CMPSC 153A: LAB 22

Alex Shortt

Lab 2B used the QP Nano framework to design our own FSM that controlled a volume indicator.

Methodology

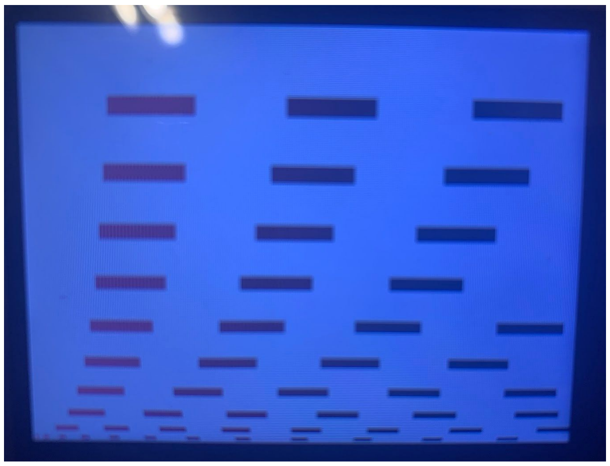
The specs of the project were - pretty complex, actually. There was a background that needed to be drawn on an LCD. Over the background, an overlay would show when you either pressed a button or interacted with the rotary encoder. It would show the volume and a given message. The controls are as follows:

(1) Rotating the rotary encoder either way will increase or decrease the volume
(2) Clicking the rotary encoder in will "mute," or bring the volume to zero. While muted, the old volume is saved and rotating the encoder does nothing.
(3) When a button is clicked, a message pops up. There's a different message for each button.

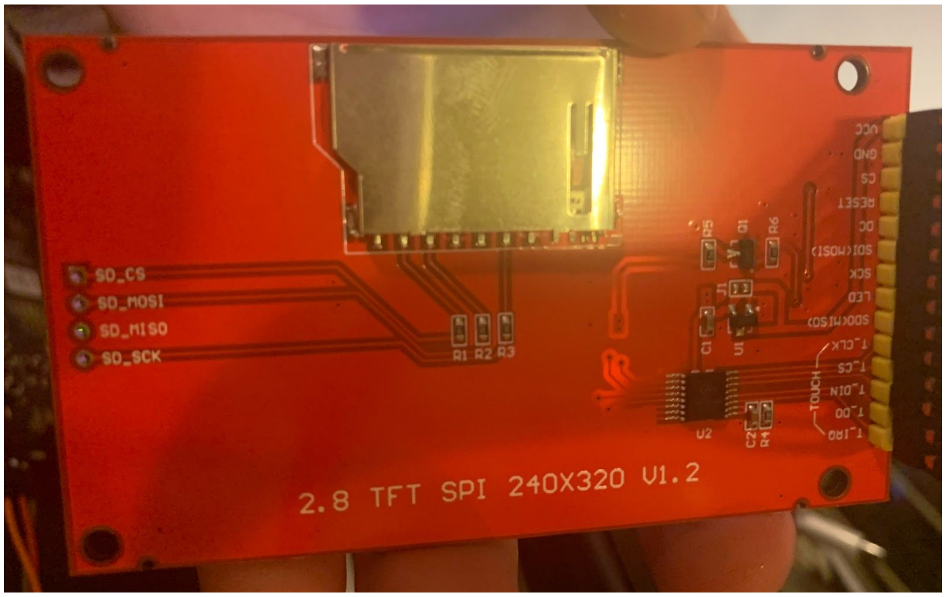
I found it easiest to implement the entire FSM in code (after drawing it out on a whiteboard), then draw to the LCD afterward. This process turned out a lot smoother than I thought it was going to!

Results

The final result for the background looks like this:



Also, this is what my LCD looked like. I ordered it online at the beginning of the quarter, so it's just been sitting in my closet this whole time.



I ended up creating a finite state machine with the following signals as input:

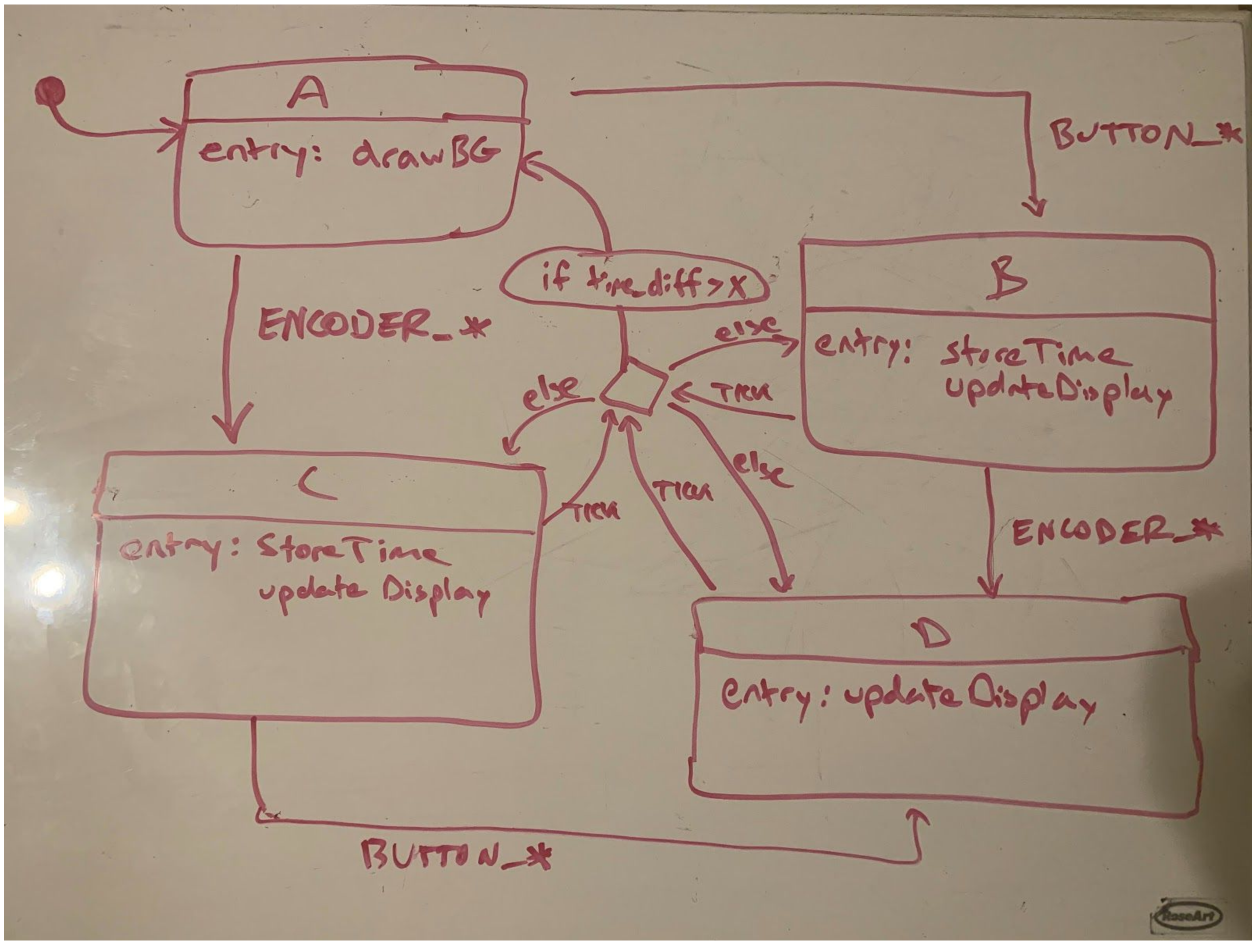
```
enum Lab2ASignals {
      ENCODER_UP = Q_USER_SIG,
      ENCODER_DOWN,
      ENCODER_CLICK,
      BUTTON_LEFT,
      BUTTON_TOP,
      BUTTON_RIGHT,
      BUTTON_BOTTOM,
      BUTTON_CENTER,
      TICK
};
```

The bsp file would set up all the interrupts and handlers for processing the inputs, including debouncing and decoding the two pins from the rotary encoder. After affirming that an input had been received, it would send it to the FSM using QActive_postISR. The BUTTON_ signals correspond to one of the five buttons on the board and the ENCODER_ signals correspond to each action with the encoder. Finally, the TICK signal was sent out in the FSM idle method, meaning that it would be called every time the FSM was idle.

This was perfect for a ticker used to check the current time for the overlay timeout. When an input was received, the overlay would be turned on and the current time (sent over from the bsp file) would be stored. Then, during every TICK signal, the program would check for the time different, and turn the overlay off after 2 seconds.

Drawing and removing turned out to be more intricate than I had thought. You can't just remove a graphic from the display, you have to draw over it. And, drawing the background is very expensive since it only has a 10Hz refresh rate. As such, I opted to draw a white box behind the volume slider and the message every time the overlay was enabled so that I could keep drawing on top without artifacts from previous frames. Then, once the overlay went away, I would redraw the entire background over again. This method allowed for a fairly smooth update on the label and the volume slider without compromising the background design too much.

Another note is that I only drew to the screen on state changes, rather than every frame. Basically, for every input change, I would redraw the label and volume portion of the screen. This meant that overall it drew to the screen much less than it needed to.



The state machine I designed is seen above. State A is the rest state, where nothing is done. If any button input was pressed, the machine would go to state B, where the time would be stored and the display would be updated to show the correct label. Every time a button was pressed, it would just update the label and update the display again. If, from A, an encoder input came in, it would transition to state C where the time would be stored and the update would be displayed. In this case, the volume display would update, but not the label display. If a button were pressed while in state B or C, it would transition to state D, where both the label display and volume display would update. In all states, every time a button signal or encoder signal was received, the current time would be stored. In states B/C/D (when the overlay was displayed), the TICK signal that came in every time the FSM went idle would trigger a check for the time difference between the current time and the stored time. If it was greater than two seconds, it would transition back to state A, where it would draw the background over the overlay displays, resetting the state machine.

Overall I thought my design turned out well, and I was happy with it. If I could change one thing, I would have modeled the state machine to be hierarchical, since some of the calls were duplicated in the state machine I designed. However, with only 4 states that pretty clearly partitioned the design of the device, I was happy with the end result. And I submitted it on time for once!