CSE 474 Lab Report #4

Luke Jiang, 1560831
Hantao Liu, 1428377

Abstract:

In this lab, we used FreeRtos to re-implement the traffic light control FSM we did in Lab #3. We also implemented the functionality of blinking three LEDs at different frequencies, and explored power mode control of this board.

Introduction:

This lab includes three sections. For section A, we implement a sleep function, which put the board into a low power mode for about three minutes and then resume the previous task.
In Section B, we displayed the coordinates of a touch on LCD and print which button is pressed.
In Section C, we first created three tasks to blink three LEDs at different frequencies. Then, we re-implemented the traffic light control system into the FreeRtos structure, where the queue is used to establish communication between the LCD touch and LED control system.

Procedure:

Section A:

Step 1:
According to the helper posts on piazza, we first set the configUSE_TICKLESS_IDLE bit to 1 in FreeRTOS.h in order to enable portSUPPRESS_TICKS_AND_SLEEP() function:

```
#ifndef configUSE_TICKLESS_IDLE
    #define configUSE_TICKLESS_IDLE 1
#endif
```

Step 2:
Then, we created a sleep() function inside freertos_demo.c:

```
void sleep() {
  for (int i = 0; i < 30000000; i++) {
    portSUPPRESS_TICKS_AND_SLEEP(1);
  }
}
```

which put the system into sleep mode.

Section B:

First of all, we have to make sure we added SSD2119.c and SSD2119.h into the project so that the functions and declaration in there can be successfully implement to activate LCD. Then we have to active LCD with its GPIO and Touch. The following code in main can get this done

```
main(void)
{
    LCD_Init();
    LCD_GPIOInit();
    Touch_Init();
```

The next step is to implement the logic for the LCD. The logic that we used for LCD is as following. First of all, we will enable the reading for touches. Then we will get the coordinate of the point that we have touched. The point that we touched will produce two values correspond to x and y. We then pass the left button value to ui8Message if the left portion is pressed, and right button value to ui8Message if the right portion is pressed.

```
while(1) {

  //if (ui8CurButtonState != ui8PrevButtonState) {
  //  ui8PrevButtonState = ui8CurButtonState;
   // if (ui8CurButtonState != 0) {
   Touch_ReadX();
   Touch_ReadY();
   long res = Touch_GetCoords();
   long yPos = res & 0xFF;
   long xPos = res >> 16;
    LCD_SetCursor(0, 0);
    LCD_PrintInteger(xPos);
    LCD_PrintChar(' ');
    LCD_PrintInteger(yPos);
    LCD_PrintChar('\n');
    if (yPos > 200 && yPos < 800) {
       ui8Message = LEFT_BUTTON;
    } else if (xPos > 200) {
       ui8Message = RIGHT_BUTTON;
    } else {
       ui8Message = NO_BUTTON;
    }
    if (xQueueSend(g_pLEDQueue, &ui8Message, portMAX_DELAY) != pdPASS) {
        // Error. The queue should never be full. If so print the
        // error message on UART and wait for ever.
        UARTprintf("\nQueue full. This should never happen.\n");
        while(1);
    }
    //}
  //}
```

Finally, we have to use function LCD_PrintString to print the message on to the board. This function is also in the SSD2119.h

Section C:

Part 1:

To implement the blinking LED functionality, we have to use a function called xTaskCreate. Rather than initialize LED_task, and switch_task, we can only need to initialize off board LEDs. Then we called xTaskCreate to turn on LED red, green, and blue with different number as parameter indicating different frequencies. After that we can call vTaskStartScheduler to start the scheduler. The code is demonstrated below:

```
int main()
{
  LED_init();
  xTaskCreate(red, "red", 1024, NULL, 1, NULL);
  xTaskCreate(blue, "blue", 1024, NULL, 2, NULL);
  xTaskCreate(green, "green", 1024, NULL, 3, NULL);
  vTaskStartScheduler();
  return 0;
}
```

Part 2:

We then have to implement the traffic light system using FreeRTOS. We use the same implementation of the switch setup as that in section B. However, we have to change LED_task to implement states according to the behavior of the lights. As it requires on the spec, the off state will change to stop state when the start button is pressed, which correspond to Left button on LCD. Then go state will change to stop state back and forth automatically when there is no button pressed. However, when the pedestrian button is pressed, which corresponds to the Right button on LCD, as well as the state is currently at go state, the state will then go to warn state which light up the yellow light. The warn state will then go to stop state automatically. The code demonstration is attached below.

```
// Loop forever.
while (1) {

  switch (curr_state) {
  case OFF_STATE:
    if (xQueueReceive(g_pLEDQueue, &i8Message, 0) == pdPASS) {
      if (i8Message == LEFT_BUTTON) {
        led_ctrl(RED_, 1);
        LCD_PrintString("left button is pressed\n");
        next_state = STOP_STATE;
      } else {
        next_state = OFF_STATE;
      }
    }
  break;
  case GO_STATE:
     if (xQueueReceive(g_pLEDQueue, &i8Message, 0) == pdPASS) {
       if (i8Message == LEFT_BUTTON) {
       led_ctrl(GREEN_, 0);
         LCD_PrintString("left button is pressed\n");
         next_state = OFF_STATE;
       } else if (i8Message == RIGHT_BUTTON) {
         led_ctrl(GREEN_, 0);
         led_ctrl(YELLOW_, 1);
         LCD_PrintString("right button is pressed\n");
         next_state = WARN_STATE;
       } else {
        led_ctrl(GREEN_, 0);
        led_ctrl(RED_, 1);
         next_state = STOP_STATE;
       }
     }
     break;
  case WARN_STATE:
    if (xQueueReceive(g_pLEDQueue, &i8Message, 0) == pdPASS) {
      if (i8Message == LEFT_BUTTON) {
         led_ctrl(YELLOW_, 0);
         LCD_PrintString("left button is pressed\n");
         next_state = OFF_STATE;
      } else {
        led_ctrl(YELLOW_, 0);
        led_ctrl(RED_, 1);
         next_state = STOP_STATE;
      }
    }
    break;
```

```
  case STOP_STATE:
     if (xQueueReceive(g_pLEDQueue, &i8Message, 0) == pdPASS) {
      if (i8Message == LEFT_BUTTON) {
         led_ctrl(RED_, 0);
         LCD_PrintString("left button is pressed\n");
         next_state = OFF_STATE;
      } else if (i8Message == RIGHT_BUTTON) {
         LCD_PrintString("left button is pressed\n");
         next_state = STOP_STATE;
      } else {
         led_ctrl(RED_, 0);
         led_ctrl(GREEN_, 1);
         next_state = GO_STATE;
      }
     }

    break;
  }

  curr_state = next_state;
}
```

Results

Section A:
We successfully implemented the sleep functionality. However, the parameter of function portSUPPRESS_TICKS_AND_SLEEP() doesn't control the duration of sleep mode. We are unable to figure out the cause of this phenomenon and we are going to consult the TA for this matter.

Section B:
We implemented the touch and display functionalities described in the previous section. The coordinates printed on the screen are not the actual coordinates one has pressed. This is a issue we have been facing since the previous lab. However, this issue doesn't cause any problem in the FSM section.

Section C:
We first created three tasks to blink three LEDs in different frequency. Then, we modified the FSM logic in Lab #2, and incorporate the LCD touch reader in Section B into the FSM. The implementation works as expected, except that sometimes it takes longer than two seconds for the system to read a touch input. This is probably due to some delay in the FreeRtos scheduler.

Conclusion

I learned the following skills in this lab:

1) Creating new tasks in FreeRTOS.
2) Modify existing tasks in FreeRTOS to incorporate new functionalities.

3) Utilizing low-power mode control in FreeRTOS

This lab is very helpful for both understanding basic RTOS concepts like tasks and scheduler, and for gaining familiarity with FreeRTOS architecture. Working on this lab make us more ready for our final project.

Reference

1. TM4C123 datasheet
2. EB-LM4F120-L35 datasheet
3. https://piazza.com/class/jikt8d6rgxa689?cid=163
4. Source code in .zip