September 11, 2020

CMU 18-643: Reconfigurable Logic: Technology, Architecture and Applications

Handout #4/Lab 1: Up and Running (150 points)

Due Friday, 9/25/2020 noon

This lab can be completed individually or in a team of 2. This lab will bring you up to speed on both:

- the Ultra96 (V2) board and the Vivado design tools suite, and
- the Intel DevCloud and the Intel FPGA SDK for OpenCL.

Proceed to borrow an Ultra96 board from the ECE Academic Services Center. (You can get an early start with installing Vivado before receiving the board.) You will receive an email from Intel about your DevCloud account for use in **Part 5**.

Except for **Part 4** (where you are asked to figure things out on your own), please feel free to post questions and answers on 18643's Piazza page to help each other out with tools and board related issues.

Expect to spend one good evening to get the Ultra96 and Vivado working. Expect to spend one or two good long evenings to run through the tutorials. The tutorials are meant to get you up to speed with a minimum set of skills to start working with the FPGA hardware and tools. Fortunately, there are a tremendous amount of supporting resources online for Vivado, Zynq, and Ultra96. You will be surprised how often googling a specific question phrase will return exactly the answer you need. *Do not wait to start, FPGA synthesis takes hours.*

Part 1: Download and Install Vivado

In this part, you will install *Vivado Design Suite - HLx Editions* on your personal computer.[1] You can download the installer from http://www.xilinx.com/support/download.html. Please select version 2018.3. Follow the instructions on the website and the installer to install the Design Edition or the System Edition. Make sure **SDK** and **SoCs** are selected for installation. Download and install can take hours depending on your network connection and computer; have something else to do while you wait. You will need to register for a free Xilinx web account to download the installer and for the activities later on.

After you have completed the installation, skip the step where you are prompted to go online to obtain a license. Instead, from the license manager (Windows version), look for "Manage License Search Path" to set XILINXD_LICENSE_FILE to 2101@xilinx-lic.ece.cmu.edu.[2] Click refresh in "View License Status" to verify you have access. The license is only accessible from CMU's network or VPN. On Linux installations, you also need to manually install cable drivers by invoking the script: <YOUR_XILINX_INSTALL>/data/xicom/cable_drivers/lin64/install_scripts/install_drivers/install_drivers .

Part 2: Going for a Quick Spin

Before doing anything with your Ultra96 board, review *Ultra96-V2 Hardware User's Guide* (https://www.element14.com/community/docs/DOC-92688/l/ultra96-v2-hw-user-guide-rev-1-0-v10preliminary?ICID=ultra96v2-datasheet-widget) for a quick overview.[3]

---

[1] A number of ECE lab machines will be configured with Vivado. However, it will be much easier for you to work on your own Windows or Linux computer with at least 8GB of DRAM. The Windows version will likely install and work properly by following the instructions; the Linux version will invariably involve some machine and OS specific tweaking. If you are a Mac person, Vivado can work from a VMware VM (free for CMU students from https://www.cmu.edu/computing/software/all/vmware/index.html) if you allocate enough memory.

[2] For Linux, set XILINXD_LICENSE_FILE in the shell.

[3] https://www.element14.com/community/community/designcenter/zedboardcommunity/ultra96 hosts other useful resources.

Work on the tutorials below. Follow the directions carefully.

- *Tutorial 01 Build a ZU+ MPSoC Hardware Platform* (http://zedboard.org/sites/default/files/design/01_Ultra96_Vivado_Intro_2018_2_01.zip)
  - **Important Deviation:** The tutorial is written for the older Ultra96-V1. Follow the instructions (page 3) to install the board definition but use the ultra96v2 folder.
- *Tutorial 02 First ZU+ Application - Hello World* (http://zedboard.org/sites/default/files/design/02_Ultra96_Vivado_Hello_World_2018_2_01.zip)
  - **Important Deviation:** When attaching the USB-to-JTAG/UART pod, your Ultra96-V2 will look different than the Ultra96-V1 in Figure 18 and 19 on Page 15. Compared to Figure 18, J2 on Ultra96-V2 has 1 extra left-most pin; J6 has 1 extra right-most pin. Ignore the extra pins when connecting the pod.
- *Tutorial 03 Generate and Run Bare Metal ZU+ Test Applications* (http://zedboard.org/sites/default/files/design/03_Ultra96_Vivado_TestApps_2018_2_01.zip)
  - In addition to simply running the applications, try out the debugger too. This full featured debugger lets you set breakpoints, take single steps, and interrogate the processor state.

Besides demonstrating the basics of working with Zynq/Ultra96 and Vivado, these tutorials will confirm that your Vivado installation and your Ultra96 board are working correctly.

Part 3: Taking a Closer Look

In this part, you are going to add a Block BRAM (BRAM) on Znyq's programmable logic (PL) fabric and to access it using memory-mapped loads and stores from the ARM processor in Zynq's processing system (PS).

To add BRAM:

1. Re-open the Vivado project from Tutorial 1[4], in the IP Integrator diagram with the PS already configured, click on "+" to add a new IP called AXI BRAM Controller.
2. Click when prompted to "Run Connection Automation"; accept all three automations offered. When you return to the IP Integrator diagram afterwards, you should see several additional IP blocks. Most importantly, the BRAM controller is connected with the PS through an AXI switch, and a BRAM block is attached to the BRAM controller.
3. Click on the "Address Editor" tab above the IP Integrator diagram. Change the "range" setting for the BRAM controller to 64K. Afterwards, make a note of the "offset address" and "high address".

Next follow the steps in Tutorial 1 starting from Step 8 to generate the new bitstream and export hardware.[5] Follow the steps in Tutorial 3 for the Memory Tests only. The BRAM you added should now be covered by the new automatically generated test code.[6] Study the source code to see how to access the BRAM region by memory-mapped loads and stores.

> **Thinking Ahead.** Following similar steps as above, you can introduce other modules that support AXI target interfaces (typically comprising a set of memory-mapped control and status registers). The modules can come from an IP library or be your custom design. Have a peek in Exercise 4A in Chapter 17 of the Zynq book to see how to create and use your own AXI target module.

[4] Instead of reopening the old project, you can also restart Tutorial 1 from scratch and stop after Step 7.
[5] You can skip Step 11 if you are adding to the old project with wrapper already created.
[6] There is a lot of smarts and automation in creating the software collaterals for the SoC you specified. Take a look through the .h files in the include directory of the bsp. You can find function or macros for most things you are likely to want to do in interacting with the hardware. The file xparameter.h has the parameters specific to your hardware instance.

Part 4: Let's see what you can do.

For this final part, you are on your own to figure out what to do. Be resourceful.[7]

Before starting work, decide how you will make the time measurements in this part. Apply your chosen methodology to experimentally determine the clock frequency of the ARM core. Check the correctness of your timing measurement by checking your answer against what you can find from tool or documentation.

Modify the Memory Tests application to do the following using memory-mapped 32-bit loads and stores:

1. Initialize all BRAM 32-bit word locations by writing increasing counts starting with 0 in word location 0.
2. Read back the BRAM word locations to verify the writes.
3. Overwrite the BRAM word locations with new values by continuing the counts.
4. Read back the BRAM word locations to verify the writes.
5. Design a microbenchmark to determine the steady-state bandwidth (in bytes per second) when reading the same BRAM word location repeatedly. Do the same for writes.
6. Design a microbenchmark to determine the steady-state bandwidth (in bytes per second) when sequentially reading through all of the BRAM word locations repeatedly. Do the same for writes.
7. Experimentally determine if memory-mapped loads and stores to BRAM are cached by the ARM core.

Submit: Create a submission directory <team_name>_lab1 to turn in the artifacts requested. <team_name> should be a concatenation of the team members' Andrew IDs in alphabetical order connected by '_'. One member of a team should submit a single file called <team_name>_lab1.zip (that is the zip of your submission directory) through Canvas.

Include in the submission directory all .c/.cpp and .h source files you introduced or modified. (Do not submit the entire Vivado or SDK project directory). Comment extensively so the intention of your program is clear. Also include a PDF file <team_name>.pdf which reports

1. The base address and size of the BRAM.
2. The frequency of the fabric and the ARM core.
3. What is the associativity, block-size, and capacity of the ARM core's L1 and L2 cache? (You can look this up or determine experimentally.)
4. The read and write bandwidths for accessing a single location repeatedly
5. The read and write bandwidths for accessing all locations sequentially over the entire BRAM.
6. Are reads and write to the BRAM cached? Explain how you determined this.
7. What happens if you access the BRAM using 64-bit loads and stores?

Part 5: Intel DevCloud Tutorial

See separate Handout #5.

[7] Bookmark this page: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0070-zynq-mpsoc-design-overview-hub.html for future reference.