ECE 153a Lab 2B Handout

The Display

Lab Overview

In Lab 2B you will connect an LCD Display to your FPGA board. This is the first lab where you will build a QP-nano automata to handle multiple peripherals. The goal of this lab is to control a graphical volume bar on the LCD using the rotary encoder to adjust the volume level. When the rotary encoder is inactive for 2 seconds the overlay graphic of the LCD will disappear, leaving only the background. A click on the rotary encoder is to be used to make the volume level 0. A second click should restore the volume to its previous value. In all cases, you should only repaint the necessary parts of the volume bar – do not repaint the entire screen on updates of the volume bar! In addition to this, four push-button switches will be used to print different text items on the LCD when they are clicked. Your system will consist of:

- TFT LCD Display (ILI9340/ILI9341)
- Rotary Encoder
- Push-Button Switches

1 Introduction

The LCD peripheral uses a SPI digital interface. A SPI interface to handle the communication with the peripheral's micro-controller (ILI9340/ILI9341) could be implemented using software bit-level protocol coding or by making use of a hardware FSM peripheral controller. The benefit of software is program level flexibility, while a hardware interface allows for a much higher speed interface (5Mb/s = 10MHz clock in this case). Typical Arduino solutions for this display use a software interface, limiting the rate to about 100kb/s. The SPI module provided by Vivado is named AXI Quad SPI and for our LCD screen to work properly we also need to add another AXI GPIO module to the block design. This additional GPIO sets the 'Data/Command' value for the Display (which ought to be part of the SPI protocol as a 9-bit word), but somehow that option does not work in these displays. Effectively, the interface needs to synchronize the actions of SPI hardware and the other GPIO interface so that commands and data are interpreted properly.

In this lab, you need to modify the FPGA firmware to add the SPI and GPIO peripherals to support the encoder, display and four push-button switches. You will then need to build QP-nano code to implement a screen overlay "volume" that displays the value of "volume" stored as a fixed point value from 0-63. The present value of "volume" is conveyed to the viewer by using a rectangular image on the LCD that is at least 1/2 of the display wide, with a bar *proportional to the set value* and a fixed background. The background can be of any color with 40×40 triangles on top of it. Each triangle will be filled with a color that should be different from the background color. It is up to you whether you create shapes with or without borders. You can be as creative as you like with your choice of background. Two examples are shown in Figure 1.

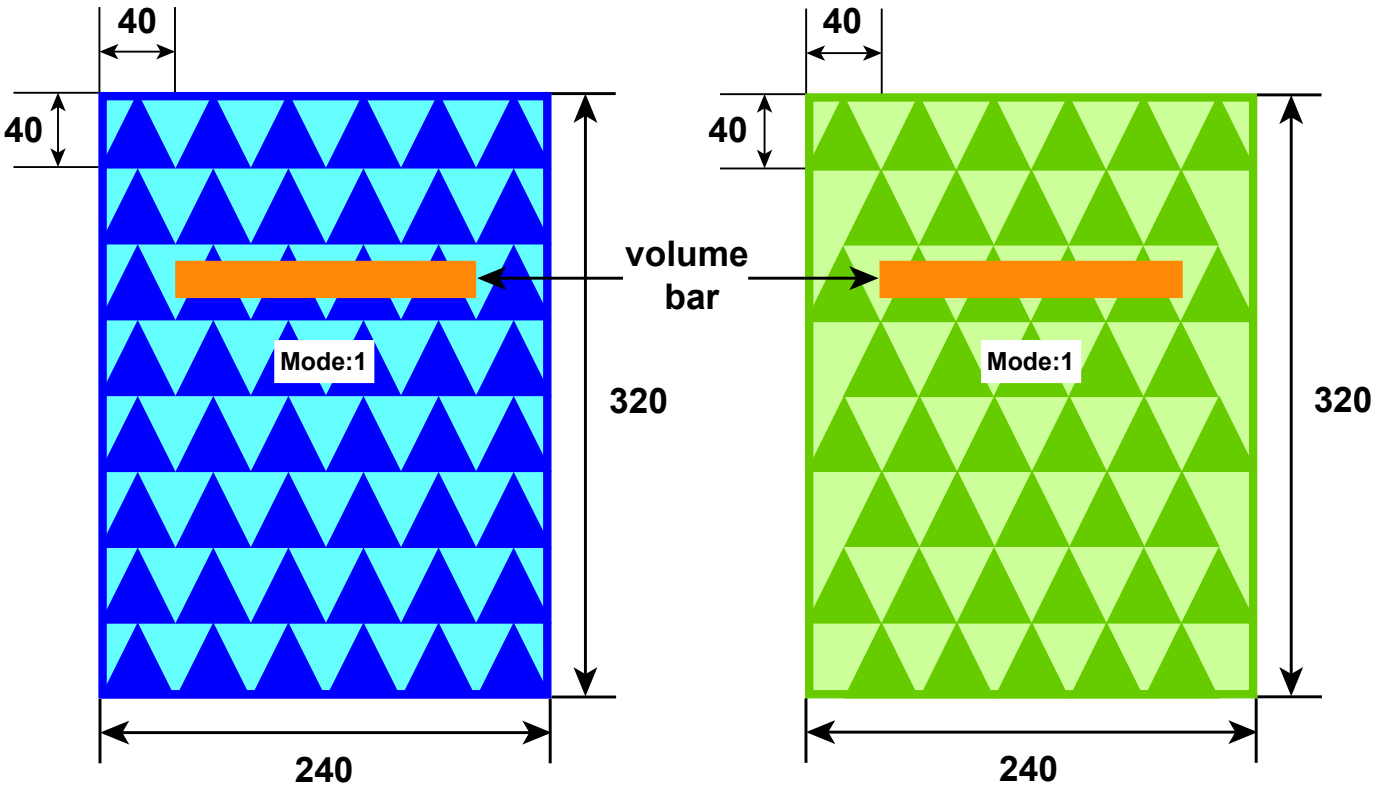


Figure 1: Two different backgrounds. Any of these can be implemented. Be creative with your design.

The overlay for the volume bar should appear on top of the background. It should appear immediately as the encoder is turned, and stay on, updating dynamically until 2 seconds after the last encoder update. In addition to the volume bar, different text items should be displayed in response to pressing of different push-button switches (For example: pressing btn1 prints "Mode:1", pressing btn2 prints "Mode:2", etc.). If there is no activity on the encoder or switches for 2 seconds, the volume bar and text disappear, leaving only the background on the display. The goal here is to get two timed protocol behaviors and timed events into the software behavior. You can arrive at the final design in several ways, but we suggest making a separate program to run the display and then incrementally merge the desired behavior into a single QP state chart.

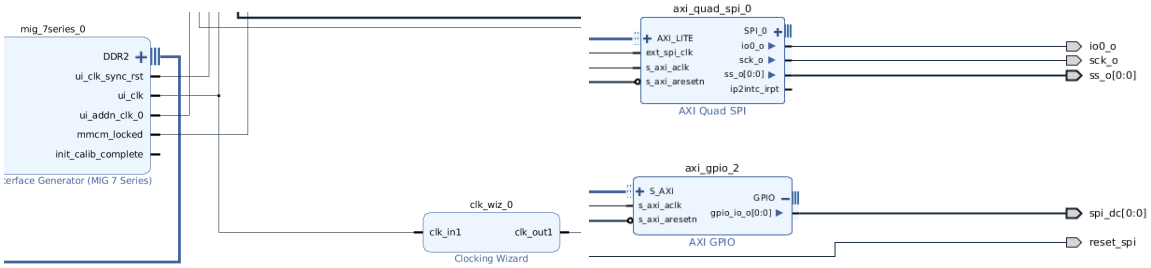


Figure 2: The modifications to the hardware Block Design are shown above. Detailed instructions are provided below. In particular, note the reset_spi port and the Clocking Wizard module.

2 Hardware Modifications

1. Add GPIO module to supply the Data / Command signal.
   (a) With the Block Design open, Right click and choose 'Add IP'.
   (b) Type 'AXI GPIO' and add the module to the design.
   (c) Left click on the newly created module and change the name to `spi_dc`.
   (d) Click on 'Run Connection Automation' and select '/spi_dc/S_AXI/'
   (e) Right click on the spi_dc module, select 'Customize Block', choose 'All Outputs' and make the GPIO width '1'.
   (f) Left Click on the '+' next to 'GPIO' to expand the IO ports.
   (g) Right click on the port dash next to the terminal labeled 'gpio_io_o[0:0]' and 'make external'. Select the port and rename it to spi_dc.
2. Createing a 10MHz SPI Clock (ext_spi_clk)
   (a) Add a new "Clocking Wizard" block.
   (b) Connect the input of this clock "clk_in1" from the "ui_clk" (main system clock) of the MIG.
   (c) Right click on the new 'Clocking Wizard', select 'Customize Block' and click on the tab 'Output Clocks'.
   (d) On the first tab, Ensure that the input clock is set to 100MHz.
   (e) Switch to the "Output Clocks" tab, ensure only clk_out1 is selected and set it to 10MHz.
   (f) This results in an effective SPI clock of 5MHz(the SPI block divides by 2). You may increase this clock to 20MHz for 10MHz SPI clock if you wish.
   (g) Uncheck the "reset" and "locked" options in the "optional input/output" section.
3. Add SPI module
   (a) With the Block Design open, Right click and choose 'Add IP'.
   (b) Type 'AXI Quad SPI' and add the module to the design.
   (c) Left click on the newly created module and change the name to `spi`.
   (d) Click on 'Run Connection Automation' and select '/spi/AXI_lite/'
   (e) Connect the Clocking Wizard pin `clk_out1` to `ext_spi_clk`
   (f) Right click on spi module, select 'Customize Block' and make the settings: **Mode**: Standard, **Transaction Width**: 8, **Frequency Ratio**: 2, **No. of Slaves**: 1, Check 'Enable Master Mode', Check 'Enable FIFO', **FIFO Depth:** 256. UNCHECK Enable STARTUPE2 Primitive.
   (g) Left Click on the '+' next to 'SPI_0' to expand the IO ports.
   (h) Right Click on the port dash next to the terminal labeled '`io0_o` (letter i, letter o, number zero, underscore, letter o) and choose 'Make External'.
   (i) Repeat the step to 'Make External' for the signals `sck_o`, `ss_o[0:0]`

(j) If any of these ports end up with a trailing "_0" on the end, remove it so they end up with the same name as the constraints below.

(k) Right click near the `btnCpuReset` pin and 'Create Port'.

(l) Name the new port `reset_spi` and select 'output'.

(m) Draw a wire to connect `reset_spi` to the same wire at `btnCpuReset`

2.1 Update Constraints File

You now have to assign pins for the GPIO and the SPI signals spi_dc, io0_o, sck_o, ss_o, and reset_spi by updating the constraints file. You may want to use a Pmod header (for example, JB) for convenience. For the rotary encoder, the Pmod header pins should have been already assigned from Lab 2A.

2.2 Export to xSDK

After modifying your hardware, make sure that the depth of the debug core (created in Lab 2A) is less than or equal to 32768. Otherwise you may run out of space on the FPGA, and implementation will fail. Run synthesis, implementation, generate a new bitstream and export your Vivado design to xSDK. Make sure the Board Support Package which you intend to use for the lab is updated. To make sure it is updated, open the BSP project, open **system.mss** and click 'Re-generate BSP Sources'. In SDK, do not forget to run the linker script, map all sections of the memory to the DDR (mig_7series_0_memaddr...) and **not** the BRAM. Also make sure that the the heap and stack sizes are 4096 (4 KB). In this and the previous lab, this step is crucial... to work correctly, the bsp, the hdf and mss files and all internals must all match the bitfile firmware you are loading. If you have doubts, archive the sdk directory, replace it using the vivado export hardware function, create a new bsp and project and import your source code.

2.3 Wiring LCD Display to Nexys4 DDR Board

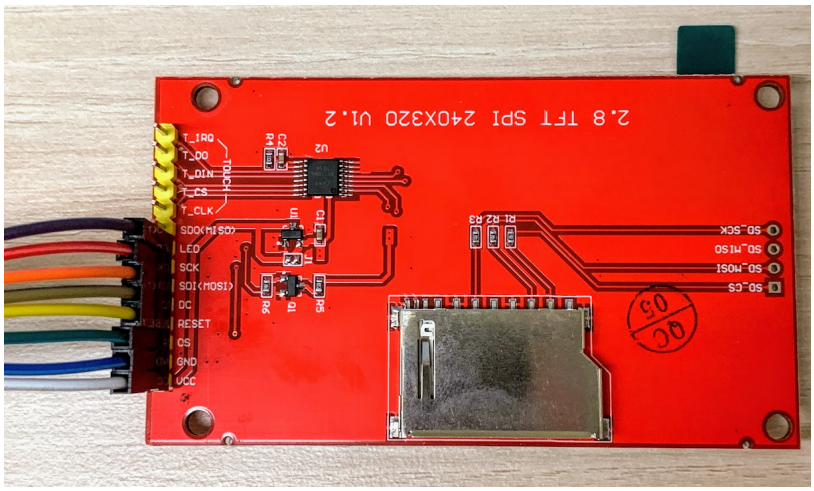
Figure 3 shows the pinouts for connecting the LCD display to the board. Please view the Nexys 4 manual for the pinout of the Pmod (or whatever pins you have used). The connection should be achieved by soldering a male pin header to the display (if it didn't already come with one soldered in), and then using male-female jumper wires back to the FPGA board. In Figure 3, we have used a HiLetgo ILI9341 2.8" SPI TFT LCD Display which already comes with the pin header.

In particular note that sticking a pin header through the holes on the LCD PCB will not result any sort of reliable connection. Pin header to pin header connections rely on spring action to achieve a usable connection, which an empty hole on the LCD board does not supply. Soldering is necessary in this case.

3 Software Test

Create a new Application project and copy four files **lcd_test.c**, **lcd.c**, **lcd.h** and **fonts.c** . Program your FPGA and run the lcd_test on your FPGA board. If the 'Hello World' displays on the screen your hardware configuration is correct.

If the LCD does not work right away, check the wiring to the LCD and check the names of your io ports. The LCD should turn white and then draw the provided test display. Note: the provided driver is minimal. More complex versions can easily be written using the data-sheet (on canvas) and on-line lcd libraries (e.g. Adafruit). I have several times in the



1. LED - Purple, 3.3V
2. SCK - Red, sck_o
3. SDI/ MOSI - Orange, io0_o
4. DC/RS - Brown, spi_dc
5. RESET - Yellow, reset_spi
6. CS - Green, ss_o
7. GND - Blue, Gnd
8. VCC - White, 3.3V

Figure 3: The Figure above shows HiLetgo ILI9341 2.8" SPI TFT LCD Display and how to connect the wires. The list of wires next to the display is in the format of LCD Pin Name - Color, signal. To connect the signal wires to your Nexys4 DDR board, follow your updated constraints file. The wire color is consistent for the signal name.

past released better, more efficient drivers, only to find that they didn't work for some fraction of the class. In reality, the lcd has its own embedded processor and command timing and response protocols are all based on internal firmware and thus vary from card to card. The provided driver was chosen as it reliably worked with all the cards (ili934x) we tried.

4 QP Nano Program Design

The goal of this lab is to control a graphical volume bar on the LCD using the rotary encoder to adjust the volume. When the rotary encoder is inactive for 2 seconds the volume bar will disappear, leaving only the background.

- Provided are two sets of example code, one is a partial example of QP nano. The other is an example of using the LCD. You should build your application around the QP nano framework.
  - Copy the QP nano example files as a template for your project.
  - Fill out **BSP_Init()** with hardware initializations for your peripherals.
  - Define interrupt service routines like in **bsp.c**. These will be similar to Lab 2a. Edit **bsp.c** to connect interrupts to your Finite State Machine. This file is heavily commented, use the comments to help fill in the missing code.
  - **lab2a.c** is an example state machine definition. Read it to understand how QP Nano models a FSM. Replace this with your own state machine.
  - **q*.c** and **q*.h** contain the QP Nano code. **Do not modify these**. Read to understand.
  - **lcd_test.c** is an example of using the LCD. Include this into your QP Nano project.
- Create a simple custom LCD function to draw your background.
- Draw a rectangular block on the LCD to represent the 'volume' the rotary encoder is at. The graphic must be proportional to the set value.
- When twisting 'left' decrease the 'volume'. Twisting 'right' increase the 'volume'.

- When the rotary encoder push button is pressed, reset the 'volume' to 0.
- When different push-button switches on your board are pressed, print different text values to the display.
- Consider the design of your GUI and how the 'overlay' should work for your volume bar. An 'overlay' is a shape (sprite) which appears on top of a **background** that could incorporate a custom image. They are things like video game characters which walk around of a fixed image of a background.
- Use the timer to measure the number of seconds since last activity on the rotary encoder or push-button switches.
- When the inactivity time is 2 seconds, leave only the background.

5 Reporting

1. Report on how you created your function for drawing the background and include a picture of your background pattern with your LCD photos.
2. Report where you acquired your LCD from, and include a photo of the front and back of the LCD with your report. (Let us know if you got it on time and if you had any problems getting it setup.)
3. Explain how you integrated the LCD and the Rotary Encoder using QP-nano.
4. Explain how you detect inactivity and how you quickly remove the overlay graphic or text.
5. Draw a statechart that models the behavior of your system.

6 Video Submission Guidelines

Each group can work together on the lab assignment, but must submit the report and video individually. Upload a short video (do not exceed 60 seconds) to Google Drive demonstrating the following:

1. Run your code and wait until the background is painted. (State your name and perm in the video to identify yourself).
2. Turn the knob of the rotary encoder in clockwise direction a few times to slowly increase the volume.
3. Stop turning the encoder and wait for 3-4 seconds. The volume bar should disappear.
4. Now start turning the encoder again. First turn it counter-clockwise a few times to decrease the volume and then turn it clockwise to increase the volume to the maximum possible value.
5. Press the push-button on the encoder to make the volume 0.
6. Again turn the encoder clockwise to increase the volume.
7. Press all the push-button switches one after the other to display text.
8. Wait for 3-4 seconds.
9. Finally, turn the encoder again a few times in each direction.