Creating and Executing an AXI Central Direct Memory Access (CDMA) Design on the Zedboard

Table 1: Revision History

Revision	Description of Change	By	Date
1.01			

Contents

1 Introduction

1.1 Purpose

An effort was undertaken to create a design that runs data over the AXI bus on the Zedboard. The purpose of this document is to define repeatable steps to complete the following objectives:

1. Create a hardware design to target a Zynq chip.
2. Import and customize a linux-based software application to run with the hardware design.
3. Build the components necessary to boot the Zynq chip with Petalinux and appropriate hardware drivers.
4. Download and run the hardware and software design on the Zedboard.

1.2 Required Hardware and Software

The following hardware and software were used to create and test the benchmarking design. Refer to table 2 for more information on these products.

- Avnet Zedboard
- Xilinx Platform Cable USB II
- Cypress USB-uart cable
- Unix machine running Centos 7
- Xilinx Vivado version 2015.3
- Xilinx SDK version 2015.3
- Xilinx UG1165 supporting resources

Title	**Link**
Avnet Website	https://avnetexpress.avnet.com/index.html
Xilinx USB II Cable	http://www.xilinx.com/products/boards-and-kits/hw-usb-ii-g.html
Cypress USB-UART Cable Cable	http://www.cypress.com/products/usb-uart-controller-gen-2
Xilinx Vivado Design Suite	http://www.xilinx.com/support/download.html
Required Repos	http://www.wiki.xilinx.com/Fetch+Sources
Xilinx UG1165 zip file	https://secure.xilinx.com/webreg/clickthrough.do?cid=383986

Table 2: Table of Reference Sites

2 Vivado Design

Vivado is used to create a hardware design to target the Zynq chip on the Zedboard platform. The current design uses a PL-based CDMA to write and read data to/from the DDR via the high performance (HP) AXI ports. The AXI general purpose (GP) port will be used with an AXI lite interface to configure the CDMA. This application mimics a Xilinx project, UG1165, but targets the Zedboard instead of the Xilinx Evaluation Board, XC702.

2.1 Create a new Project in Vivado

1. Open a new Linux terminal window and set up the environment:

   ```
   $export CROSS_COMPILE=arm-xilinx-linux-gnueabi
   ```

2. Start up a new Vivado project running the following command in the Linux shell.

   ```
   $vivado&
   ```

3. In the Quick Start screen, click on "Create New Project"
4. In the New Project Wizard, click "Next"
5. Type in the project name and location. For this effort, the project name is "pcie_benmarking" and the location is /home/devel/. Leave the box marked "Create project subdirectory" checked. Then click "Next"
6. Leave the circle marked "RTL Project" checked and companion box unchecked. Click "Next"
7. On the "Add Sources" screen, click "Next"
8. On the "Add Existing IP" screen, click "Next"
9. On the "Add Constraints" screen, click "Next"
10. On the "Default Part" screen, click on the "Boards" Icon, select "Zedboard Zynq Evaluation and Development Kit", then click "Next"
11. On the "New Project Summary" Page, click "Finish"

2.2 Invoke IP Integrator

1. In the "Flow Navigator" pane, under the "IP Integrator", click on "Crete Block Diagram"
2. Leave all options as they are in "Create Block Design" window, and click "ok"

2.3 Customize CDMA

1. In the "Diagram" window, right click and select "Add IP..."
2. In the Dialog Box, search for "CDMA" and double click on "AXI Central Direct Memory Access"
3. Double click on the CDMA block and make the following changes:
   - Uncheck "Enable Scatter Gather"
   - Change "Write/Read Data Width" to 1024
   - Change "Write/Read Burst Size" to 32
4. Click "ok"

2.4 Customize ZYNQ Processing System

1. In the "Diagram" window, right click and select "Add IP..."
2. In the Dialog Box, search for "zynq" and double click on "ZYNQ7 Processing System"
3. Double click on the Zynq block and make the following changes:
   - Click on the "Presets" icon and select "ZedBoard"
   - Under PS-PL Configuration, click on HP Slave AXI Interface, and check the boxes for "S AXI HP0 interface" and "S AXI HP2 interface"
   - Under PS-PL Configuration, click on AXI Non Secure Enablement, and then on GP Master AXI Interface, and check the box for "M AXI GP0 interface"
   - Under Clock Configuration, change the CPU clock to "50" (MHz)
   - Under Interrupts, check the "Fabric Interrupts" box, select PL-PS Interrupt Ports, "check IRQ_F2P".
4. Click "ok"

2.5 Customize Processor System Reset

It is recommended to instantiate a Processor System Reset IP block to support the ZYNQ Processing System.

1. In the "Diagram" window, right click and select "Add IP..."
2. In the Dialog Box, search for "reset" and double click on "Processor System Reset"
3. Double click on the block and change both "Ext Reset Active Width" and "Aux Reset Active Width" to 1.
4. Click "ok"

2.6 Customize Clock Buffer

1. In the "Diagram" window, right click and select "Add IP..."
2. In the Dialog Box, search for "buffer" and double click on "Utility Buffer"
3. Double click on the block and select "BUFG".
4. Click "ok"

2.7 Customize Axi Interconnects

1. In the "Diagram" window, right click and select "Add IP..."
2. In the Dialog Box, search for "interconnect" and double click on "Axi Interconnect"
3. Select the block that appears, and copy-paste so that two interconnect blocks are on the screen.
4. Double click the first block, and change "Number of Master Interfaces" to 1.
5. Double click the second Axi Interconnect block, and ensure "Number of Master Interfaces" is 2 and "Number of Slave Interfaces" is 1.

2.8 Connect the Clocking

1. Connect FCLK_CLK0 of the Processing System block to BUFG_I[0:0] of the Utility Buffer.
2. Connect BUFG_O[0:0] of the Utility Buffer to the following:
   (a) slowest_sync_clk of Processor System Reset
   (b) ACLK[0:0], S00_ACLK[0:0], and M00_ACLK[0:0] of the first AXI Interconnect block.
   (c) m_axi_aclk and s_axi_lite_aclk of AXI Central Direct Memory Access block.
   (d) ACLK[0:0], S00_ACLK[0:0], M00_ACLK[0:0], and M01_ACLK[0:0] of the second AXI Interconnect block.
   (e) M_AXI_GP0_ACLK, S_AXI_HP0_ACLK, and S_AXI_HP2_ACLK of ZYNQ7 Processing System

2.9 Connect the Resets and Interrupt

1. Connect FCLK_RESET0_N from ZYNQ7 Processing System to ext_reset_in of Processor System Reset.
2. Connect interconnect_aresetn[0:0] to the following:
   (a) ARESETN[0:0], S00_ARESETN[0:0], and M00_ARESETN[0:0] of the first AXI Interconnect block.
   (b) s_axi_lite_aresetn of the AXI Central Direct Memory Access block
   (c) ARESETN[0:0], S00_ARESETN[0:0], M00_ARESETN[0:0] and M01_ARESETN[0:0] of the second AXI Interconnect block.

3. Connect cdma_introut of the AXI CDMA block to IRQ_F2P[0:0] of the ZYNQ7 Processing System.

2.10 Connect the Data Paths

1. Connect M00_AXI of the first AXI Interconnect block to S_AXI_LITE of the AXI CDMA block.
2. Connect M_AXI of the AXI CDMA block to S00_AXI of the second AXI Interconnect block.
3. Connect M00_AXI of the second AXI Interconnect block to S_AXI_HP0 of the ZYNQ7 Processing System block.
4. Connect M01_AXI of the second AXI Interconnect block to S_AXI_HP2 of the ZYNQ7 Processing System block.
5. Connect M_AXI_GP0 of the ZYNQ7_Processing System block to S00_AXI of the first AXI Interconnect block.

2.11 Connect External Ports

1. At the top of the block diagram, there should be a bar that reads "Designer Assistance Available". Click on "Run Block Automation"
2. In the menu that appears, make sure "processing_system7_0 is checked. Note under "Options", that the automation will make the FIXED_IO and DDR ports external. Make sure "Apply Board Preset" is checked. Click "OK".

2.12 Modifying Addresses

1. Select the Address tab, which is near the top left corner of the IP Integrator block diagram.
2. Expand the axi_cdma_0 line to see two cells with base names HP0_DDR_LOWOCM and HP2_DDR_LOWOCM. These are the two memory locations at which the AXI ports can access the DDR for writing and reading.
3. Click on Offset Address. Enter 0x10000000 as the address for HP0_DDR_LOWOCM and 0x18000000 of HP2_DDR_LOWOCM.
4. Change the Range of both HP0_DDR_LOWOCM and HP2_DDR_LOWOCM to 128M.

2.13 Finalizing the Hardware Design

The final design appears in Figure 1

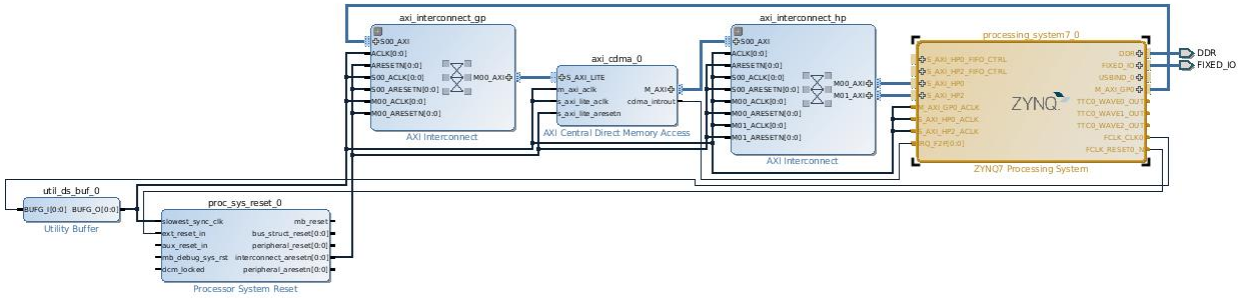

Figure 1: Worker Control Interface - Separation of Concerns

1. Along the main Vivado toolbar, there is a yellow box icon with a green checkmark. Click this icon to validate the design. When the "Validation Successful" dialogue block opens, click ok.

2. In the Sources tab/window, expand Design Sources, right-click on the top level design name, and select "Create HDL Wrapper".

3. In the dialogue box that appears, select "Let Vivado manage wrapper and auto-update" and then click ok.

4. In the Sources tab/window, right-click in the top level *.bd file, and select "Generate Output Products". In the dialogue box that opens, click "Generate"

2.14 Generating Bitstream and Exporting to SDK

1. In the Flow Navigator pane of Vivado, under Program and Debug, click on Generate Bitstream.

2. A pop-up window will ask whether or not to perform synthesis and implementation first. Click "yes". Bitstream generation can take 10 minutes or so, depending on design and computer resources.

3. Once bitstream generation is complete, a "Bitstream Generation Completed" window will appear. Select "View Reports" to see implementation results such as device utilization, place and route details, etc.

4. To export the hardware design to SDK, click on the File menu in the upper left corner of the Vivado screen. Navigate to Export, and click on Hardware.

5. In the Export Hardware window that appears, check the box marked "Include Bitstream". Keep the "Export to:" field as <Local to Project>, and click "ok". This will create a subdirectory in the vivado project folder which will contain the SDK workspace. The subdirectory will be called <project name>.sdk. Vivado will populate the subdirectory with initalization code and settings for the Zynq processor, DDR, clocks, PLLs and MIOs.

6. Note - If SDK does not automatically open, from the File menu, select "Launch SDK". Keep both the Exported location and Workspace as <Local to Project>. Click "ok".

3 Running an Application on Zedboard Baremetal

4 Running an Application on Zedboard with Linux

4.1 Creating the CDMA Application Project in SDK

Continuing from the end of section 2.14, SDK is now open. In the Project Explorer window, the hardware description file (*.hdf) generated from the vivado design can be seen as system.hdf under <vivado_design_name>_wrapper_hw_platform_0.

1. Select File $->$ New $->$ Application Project
2. In the dialogue box that appears, make the following selections:
   - Project Name: linux_cdma_app
   - Use Default Location: yes
   - OS Platform: linux
   - Hardware Platform: <vivado_design_name>_wrapper_hw_platform_0 (as described in subsection 2.1, item 5)
   - Processor: ps7_cortexa9_0 (first core of the dual core A9 processor)
   - Language: C
   - Board Support Package: Create New: linux_cdma_app_bsp
3. Click Next.
4. Under Avalable Templates, select Empty Application, and select "Finish"
5. Two new items will appear in the project explorer: linux_cdma_app and linux_cdma_app_bsp.
6. Right click on the src directory under the linux_cdma_app, and select Import.
7. Click on General to expand, and select File System. Click Next.
8. Browse to the unzipped UG1165 support files, and select "linux_cdma_app.c". Add the file, and click Finish. Temporarily ignore any warnings or errors.
9. Open cdma_app.c in an editor and make the following changes:
   (a) Include a missing library
   ```
   #include <unistd.h>
   ```
   (b) Change CDMA_BASE_ADDRESS to match offset address of axi_cdma_0 from vivado address editor
   ```
   #define CDMA_BASE_ADDRESS 0x7e200000
   ```
   (c) Change DDR_BASE_ADDRESS to match offset address of S_AXI_HP0*/
   ```
   #define DDR_BASE_ADDRESS 0x10000000
   ```

(d) Change DDR_BASE_WRITE_ADDRESS to match offset address of S_AXI_HP2*/

```
#define DDR_BASE_WRITE_ADDRESS      0x18000000
```

(e) Change DDR_MAP_SIZE to match size of S_AXI_HP0 and S_AXI_HP2*/

```
#define DDR_MAP_SIZE 0x5000000
```

(f) Halve BUFFER_BYTE_SIZE

```
#define BUFFER_BYTESIZE 131072
```

10. The application may still fail to compile or link due to a current error in the way Vivado populates the SDK settings when it exports the design. If this is the case, review compiler and linker scripts. Right click on linux_cdma_app and go to Properties –> C/C++ build –> Settings –> ARM Linux gcc compiler/linker. Some of the options under the field "All options" may need to be removed.

4.2 Repos and Downloads

The following repos are required to build the linux kernal and run the design on hardware. See table 2 for more information.
Download these repos into the top level of the Xilinx directory on the Linux machine. For the git repos, using the following command:

```
$git clone https://path.to.repo.git
```

linux-xlnx.git

https://github.com/Xilinx/linux-xlnx.git. This repo contains the Xilinx release of the Linux kernel. Specifically, uImage is needed for both SD card booting and JTAG booting.

device-tree.git

https://github.com/Xilinx/device-tree-xlnx.git. This repo will be a necessary inclusion to the SDK environment in order to generate the customized device tree (devicetree.dts) for the zedboard.

dtc.git

https://git.kernel.org/pub/scm/utils/dtc/dtc.git. For this effort, this repo will only be used to convert devictree.dts to system.dtb, a recognizable u-boot format.

2015.4-zed-release

http://www.xilinx.com/Zynq+2015.4+Release - this download contains files necessary for a basic zedboard booting. This effort uses only uramdisk.image.gz which is the zedboard-customized file system image.

4.3 Environment

Set the environment as follows. The provided path values assume the git repos are installed at /opt/Xilinx.

```
$export CROSS\_COMPILE="arm-xilinx-linux-gnueabi-"

$export PATH="/opt/Xilinx/dtc:$PATH"

$export PATH="/opt/Xilinx/u-boot-xlnx/tools:$PATH"
```

4.4 Creating Zedboard Boot Files

These are the files necessary to boot and run a Linux Application on the Zedboard. Creation and utilization of these items will be described in the sections that follow.

1. **Hardware Bit File** - design.bit
2. **First Stage Boot Loader** - fsbl.elf
3. **File System Image** - uramdisk.image.gz
4. **Second Stage Boot Loader** - u-boot.elf
5. **Linux Kernel Image** - uImage
6. **Device Tree** - devicetree.dtb

4.4.1 Hardware Bit File

The Hardware Bit File, design.bit was created in subsection 2.14. It was exported to the Xilinx SDK environment, and can be found at <Vivado Project Location>/<Vivado Project Name>.sdk/<Vivado Block Design Name>_wrapper_hw_platform_0/<Vivado Block Design Name>.bit

4.4.2 First Stage Boot Loader

The first stage bootloader is customizable for the specific platform, chip and peripherals being used. In this effort it is created within Xilinx SDK. It will load the Zynq chip and initialize it based upon the hardware design.

1. In the SDK environment, within the same workspace as begun in subsection 4.1, select New –> Application Project.
2. In the dialogue box that appears, make the following selections before clicking Next:
   (a) Project Name: fsbl
   (b) Use Default Location: yes
   (c) OS Platform: standalone
   (d) Hardware Platform: <vivado_design_name>_wrapper_hw_plaform_0 (as described in subsection 2.1, item 5)

(e) Processor: ps7_cortexa9_0

(f) Language: C

(g) Board Support Package: Create New: fsbl_bsp

3. In the next page of the dialogue box, select Zynq FSBL Template, and click Finish. **fsbl.elf** is created under fsbl/binaries.

4.4.3 File System Image

In this effort, the rootfs (root filesystem) did not need to be rebuilt from scratch. A standard rootfs is included in the 2015.4-zed-release archive downloaded in subsection 4.2. It is found in 2015.4_zed_release/zed/ and is called **uramdisk.image.gz**.

4.4.4 Second Stage Boot Loader

The SSBL is created using the git repo downloaded in subsection 4.2. It will load Linux onto the Zynq A9 Processor.

1. In a terminal window, cd to /opt/Xilinx/u-boot-xlnx. Make sure the environment is set up as described in subsection 4.3
2. Type the following and hit enter, to configure the make process for the Zedboard:

   ```
   $make zynq_zed_config
   ```

3. Type the following and hit enter, to rebuild u-boot for the Zedboard and to create the mkimage executable (can be used later to wrap u-boot headers).

   $make

   *Note - compiling u-boot also creates the mkimage utility, which will be used in subsection 4.4.5, item 4, to wrap the linux kernel in a format that the u-boot recognizes.

4. Rename u-boot to **u-boot.elf**

4.4.5 Linux Kernel Image

1. cd to /opt/Xilinx/linux-xlnx and
2. type the following to configure the kernel based on the contents of linux-xlnx/arch/arm/configs/xilinx_

   ```
   make ARCH=arm xilinx_zynq_defconfig
   ```

3. type the following for an interactive menu allowing additional kernel configuration:

   ```
   make ARCH=arm menuconfig
   ```

4. type the following to create the kernel:

   ```
   make ARCH=arm UIMAGE_LOADADDR=0x8000 uImage
   ```

   uImage will be created at /opt/Xilinx/linux-xlnx/arch/arm/boot.
   *Note - If uImage does not appear at this location, but Image and zImage is created, then the mkimage utility (in the u-boot-xlnx repo) has not been added to the build environment path as described in subsection 4.3.

4.4.6 Device Tree

The Device Tree identifies all of the hardware devices for the Linux kernel.

1. cd to /opt/Xilinx/dtc and type

   $make

2. Ensure the terminal window PATH points to dtc, as described in section 4.3.
3. In the SDK environment, continuing in the workspace from subsection 4.4.2, select from the main menu, Xilinx Tools –> Repositories.
4. In the dialogue box that appears, click "New" in the Local Repositories area.
5. Browse to the directory /opt/Xilinx/device-tree-xlnx, and click "ok".
6. From the main menu, select File –> New –> Board Support Package.
7. In the dialogue box that appears, ensure the following options are selected.
   (a) Project name: devie_tree_bsp
   (b) Use default loaction: checked
   (c) Target Hardware Platform: <Vivado_design_name>_wrapper_hw_platform_0
   (d) Target CPU: ps7_cortexa9_0
   (e) Board Support Package OS: device_tree
   *Note "device_tree" will only appear in the Board Support Package OS type list if the device-tree-xlnx repository was correctly added as described in list item 5 above.
8. Click "Finish".
9. System.mss will be created, and can be viewed in the Project Explorer pane under device_tree_bsp_0. The file can be modified by double-clicking it, and then clicking the button "Modify this BSP's Settings"
10. In the terminal window, cd to the path of system.mss, and type

    ```
    dtc -I dts -O dtb -o devicetree.dtb system.dts
    ```

    This converts the name and file type of system.dts to devicetree.dtb, which is necessary for u-boot to recognize the device tree upon loading.

4.5 Booting Zedboard With an SD Card and Running the Linux Application

4.5.1 Board Settings

1. The MIO bank on the Zedboard must have the following values to enable booting from SD card:
   - MIO6 GND
   - MIO5 3.3V

- MIO4 3.3V
- MIO3 GND
- MIO2 GND

2. Ensure the Platform USB Cable is connected between the host machine and the Zedboard JTAG port.
3. Ensure the UART cable is connected from the host machine to the Zedboard UART port.
4. Ensure the UART port is enabled on the Zedboard by shunting the JP3 pins.
5. Ensure the power cable is plugged into the board.
6. Ensure the ethernet cable is plugged into the board.

4.5.2 Creating BOOT.bin

The Zynq Boot Image is a compilation of elements already created in previous sections. In this effort, SDK is used to put the necessary components into BOOT.bin

1. In the SDK environment used in subsection 4.4, from the main menu, select Xilinx Tools –> Create Zynq Boot Image.
2. In the dialogue box that appears, select the following parameters:
   (a) Output File Path: path/to/<Vivado_design_name>_wrapper_hw_platform_0
   (b) Use Authentication: unchecked
   (c) Use Encryption: unchecked
   (d) Boot image partitions: Click "Add" to add the files below. *Note - order is important!
      i. fsbl.elf - located in workspace under fsbl/binaries
      ii. design.bit - located in workspace under/<Vivado Block Design Name>_wrapper_hw_platfo Block Design Name>.bit
      iii. uboot.elf - located at /opt/Xilinx/u-boot-xlnx/uboot.elf
3. click "Create Image".
4. BOOT.bin will be created in workspace under fsbl/bootimage

4.5.3 Loading and Booting from the SD Card

1. Load the following files onto an SD Card to boot and run a linux application:
   - **Zynq Boot Image** - BOOT.bin
   - **Device Tree** - devicetree.dtb
   - **Linux Kernel** - uImage
   - **File System Image** - uramdisk.image.gz
2. Remove the SD card from the computer and (with Zedboard off) insert into the flash port on the underside of the Zedboard.

3. Flip the Zedboard switch into the "on" position.
4. Open a serial terminal window emulator. In this effort minicom is used. To open a minicom window and communicate with the Zedboard via serial port, type:

   ```
   minicom -b 115200 -D /dev/ttyACM0
   ```

   If the initial boot is finished when the minicom screen links to the serial port, the screen will display:

   ```
   Zynq>
   ```

5. In the minicom window, type "boot" and hit enter.
6. If Linux boots successfully, boot information will scroll by, ending with the following:

   ```
   Starting syslogd/klogd: done
   Starting tcf-agent: OK
   zedboard-zynq7 login:
   ```

7. Type "root" for the Zedboard username/password and hit enter.
8. Type "ifconfig" to display the IP address acquired by the Zedboard.

4.5.4 Running the CDMA Application

The CDMA Application modified in subsection 4.1 will now be tranferred to and run on the Zedboard.

1. In a terminal window, cd to linux_cdma_app/Debug/
2. Type the following:

   ```
   sudo scp ./linux_cdma_app.elf <IP address of Zedboard>:/home/root
   ```

3. In the Project Explorer pane of Xilinx SDK, right click on cdma_app and select Run As –> Launch on Hardware (GDB)
4. In the minicom window, type:

   ```
   ./linux_cdma_app.elf
   ```

   Successful execution will yield the following display in the minicom window:

   ```
   Entering Main
   /dev/mem opened.
   Memory mapped at address 0xb1fc4000.
   DATA Transfer is Successfull
   ```

4.6 Booting Zedboard Over JTAG and Running the Linux Application

Booting over JTAG allows one useful benefit: The ability to utilize Xilinx SDK's GDB Debug environment while executing an application on the Zedboard.

4.6.1 Board Settings

1. The MIO bank on the Zedboard must have the following values to enable booting over JTAG:
   - MIO6 GND
   - MIO5 GND
   - MIO4 GND
   - MIO3 GND
   - MIO2 GND
2. Ensure the Platform USB Cable is connected between the host machine and the Zedboard JTAG port.
3. Ensure the UART cable is connected from the host machine to the Zedboard UART port.
4. Ensure the UART port is enabled on the Zedboard by shunting the JP3 pins.
5. Ensure the power cable is plugged into the board.
6. Ensure the ethernet cable is plugged into the board.

4.6.2 Loading and Booting over JTAG

In the following instructions, fsbl.elf is not used, as it has not been shown to work in this effort. Instead, a tcl script released by Xilinx is used.

1. Withing the Xilinx SDK workspace, use the SDK Terminal for serial input and output. If one is not open in the C/C++ perspective, then in the main menu click on Window –> Show View –> Other –> Xilinx –> SDK Terminal, and then click "OK".
2. In the SDK Terminal window, click on the green "+" sign to open a new serial port. Set the baudrate to 115200. In this effort the port was located at /dev/ttyACM0. Click OK.
3. Also within the Xilinx SDK workspace, locate the Xilinx XMD Console. If it is not open, on the main menu, select Xilinx Tools –> XMD Console.
4. In the XMD console, type and enter each of the following commands:
   *Note - Alternatively, these commands may be put in a file and run by simply typing "source <filename>"
   (a) cd <path/to/<Vivado Block Design Name>_wrapper_hw_platform_0/>
   (b) connect arm hw
   (c) fpga -f <Vivado Block Design Name>.bit
   (d) source stub.tcl
   (e) target 64
   (f) source ps7_init.tcl
   (g) ps7_init

(h) ps7_post_config

(i) dow -data /opt/Xilinx/linux-xlnx/arch/arm/boot uImage 0x03000000

(j) dow -data device_tree_bsp_0/devicetree.dtb 0x02a00000

(k) dow -data /opt/Xilinx/2015.4_zed_release/zed/uramdisk.image.gz 0x02000000

(l) dow /opt/Xilinx/linux-xlnx/u-boot.elf

(m) con

5. In the SDK Terminal window, some u-boot output will appear. When "Hit any key to stop autoboot" appears, hit any key and the "zynq-uboot" prompt will appear.

6. In the SDK Terminal window, type:

   bootm 0x03000000 0x02000000 0x02a00000

7. After Linux boots, the prompt will be zedboard-zynq7. Enter the username/password of "root".

4.6.3 Running the CDMA Application

1. In Xilinx SDK, right click on linux_cdma_app and select Debug as –> Debug Configurations...

2. In the dialogue box that appears, in the left menu pane, select "Remote ARM Linux Application".
   *Note - If that listing is not available, click the pane menu pull-down icon "˅" and uncheck "Filter Configuration Types".

3. Once "Remote ARM Linux Application is selected", click the "New" icon to create a new ARM Linux Application configuration>

4. In the dialogue box that appears, next to Connection, click "New...". A "New Connection dialogue box will appear. Select the following options and then click "Finish":

   (a) System type: Linux (click "Next)

   (b) Parent profile: <default>

   (c) Host name: <IP address of Zedboard>

   (d) Connection name: <IP address of Zedboard>

   (e) Description: <IP address of Zedboard>

5. In the Debug Configurations dialogue box, select the following options:

   (a) Connection: <IP address of Zedboard>

   (b) Project: linux_cdma_app

   (c) Select configuration using 'C/C++ Application': checked

   (d) C/C++ Application: Debug/linux_cdma_app.elf

   (e) Remote Absolute File Path for C/C++ Application:/home/root/linux_cdma_app.elf // *Note - Make sure the elf executable is listed in the path for it to be recognized!

(f) Skip download to target path: unchecked

(g) Click "Apply"

6. In the dialogue box that appears, select "Xilinx C/C++ application(GDB).
7. Click on the "New" icon and enter the following parameters:
   (a) Debug Type: Standalone Application Debug
   (b) Connection: zed_ssh
   (c) Device: Auto Detect
   (d) Hardware platform: <Vivado Block Design Name>_wrapper_hw_platform_0
   (e) Processor: ps7_cortexa9_0
   (f) Bitstream file: <Vivado Block Design Name>.bit
   (g) Initialization file: ps7_init.tcl
   (h) Reset selection: No reset
   (i) Run ps7_init: unchecked
   (j) Run ps7_post_config: unchecked
   (k) Enable Cross-Triggering: unchecked
8. Click "Apply"
9. Return to the Remote ARM Linux Application - linux_cdma_app Debug configuration pane. Click "Close".
10. In the SDK Main Menu, click Window –> Perspective –> Open Perspective –> Other –> Remote Systems Explorer.
11. Within the Remote System Explorer perspective, click Window –> Show View –> Other –> Remote Systems –> Remote Monitor.
12. Within the Remote Monitor pane, right click on SSH Terminals and select Connect...
13. In the dialogue box that appears, enter the User ID and Password (root) and click "OK".
14. If a warning appears about inserting a new key into .ssh/known_hosts, click "Yes".
15. The SSH Terminals should now appear as Connected in the Remote Monitor Pane. *Note - In this effort, connecting via SSH automatically by clicking on Debug In the Debug Configurations dialogue box did not work. Thus the manual connection described in steps 10 through 15.
16. Return to the C/C++ perspective.
17. Right click on linux_cdma_app and select Debug As –> Debug Configurations.
18. Navigate to Remote ARM Linux Application –> linux_cdma_app Debug.
19. Click on "Debug" in the lower right corner of the dialogue box.

20. The Debug perspective will automatically open. If linux_cdma_app.c is not already open, from the main menu select File –> Open File, browse to the file and click "OK". Set and step through breakpoints using the menu icons.

21. Click on the Console tab to view program output.

22. A successful program run will end with the Console messages below:

```
DATA Transfer is Successful

Child exited with status 0
GDBserver exiting
root@zedboard-zynq7:~# exit
logout
```

5 Appendix

5.1 Troubleshooting