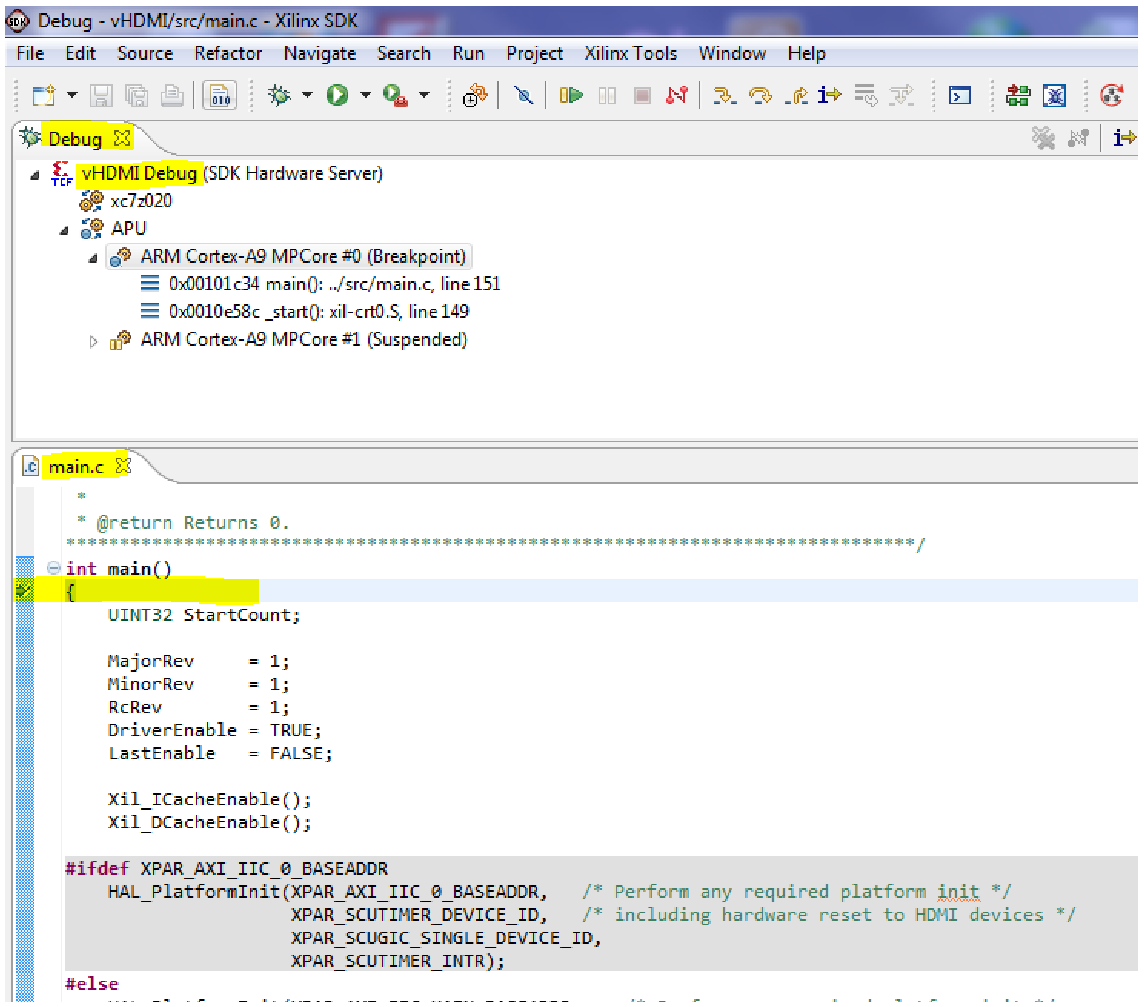

Create, manage, and run configurations
Debug a program using System Debugger.
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
Launch Group
Remote ARM Linux Application
Target Communication Framework
Xilinx C/C++ application (GDB)
vHDMI.elf
Xilinx C/C++ application (System Debugger)
vHDMI Debug

Figure 23 – Create New System Debugger Configuration

[4] If you wish, you may use a serial terminal program such as Tera Term instead of the SDK Terminal tab.
[5] If you do not see any output in your serial terminal, try toggling the connection and run the application again. Also ensure you have connected to the correct COM port.

5. Click the **Debug** button. Click **Yes** to confirm a switch to the Debug Perspective[6]. The Application will launch in the symbolic debugger and stop automatically at the first executable statement. You may use the standard debug controls to operate the application in debug mode. If you need instructions on how to do this, please reference the Avnet *Developing Zynq Software with the Xilinx SDK Speedway.*

   Serial output will be available under the **Terminal 1** tab.


Debug - vHDMI/src/main.c - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
Debug
main.c

```
vHDMI Debug (SDK Hardware Server)
    xc7z020
    APU
        ARM Cortex-A9 MPCore #0 (Breakpoint)
            0x00101c34 main(): ../src/main.c, line 151
            0x0010e58c _start(): xil-crt0.S, line 149
        ARM Cortex-A9 MPCore #1 (Suspended)
```

```
 *
 * @return Returns 0.
*****************************************************************************/
int main()
{
    UINT32 StartCount;

    MajorRev     = 1;
    MinorRev     = 1;
    RcRev        = 1;
    DriverEnable = TRUE;
    LastEnable   = FALSE;

    Xil_ICacheEnable();
    Xil_DCacheEnable();

#ifdef XPAR_AXI_IIC_0_BASEADDR
    HAL_PlatformInit(XPAR_AXI_IIC_0_BASEADDR,   /* Perform any required platform init */
                     XPAR_SCUTIMER_DEVICE_ID,   /* including hardware reset to HDMI devices */
                     XPAR_SCUGIC_SINGLE_DEVICE_ID,
                     XPAR_SCUTIMER_INTR);
#else
```

Figure 24 – Application Debug

This concludes the ZedBoard Bare Metal HDMI Reference Design. You may close all open applications and turn off power to your ZedBoard.

[6] HINT: Click the checkbox to remember your decision to avoid the pop-up box in the future.

Appendix I: Where to Get More Information

ZedBoard.org

ZedBoard documentation page
http://www.zedboard.org/documentation/1521

ZedBoard reference designs and tutorials
http://www.zedboard.org/design/1521/11

Avnet Speedway: Developing Zynq Software with the Xilinx SDK
http://www.zedboard.org/support/trainings-and-videos

Ubuntu on the Zynq-7000 SoC Tutorial
http://www.zedboard.org/design/1521/11

ZedBoard Forum Support
http://www.zedboard.org/forums/zedboard-hardware-design

ADI

ADV7511 Product Page
http://www.analog.com/en/audiovideo-products/analoghdmidvi-interfaces/adv7511/products/product.html

ADAU1761 Product Page
http://www.analog.com/en/audiovideo-products/audio-signal-processors/adau1761/products/product.html

Wiki Instruction page
http://wiki.analog.com/resources/fpga/docs/hdl
http://wiki.analog.com/resources/fpga/docs/hdl/vivado
http://wiki.analog.com/resources/fpga/xilinx/kc705/adv7511 (bare metal software)

ADI HDL Reference Designs HDL User Guide
http://wiki.analog.com/resources/fpga/docs/hdl/github

ADI Support
http://ez.analog.com/community/fpga

ADV7511 Xilinx Evaluation Boards Reference Designs (ZedBoard)
http://wiki.analog.com/resources/fpga/xilinx/kc705/adv7511

Xilinx Website

Xilinx Design Tools Installation and Licensing Guide
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_4/ug973-vivado-release-notes-install-license.pdf

Zynq
http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm

Xilinx Silicon and Tools Forum Support
http://forums.xilinx.com/

Other Internet Resources

http://www.hdmi.org/
http://en.wikipedia.org/wiki/HDMI

Revision History

Version	Date	Author	Details
1.0	June 5, 2014	Avnet	First Release, Vivado/SDK 2013.4