Sidharth Daga, Nick Khormaei

3/1/2023

EE 474

Lab 4 Report

Procedure

Task 1:

For task 1a, we first gathered the materials and figured out how to set up the LCD on the TIVA board. To approach the setup part of things, we followed the directions layed out in the lab spec and configured the correct registers.

```
289     // D2
290     SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R3;    // activate port D
291     wait++;                                     // wait for port activation
292     wait++;                                     // wait for port activation
293     GPIO_PORTD_DIR_R |= 0x4;   // make PD2 an output
294     GPIO_PORTD_AFSEL_R = 0x00;     // disable alternate functions
295     GPIO_PORTD_DEN_R  |= 0x4;       // enable digital I/O on PD2
296
297     // K5
298     SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R9;             // activate port K
299     wait++;                                     // wait for port activation
300     wait++;                                     // wait for port activation
301     GPIO_PORTK_DIR_R |= 0x20;                      // make PK5 an output
302     GPIO_PORTK_AFSEL_R = 0x00;                    // disable alternate functions
303     GPIO_PORTK_DEN_R  |= 0x20;                     // enable digital I/O on PK5
304
```

Figure 1. Registers configured for LCD

From there, we had to call the LCD_ColorFill function to produce a color that filled the screen.

```
int main()
{
  LCD_Init();
  unsigned short color = Color4[5];
  LCD_ColorFill(color);
  while (1) {};
  return 0;
}
```

Figure 2. LCD_ColorFill function

For task 1b, we used a similar approach but instead we had to print the temperature of the board and have it change depending on the button that was pressed. We approached this by using

our previous code we had done in lab 3, then tried to implement the LCD in it. We used the sprintf function as well to put the float to a string (char array) then have the cursor reset back to the top left so we would have the temperature print there.

```
sprintf(temp_board, "The current temperature is %.2lf C, %.2f F.\n", ctemp, ftemp);
sprintf(freq_board, "The current clock frequency is %d MHz.\n", freq);
```

Figure 3. Code for sprintf and the temperature readings

For task 1c, we were replicating the temperature readings except now we had to use virtual buttons on the screen. We configured it to have conditions for the dimensions of x and y on the screen to change the clock speed depending on if the virtual button was pressed. This then changed the reading of the temperature.

```
if ((x <= 2200) && (x >= 1900) && (y <= 1800) && (y >= 1400)) { // if s
  freq = PRESET1; // increase frequency
} else if ((x <= 2500) && (x >= 2300) && (y <= 1800) && (y >= 1400)) {
  freq = PRESET3; // decrease frequency
}
```

Figure 4. Code for virtual buttons

Task 2:

For task 2a, we approached this by using what we had done in task 1 and our knowledge of timers and interrupts. This entailed us combining information from past labs and this one. We ran into some trouble with the traffic lights changing too quickly, but we were able to get that figured out by adjusting our clock rate to 60000000 and fixing the bit amount.

```
2
3   bool sw1_pressed() {
4     unsigned long x = Touch_ReadX();
5     unsigned long y = Touch_ReadY();
6     if ((x < 2900) && (x > 2500) && (y > 2000) && (y < 2700)) {
7       return true;
8     }
9     return false;
0   }
```

Figure 5. Code from 2a

For task 2b, it was very similar but now we had to use RTOS. This was a bit tricky because we had never done this before but after rewatching the lectures we were able to get a better grasp of how it worked.

```
while (1) {
  curr_ped_tick_time = xTaskGetTickCount();
  unsigned long x = Touch_ReadX();
  unsigned long y = Touch_ReadY();
  if ((x < 2900) && (x > 2500) && (y > 1000) && (y < 1900)) {
    if (curr_ped_tick_time - prev_ped_tick_time >= 2000) {
      pedestrian_status = 1;
      prev_ped_tick_time = curr_ped_tick_time;
```

Figure 6. Code for RTOS for task 2b

Results

Figure 7. LCD screen changing color

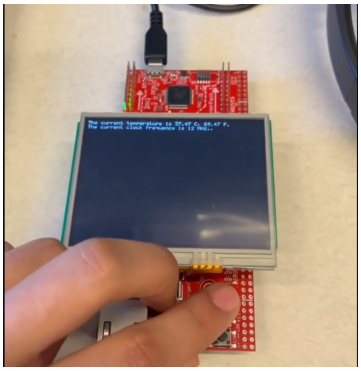

Figure 8. Temperature outputted on LCD

Figure 9. Virtual buttons on LCD

Task 1:

Overall, everything worked out great as in part 1a. We set up the registers outlined in the directions and got the LCD to turn to a dark purple.

For task 1b, we used sprintf and got the LCD to output the current temperature reading of the board. We also used polling and got the on board buttons to change the clock frequencies which increased/decreased the temperature.

For task 1c, the output was almost the same as 1b except we created the virtual buttons, so when someone pressed switch one or two it would increase/decrease the clock speed which then increased/decreased the temperature of the board.

Task 2:

For task 2a, we repeated the traffic light controller like in lab 1. This worked out great as the system followed how the directions had it lined up. For example, the Pedestrian button is pressed and held by the user for 2 seconds while the system is in the go state, the system should immediately transition to the warn state, and remain there for 5 seconds before moving to the stop state.

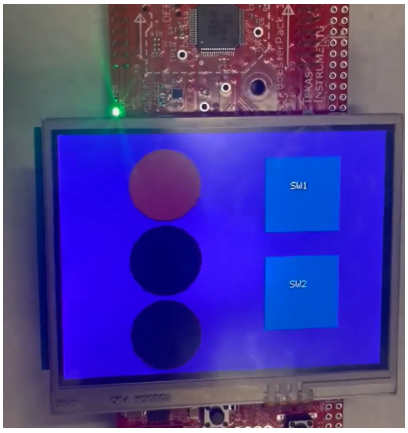

Figure 10. Results of task 2a

For task 2b, we repeated the traffic light controller like in task 2a but this time with RTOS. This worked out great as the system followed how the directions had it lined up. For example, the Pedestrian button is pressed and held by the user for 2 seconds while the system is in the go state, the system should immediately transition to the warn state, and remain there for 5 seconds before moving to the stop state.