Microprocessor-Based Systems (E155)

D. Harris and M. Spencer **Fall 2015**

Lab 1: Utility Board Assembly

Introduction

In this lab you will assemble and test your MiddPi Mark IV utility board containing an Altera Cyclone IV FPGA, Flash configuration memory, LEDs, switches, a clock oscillator, H-Bridges and an ADC. You will be using the board for the remainder of the semester, so it is very important to assemble it properly. But don't panic! You'll quickly learn to troubleshoot and repair issues. If you totally cook your board, you may obtain parts for another one from the stock room.

Objectives

- Learn how to solder
- Assemble your MuddPi Mark IV utility board
- Write a Verilog module to control LEDs and a 7-segment display
- Program the FPGA with your Verilog code
- Test and debug the utility board
- Interface the 7-segment display to the utility board

Background

In the 1980's and 1990's, digital design projects were built from a truckload of chips, each containing a few logic gates such as 74xx series logic gates or simple programmable array logic chips (PALs). Such projects involve placing and wiring together dozens of chips on a breadboard. It was easy to make a wiring mistake or burn out a chip and spend hours tracking down the problem. Now you can perform all of your digital logic on a single field-programmable gate array (FPGA) to greatly reduce the necessary wiring and number of chips. Later in the course, you will use a Raspberry Pi to write programs in assembly language and C that can interface with external hardware and the FPGA.

You will need to connect your FPGA to the real world to get inputs and outputs. In particular, you will often find it useful to have a clock oscillator, switches, and LEDs. Moreover, the FPGA comes in a 144-pin thin quad flatpack (TQFP) package. This is a surface mount (SMT) part so the pins of the chip do not go through the printed circuit board (as with the other through-hole parts you'll be working with). SMT components are mounted on pads on top of the PCB. Their pins are so small that special soldering skills and tools are needed to reliably attach them to the board. Your board comes with the

FPGA already attached, but you will get to solder the rest, including a few easier SMT components.

The FPGA you will use in this course is the Altera Cyclone **EP4CE6**, which contains 6,272 Logic Elements. You can find the datasheet on the class web page. You will need to become familiar with the internals of the FPGA as you progress in this class.

The board was designed to maximize your access to the capabilities of the FPGA from a breadboard. Therefore, as you will notice, the board has a single row of 58 header pins (on one of the long sides of the board) that give you access to most of the FPGA pins. The other pins in the FPGA are power, ground or unused. Furthermore, the pins are labeled for your convenience. The other characteristics of the utility board will be explained later, in the Discussion section of this guide.

The MuddPi Mark IV board schematic is at the end of this manual. Study this schematic to understand what goes on in the board. You will need this information to use and debug your board, and you are likely to be asked about some details of the board during your checkoff. The parts in the schematics are labeled using R for resistors, C for capacitors, and U for units (chips and other large parts).

Component Discussion

The figures at the end of this document include both the bill of materials (BOM) and the utility board schematic. Major components include the power supply, header pins, clock generator, reset pushbutton, the JTAG programmer connector, DIP switches, LEDs, H-Bridges, and the ADC. What follows is more information several key components on the board.

Read through and understand this section that describes each of the board components. Identify the components in the schematic and think about how it operates. The subsequent section will guide you through the assembly process.

Power Supply

The utility board is supplied power by the header pins Vin and GND. The input supply should be at least 5 volts. The input is regulated down to 3.3 V, 2.5 V, and 1.2 V to serve various components on the board. The 3.3 V regulated voltage is also available at another header pin to serve other components you might want to connect to your board. Be sure not to connect this 3.3 V output to the 3.3 V output of another board such as the Raspberry Pi because the two voltages will be subtly different and will cause contention and likely board damage. Also, don't short the 3.3 V output to ground.

In the lab, the easiest way to supply power is from a 5 V benchtop power supply to the Vin header pin. If you do not have a power supply available, a DC transformer is an inexpensive alternative. For unteathered applications such as robotics, you can also use a battery pack providing at least 5 V.

The FPGA I/O's will operate at a voltage we will call V_{CC}. V_{CC} is 3.3 V in our FPGA. The FPGA contains tiny logic transistors that would burn out at 3.3 V, so the 1.2 and 2.5 V supplies are needed to operate the digital logic and the analog circuitry within the FPGA.

Header Pins

The board provides one 58-pin row of male header pins that tap out signals from the FPGA, LEDs, switches, and power and ground. The header pins will be installed along the left side of the utility board.

Additionally, the board has an 18 pin female header connection at the bottom of the board which grants external access to the H-Bridge and ADC pins. Through this interface, your Raspberry Pi can command the ADC and the H-Bridge.

Clock Generation

In digital electronics, the term clock refers to a square wave that oscillates between GND and V_{CC}. Typically, clocks are used to sequence the action of a circuit. For instance, a flip-flop samples its input every time the controlling clock rises from logic 0 to logic 1. The clock signal is sent to the FPGA. The utility board provides 40 MHz clock from a crystal oscillator.

Reset Pushbutton

There is one reset pushbutton switch on the board which resets the FPGA. Upon reset, the FPGA, will have its memory cleared. If the PROM is programmed, the FPGA will immediately reprogram itself from the PROM. You will notice that these pushbuttons actually have 4 pins. However, only two pins are used, while the other two are left unconnected (and simply provide mechanical stability).

JTAG Programmer Jack

In the top right corner, the double row of header pins (marked JTAG) is for programming the FPGA.

DIP Switches

The DIP switches provide a convenient source of inputs to circuits in your FPGA. When the switch is closed, it delivers a high output. When the switch is open, a pull-down resistor produces a low output.

LED array

There is a 10 segment red light-emitting diode (LED) array on the board. The 'ON' LED (far left LED, with the pin header towards you) simply tells that the board is powered ON and receiving 3.3 V. The LED immediately to the right of the ON LED is used by the JTAG programmer. The other 8 red LEDs can be driven by the FPGA. Notice that there are 8 header pins labeled LED0 – LED7. These can be used in case you want the outputs of the LEDs for an external device. If a voltage is applied at one of the LED pins, the corresponding light is off for ground (GND) and on for voltage 3.3V (V_{cc}).

LEDs glow nicely when given about 5 mA of current but may burn out when given more than about 20 mA. Therefore, it is important to include a current-limiting resistor to prevent LEDs from burning out, or from overheating the driver or simply wasting power. A 330 Ω resistor does the job nicely, allowing $\frac{V_{CC}-V_{diode}}{R} = \frac{3.3V-1.7V}{330\Omega} = 4.8mA.$ of current to flow. The diode drop of 1.7V is typical of red LEDs, but silicon diodes and other LED colors will have different values.

H-Bridges

The H-Bridges are ICs that help to control DC motors. DC motors have a two terminal interface. When the voltage of one terminal exceeds the voltage of the other, the motor spins in one direction. When the voltage of the other terminal is higher, the motor spins in the opposite direction. The motors draw a large amount of current, far exceeding the capacity of a typical digital circuit. H-Bridges allow you to use two low-current digital control signals to specify which direction you want the motor to spin. The H-bridge powers the motor from an external battery source. This board has a screw terminal in the lower right corner to provide an interface to the board's H-bridges: pairs of DC motor wires can be connected there. The H-Bridges can control motors connected to the pairs 1Y/2Y, 3Y/4Y, 5Y/6Y, and 7Y/8Y. These motors are powered from the Vmot and GND terminals. Read the L293D datasheet for more information.

ADC

The ADC (Analog to Digital Converter) can measure analog voltages on two channels (CH0 and CH1) and output the measurement as a 10 bit digital value over a serial peripheral interface (SPI). You can read more about its specific details on the MCP3002 datasheet.

FPGA

The FPGA contains a large array of configurable logic blocks (CLBs) and programmable interconnections. The configuration information is stored in static random-access memory (SRAM) within the FPGA. Therefore, the information is lost when power is removed and the device must be reconfigured each time it is powered up.

The FPGA can be configured via the Programmable Read Only Memory (PROM) on board. On powerup, the FPGA will serially configure itself from the PROM if the PROM has been programmed. It will also reprogram itself from the PROM when you push the reset button. Alternatively, designs can be programmed directly onto the FPGA in the Quartus software via the JTAG connection. This will overwrite the design already loaded from the PROM, but will only last until the FPGA is reset or the power is disconnected.

Assembling the Board

This section will give you tips on how to assemble the utility board. Following these tips will help you complete this lab quickly and in a logical way that places the flattest components first to make soldering easier.

☐ Identify the Component Side of the Utility Board and Prepare to Solder

The utility board has two sides. The *component side*, with the white silk screen markings indicating component placement, should go up. All components except the 58-pin header are placed on the component side. The through-hole parts (except the 58-pin header) are soldered on the reverse *solder side*. The surface-mount components are soldered using solder paste on the component side.

Using a multimeter, measure the resitance between the pins labeled Vin and GND on the board, and verify that the resistance is at least tens of kΩ. If the resistance is low, you have a short circuit on your bare board and should get a new one. As you assemble your board, occasionally check the resistance between Vin and GND. If it is ever less than 10 Ω, you've introduced a short and should debug it before continuing.

The FPGA and voltage regulators should already be soldered to your board. The FPGA is a fine-pitch SMT component that requires practice to attach without creating solder bridges between pins. The voltage regulators have large solder pads which also require careful soldering techniques. If your board malfunctions and you suspect a bridged or bad connection on one of the SMT chips, check the connections under a microscope.

Note: CMOS components such as the FPGA are sensitive to static electricity. Before you touch your board or solder anything, discharge your body by touching a large metal object. This is good practice when handling electronic components in any lab.

When you begin soldering, moisten the sponge. When the iron first heats up, tin the tip by applying a generous amount of solder all over the tip, then wiping off the excess on the sponge. Periodically repeat as you work to keep the tip looking silvery rather than black and blistered. The soldering iron temperature should be set to a medium value (2.5 – 3.5 on the MicroP's Weller stations). Higher temperatures will cause tips to oxidize and blister more quickly. Similarly, leaving the iron at high temperatures will cause damage to the tip, so the iron should be shut off unless you are actively using it (i.e.: holding it in your hand with intent).

Having a clean, well-maintained, solder iron tip is critical to being able to solder quickly. Further, the soldering irons are a shared resource to any damage that you do to your iron will be felt by everyone that uses it after you. Please mind these tip maintenance techniques carefully.

To solder through-hole components, touch the tip of the iron to the pad on the board at the same time it touches the lead of the component. Apply the solder to the joint, not the tip of the iron. The solder should smoothly adhere to both the pad and the component pin rather than balling up on the component. The connection should appear shiny; a gray color indicates a possibly unreliable "cold solder" joint.

To solder surface-mount components, apply a small drop of solder paste to each pad. Use tweezers to place the component into the desired placement. Use the tweezers to hold the component in place while you touch the tip of the iron to each pad.

If you are in doubt about the quality of your solder joints, ask early on rather than doing all of them first and discovering that your connections are intermittent or unreliable. Some of the components will be soldered close to vias (the small holes through the board used to connect between wiring layers on the printed circuit board). Be sure excess solder does not bridge to the via, creating a short circuit. When you are done, tin the iron one last time to protect the tip before turning it off. You can use a multimeter to test for conductivity across solder joints to be sure they are not cold.

A Safety Note: You may wish to use safety goggles while soldering, especially if you don't wear glasses. Also, be sure to hold the ends of leads as you cut them after soldering so they don't fly into the eye of the person working across the room. Solder contains lead, so wash your hands afterward.

☐ Install the H-Bridge Sockets

First, install the two 16 pin DIP sockets for the H-Bridges at U10 and U11. Place the notches in the socket to line up with the notches on the silk screen. You will later place L293D H-Bridges in these sockets, which can drive simple DC motors through a simple interface.

☐ Install the Resistor Networks

Next you should install the resistor networks at R7, R8, and R9.

R7 and R8 are 8-terminal isolated resistor networks for the LEDs. 'Isolated' means that it contains a set of totally independent resistors. These particular resistor networks are sets of 5 330 Ω resistors. Pins 1 & 2 are one resistor, pins 3 & 4 are another, and so on. Polarity does not matter. You may find it helpful to bend one pin at each end of the resistor network to hold it in place while soldering.

R9 is a 5-terminal bussed resistor network. 'Bussed' means that it contains resistors that are all connected to a single common pin. This particular one is a 5 pin component that has 4 2.2 kΩ resistors. Pins 2, 3, 4, and 5 are terminals of the four resistors. Pin 1 is the common pin. On the board, this is connected to ground. Since R9 is not symmetric, **polarity matters**. Pin 1 is marked on the component with a small circle. Similarly, pin 1 on the board is marked with a square terminal instead of a round terminal. Make sure these are aligned or you may short circuit your board.

☐ Install the PROM and the Oscillator

Next, install the PROM and the oscillator. These parts are surface mount (SMT) parts. Soldering a surface mount device would be easiest if you had three hands: one to hold the part with tweezers, a second to hold the soldering iron, and a third to apply the solder. If you do not have three hands, solder paste is handy. Apply some paste to the pads, then place the part on the board and hold it with tweezers in one hand while using the soldering iron in the other hand to melt the paste., while a silvery appearance covering the pad and pin usually means a good connection.

If you don’t have solder paste, another option is to apply a blob of solder to one of the pads, then place the part on the pad, hold it with tweezers, and remelt the solder with the iron.

Be mindful of how much solder you use while connecting surface mount parts. Using too much solder can bridge between pins, causing shorts. Use solder wick to remove the bridge. Excess solder can also bridge to nearby vias. Finally, extra solder will exert forces on your part as it cools, which makes it harder to position precisely. This is the most difficult SMT part you are installing. You may wish to use a multimeter to check continuity of your joints. If you later encounter an error programming the PROM, check the connections again.

The small PROM is not built to endure extended heat from the soldering iron. Make your connections deliberately and then remove the iron so that you don’t overheat the pins.

The polarity of both the clock and the PROM matter. Be sure the notch on the clock chip aligns with the notch on the board’s silkscreen. Also be sure the dot on the PROM is in the lower left hand corner, diagonally opposite the silkscreened text “MEM.”

☐ Install the Resistors and Bypass Capacitors

Now install the resistors and bypass capacitors. For this board, all of these components are 0805 SMT components. The polarity of these components does not matter so you can insert them whichever way you’d like. However, some of the components have labels, and it is good practice to install labeled components in the same direction, making it easy to read them. See Table 1 for resistor and capacitor values. The resistors and capacitors are stored in drawers in the parts cabinet, and you should collect them from there.

The labels on these components have meaning in the same way that the colored stripes have meaning on through hole resistors. SMT passives are typically labeled with three numbers, for instance 103. This is a notation that is similar to scientific notation which denotes the number composed of the first two digits multiplied by ten raised to the third digit. So 103 indicates 10 times 10 to the third or 10,000. 224 indicates 22 times 10 to the fourth, or 220,000. This number has different units on resistors and capacitors. In resistors it denotes the number of Ohms of resistance and in capacitors it denotes the number of picofarads of capacitance.

The bypass capacitors are important to reduce fluctuations on voltage lines, notably the power supply nets. Large capacitors near the voltage regulators are there to supplement the voltage regulator for spikes of current draw. Though the voltage regulators can supply current sufficient to meet the average demand of the circuits on the board, they cannot respond quickly enough to deliver high frequency spikes in current demand. Such spikes would cause the voltage to briefly droop, if the bypass capacitors were not there to supply the current.

A typical digital system uses several sizes of bypass capacitors. Larger capacitors store more charge and can deliver more current, but react more slowly. Smaller capacitors can rapidly deliver charge to handle short spikes of demand. Some components that have especially demanding current profiles, like the FPGA, need to be bypassed with both large and small capacitors.

Note that the chips and board have some inherent capacitance as well, so the voltage might be stable enough even without bypass capacitors. However, noise-related problems are so difficult to reproduce and fix that it is better practice to put a generous number of inexpensive bypass capacitors on the board than to skimp on bypassing and hope the board works anyway.

Part	**Reference Designations**
1 kΩ resistor	R2-3
10 kΩ resistor	R1, R4-6
0.01 μF capacitor	C1,C4, C7
0.1 μF capacitor	C2, C5, C8, C10
1 μF capacitor	C11
10 μF capacitor	C3, C6, C9, C12-13

Table 1: Resistor and capacitor component values.

☐ Install the Pushbutton Switch

There is one small pushbutton switches that come with your kit, which will be installed at U5. When installing this, you will have to align it in its holes and push until it snaps into location. Do not put your finger directly underneath the board because the sudden popping into position could stab your finger. A thin pair of pliers helps push the legs through. The switch will only align in two positions on the board; either alignment will work.

☐ Install the ADC Sockets

Install the 8 pin DIP socket for the ADC at U8. You will later place an MCP6002 ADC in this socket, which can read analog values and output 10 bit digital values through a serial SPI interface.

☐ Install the DIP Switches and LED array

The DIP switches are to be placed at the bottom of the board at U12. Align the writing on the DIP switches with the silk screen writing on the board.

Insert the LED Array at U9. Align the notch on the part with the square solder pad on the board for correct alignment.

☐ Install the 18 pin Female Header

Install the 18 pin Female Header at H3. This will provide an external interface to access the H-Bridge and the ADC from an external controller such as your Raspberry Pi.

☐ Install the Motor Terminals

Install the motor screw terminal headers at H4. Make sure the terminals are facing away from the board so you can easily access them in future labs. i.e.: you should be able to stick a wire in to the side of the terminal facing away from the board.

☐ Install the JTAG connector

Install the JTAG connector at H2. The connector is symmetrical, so rotation does not matter.

☐ Insert the H-Bridge and ADC

Insert the L293Ds in the H-Bridge sockets in U10 and U11, and the MCP3002 in the ADC socket in U8. The dot on the ICs signify pin 1, and the ICs should be placed such that the dot is in the upper left corner. These components are placed in sockets rather than soldered directly to the board so that they are easily replaceable.

☐ Install the 58-pin Row of Header Pins

This is the *only* part that is installed on the bottom and soldered on the top (it will be inserted into the breadboard). Place the part so that the long pins extend below the board, with the black bumper up against the bottom of the board. The short pins go through the board and are soldered on top. Watch carefully that you do not use too much solder because it may bridge to one of the vias. When you place the header pins, try to keep all of them perpendicularly aligned so that placing the utility board in your breadboard isn't too much of a headache. As the header pins come in strips, you will need to use one long strip and cut one shorter strip.

Now hopefully you should be done with assembling your board. Before proceeding with the lab, you should check all of your solder joints. They should look reliable, filling their specific holes in the board. If some of your soldering looks more like a bubble on the top of the pin, you probably have a bad connection there. Also look for solder jumping across pins or to vias (solder bridges) and cold solder joints (gray, dull joints).

After you've checked your soldering and verified with a multimeter that there is no power to ground short on the board, your board is done!

☐ Place the Utility Board on the Breadboard

Now you have to place your board on the breadboard. It should fit in the right-most side of the breadboard.

☐ Test the Power Supply

After you've placed the utility board, connect a red wire from the red Vin banana plug connector to the upper power row. Then, from this row, connect a wire to the row of pins next to the Vin pin on the utility board. Then connect a black wire from the GND banana plug connector of the breadboard to the second power row (below the red wire). Connect this row to the Gnd pin of the utility board. Make sure these wires do not touch each other.

Turn on a benchtop power supply and check the 0-6 V scale indicates 5 V. Adjust it if necessary. Please keep the supply set to 5 V. However, it is good practice to check the supply before connecting it to your board because the knobs occasionally get bumped and you don't want to connect an incorrect voltage that damages your board or causes erratic operation.

When the power supply is set correctly, turn it off. Get a red and a black pair of banana plugs and connect the power supply to your breadboard (never connect power circuits while the supply is turned on). Turn the supply back on. The power LED should turn on. Feel your FPGA and make sure it does not get very warm. Also check your other components, especially the voltage regulators. The current from the supply should be around 80 mA. Check the 3.3 V output pin and confirm that the voltage regulator is producing the correct value. Also check that the capacitors C3 and C6 next to the 1.2 and 2.5 V regulators have 1.2 and 2.5 V on one of their leads.

☐ Test the LEDs and switches

You can test your LEDs and switches by connecting a wire in the breadboard between one of the switch outputs and one of the LED inputs. Toggle the switch and check that the LED toggles. Repeat to check all the LEDs and switches. Remove the wire when you are done so that it doesn't interfere with subsequent testing of the FPGA. If anything is amiss, turn off the power supply immediately before damaging any parts. Look for solder bridges on your board.

☐ Test the oscillator

Verify that the oscillator was installed correctly by using the oscilloscope to view the oscillator signal. You should use the analog side of the scope. The probes are in the pouch above the oscilloscope. Make sure to attach the probe ground wire to your circuit ground. The clock signal is pin 3 of the chip. Adjust the voltage and time scales so you can see a nice signal. Use either the cursors or the Quick Measure feature to verify that the correct frequency clock (40 Mhz, which corresponds to a 25ns period) is being produced.

☐ Clean Up

Clean up your lab station. Discard the refuse you accumulated while soldering. Tin the iron and turn it off. Please try to keep the lab clean and neat as you work because you share it with many others.

FPGA Design

Your next goal is to write some Verilog modules to test the hardware on your board and operate a 7-segment display. By writing a module to control the LEDs based on the switches and clock, you will show that these components work and also that the FPGA, PROM, and power supply are functional. The system should have the following inputs and outputs:

clk	input	the 40 MHz clock
s[3:0]	input	the four DIP switches
led[7:0]	output	the 8 lights on the LED bar
seg[6:0]	output	the segments of a common-anode 7-segment display

The following tables define the relationship of the LEDs to the switches and clock:

S0	LED0	LED1
0	OFF	ON
1	ON	OFF

S1	LED2	LED3
0	OFF	ON
1	ON	OFF

S2	LED4	LED5
0	OFF	ON
1	ON	OFF

S3	S2	LED6
0	0	OFF
0	1	OFF
1	0	OFF
1	1	ON

LED7
Blink at ~2.4 Hz

Table 3 – I/O Requirements

The 7-segment display should display a single hexadecimal digit specified by s[3:0]. Remember that you will be using a common anode display. The anode (positive terminal) of all of the LEDs is tied to 3.3 V through a single ("common") pin. Each segment's cathode (negative terminal) is connected to a pin. Therefore, you will need 7 separate control signals. Remember that a logic 0 applied to the cathode will turn on the segment. The segments are defined as shown below. Let seg[0] be A and seg[6] be G.

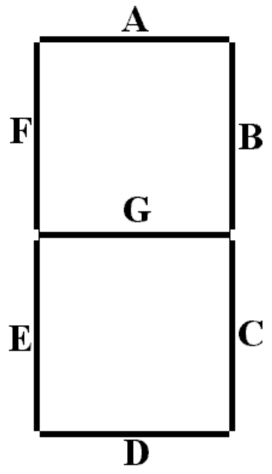


Figure 2 – Seven Segment Display layout

Launch the Quartus II software (version 15.0 Web Edition, find it by searching for 'Quartus' in the start menu search bar) and start *New Project Wizard* from the File or startup menu.

Your first decision is where to keep your files. Ideally, they would go on your Charlie home directory. However, Quartus can be painfully slow accessing files on the file server over the network. A better choice is to keep your files on the local C drive while you are working and then to copy them back to your Charlie account before you log out. Please clean up after yourself by deleting the project from the C file when you are done, and remember that it is an honor code violation to refer to somebody else's code that was left on a computer. Note that some CAD tools have trouble with filenames having spaces and other special characters, exceeding 8 characters, or in paths not starting with a letter drive. The best way to avoid these problems is to choose short alphanumeric filenames.

Create a working directory for the project such as c:\e155\xx\lab1_xx where 'xx' area your initials. Name the project lab1_xx and click Next.

Select *Empty Project* on the Project Type page and then click *Next* at the Add Files page, since we will not be adding any existing files to the new project. Now we will tell Quartus which FPGA we are using. Under Family, select **Cyclone IV**. Under Available Devices, select **EP4CE6E22C8**. If you ever forget which device to use, the part number is written on the FPGA.

On the next page, select Pick *ModelSim-Altera* as the Simulation tool and *SystemVerilog HDL* as the format. Click Next and verify the settings for the project. When you are satisfied, click Finish.

Choose *File* -> *New* and create a SystemVerilog HDL file. Save the file as lab1_xx.sv in your project directory and check the box to add the file to the current project. Create modules to perform the functions described above. The 7-segment display decoder should be combinational logic. Use a reasonable amount of hierarchy, but be mindful that SystemVerilog is an expressive language and that overly deep hierarchies are hard to follow. Name the top level module lab1_xx. The 7-segment display code, for example, will be reused in future labs, so it should be a module of its own. Every module should begin with a comment section that includes your name and email address, the date of creation, and a brief summary of its purpose, so that somebody else can understand what the module does and get ahold of you if they need help. Comment the modules as appropriate.

Logic Simulation

The next step is to simulate your logic with ModelSim.

Check that Altera has the correct path for ModelSim by invoking *Tools → Options*. Under EDA Tool Options, check that the ModelSim-Altera path is

```
C:\altera\15.0\modelsim_ase\win32aloem
```

and set it if necessary. EDA stands for Electronic Design Automation and is the industry name for computer-aided design (CAD) tools for electronic design.

For unknown reasons, Quartus wants you to compile your code before simulating. To do this, select *Processing → Start Compilation* to compile your design. Ignore warnings about missing pin assignments or timing violations.

Invoke Modelsim by choosing *Tools → Run Simulation Tool → RTL Simulation*. RTL stands for Register Transfer Level code (your SystemVerilog code). The ModelSim window will open. Open your sv file and compile it using *Compile → Compile …* Get in the habit of watching the transcript window to look for errors and to famiarlize yourself with what a good run looks like. If you see errors, close ModelSim, correct your Verilog code in Quartus, and reopen ModelSim. You may also edit your file from ModelSim if you are careful to import changes back to Quartus.

At this point it is possible that you will see the error `"** Error: (vlog-19) Failed to access library 'work' at "work"`. In that case, open your SystemVerilog file in Modelsim, click the "Compile" button, find your file in the resulting dialog box, click compile, and click "yes" when prompted to create the library "work".

In ModelSim, simulate your design by choosing *Simulate → Start Simulation*. Click on the + symbol next to the work library and select your code (lab1_xx).

If the wave pane isn't open, open it by choosing *View → Wave*. View all of the inputs and outputs of your design by selecting them in the Objects window and dragging them to the Waves window. In a more complicated design, you may wish to examine internal signals as well.

Manually test your design by forcing the inputs to specific values. In the transcript window, type:

```
force s 0000
run 100
force s 0001
run 100
force s 0010
run 100
...
```

You should see the led and seg outputs displaying appropriate values. Note that the clock is not driven and you have not reset any registers so led[7] will be an x.

Check the outputs against your expectations. If you find any discrepancies, fix the code and resumulate. A helpful shortcut to avoid restarting ModelSim is that you can edit the modue by finding it in under "work" in the library pane, right clicking, and choosing *Edit*. Make your fixes, then right click again and choose Recompile. Then type `restart -f` in the transcript window to restart simulation without having to set up the waveforms window again. When you return to Quartus, you'll find your corrected code.

Pin Assignment

Next, assign pins to relate the signal names in your Verilog code to physical pin numbers on the FPGA. Launch *Assignments → Pin Planner*. A table listing all inputs and outputs for the project should appear. Under Location, type the pin number to associate with the given signal. For example, SW0 is mapped to pin P76, so enter PIN_76 as the value for s[0].

The FPGA pinouts are shown in the Utility Board Schematic. Most of the user input/output (I/O) pins are tapped out to the headers and labeled on the board silk screen. Some have special functions; for example, FPGA pin P68 is connected to LED0. The clock is connected to pin P88 and not to a header pin because a 40 MHz clock would be degraded by the parasitic capacitance and inductance of the breadboard.

The pin numbers for the LEDs and switches are marked on the board's silkscreen. For the outputs for the 7-segment display, you may select any I/O pins you'd like. For example, you could use pins 51, 52, 53, 54, 55, 58, and 59 since they are contiguous on the board.

Note that the Pin Planner defaults to assuming that each bank of I/O pins receives 2.5 V and uses 2.5 V outputs. However, our system uses a 3.3 V I/O. Fortunately, leaving the voltages at their default value in Pin Planner works and generates 3.3 V outputs.

7-Segment Display Circuit

The 7-segment display will be used throughout the class for general output of numbers. In this lab assignment, though, it will be used to output the hexadecimal number entered by the user through the DIP switches.

Each segment of the display works as an independent LED. Therefore, the same current-limiting concern with the LEDs applies to the display as to the on-board bank of LEDs. You can limit the current into each segment of the display the same way you did for the LEDs on board, adding a suitable resistor to provide roughly 5-20 mA of current. You can find resistors and other such components in the supply cabinet or in the stockroom.

Consult the data sheet for the pinout of the common anode dual seven segment display. All seven segments share the same anode, which should be connected to VCC (3.3 V). Each of the segments has its own cathode, which can be pulled to 0 to turn on the segments.

Be sure to turn power off before wiring circuits on your board. You can choose either side of the display to use in this lab. After deciding on which side to use, you will need to connect the V_{DD} pin of that side (either V_{DD1} or V_{DD2}) to 3.3 V. Then connect the input pins of the same side of the display to the header pins you chose. Remember to add suitable resistor between each of the inputs to the display and the header pins. These LEDs are common anode LEDs. That is, all the anodes from the LEDs are connected to a single V_{DD} (V_{DD1} or V_{DD2}). You are driving the cathode of each LED. Given this information, you might need to modify your Verilog file. Do so in the simplest way possible.

Generating the FPGA Configuration Files

Now you will synthesize your HDL into a programming file to be transferred onto the FPGA. This outputs an SRAM Object File (.sof) in your project directory that can be used to program the FPGA directly over JTAG. Be sure your SystemVerilog files are saved, and choose *Processing* □ *Start Compilation*. To help sort the many messages that the compilation process generates, click a tab under the Message area to see only that type of message. If compilation is successful but generates warnings, check the Warning and Critical Warning tabs for errors relevant to your design. Warnings about incomplete I/O assignments may be ignored if you have in fact assigned all relevant I/O pinsMissing Synopsis Design Constraints file warnings and timing analysis violations may also be ignored.

Note that Quartus seems to crash from time to time while compiling. If it does, restart and try again. If you find a consistent pattern of what causes the crashes, please let the instructor know.

Launch *Tools* □ *Netlist Viewers* □ *RTL Viewer* and examine the RTL schematic of your design. This shows the logic synthesized from your Verilog design. Ensure the hardware matches your expectations.

Look at the Compilation Report tab. In the Flow Summary, you should see a total number of registers and pins that match your expectations. Under Analysis & Synthesis, you can see how the logic blocks and registers are broken down in each module. Under Fitter, the Pin-Out File should match the pin assignments you intended.

Programming the FPGA via JTAG

Next, you will load your design onto the FPGA and PROM with an Altera USB Blaster programming device and the chip's JTAG[1] interface. Run *Tools* □ *Programmer* to bring up the Programmer window. In the top left corner, check that *USB Blaster* is selected and use *Hardware Setup* to choose it if necessary. In the top-right corner, set *Mode* to JTAG. Your lab1_xx.sof file should appear already in the programmer window, and list EP4CE6E228C as the associated device. If it does not appear, click the Add File… button and find the file by hand in your project directory. Be sure the Program/Configure box for your .sof file is checked.

You are now ready to program the FPGA. Connect the USB Blaster to the JTAG port with the ribbon cable leading off to the right of the board. Turn on the power supply. In the programmer window, press Start to begin the process. Check the Messages pane for programmer output, and verify that there were no errors. Look for "Configuration succeeded" and "Successfully performed operation(s)" messages.

At this point, with your design loaded, you should see the hexadecimal digit currently set on the switches displayed on the 7-segment display. Debug your design until it is fully functional. Good luck!

If you have problems, the following checklist may help:

1. Check the voltage outputs on each of the voltage regulators. Also check it on the adjacent capacitors (C3,C6,C9) to detect a bad connection between the regulator output and the PCB. This is one of the most common soldering problems.
2. Check that the power supply is hooked up correctly. Measure with the volt meter that you have 5 volts between the V_{in} and GND pins on your FPGA board. Check that the current out of the power supply is less than 200 mA. If the current is high, look for

[1] JTAG stands for the *Joint Test Access Group.* The JTAG interface is a serial interface used to communicate with chips on a board using a few pins during testing and configuration. JTAG is also called Boundary Scan.

power/ground shorts on your board. Otherwise, if the voltage is incorrect, adjust the power supply to produce the correct output.

3. If the JTAG programmer indicates “Program Failed,” first, make sure your board is powered, close the Programmer window and try again. Sometimes programming fails if you power on your board after you open the Programmer window. If programming still fails the problem could be in your board or in the programmer. If programming works on a classmate’s board using the same programmer, you may have a bad solder joint or other board error. If the programmer doesn’t work on multiple boards, please report it to the instructor.
4. If your power is good but the FPGA still refuses to respond, check the NCONFIG signal (on the RESET button). It should be 1 when the button is not pressed.
5. If the LEDs flicker on and off when you tap on the board or toggle switches, you probably have a loose connection or marginal solder joint somewhere. Two common problems in the past are poor banana plugs that are too small for the recepticals and inadequate solder joints on the voltage regulators.
6. If the LEDs do something but do not function correctly, the problem is likely in your Quartus schematics or implementation. Check the implementation report file that the pin mapping is what you want. Check for any logic bugs.
7. When exiting the programmer window, don’t save anything. That way the design programmed to the PROM or FPGA is always the most recent build.

Programming the PROM with the Serial Flash Loader

Programming over JTAG only changes the volatile SRAM of the FPGA itself and does not write to the non-volatile PROM. The design will only stay on the FPGA until it is reset or power is disconnected (try it!). Programming the PROM directly would require adding an additional programmer port to the board, connected to the Active Serial (AS) interface between the PROM and the FPGA. Instead, it can be programmed over JTAG through the FPGA, using the Serial Flash Loader function in Quartus. JTAG is used to write a temporary design to the FPGA that effectively bridges the JTAG and AS interfaces, allowing the USB Blaster to talk to the PROM without being physically connected to it.

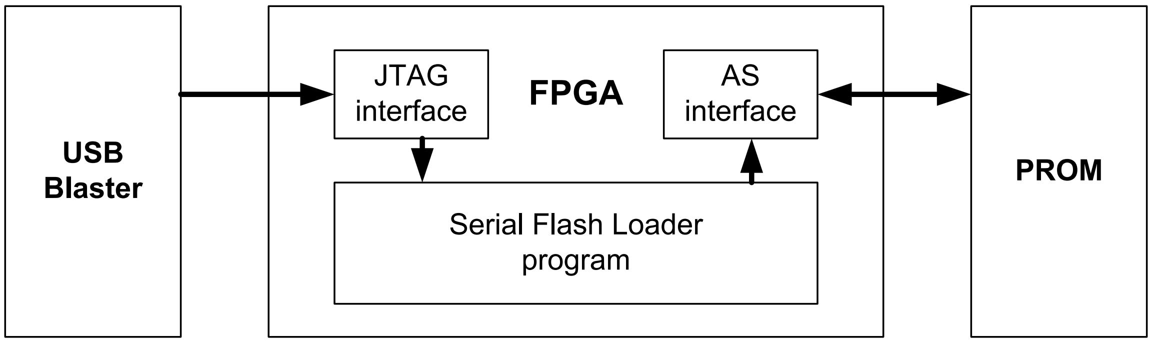


Figure 3 – Serial Flash Loader block diagram. A program bridges the JTAG and Active Serial interfaces to let the USB Blaster talk to the PROM.

When debugging your design, simple JTAG programming has advantages, namely that it is a much faster process. Losing the contents of the FPGA with every reset is less important if your design is being changed with every test. The Serial Flash Loader is much slower than JTAG programming and requires extra steps to complete, but should be used whenever the design is unlikely to be changed in the short term.

Always write your finished labs to the PROM before checkoff. Also remember that whatever design you last programmed to the PROM will be loaded immediately when the FPGA turns on. Forgetting this could cause you massive confusion when debugging, especially in Labs 5-7 when old code on the FPGA might interfere with pins you are trying to use with your Raspberry Pi.

Compile your project to generate an up-to-date .sof file. Next, you will convert the .sof to a JTAG Indirect Configuration (.jic) file for use with the Serial Flash Loader. Choose *File → Convert Programming File*s. Set the programming file type to *JTAG Indirect Configuration File* and the configuration device to *EPCQ16*. Pick a simple file name, such as lab1_xx.jic.

Under Input Files to Convert, select SOF Data and click Add File. Find your lab1_xx.sof. It is in the output_files directory. Press OK. Now highlight Flash Loader and click Add Device. Select Cyclone IV E in the left pane, then EP4CE6 in the right pane, and click OK. Click *Generate* and verify the .jic was generated successfully, then close the window.

Launch the Programmer and remove your old lab1_xx.sof from the programming file list. Now add the lab1_xx.jic file you just generated. Check both Program / Configure boxes and click Start. Watch the Messages pane for any errors. When the process completes, your design is loaded on the PROM. It will automatically be loaded onto the FPGA after reset or power-on until the Serial Flash Loader is used again to replace it.

If the Messages pane displays the error “Can't recognize silicon ID for device 1,” then it is likely that there is a bad joint on your PROM. Inspect the solder carefully, and reapply heat to the junctions to make sure there is good contact.

If you want to erase the EPROM, change the EPCQ16 option from Program/Configure to Erase and click Start again.

When you are finished, save you files somewhere (such as your network drive or a USB key) and remove the files from the local machine. Good luck!

You will test the ADC and H-Bridges in future labs.

What to Turn In

Turn in a lab report with the following sections:

- **Introduction**: Briefly explain what was done.
- **Design and Testing Methodology**: Explain how you approached the design of this assignment from both a software and a hardware standpoint (as appropriate). Include how you tested your design. These test should convince the reader that the requirements of the assignment have been met.
- **Code and Schematics**: Include your Verilog code and schematics of your circuits.
  - Quality and clarity of your code is import. Make sure it is well commented.
  - Schematics of your breadboarded circuits should be sufficient for another engineer to understand and reconstruct the circuit on the breadboard. Always use standard symbols for standard components such as resistors, switches, transistors, diodes, etc. The reader shouldn't have to open Quartus to relate your Verilog code to the schematic. Don't assume that the reader has memorized the pinouts of any chips. There is no need to draw any of the circuitry on the board; just refer to it by the pins number and name.
- **Results and Discussion**: Did you accomplish all of the prescribed tasks? If not, what are the shortcomings? How might you address them given more time? As appropriate, how did the design perform (ex. How fast/accurate/reliable was it?). Is there anything you would do differently if you were to redo the lab? Is there anything else interesting worth mentioning?
- **Conclusions**: Briefly summarize what was done and how it performed.
  - How many hours did you spend on the lab? Any comments, suggestions, or complaints about the assignment? This will not count toward your grade, but will help refine the lab for the future.

The report does not need to be long, recall that the syllabus limits you to one page, but it should be complete. Some individual questions may not apply to this particular lab, but are listed to give you a general idea of what is desired. Future reports will follow a similar format and the questions may be more applicable in these instances.

Have your lab checked off by the instructor. You will need to demonstrate that the board and 7-segment display operate correctly. You will also be asked a question about some part of the lab or your board. You should be thoroughly familiar with all of the lab and the components of your board to be able to answer the question. The oral exam is typically in the form of a "Fault-Tolerance Question." What would happen if a particular wire is broken or a pin is shorted to V_{CC} or GND? Be prepared for any other questions about your lab, however.

Acknowledgments:

This lab manual and previous utility boards have been developed by Profs. Matthew Spencer, Matthew Watkins, David Money Harris, and Sarah Harris, Alex Alves '16,

Gourav Khadge '15, Leo Altmann '11, Christian Jolivet '11, Elizabeth Reynolds '02, Marty Weiner '02, Fernando Mattos '00, and Eliot Gerstner '99.

Appendix: FPGA Initialization Sequence

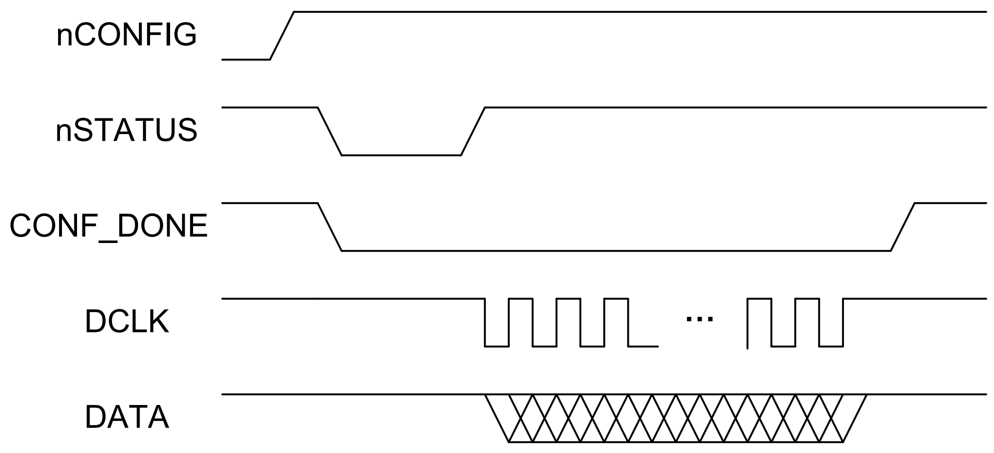
If your FPGA board is assembled incorrectly or your PROM is programmed wrong, you may find that your FPGA fails to initialize. This sheet documents the initialization process to help you track down your problem. The initialization sequence is described in more detail in pages 8-6 through 8-10 of the Cyclone 4 Device Handbook (vol. 1).

The MSEL[2:0] pins on the FPGA are used to select a configuration scheme. They are connected to GND, VCC and GND respectively to indicate Active Serial configuration at 3.3V levels.

When the FPGA powers on or you release the reset switch (pulling nCONFIG from low to high), the device begins initialization, pulling nSTATUS and CONF_DONE low while clearing internal memory and preparing for configuration. In < 200ms, nSTATUS is then pulled high to indicate the start of configuration.

During configuration, the FPGA generates a 40 MHz clock via an internal oscillator and transmits the signal via DCLK to the EPCQ16 PROM. The PROM responds by sending serial configuration data back over the DATA connection. The FPGA has 3,000,000 bits of internal state, so serial configuration via PROM should complete in about 100ms. When configuration is complete, CONF_DONE is pulled high again, and the device enters user mode < 0.5ms later. If CONF_DONE is never asserted or nSTATUS is pulled low after initialization completes, a configuration error has occurred. Check the soldering on the FPGA and PROM. CONF_DONE is also output to the second LED on the 10LED array.

The configuration waveforms are shown below:


nCONFIG
nSTATUS
CONF_DONE
DCLK
...
DATA