Lab 4: Pong

ESE3500: Embedded Systems & Microcontroller Laboratory
University of Pennsylvania

In this document, you'll fill out your responses to the questions listed in the Lab 4 Manual. Fill out your name and link your Github repository below to begin. Be sure that your code on the repo is up-to-date before submission!

Student Name: Emily Shen
Pennkey: shenyit
GitHub Repository: https://github.com/ese3500/lab-4-pong-EmilyYShen

1.
The controller differentiates between a command or a data packet based on the contents of the transmitted data. The ST7735 controller uses a parallel interface for communication with the host processor. The interface includes data lines, a chip select line, a write enable line, and a data/command select line (DC). When a byte of data is transmitted to the ST7735 controller, the state of the DC line determines whether the byte is interpreted as a command or a data packet.

When the DC line is set to low, the byte of data is interpreted as a command. Commands are used to configure the controller's internal registers, such as setting the display area or changing the color mode. The specific command is determined by the value of the byte that is transmitted.

When the DC line is set to high, the byte of data is interpreted as a data packet. Data packets contain the pixel values that are displayed on the screen. The data is stored in the controller's internal memory, and the controller uses the memory to drive the display.

2.
This means that each 16-bit color value must be split into two 8-bit packets for transmission.

To send a 16-bit color value, the driver first splits the value into two 8-bit packets, differentiated as the most significant byte (MSB) and least significant byte (LSB). The driver then sends the MSB packet to the controller using one of the aforementioned functions, and then sends the LSB packet using the same function. This process is repeated for each color value that needs to be sent to the controller.

3.
The LCD_setAddr function is used to set the pixel memory address of the ST7735 controller for pixel data to write to. This function is typically called by the driver code before sending pixel data to the controller.

The LCD_setAddr function takes four arguments: x0, y0, x1, and y1. These arguments specify the rectangular area of the display to which the subsequent pixel data should be written. The function uses these arguments to send a series of commands to the ST7735 controller that set the memory window to the specified area.

There are three commands that are sent:

CASET (Column Address Set): This command sets the range of columns in the display memory that subsequent pixel data writes will address. The x0 and x1 arguments are used to set the start and end column addresses.

RASET (Row Address Set): This command sets the range of rows in the display memory that subsequent pixel data writes will address. The y0 and y1 arguments are used to set the start and end row addresses.

RAMWR (Memory Write): This command sets the controller to write mode, allowing subsequent pixel data to be written to the specified memory window.

By setting the memory address with LCD_setAddr before writing pixel data, the driver code can ensure that the data is written to the correct area of the display.

4.
rgb565 is a function that converts a 24-bit (8-8-8) RGB color value to a 16-bit (5-6-5) value which is used by the controller. This conversion is done because a 16-bit value is more optimal for use by the controller. This is because it allows for easier and faster communication, and also less space is used up in the memory. There will be information lost during this conversion, as certain shades of colors will not show up, and there will not be as much variance of available shades between one primary color and another primary color. However, the human eye cannot really discern between these different shades, so this loss of information is not important. Furthermore, the loss is not too significant for our particular application, which is a video game. It may be significant if the color is needed for a high quality photograph, so the importance also depends on the application.

5.
Yes, debouncing of the "buttons" used to control the paddle may be needed here to ensure accurate and reliable control of the paddle. When a slider is moved, it can cause rapid changes in the signal which may result in multiple detections of the signal or inaccurate readings. Without debouncing, the slider signal may bounce back and forth between high and low states rapidly, causing the paddle to move erratically or not move at all. This is seen slightly with the paddle movement in the video taken of part E, where the movement of the paddle in part E is less smooth than the movement of the paddle in part B.

Debouncing is when false detections are eliminated to ensure that the signal is stable before it is detected. It can be done using software or hardware methods, but it depends on a specific implementation. Debouncing ensures that only one signal is registered for each value of the slider, thus reducing any false readings.

If the Blynk app is used to control the paddle, it is also possible that the app itself may perform debouncing before sending the signal to the NodeMCU.

Part B: Pong
Github link: https://github.com/ese3500/lab-4-pong-EmilyYShen

Link to video:
https://drive.google.com/file/d/1ah3SJVIr8jmeWrGi1IyYGsTXFwvlBICV/view?usp=sharing

In this video, I demonstrate all the features of the pong game that I implemented. For the computer paddle to move, I have a function called “setComputerPaddle” that moves the paddle steadily up and down the screen at the same speed. When the paddle touches a wall (either the top or the bottom of the screen), the paddle will switch direction and move in the opposite direction. When looking at the screen, the computer paddle is on the right side along the y axis whereas the user paddle is on the left side along the y axis. Both the user paddle and the computer paddle have the same width and height. The user paddle moves up when the joystick is moved up (or in the video, to the right) and the user paddle moves down when the joystick is moved down (or in the video, to the left). The paddle will continue to move up if the joystick is held in the up position, and vice versa if the joystick is held in the down position. The function “updateUserPaddle” reads in the ADC value for pin A0 and determines whether the paddle should move up or down, or if the joystick is in the neutral position, the paddle will stay in the same position. This function also checks if the paddle is already at the bounds of the screen; if it is, then

the paddle will not continue to move in that direction even if the user is toggling the joystick.

In the Initialization of the game, as well as every time a game is reset, the x velocity (called “vX”) and the y velocity (called “yX”) will be updated to a random value between -5 and 5. Note that no matter what the value is, the random value is never set to 0, as otherwise the ball would not be able to touch the paddles. Then, in the function called “updateBallPosition”, the ball will start to move in the random direction at a random speed that is set by the variables “vX” and “yX”. Note that in the function “updateScoreboardUser” and “updateScoreboardComputer”, every time a game ends, the ball position is reset to the middle of the screen and the new random “vX” and “vY” values are determined. The ball also follows proper game dynamics, so it interacts with other components in the game such as the paddles and the boundaries of the game. In the function “checkOutOfBounds”, if the ball is at the top or bottom of the walls, or if it is touching a paddle, the corresponding velocity changes to the negative value of that velocity (not that the velocity can become negative or positive depending on its initial value) so that the ball will stay on the screen.

There is also a scoreboard on the top of the screen. This is Initialized at the beginning of the game and also updates throughout the entire game. The two numbers in the boxes in the middle indicate the current points that the user and the computer game for this particular round. The function “checkOutOfBounds” checks if the computer or user has missed the ball. If the computer misses the ball, then the function “updateScoreboardUser” will increment the current points that the user has by one and update this on the screen. Furthermore, the green LED will flash to indicate that the user has won this point. Similarly, if the user misses the ball, then the function “updateScoreboardComputer” will increment the current points that the computer has by one and update this on the screen. Furthermore, the red LED will flash to indicate that the computer has won this point.

In these two functions, there is another function called “checkRoundFinish” that checks if either the user or the computer has reached 5 points in this round. If they have, then that means the round has ended and the board will be reset, including the current points, which will be reset to 0 points for both the user and the computer. If the user wins, the number of wins will be incremented by 1 and displayed on the screen that reads “Wins:” on the user side (left side). If the computer wins, the number of wins will be incremented by 1 and displayed on the screen that reads “Wins:” on the computer side (right side). Depending on who wins, the corresponding LED will also light up. Then, the next round will start, and this will be indicated by text on the screen that denotes the current round

number. Furthermore, in this function called “checkRoundFinish”, since the game is best of 3 rounds, if the number of wins for the user and computer add up to 3, then a black screen will show up with text indicating who won. Furthermore, an LED will light up - it will be red if the computer won this game or green if the user won.

The buzzer sounds with a clack sound every time the ball hits the horizontal boundary surfaces, if the ball hits the paddle or if the ball hits the paddle. This is done in the “checkOutOfBounds” function.

Part E: Blynk
Link to video:
https://drive.google.com/file/d/1guYL0s57Dfj8ZWDD44-EcZPI1KWHlHxF/view?usp=sharing

None of the code from Part B was changed for this section. I chose to use a slider to control the user paddle wirelessly. As shown in the video, if I move the slider to the left, then the user paddle moves up the screen along the y-axis. If I move the slider to the right, then the user paddle moves down the screen along the y-axis. If I keep the slider relatively in the middle, then the user paddle also stays in the middle of the screen along the y-axis.

I used V2 (virtual pin 2) for the slider. Thus, as seen in the arduino IDE code, I use the function BLYNK_WRITE(V2) to get the value from virtual pin 2. The values for V2 are controlled by a slider, as shown in the video. The sensor value is a variable that takes and stores the current value of V2. Then, I use analogWrite to send this particular value that was recorded to A0, which is where I am reading the ADC input to move the user paddle.

Part F
N/A