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

ESE532, Fall 2017 HW1: Hello World and Debug Wednesday, August 30

Due: Friday, September 8, 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. Recently, SDSoC has been combined with SDAccel, so we will typically refer to the software as SDx.

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).

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.

SDx Software

We have installed SDx on the PCs in the Detkin and Ketterer Labs. If you would like to run the software on your own system, you can download it from the Downloads page on the Xilinx website. We require SDx 2017.1 because we test our instructions on that version. We cannot support or accept homework based on other versions. Be aware that SDx has strict operating system requirements, and that backward compatibility is not 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. We advise against running SDx in a virtual machine because we have not seen any successful cases. Typically, communication with the ZedBoard is the issue. Should you manage to make this work anyway, please share how you did this with us. Be aware that the complete installation of SDx needs a lot of disk space (around 79 GB). We will not use SDAccel or the MPSoC platform, so you can deselect the MPSoC platform, which should save you about 39 GB.

Windows

If you install SDx on your own Windows-based system, you can use a license from the license server. In order to use it, 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`. You can also set the variable in the Xilinx License Configuration Manager (XLCM). In the past, we have seen an issue where SDx complains that you have to install the Visual Studio 2012 Redistributable. However, even after installing it, SDx keeps complaining. To solve this, you can remove the file `Vivado\tps\win64\xvcredist.exe` in your SDx installation root. More information can be found here.

Linux

If you install SDx on Ubuntu, you should pay special attention to the instructions provided in the manual about installing certain 32-bit libraries and creating a symbolic link for `gmake`. 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`.

First Application

1. Create a new *Workspace* directory in which you would like to store all your SDx project files. In the labs, you can make a directory under `S:\`, your Eniac home directory. Choose a directory with a short path name because SDx cannot handle paths with more than 260 characters, and it needs many of the characters for the file hierarchy that it generates in your directory. In Windows, make sure that you are not using UNC addresses, such as `\\system\dir\dir\`, when you are using network resources, but use network drives, such as `S:\`.

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

3. Launch SDx and switch to the workspace directory that you created. In the labs, you can start SDx via *All Programs* → *ESE Lab Software* → *Xilinx Design Tools* → *SDx 2017.1* → *SDx IDE 2017.1*. You can find instructions to perform this and the next step in the SDx Environment User Guide: Getting Started.

4. Create an empty *SDx project* for the ZedBoard with a *Standalone* operating system. The ZedBoard is indicated as the *zed* platform in SDx. 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 SDx (not another text editor). Students familiar with the Eclipse IDE should have no trouble using SDx as it is based on Eclipse. Other students may find the Eclipse Documentation useful.

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 try to compile it using a regular C compiler such as Visual C or GCC, but this is not strictly necessary, so you can ignore this if you do not have a compiler handy.

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

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

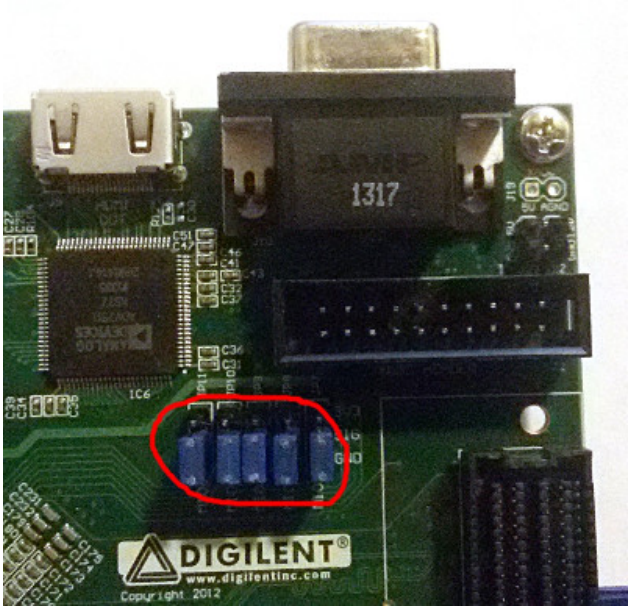

Figure 1: Jumpers in JTAG Mode

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 SDx as shown in Figure 2. If you connect other ports than the ones shown, it will not work. Be aware that there have been USB cables in the labs that only convey power. We will use the cables for transmitting data, so power-only USB cables are not suitable. You can check whether a cable can transmit data by measuring the resistance of the middle two pins of a connector from one side of the cable to the other. The resistance should be low.

10. Connect the board to an outlet using the power adapter, and power on the board. Note that the board cannot be powered via USB.

11. 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. If the drivers are not installed automatically, you can install the driver manually. Go to *Devices and Printers*. Right-click on the *Cypress USB2UART* device, select *Update device drivers*. In the following dialogs, make sure that you search for the drivers online.

12. 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,

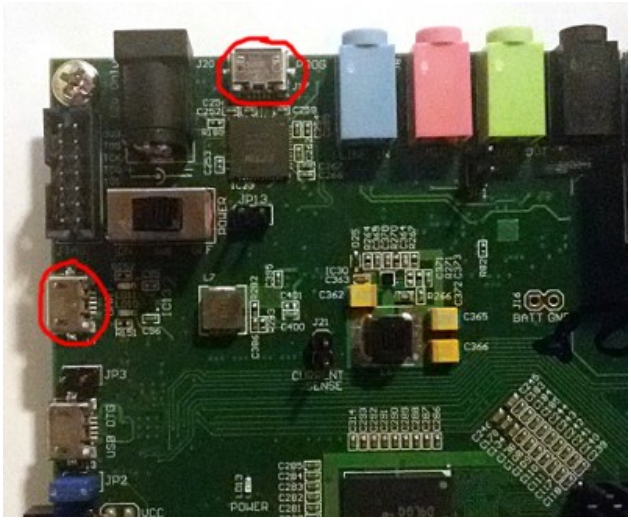

Figure 2: USB Connectors to be Connected

the application cannot be launched on the FPGA.

13. If you are using Linux, you will have to change the permissions of the serial device each time you connect the ZedBoard to your PC. To avoid this on Ubuntu, you have to add your user to the `dialout` user group with the command `sudo adduser $USER dialout`. You will have to log out and back in again before this takes effect.

14. Launch the application in debug mode. Right-click on your project in the *Project Explorer*, and choose *Debug As* → *Launch on Hardware (SDx Debugger)*. The configuration will now be uploaded to the FPGA. Occasionally, SDx warns `No ARM device found on the board`. This is often resolved by power cycling the board and relaunching.

15. 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 SDx. To switch to another perspective, one presses the buttons in the top-right corner of SDx. The *SDx* 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.

16. Switch to the *SDx 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 observing 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. Once you have completed the dialog, dismiss it by clicking *OK*.

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

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

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

Should you decide to change your code after running it, note that SDx does not support hot swapping of code similar to Java in Eclipse. You will have to build your entire application again and relaunch your debug session. Further note that resuming the application when it has finished will not restart it. Instead, it will keep running in an infinite loop. To restart it you will have to *relaunch* it.

As you probably have noticed by now, building a project can consume much time. That is because SDx builds both hardware and software. Building hardware is notoriously slow, and we must plan for the time required. That said, there are a few things that may reduce the time somewhat. If you run your code on a network drive (such as S: in the labs), you should consider moving your code to a local drive. Moreover, make sure that you build only the projects that you want to build. In other words, select your project and choose *Build Project* instead of building the entire workspace. You can also close projects to avoid building them.

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. The code should print another message, but due to a bug, it doesn't.

```
#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;
  printf("The secret message is: ");
  while (i < len(msg))
    printf("%c", rot13(msg[i++]));

  return 0;
}
```

3. Build and run the new project.

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

Your writeup should follow http://www.seas.upenn.edu/~ese532/fall2017/handouts/writeup_guidelines.pdf
Your writeup should include the following:

1. **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 SDx (each 3 lines max.):
      i. Add a breakpoint.
      ii. Remove a breakpoint.
      iii. Inspect a variable value.
      iv. Step through program execution without a breakpoint.
2. **Debug**
   (a) Provide the SDK Terminal window for the correct output.
   (b) Provide code for the corrected function.
   (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) (5 lines max.)

3. **C** – answer the following questions about C pointers, compilers, and executables.

   (a) Considering the following code, give an expression to obtain the address of `b` that can be accessed via the third element of x. (1 line)

```
struct s2
{
  float a;
  int b;
};

struct s1
{
  int c;
  s2 * * d;
};

s1 x[5];
```

   (b) How do you declare a pointer that points to the following two-dimensional array? (1 line)

```
int a[2][3];
```

   (c) The following array will be stored as a sequence of bits in the ARM memory. We could also consider these bits as a sequence of 16-bit signed integers. Show code that prints those integers.

```
double a[] = {3.14, 2.71};
```

   (d) What is the purpose of the preprocessor, compiler, and linker? (each 3 lines max.)

   (e) How do you inform the preprocessor where to search for header files that are included with `#include`? (3 lines max.)

   (f) How can you resolve an error such as `undefined reference to ...`? (3 lines max.)

   (g) How does SDx know where functions and data are located in the executable when you are debugging? (3 lines max.)