1 | LCD Display for Volume Rotary Encoder and Mode Push Buttons

Oviya Seeniraj

2 | Purpose and expected goals

The main purpose of this lab is to understand and harness a QP-Nano FSM in order to implement graphical volume bar controlled by a rotary encoder and mode declaration controlled by push buttons on a Nexys A7 board. We expect graphics generated by each of these peripherals (aka overlaid graphics) to disappear after 2 seconds, and the encoder's push-button should reset the volume bar to 0.

3 | Methodology

a. Overview of Design Tasks

- Configure FPGA to interface with the LCD through SPI and GPIO peripherals
- Create firmware that utilized QP-Nano automata for state management across multiple peripherals by
    - Configuring SPI and GPIO for display control
    - Designing QP-Nano state machines to handle input and overlay display behavior
    - Developing functions to dynamically adjust the volume bar overlay based on encoder input
    - Implementing an inactivity timer to manage overlay display time

b. Assumptions for Design Analysis and Procedures

- The SPI clock frequency was assumed to provide reliable communication at 5 MHz
- The overlay timer functionality was based on the AXI timer with a 2-second timeout as the inactivity threshold

c. Observations from Design Tasks

- Volume control responsiveness and overlay updates were smooth, as we only repaint the necessary pixels in response to encoder adjustments
- Using a QP-Nano FSM handled interrupts efficiently which reduced latency for both peripherals

d. Plan for Design Testing

- Accurate rendering of the volume bar proportional to encoder adjustments
- Correct inactivity timer functionality to clear the overlay after 2 seconds
- Immediate update of overlay text upon button presses, without affecting the background

4 | Results

a. Metric Results: I met all my goals. Volume adjustments displayed correctly on my screen, overlay elements cleared after 2 seconds of inactivity, and Text updates for each push-button functioned as intended.

b. Limitations of the Design: The SPI communication speed was limited to 5 MHz due to hardware constraints, slightly reducing the refresh rate of the display updates.

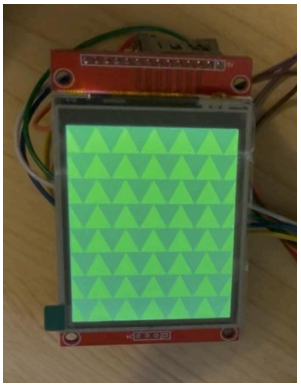
c. Design Test Results: The inactivity timer met the 2-second goal.

d. Road Blocks and Parts Not Meeting Spec: Initial difficulties in configuring GPIO resulted in delayed integration of the push-button functionality. However, this was resolved by remapping the constraints file and renaming ports - Vivado was the real roadblock here, along with understanding the right QP Nano FSM Hierarchy.

e. Issues and Sources of Errors: I had a few issues with getting my board to connect properly and had to reprogram and rebuild often. I had to redo my GP Nano FSM structure a few times as well - I adjusted my code several times.

5 | Reporting

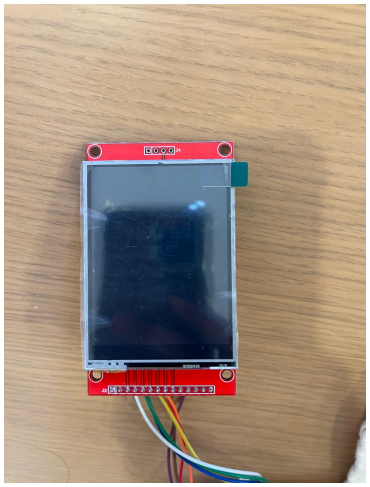
Report on how you created your function for drawing the background and include a picture of your background pattern in your report.



I replicated the green and orange background from the report. The background consists of a grid of alternating triangles, arranged in 8 rows and 6 columns. Each triangle is 40 pixels tall and 40 pixels wide, and the color pattern is created by adjusting the number of green pixels in each row of the triangle. To do this, a helper function draws each triangle within a 40x40 box, starting with 2 green pixels in the top row, then increasing by 2 every two rows until reaching 40 green pixels at the bottom (can calculate width of pixels using 2*ceil(y/2)). The remaining pixels in each row are filled with cyan, creating a stacked, alternating pattern of neon green and dark green triangles across the screen. This design is repeated across the entire screen by rendering each 40x40 triangle in its respective grid position. The function operates row by row, calculating the correct number of green pixels for each row and ensuring the pattern aligns across the 8-row grid.

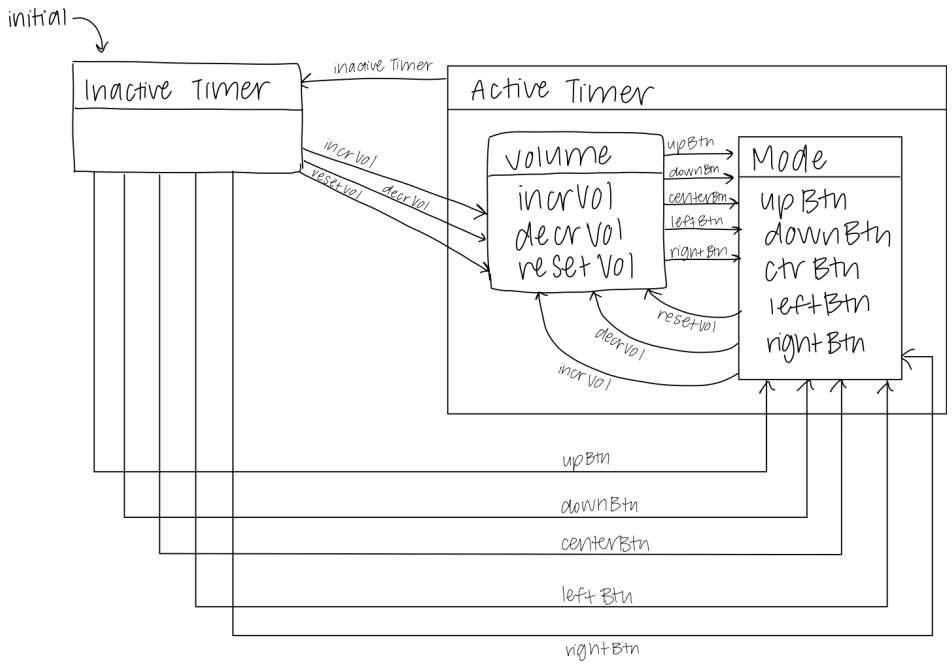
Report where you acquired your LCD from, and include a photo of the front and back of the LCD with your report. (Let us know if you got it on time and if you had any problems getting it setup.

I got my LCD from the ECE shop in Harold Frank Hall. I had no problems setting it up, although I often had to re-program my board and rebuild the same code to get it to run correctly.



Explain how you integrated the LCD and the Rotary Encoder using QP-nano. Draw the statechart/UML FSM you built and answer these questions by referring to the chart.

I integrated the LCD and Rotary Encoder using QP-Nano by adding 3 extra signals to QP Nano coming from the rotary encoder code from lab 2A - a left turn, a right turn, and a click. Each of these triggers a different transition by the QP-Nano. The QP-Nano makes state transitions and handles them as needed by calling helper functions to draw or reset rectangles on the screen to represent the volume. Similarly, each push button also generates a signal. All of these push button signals are handled by one helper function to print text. I created these helper functions for my drawings using built in functions like setXY(x1, y1, x2, y2), clrXY(), setColor(r, g, b), setColorBg(r, g, b), clrScr(), fillRect(x1, x2, y1, y2), and LCD_Write_Data(color).



Explain how you detect inactivity and how you quickly remove the overlay graphic or text.

The AXI timer handles the overlay display by tracking input activity and hiding the overlay after 2 seconds of inactivity. Each time an interrupt occurs (such as a button press or encoder adjustment), the timer resets, updating the time of the latest input in a variable. The system checks if the current time exceeds the last recorded input time by 2 seconds in a function. If it does, it sends a signal that the timer is now inactive to the finite state machine, which causes a transition to an inactive state, which hides the overlay.

To quickly remove the overlay, we create a function that determines the color of each pixel in the overlay area (either green or forest green) based on its position. It only overwrites specific pixels - it doesn't affect the whole screen. This ensures that non-overlay elements on other parts of the display are less affected and that there is less work to be done by the function, resulting in a faster action.

6 | Video Submission

https://drive.google.com/file/d/1H590OWapIhs1SOLoj_mR1gBTsRNCB8_2/view?usp=drivesdk