University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Spring 2017 HW1: Hello World and Debug Wednesday, January 11

Due: Friday, January 20, 5:00PM

This first assignment is designed to introduce you to the software and hardware that will be used throughout the course. You will use the Xilinx SDSoC (Software-Defined System on Chip) software to run a "Hello, World!" application on one of the ARM Cortex-A9 cores on a ZedBoard. Afterwards, you will learn the skills need to locate a bug in SDSoC.

This assignment is more scripted than future assignments. The goal is get you started quickly on the platform and toolset. You should take the time to explore and understand the tools. While the assignment is fairly detailed, it does not go so far as to tell you every button to press. You will need to consult other documentation identified and experiment to complete many steps.

Like most modern tools you will use, these tools are complicated and their behavior at times will be obscure. While we have given you instructions here, you will likely find them incomplete or do something not quite as intended and find yourself trying to figure out where to go. We expect it will take some reading, searching, and experimentation to get back on track (it took us a bit of all of these to get this far). Since this is a new tool set for all of us, we don't have a ready FAQ of all the problems you may encounter. As part of this exercise, we are asking you to document the problems you run into and how you overcame them. This is both for your own introspection and self-learning and as a way for us, as a community, to begin to collect and share the FAQ. As an example (and as assitance for problems we did run into), we include notes of this style from when Prof. DeHon worked through the assignment.

Experience Note: My notes will be formatted like this.

You probably want to read through the entire assignment (including the "Homework Submission" section at the end) before you start to work the assignment.

Collaboration

In this assignment, you may work in pairs. You can certainly do this assignment alone. Everyone should be learning how to use all the tools, so don't take this as an opportunity to compartmentalize who knows how to do what. Our rationale for pairs for this first assigment is so that you can reason together on how to get the tools to work. For this assignment, every student can choose the partner of her/his choice. For future assignments, you will have

a different partner that we will assign—all the more reason to make sure everyone knows how to use all the tools. If you cannot find a partner, we encourage you to look on Piazza for other students without partners. Piazza has functionality that allows you to search for teammates. In the event that you are unable to find a partner, contact the instructor or TA.

Each student should submit a complete report. You are free to share information about how to perform certain activities using the Xilinx software with other students. In fact, we encourage you to share any difficulties that you have with the software and general solutions or workarounds for them on Piazza. However, you are not allowed to share the results (e.g. output files, numbers, graphs) that you obtain using the software outside of the pair working together on the assignment. Moreover, final results should be analyzed, and conclusions should be drawn individually.

All students must follow the Code of Academic Integrity. Infringement of the code can have severe consequences for you and your partner, such as failing the course or cancellation of your student visa if you are an international student, so please familiarize yourself with them. See the course policies on the course web page `http://www.seas.upenn.edu/~ese532` for full details of our policies for this course.

SDSoC Software

We have installed SDSoC on the PCs in the Ketterer Lab. If you would like to run the software on your own laptop, you can download it from the Downloads page on the Xilinx website. Be aware that SDSoC has strict operating system requirements, and that backward compatibility is not always guaranteed. SDSoC 2016.2, for example, supports Ubuntu 14.04, but Ubuntu 16.04 is not supported. The software requirements can be found in the SDx Environments Release Notes, Installation, and Licensing Guide. A license can be obtained from the license server. In order to use the license server, you should set the environment variable `LM_LICENSE_FILE` or `XILINXD_LICENSE_FILE` to `2100@potato.cis.upenn.edu:1709@potato.cis.upenn.edu:1717@potato.cis.upenn.edu:27010@potato.cis.upenn.edu:27009@potato.cis.upenn.edu`. In Windows, you can also set the variable in the Xilinx License Configuration Manager (XLCM).

First Application

1. Create a new *Workspace* directory in which you would like to store all your SDSoC project files. In the Ketterer Lab, you can make a directory under `S:\`, your Eniac home directory.

2. If you are using Linux, source `<SDSoC directory>/2016.2/settings64.sh` to prepare your shell for running SDSoC.

3. Launch SDSoC and switch to the workspace directory that you created. In the Ketterer Lab, you can start SDSoC via *All Programs* → *ESE Lab Software* → *Xilinx (Ketterer*

Lab) → *SDSoC 2016.2* → *SDSoC 2016.2*. You can find instructions to perform this and the next step in the SDSoC Environment User Guide: Getting Started.

> Experience Note: When I first tried this, it complained that the workspace could not be created or was read-only. However, clicking through the windows, it eventually allowed me specify the directory that I created in step 1.

4. Create an empty *SDSoC project* for the ZedBoard with a *Standalone* operating system. The ZedBoard is indicated as the *zed* platform in SDSoC. An operating system, e.g., Linux, provides more drivers and libraries to the application than the standalone configuration. A drawback is that an operating system will also consume more resources. In this assignment, we will only use the serial port, for which the standalone configuration is sufficient.

5. Create a new file called `hello_world.c` in the `src` directory using SDSoC. Students familiar with the Eclipse IDE should have no trouble using SDSoC as it is based on Eclipse. Other students may find the Eclipse Documentation useful.

   > Experience Note: You need to tell SDSoC about the file. I found that I needed to use the New function, then select "Other", then"C/C++", then "Source file" to get an editor up for a new file where I could add code.

   > Experience Note: When I tried just putting a file I created elsewhere in the directory, it complained there was nothing to build. I eventually decided it was necessary to "import" my source file in order for the tool to know about it. This mostly worked, but the tool also flagged my include of `stdio.h` as unresolved. Nonetheless, it seemed to work.

6. Write C source code for printing a cheerful message such as `Hello World` in the `hello_world.c` file, and save it. To verify whether your code works, you could compile it using a regular C compiler such as Visual C or GCC.

7. Build the project. You can refer again to the earlier mentioned SDSoC Environment User Guide. This will take longer than a regular C application because SDSoC will also generate a hardware configuration for the FPGA.

   > Experience Note: I had problems with my build failing. Eventually, I noticed that I had exceeded my eniac quota and there wasn't space to store the build; the error messages weren't helpful in leading me to this potential failure cause. Once I cleared up some space on my account, it finally built.

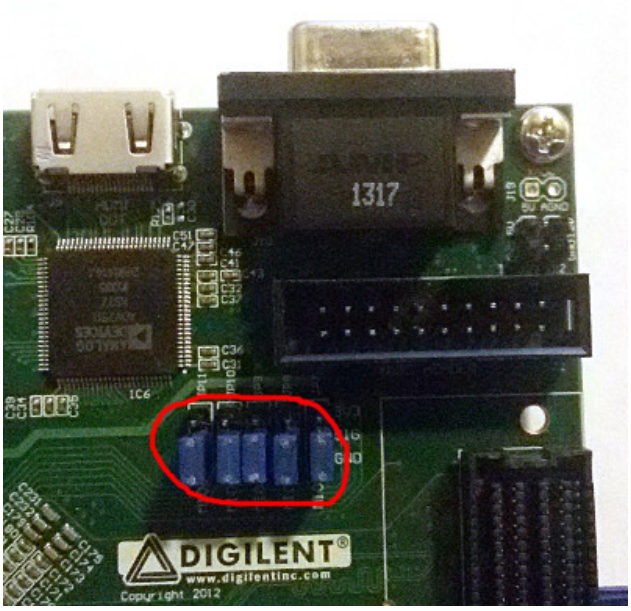

Figure 1: Jumpers in JTAG Mode

8. Set the configuration mode of the ZedBoard to JTAG mode while the board is powered off. The mode is set by placing the MIO3, MIO4, and MIO5 jumpers above the Digilent logo on the board to the bottom position, connecting the SIG to GND. This is illustrated in Figure 8. In JTAG mode, the FPGA configuration is transmitted to the FPGA via a USB cable. In SD card mode, the user copies the configuration to an SD card, which must be transferred physically to the ZedBoard. JTAG mode is better fit for short debug cycles because the configuration can be performed automatically when the application is started. More information on the different configurations can be found in the ZedBoard Hardware User's Guide.

9. Connect the two USB connectors closest to the power connector labelled `PROG` and `UART` to the PC running SDSoC as shown in Figure 9.

10. Power on the board. If you are using Windows, it may ask you whether you would like to install the USB-UART and Digilent device drivers. In that case, give Windows permission to install the drivers.

11. If you are using Linux, make sure that `hw_server` is running with root permissions. If that is not the case, kill the process and start it again as root. Without root permission, the application cannot be launched on the FPGA.

12. Launch the application in debug mode. Right-click on your project in the *Project Explorer*, and choose *Debug As* $\rightarrow$ *Launch on Hardware (SDSoC Debugger)*. The configuration will now be uploaded to the FPGA.

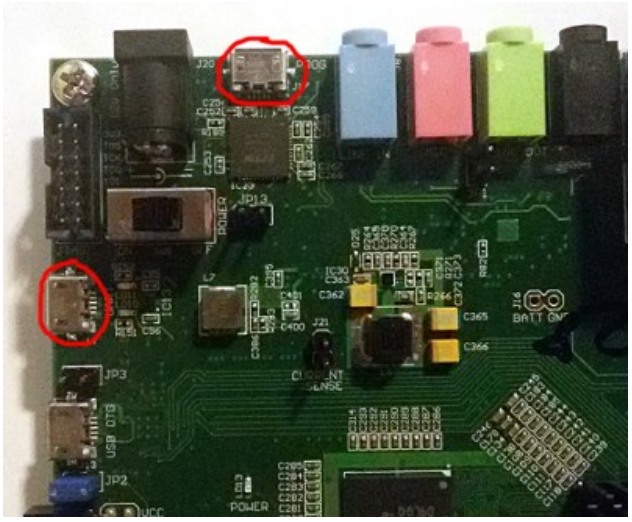

Figure 2: USB Connectors to be Connected

> Experience Note: I got an error message saying that it could not find the ARM device on the board. Ultimately, it turned out I had plugged the USB connectors into the wrong ports. There are more than two USB micro B ports. Sadly, I spent an hour on this.

13. A dialog will pop up requesting your approval to switch to the *Debug* perspective, which you should confirm. The perspective affects the menus, views, and toolbars of SDSoC. To switch to another perspective, one presses the buttons in the top-right corner of SDSoC. The *SDSoC* button will return you to the perspective that we used so far. For now, stay in the *Debug* perspective. The first executable line of your code should be highlighted in the view in the center.

> Experience Note: If the *Debug As → Launch on Hardware (SDSoC Debugger)* didn't work for you the first time, you'll now find yourself in this *Debug* perspective and need to relaunch the deugger. Look for the icon with the bug and bring up its menu. There should be an option there (likely top one at this point) that will allow you to launch the debugger on your specified project.

14. Switch to the *SDK Terminal* tab in the view at the bottom center of the window. Press the plus button in the top right of the view to add a new serial connection to the board. Enter the name of the port and keep the default baud rate of 115200 symbols/s. In Windows, the port name will be `COMXXX`, where `XXX` is a number. You can determine the number by opening the device manager to see all available ports, and determine which port disappears when the UART connector on the board is disconnected. In Linux, the port name is usually `/dev/ttyACMXXX`. Here, you can also disconnect and reconnect the cable to see which device file is associated with the serial cable. It is also necessary to give the user running SDSoC permission to access the device file in Linux. Once you have completed the dialog, dismiss it by clicking *OK*.

Experience Note: I never found the "plus button". Nor did the SDSoC Environment User Guide document help. Ultimately, I had to type "SDK Terminal" into the "Quick Access" text box on the top right. Note that the "SDK Terminal" is different from the "TCF Debug virtual terminal" that comes up by default.

15. Press the *Resume* button in the main SDSoC toolbar, or select *Run* → *Resume* from the menu.

16. The application will continue to run. The message that you wrote will be displayed in the SDK terminal.

17. Include the build log in the *Console* window and the program output from the *SDK Terminal* window in your report.

Debug An Application

1. Create a new project with the same settings as the previous one.

2. Create a new source C file and paste the following code in it, which should print another message:

```
#include <stdio.h>

int len(char * s)
{
  int l = 0;
  while (*s) s++;
  return l;
}

int rot13(int l)
{
  if (l >= 'A' && l <= 'Z')
    l = (l - 'A' + 13) % 26 + 'A';
  if (l >= 'a' && l <= 'z')
    l = (l - 'a' + 13) % 26 + 'a';
  return l;
}

char * msg = "Jryy Qbar!!!\n";

int main()
{
  int i = 0;
```

```
  while (i < len(msg))
    printf("%c", rot13(msg[i++]));

  return 0;
}
```

3. Build and run the new project.

   Experience Note: Is there a way to just rebuild the C code and not the rest of the platform? I haven't figured that out, yet.

4. You will notice that the output is not correct. Use the debugger to locate the bug in the code. Specifically, we are asking you to **not** simply perform `printf` debugging where you insert code and recompile. If you haven't used a debugger before, take the time now, during this easy assignment, to learn the basic tricks of the debugger. Learn how to set breakpoints, step through code, and inspect variables. Instructions on how to set breakpoints, view variable values, and more can be found in the Eclipse documentation.

5. In your report, describe how you found the bug, how you changed the code, and show the message that should have been displayed.

Homework Submission

Write the answers to the questions in an electronic document in Microsoft Word or PDF format, and submit it on Canvas. We do not accept handwritten documents or figures. To be eligible for the full grade, your homework must be received before the deadline. Keep your answers concise. When you include results, integrate them properly in your report, by adding captions and references for example. When you refer to them in your text, clearly indicate the particular result that you are calling out. Include your name and the name of your partner on your submission. Your writeup should include the following:

1. **Learning and navigating the tools.** As noted at the beginning, document all the problems you encountered in working the assignment and how you solved them. For each, include:
   (a) Step (if applicable)
   (b) OS where you were running (possibly some of you will do parts of the problem on multiple OSes)
   (c) Error message or unexpected behavior you encounted
   (d) How the problem was resolved
   (e) How you found the solution (e.g., how did you experiment or reason through an answer, which document had the answer, where on the web did you find an answer, which student or TA was able to point out the answer)
2. **Tools**
   (a) Provide the code for your "First Application"
   (b) Provide build log from your successful build of your "First Application"
   (c) Provide the SDK Terminal log from your successful execution of your "First Application"
   (d) Describe how you do the following in SDSoC
      i. Add a breakpoint
      ii. Remove a breakpoint
      iii. Inspect a variable value
      iv. Step through program execution without a breakpoint
3. **Debug**
   (a) Provide the SDK Terminal window for the correct output
   (b) Provide code for corrected function(s)
   (c) Include a description of how you used the tools to identify the bug (or confirm a hypothesis you formed about the location of the bug)