Lab 1: Development Board Assembly and Testing

1 Introduction

In this lab you will assemble and test your µMudd Mark VI daughter boards and test them with the Nucleo F401RE microcontroller unit (MCU) and MAX1000 field-programmable gate array (FPGA) development boards. There are two daughter boards: an Arduino R3 form factor shield which interfaces the MAX1000 board with the MCU board and a T-shaped connector board which is used with a ribbon cable to enable easy connections to a breadboard. You will be using these boards for the remainder of the semester, so it is very important to assemble them properly. But don't panic, you'll quickly learn to troubleshoot and repair issues.

2 Learning Objectives

By the end of this lab you will have...

- Learned how to solder
- Assembled and tested your µMudd Mark VI board
- Written a Verilog module to control LEDs and a 7-segment display
- Programmed the FPGA with your Verilog code
- Gained confidence in building, assembling, testing, and debugging circuits
- Interfaced the 7-segment display to the board

3 Before You Start

Before you start working on this lab, you should familiarize yourself with the documentation for both the MAX1000 and Nucleo-F401RE boards. The website has PDF links for the MAX1000 User Guide and the Nucleo-F401RE User Manual. These two documents contain information that will be helpful for answering questions that you may have during the course of this lab.

4 Requirements

Follow the steps in this guide to test and assemble your µMudd Mark VI board. Write some Verilog code to exercise the FPGA using the switches, LEDs, and a 7-segment display to ensure your board is operational. Simulate and synthesize your code, then upload it to the Flash memory and re-test the board. Hook up a 7-segment display and demonstrate that it works.

5 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 the Cortex-M4 microcontroller to write programs in assembly language and C that can interface with external hardware and the FPGA.

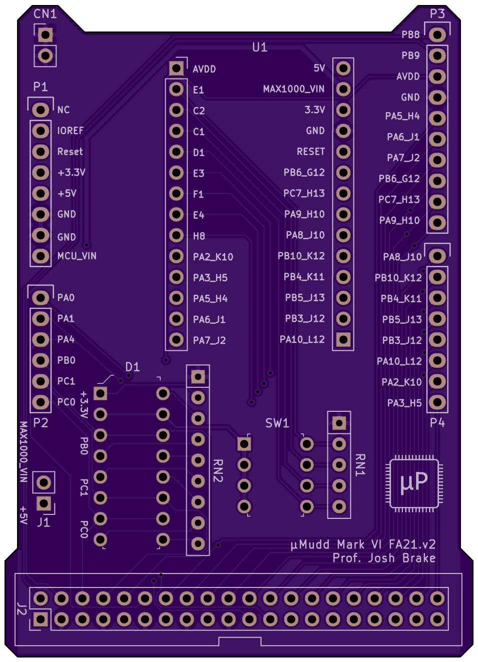

Figure 1: Shield board PCB

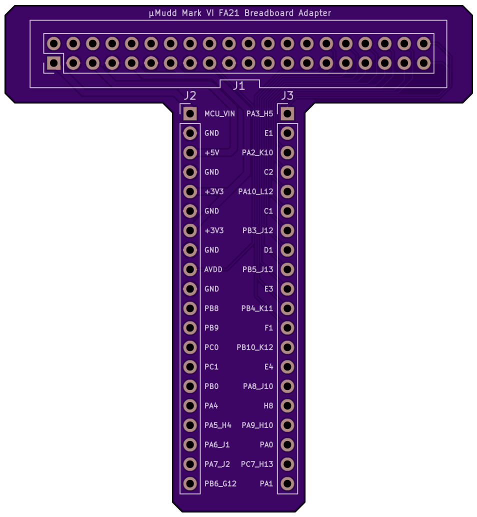

Figure 2: Breadboard adapter PCB

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 switches and LEDs. There are several LEDs and a button on the MAX1000 board itself and a sliding DIP switch array with 4 switches on the µMudd shield.

The FPGA comes in a 169-Ball Low-Profile Fine-Pitch Ball Grid Array (LFBGA) package. It is the large black square chip in the center of your MAX1000 board. You'll notice that there are no visible pins around the sides of the chip since they are arranged on the bottom of the chip in an array of tiny solder balls. This is convenient for saving space, but makes debugging a challenge since you cannot directly access any of the physical pins to probe them. This is one example of a surface mount (SMT) part. The pins on SMT parts do not go through the printed circuit board (as with the other through-hole parts you'll be working with) but 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. You can solder some SMT components by hand, but smaller or more specialized parts (such as the tiny resistors you see on the MAX1000 board or the ball grid array FPGA chip) can be challenging or impossible to solder by hand and need specialized equipment to solder.

The FPGA you will use in this course is the Intel MAX10 10M08SAU169C8G, which contains 8,000 Logic Elements. You can find the datasheet on the class web page. You will need to become familiar with the internals as you progress in this class.

The two daughter boards were designed to maximize your access to the capabilities of the FPGA and MCU from a breadboard. There are several MCU and FPGA pins which are connected on the shield PCB to facilitate easy and high-fidelity connections between the two devices. The cobbler board is designed with two rows of headers that plug into a breadboard and give you access to many of the FPGA and MCU pins. Furthermore, the pins are labeled for your convenience. The other characteristics of the board will be explained later, in the Discussion section of this guide.

The board schematics are included at the end of this manual. Study these schematics to understand how the boards work because you will need this information to use and debug your board and are likely to be asked about some subsystem during your checkoff.

6 Component Discussion

The figures at the end of this document include both the bill of materials (BOM) and the board schematics. Major components include the Nucleo F401RE board, MAX1000 board, header pins, resistors, DIP switches, and LEDs. What follows is more information about 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 they operate. The subsequent section will guide you through the assembly process.

6.1 Power Supply

There are several ways that power can be supplied to the board. Be very careful when connecting power to the devices to ensure that you supply voltages that are within the maximum operating ranges. Supplying a board with voltages outside of the maximum operating conditions is a recipe for a burned out board.

Both the STM32F401RE and MAX1000 boards have their own on-board voltage regulators. The MCU can take voltages between 7 - 12 V on `MCU_VIN` and the MAX1000 takes a voltage of 5 V on `MAX1000_VIN`. Details on powering the devices can be found in section 6.3.2 of the Nucelo-64 User Manual (MCU) and section 3.3.10 in the MAX1000 User Guide (FPGA).

Both devices can be powered independently via their own USB connections, but also can be reconfigured to be powered via a direct connection to a voltage source. The recommended way to power the combined boards is by providing power to the Nucleo board either via USB or the `MCU_VIN` pin by selecting the proper position for jumper JP5. To power the MCU board via USB, JP5 should be configured as U5V. If VIN or E5V is to be used, JP5 should be set to select E5V. See Tables 7 and 8 in section 6.3.2 of the Nucleo-64 User Manual (UM1724) for more information. After properly configuring the power supply for the Nucleo board, the jumper JP1 on the µMudd Mark VI Shield should be connected (i.e., shorted) to connect the `MAX1000_VIN` to the +5 V output of the Nucleo board. Connecting the boards in this way enables you to power both devices from the MCU power supply instead of needing two separate USB connections.

6.2 Nucleo-F401RE Development Board

The Nucleo-F401RE development board provides a convenient platform to test and develop embedded systems applications. The board is focused around an STM32F401RE MCU and provides voltage regulators to provide power to the chip, various breakout headers (both Arduino and Morpho headers) which allow access to the pins of the microcontroller chip, a reset button and user input button, configuration jumpers, and an embedded ST-LINK programmer/debugger.

6.2.1 Programming

The STM32F401RE has an embedded ST-LINK/V2-1 programmer and debugger as described in section 6.2 of the Nucleo-64 User Manual. The main purpose of the ST-LINK is to provide the ability to interface with the serial wire debug (SWD) interface of the MCU via USB.

There are a variety of integrated development environments (IDEs) which can be used to interact with the device including PlatformIO and Keil µVision. The IDEs provide a helpful wrapper around compiler toolchains such as GNU Compiler Collection (GCC) toolchain to facilitate easy programming and debugging. In this course you are welcome to use any IDE you would like, although instruction and support will be focused around PlatformIO which is based on the GCC toolchain.

6.3 MAX1000 Development Board

The MAX1000 board is developed by Arrow and focused around an Intel MAX 10 10M08SAU169C8G field programmable gate array (FPGA). Figure 1 of the Intel MAX 10 FPGA Device Overview explains the naming of the chip.

The MAX 10 FPGA is very similar to the Cyclone IV architecture that is used to introduce FPGAs in E85. At its root, the FPGA consists of Logic Elements (LEs) which can be internally connected within the FPGA. Each LE consists of a 4-input look-up table (LUT) and a programmable D flip-flop.

The 10M08SAU169C8G is member code 08 and thus according to Table 4 of the Intel MAX 10 FPGA Device Overview, it includes the following resources.

Resource	**Quantity**
Logic Elements (LE)	8000
M9K Memory	378 Kb
User Flash Memory	1376 Kb
18 x 18 Multiplier	24
Phase-locked Loop (PLL)	2
Analog to Digital Converter (ADC)	1

Table 1: 10M08SAU169C8G Resources

6.3.1 Programming

The FPGA is programmed using the onboard Arrow USB programmer via Quartus. After setting up and compiling the Quartus project, Quartus will generate bitstream files which are used to configure the FPGA. More details will be provided later in this lab writeup.

6.4 Header Pins

The boards provide several sets of header pins that tap out signals from the FPGA, MCU, LEDs, switches, and power and ground.

Item	Quantity	Manufacturer P/N	Relevant Identifiers
14-pin female header	2	PPTC141LFBN-RC	Shield: U1
40-pin male vertical keyed header	2	SBH11-PBPC-D20-ST-BK	Shield: J2; Breadboard adapter: J1
40-pin male breakable header	2	PREC040SFAN-RC	Shield: J1; Breadboard adapter: J2, J3
Arduino stackable headers kit	1	PRT-11417	Shield: P1, P2, P3, P4

Table 2: Header pin components

The two 14-pin headers are used to connect the MAX1000 FPGA board to the shield. You will also need to break off and solder two 14-pin male header pin strips to the MAX1000 board.

The Arduino headers are stackable, with pins which extend below the board and receptacles which are on the top of the board. These should be inserted from the top and then soldered on the bottom of the board. Make sure to match the correct length pin with the corresponding location on the shield board (P1, P2, P3, or P4). Also make sure to not clip the leads on these later on since you need them to stack on to the MCU board!

There are also two places for 2-pin headers to be installed on the shield at CN1 and J1. These should be installed on the top of the board with the short part of the header facing down through the board and should be soldered from below. There is a small plastic jumper connector which can be connected to either CN1 or to J1 when in use. CN1 is simply used as a place to store the jumper connector when not in use; both pins of CN1 are connected to ground. When connected to J1, the VIN input of the MAX1000 board (`MAX1000_VIN`) is connected to the +5 V output of the Nucleo board. This enables the whole system to be powered from a single power source through the Nucleo boards voltage regulators.

The 2×20 pin vertical keyed headers are used to attach the ribbon cable that connects the shield and the breadboard adapter. Be aware that these are keyed and so they must be installed in the correct orientation. The opening in the plastic on the header (the key) must be properly aligned with the divot shown on the board silkscreens.

6.5 Bar LED Array

There is a 4-segment light-emitting diode (LED) array on the board. The top LED is connected to the +3.3 V output of the Nucleo board to indicate when power is applied. The other 3 LEDs are connected to GPIO pins on the MCU PB0, PC1, and PC0.

There are a few concerns that must be addressed whenever using LEDs. One is that 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, overheating the driver, or simply wasting power. A 330 Ω resistor does the job nicely, allowing 3.3 V/330 Ω = 10 mA of current to flow. (In actuality, the current is about half of this. There is a voltage drop across the diode (about 1.7 V for red), which makes the actual current (3.3 V-1.7 V)/330 Ω = 4.8 mA. This is still around the desired 5-20 mA range and so still works).

6.6 Dual inline package (DIP) switches

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

6.7 Resistor Arrays

The resistor networks or arrays are convenient ways to package a series of resistors into a compact package. Resistor arrays come in two varieties: isolated and common bussed. Isolated arrays contain sets of individual resistors grouped together in a single package where each terminal of each resistor is exposed. In comparison, bussed resistor arrays connect all of one side of the resistors together to a single pin. Common bussed resistors are thus more compact and pin-efficient compared to isolated resistors (for N resistors, common bussed arrays have N+1 pins whereas isolated arrays have 2N pins). Both resistor arrays on the shield board (RN1 and RN2) are common bussed array. Since one of the pins is common to all resistors, you should take care to make sure the orientation of the array is correct. The common pin, indicated by a dot on the side of the array, should be connected to the corresponding pin 1 on the board which is indicated by a square pad.

6.8 Ribbon Cable

The 40-pin ribbon cable is used to connect the shield with the breadboard adapter PCB. The cable is keyed (i.e., has the bump on one side) to ensure that it can only be inserted the correct way.

6.9 Header Receptacle

Although the ribbon cable is keyed to ensure correct alignment, it is important to make sure that the header is installed correctly. Make sure to line up the opening in the plastic shrouding with the notch marking on the PCB silkscreen.

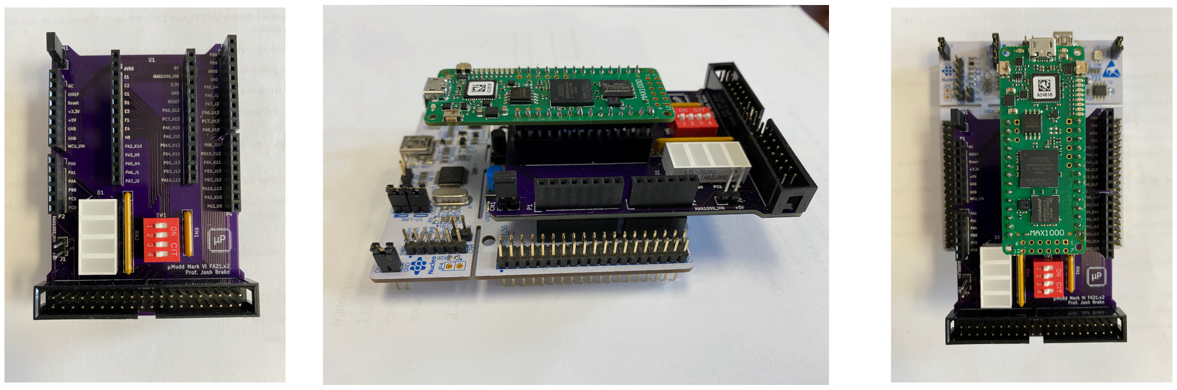
Figure 3: Photos of finished boards.

7 Assembling the Boards

This section will give you tips on how to assemble the 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.

Figure 3 shows a few photos of the boards to give you an idea of what your final product should look like.

7.1 Getting Started

The board has two sides. The component side, with the white silk screen markings indicating component placement, should go up. All components except the pin headers on the breadboard adapter and the MAX1000 board are placed on the component side. The through-hole parts are soldered on the reverse solder side.

Using a multimeter, measure the resistance between the pins labeled `MCU_VIN` and `GND` on the board, and verify that the resistance is nearly infinite. 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 `MCU_VIN` and `GND`. If it is ever less than 10 Ω, you've introduced a short and should debug it before continuing.

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.

For 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 possible unreliable "cold solder" joint.

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

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.

7.2 Soldering the MAX1000 board headers

The first step is to solder 2, 14-pin headers onto the MAX1000 board so that you can plug it into a breadboard or the shield. First break off two 14-pin sections from the 40-pin strips of headers in your kit. The best way to do this is to use a pair of pliers (available in the toolkits at each soldering station in the lab) to grab the last header in the 14-pin segment and then carefully apply pressure to snap the headers at the correct location.

The easiest way to solder these headers is to place the long half of the headers into a breadboard at the right width and then place the MAX1000 board on top. Then you can easily go down the line and solder each pin on the board without needing to worry that it will move around on you.

After you have finished this step, you are ready to move on to the shield PCB.

7.3 Soldering the Shield PCB

For this PCB we will start at the center of the board and move outward to make it easier to access the components without others getting in the way.

7.3.1 MAX1000 Shield Headers

First, take the 2, 14-pin female headers which are for the MAX1000 board to connect to. These should be placed in the holes for components U1. It is at this point that you will likely experience a common issue when soldering: the need for an extra hand or two. In order to solder the components on, you can use one of the PCB clamps or a "third hand" tool in the lab to hold the PCB. Then, hold the component on its correct location on the PCB with one hand while you solder it with the other. You can either ask a friend to help you apply solder to the joint while you work the iron, but if no one is available, you can simply put a small amount of solder on the tip of the hot iron and then apply it to one of the pins of the header. In order to make sure the component is attached firmly, one good strategy is to solder the first and last pin of the component first, ensure that the alignment looks good, and then proceed to solder the rest of the pins in between.

After you have finished soldering the headers for U1, you can test that it fits well by plugging in your MAX1000 board. It may require a bit of force to plug and unplug, but it is ok to press firmly when plugging it in. Do be careful when unplugging it to apply the force slowly so that you remove the board easily and without bending the pins.

7.3.2 Resistor Networks, DIP Switches, and Bar LED

Next install the resistor networks RN1 and RN2. Make sure to orient both arrays such that pin 1 of the arrays (indicated by a dot on the side of the array package) is oriented properly and matches up with pin 1 on the component footprint which is indicated by a square copper pad.

The DIP switches and bar LEDs should be soldered next. Again, make sure that the components are properly oriented. The datasheets provide information about the pin orientation and arrangement, so check if you have any doubt. Most components have some recognizable feature on the package to indicate the location of pin 1 or allow you to figure it out such as a notch, beveled corner, or physical marking such as a dot. Pin 1 on the PCB is indicated by a square copper pad as compared to a circular one.

7.3.3 Stackable Headers and Jumpers

Next install the stackable headers (P1, P2, P3, and P4) and jumpers (CN1 and J1). The stackable headers are already in the correct sizes so just make sure to put the right size header in the right spot. The two, two-pin headers CN1 and J1 should be snapped off one of the 40-pin male headers and installed on the board so that the long side of the headers and the plastic shroud is on the component side of the board. DO NOT clip the leads on P1-P4!

7.3.4 40-pin Keyed Header

The last component to install on the board is the 2×20 pin header for the ribbon cable to connect the shield and breadboard adapter. Make sure to correctly orient the opening of the plastic shroud with the marking on the silkscreen. Be careful to make sure the connector is aligned properly before soldering all the pins by soldering one pin on either end of the header first, then checking to make sure it is aligned properly and flush with the board, and then soldering the remaining pins. Soldering is a reversible process, but desoldering is roughly 100× more challenging. Similar to the old adage "measure twice, cut once", "align and mount twice, solder once."

8 Testing the Boards

At this point, your boards should be fully assembled. Congratulations! See Figure 4 for a few photos showing the completed boards for reference.

To test your board, you will upload a simple circuit on the MAX1000 FPGA board which generates a ̃1 Hz clock signal and blinks two LEDs on the MAX1000 board. LED1 (Pin A8) is connected to a simple clock divider and indicates that your FPGA programming is working properly. LED2 blinks based on an output signal from the MCU. The MCU reads in the slowed clock signal and then echoes that signal back to the FPGA as well as blinking one of the LEDs on the μMudd shield board. This will ensure that the boards are correctly interfaced and provide a simple example of how to program both the FPGA and MCU. Sample code for the test (MCU and FPGA) is provided on the course website.

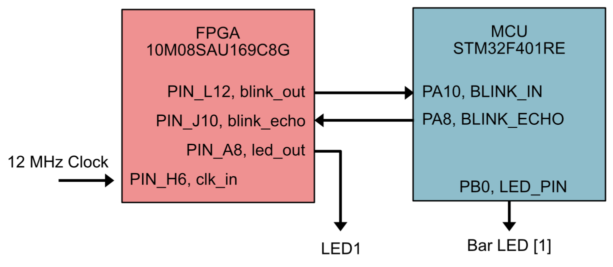

Figure 5: Block Diagram of Blink Test Circuit

8.1 MCU Programming

To program the STMF401RE MCU, we will use PlatformIO. PlatformIO is an integrated development environment (IDE) which is used as an extension to Visual Studio Code (VS Code). It uses a build system based on the GNU Compiler Collection (gcc) which is an extremely popular and well developed open-source collection of tools to compile code. In future labs we will talk in more detail about the compilation process of going from C code to machine code running on your MCU, but for now we will deal with it at a higher level of abstraction.

To begin, install VS Code on your machine if you do not already have it installed. Then, inside VS Code navigate to the extensions menu, search for, and install PlatformIO. It make some time for PlatformIO to download and install.

After installing, PlatformIO open the main PlatformIO menu by clicking on the alien icon in the sidebar. Select the menu option to create a new project, give it a name, and select the STM32F401RE as the target platform. You can narrow down the options by beginning to type a few relevant characters such as "F401RE" into the selection field. Select CMSIS for the framework and uncheck "use default location" and select the location to save the project.

After creating the project, PlatformIO will automatically install the required support packages needed to compile, upload, and debug code on the MCU. Because of this, there may be a delay when creating this initial project that will not be necessary for future new projects.

After PlatformIO is finished installing the necessary components, it will open the project with a few subfolders.

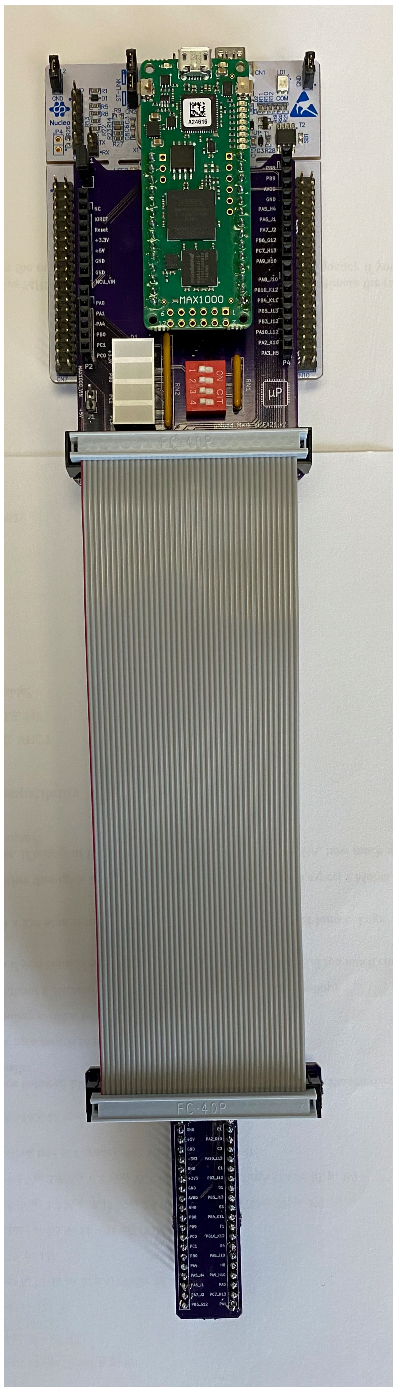
Figure 4: Photo of assembled boards.

Download the source code files (`main_blink.c` and the simple libraries for the reset and clock control (RCC) and GPIO) from the website and place all of the files in the `src` subfolder.

PlatformIO should select the correct upload protocol by default, but to make sure, add the line `upload_protocol = stlink` to your `platformio.ini` file under the line which says `framework=cmsis`.

Compile and build the project by clicking on the PlatformIO alien in the sidebar and select build under `Project Tasks > generic` You can also click the checkmark icon in the bottom bar. Then upload the project by selecting the option under the Project Tasks menu or clicking the arrow icon in the bottom bar.

Next we will go through the process of compiling Verilog and configuring the FPGA. Download the `fpga_blinkpcb.sv` file from the course website. This contains the Verilog code for the project.

Helpful Keyboard Shortcuts

In addition the menu option and shortcut buttons, PlatformIO has several helpful keyboard shortcuts.

Three that you will find yourself using frequently and may be worthwhile memorizing shown in the table below.

Task	Bottom Bar Icon	Keyboard Shortcut
Build	Checkmark	`ctrl + alt + b`
Upload	Arrow	`ctrl + alt + u`
Debug		`F5`

8.2 FPGA Programming

Open up Quartus and select the option to create a new project. In the Wizard, select a directory and name the project project. Under “Family, Device, and Board Settings”, select the FPGA on the MAX1000 board, the 10M08SAU169C8G. Under Simulation, select “Tool Name” = “ModelSim-Altera” and “Format(s)” = “System Verilog HDL”.

After you finish configuring the project and close the wizard, there are also a few other settings you will need to configure for the device which are different than the defaults.

- Under `Assignments > Settings` and then under “Operating Settings and Conditions > Voltage” set VCCA and VCC_ONE voltages to 3.3 V
- Under `Assignments > Device` click “Device and Pin Options” and under the Voltage tab set “Default I/O Standard” to 3.3-V LVTTL.
- Under `Assignments > Device` click “Device and Pin Options” and under the “Unused Pins” tab, make sure “Reserve all unused pins” is set to “as inputs that are tri-stated with weak pullup” (this should be the default and will make sure that there are no issues with pins that are shared between the MCU and FPGA).

9 Clean Up

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

10 FPGA Design

Your next goal is to write some Verilog modules to further test the hardware on your board and operate a 7-segment display. The system should have the following inputs and outputs:

Signal	Signal Type	Description
`clk`	input	12 MHz clock on FPGA
`s`[3 : 0]	input	the four DIP switches
`led`[7 : 0]	output	the 8 lights on the LEDs on the MAX1000 board
`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

10.1 Design and Synthesis in Quartus

The 7-segment display should display a single hexadecimal digit specified by `s[3:0]`. Be sure each digit can be distinguished from other digits (e.g. b and 8 should look different). 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.

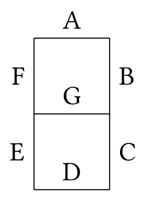

Figure 6: Seven segment display layout

Launch the Quartus Prime software 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. An alternative 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.

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 MAX 10. Under Available Devices, select 10M08SAU169C8G. 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. 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 a hold of you if they need help. Comment the modules as appropriate.

10.2 Logic Simulation in ModelSim

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:\intelFPGA_lite\18.1\modelsim_ase` and set it if necessary. Note that the path for your machine may be different and you should replace the folder (i.e. `intelFPGA_lite`) and version number (i.e. `18.1`) of the above path to match yours. 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. Get in the habit of watching the transcript window to look for errors and to familiarize yourself with what a good run looks like. If you see errors, close ModelSim, correct your Verilog code in Quartus, and reopen ModelSim.

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 resimulate. A helpful shortcut to avoid restarting ModelSim is that you can edit the module 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.

10.3 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, switch 1 on SW1 (connecting pins 1 and 8) is mapped to pin E1, so enter PIN_E1 as the value for s[0].

The FPGA pinouts are shown in the 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 E1 is connected to LED1. 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. Do make sure that these pins are not being used for other purposes. For example, you are fine to use any of the pins which are mapped to the MCU (e.g., `PA3_H5`) but you should avoid E1, C2, C1, or D1 as these are connected to pulldown resistors since they are used for the DIP switch.

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 3.3 V I/O. Fortunately, leaving the voltages at their default value in Pin Planner works and generates 3.3 V outputs.

10.4 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 VDD pin of that side (either VDD1 or VDD2) 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 VDD (VDD1 or VDD2). 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.

10.5 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 using the Arrow USB programmer. 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 pins. Missing 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.

10.6 Programming the FPGA

Next, you will load your design onto the FPGA and PROM with an Altera USB Blaster programming device and the chip's JTAG 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 EP3C5E144 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 board to the computer using the USB micro-B connector. 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!

10.7 Programming the Flash on the MAX1000

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!). To program the FPGA so that the program persists across resets, you need to program the internal configuration flash memory (CFM). To do this, in the Programmer window, select the `.pof` file (programmer object file) that is generated when you synthesize your project. Then, make sure the checkboxes under "Program/Configure" are checked and click "Start." See Chapter 6 - "Configuring the MAX1000" in the MAX1000 User Guide (available on the website) for more detailed instructions.

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. Programming the CFM 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 CFM before checkoff. Also remember that whatever design you last programmed to the CFM 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 MCU.

Compile your project to generate an up-to-date `.sof` and `.pof` file.

Launch the Programmer and remove your old `lab1_xx.sof` from the programming file list. Now add the `lab1_xx.pof` 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 CFM. 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 you choose Tools -> Options -> Initiate Configuration after Programming, you won't have to hit reset after the first programming.

If you want to erase the CFM, change the CFM and UFM options 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!

11 What to Turn In

For this lab and all subsequent ones, turn in a lab report. The report should not exceed one page of text, plus schematics, code, and anything else appropriate, such as diagrams, simulation results, or calculations. The report typically has 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 tests should convince the reader that the requirements of the assignment have been met.
- Technical Documentation: Include your Verilog code and schematics of your circuits.
  - Quality and clarity of your code is important. Make sure it is adequately commented. Succinct code using standard idioms is best.
  - 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. Give the component name or part number such that the reader could order parts and replicate your circuit, or look up components in a datasheet where necessary. The reader shouldn't have to open Quartus to relate your Verilog code to the schematic, and shouldn't need to refer to a data sheet to wire external components or understand what the connections are. Don't assume that the reader has memorized the pinouts of any chips. Therefore, you'll need to label both the pin number and pin name for each pin you use from the FPGA board or other component (e.g., 7-segment display). 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 but 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 Vcc or GND? Be prepared for any other questions about your lab, however.

12 FAQs and Hints

If some of you are seeing issues with specific pins on your FPGA there could be a few different reasons and corresponding steps to take to troubleshoot.

- You could have an electrical short which is causing some pins to be electrically connected that should not due to solder bridges. Double check to make sure that the pins are isolated and not connected together by checking the resistance between the pins with a multimeter.
- You are trying to use a pin that is shared between the FPGA and the MCU (e.g., `PA8_J10`) and the MCU is not leaving that pin floating or it is driving it as an output. For example, if you leave the test code on the MCU from the first part of the lab, PA10 will be driven which means that you can't use `PA10_L12` as a pin for your seven segment!). You can get around this by wiping the code on the MCU by uploading a file with a blank main function.
- You should also read through the Nucleo board documentation to see what pins have special functions that are used by default by the MCU. Hint: You likely want to stay away from using the pins that the MCU uses for serial communication by default.
- You may also see a similar issue if you are using a pin which is dedicated to other hardware on the board (e.g., `E1` which is wired to the DIP switch)