Lab 4 Proximity Sensor – ECE 5780

Nate Sheffield
A02268057
Nathan Critchfield
A02283426

Objective

The purpose of this lab is to implement a program using FreeRTOS to write to and read from a proximity sensor using UART and incorporate it into our audio amplifier system.

Procedure

For this lab we started with our completed Lab 3. We began by adding the setup for an additional UART port. We added the USART 3 on pin PC4 and PC5. This UART port we connect to the ultrasonic sensor. We added additional functions to our code for reading and writing to and from the added UART port. We then added the additional queues and support code to transfer the data from the UART port to the rest of the code. The additional code provided the functionality to move the receive the additional 't' and 'p' commands from the PC, read the temperature or distance from the sensor and report that reading back to the PC. This functionality was all implemented without affecting the speaker and LED functionality of the project.

Results

For our lab to produce the desired results we had to modify our existing Lab 3 code to be able to incorporate the proximity sensor and connect to it via UART. In order to control both the input from the serial terminal and the proximity sensor we had to implement a second UART. For this lab we were primarily focused on being able to write to the proximity sensor and read from it either a temperature or a proximity measurement. In order to implement this we modified our code so that whenever a "t" was sent through the serial terminal our USART2 interrupt handler passes the data to our proximity task that then writes 0x50 to the sensor and then receives back a temperature measurement. If a "p" was sent our proximity tasks writes 0x55 to the sensor and receives back a distance measurement using ultrasonic sound waves. After writing either 0x50 or 0x55 to the sensor our USART3 read tasks polls for 1 byte of data in a temperature reading and for 2 bytes if it is a distance reading. Then it sends the data to our USART2 write function to write the temperature or distance reading to the serial terminal. Additionally, we wrote our code to be capable of running the proximity sensor and the audio amplifier circuit in such a way that even if we are polling for data from the sensor it does not inhibit the sound being produced from the audio amplifier circuit.

While writing the code for this lab we had lots of issues with trying to write to and read from the proximity sensor correctly. In our lab, we were able to successfully initialize USART3 for the proximity sensor and then connect that UART to the proximity sensor pins and then take the

output from the sensor and pass it to the USART2 write function to write to the serial terminal. However, when we tried to write then read to the sensor we received no data and our code would stall waiting for data from the sensor. After some adjusting we were able to fix our USART3 write function to write the correct value to request a temperature or distance measurement but we still weren't receiving data from the sensor. After a while we found that we had our RX and TX wires between the STM32 and our sensor. Normally for UART, the RX pins connects to the TX pin and vice versa. However, for the proximity sensor the TX pin is directly connected to the other TX and same for RX. After we realized this we were able to get the data written to our sensor but we still were unable to receive data. While debugging our code we commented out the polling function and we were able to write data to the serial terminal but it only contained our default value of 0 which meant we were not getting any data from the sensor. After lots of trial and error we found that the way we were trying to read the bytes from the sensor was not functioning so we redid our USART3 read code. We set it up so that if a "t" was sent we poll for 1 byte of data then send it to our USART2 write function and if a "p" was sent we poll for 1 byte then shift it over by 8 then we poll for the 2nd byte and then send it to USART2 write. After making these changes we were finally able to receive data. However, we found that when we requested a proximity measurment our code would stall. This happened because it would poll and receive the 1st byte but then it would get interrupted by the task scheduler and exit the function and when it returned it would get stuck polling for the 2nd byte but now that byte did not exist anymore so it would stall the code. We then realized that because our LED, button and proximity sensor tasks were all the same priority that our proximity task would get interrupted in the middle of polling for data. We then called the vTaskPrioritySet function to change the priority to be higher while in the proximity task and then reset the priority to its original value as we left the proximity sensor task. This changed then worked great and we were able to request and receive temperature and distance data correctly.

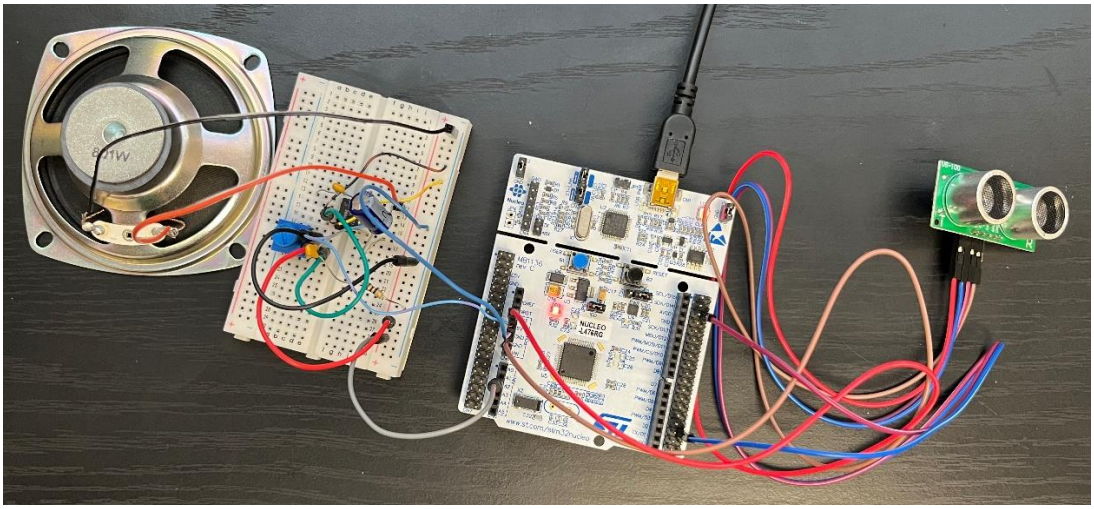

Figure 1. Proximity Sensor and Audio Amplifier Circuit connected to STM32 Nucleo Board

Figure 2. Temperature and Proximity Outputs from Proximity Sensor on Serial Terminal

Conclusion

In conclusion, we learned a lot from the issues of this lab. We learned that for the proximity sensor the TX pin connects to the TX pin and the same for the RX pin. We also learned more about the importance of keeping track of your task priorities and how if all tasks have the same priority, they can interrupt other tasks in the middle when you do not want them to interrupt. We also learned about the importance of matching up your data types and making sure bites do not get over written when saving data from the proximity sensor. Likewise, it is important to make sure you are writing the data out correctly to the terminal to display the data in the right format. Overall, this was a challenging lab for us but in the end, we were able to accomplish all the lab requirements successfully without inhibiting any of our functions we implemented previously.

Appendix

Main.c code

```
#include "FreeRTOS.h"
#include "stm32l476xx.h"
#include "system_stm32l4xx.h"
#include "task.h"
#include "timers.h"
#include "stdint.h"
#include "queue.h"
#include "stdio.h"

#include "init.h"

int main(void) {
    //Initialize System
    SystemInit();
    clock_Config();
    but_led_queue = xQueueCreate(2,sizeof(uint8_t));
    but_tim_queue = xQueueCreate(2,sizeof(uint8_t));
```

```
18.      uart2_to_uart3_queue = xQueueCreate(4,sizeof(uint8_t));
19.      //uart3_prox_sensor_queue = xQueueCreate(8,sizeof(uint8_t));
20.      gpio_Config();
21.      USART2_config();
22.      USART3_config();
23.      timer_Config();
24.      DAC_Config();
25.
26.               //Set the priority for the interrupts
27.      NVIC_SetPriority(USART2_IRQn,0x07);
28.      NVIC_SetPriority(TIM4_IRQn,0x07);
29.
30.      //Task for LED
31.      if(xTaskCreate(LED_task, "LED", 32, NULL, 2, NULL) != pdPASS){
32.               while(1);
33.      }
34.      //Task for Button
35.    if(xTaskCreate(Button_task, "Button", 32, NULL, 2, NULL) != pdPASS){
36.               while(1);
37.      }
38.
39.     if(xTaskCreate(prox_sensor_task, "Proximity Sensor", 256, NULL, 2, &prox_handle) != pdPASS){
40.               while(1);
41.      }
42.
43.      //Start Task Scheduler
44.      vTaskStartScheduler();
45.      while(1);
46. }
47.
48. //Function to toggle led_state
49. void LED_task(void *pvParameters){
50.      static uint8_t buffer[1];
51.      buffer[0] = 0;
52.      while(1){
53.               if(uxQueueMessagesWaiting(but_led_queue) > 0){
54.                        if(xQueueReceive(but_led_queue,buffer,50)== pdTRUE){}
55.               }
56.               //If the LED is off turn it on
57.               if(buffer[0] == 1){
58.                        GPIOA->BSRR |= GPIO_BSRR_BS5;
59.               }
60.               //If the LED is on turn it off
61.               else if(buffer[0] == 0){
62.                        GPIOA->BSRR |= GPIO_BSRR_BR5;
63.               }
64.      }
65. }
66.
67. //Function to read in button state and led_state
68. void Button_task(void *pvParameters){
69.      static uint8_t buffer[1];
70.      buffer[0] = 0;
71.      uint32_t button_in;
72.      while(1){
73.               //Read in the value of the button
74.               button_in = GPIOC->IDR;
75.               button_in &= GPIO_IDR_ID13_Msk;
76.
77.               //If the button is pressed toggle the LED
78.               if(button_in == 0){
79.                        while(button_in == 0){
80.                                 button_in = GPIOC->IDR;
81.                                 button_in &= GPIO_IDR_ID13_Msk;
82.                        }
```

```
                        if(buffer[0] == 0){
                                buffer[0] = 1;
                                //Send led_state to queue for LED Task
                                xQueueSendToBack(but_led_queue,buffer,50);
                                //Send led_state to queue for TIM4_IRQHandler
                                xQueueSendToBack(but_tim_queue,buffer,50);
                        }
                        else {
                                buffer[0] = 0;
                                //Send led_state to queue for LED Task
                                xQueueSendToBack(but_led_queue,buffer,50);
                                //Send led_state to queue for TIM4_IRQHandler
                                xQueueSendToBack(but_tim_queue,buffer,50);
                        }
                }
        }
}

void TIM4_IRQHandler(void){
        static uint32_t sine_count = 0;
        static uint8_t buffer[1];

        const uint16_t sineLookupTable[] = {
        305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,
        609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,
        365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,
        13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,
        188, 216, 245, 275};

        //if there is a message waiting in the queue from ISR
        if(uxQueueMessagesWaitingFromISR(but_tim_queue) > 0){
                xQueueReceiveFromISR(but_tim_queue,buffer,NULL);
        }
        //if the LED is on
        if (buffer[0] == 1){
                sine_count++; //Increment to the next value in the table
                if (sine_count == 64){
                        sine_count = 0;
                }
        }
        //Assign DAC to Sine_Wave Table Current Value
        DAC->DHR12R1 = sineLookupTable[sine_count] + 45;
        TIM4->SR &= ~TIM_SR_UIF; //Clears Interrupt Flag
}

void USART2_IRQHandler(void){
        uint8_t uart_buffer[1];
        uart_buffer[0] = (uint8_t)(USART2->RDR);         //Get serial data
        change_note(uart_buffer[0]);
        //If uart_buffer = t or p send into the queue
        if (uart_buffer[0] == 't' || uart_buffer[0] == 'p'){
                xQueueSendToBackFromISR(uart2_to_uart3_queue,uart_buffer,NULL);
        }
}

void change_note(uint8_t uart_buffer){
        if(uart_buffer == 'a'){
               TIM4->ARR = 0xFFFF008E; //2 MHz/(142) = 14.080 kHz interrupt rate; 220 Hz sine wave
        }
        else if (uart_buffer == 'b'){
                TIM4->ARR = 0xFFFF007E; //126; 246.94 Hz
        }
        else if (uart_buffer == 'c'){
                TIM4->ARR = 0xFFFF0077; //119; 261.63 Hz
        }
```

```
148.      else if (uart_buffer == 'd'){
149.                TIM4->ARR = 0xFFFF006A; //106; 293.66 Hz
150.      }
151.      else if (uart_buffer == 'e'){
152.                TIM4->ARR = 0xFFFF005E;       //94; 329.63 Hz
153.      }
154.      else if (uart_buffer == 'f'){
155.                TIM4->ARR = 0xFFFF0059; //89; 349.23 Hz
156.      }
157.      else if (uart_buffer == 'g'){
158.                TIM4->ARR = 0xFFFF004F; //79; 392.00 Hz
159.      }
160.      else if (uart_buffer == 'h'){
161.                TIM4->ARR = 0xFFFF0046; //71; 440 Hz (High A)
162.      }
163. }
164.
165. void prox_sensor_task(void *pvParameters){
166.      uint8_t uart_buffer[1];
167.      uint16_t measurement;
168.
169.      while(1){
170.                if(uxQueueMessagesWaiting(uart2_to_uart3_queue) > 0){
171.                          if(xQueueReceive(uart2_to_uart3_queue,uart_buffer,50)== pdTRUE){
172.                                    vTaskPrioritySet(prox_handle,3);
173.                                    if (uart_buffer[0] == 't'){
174.                                              USART3_write(0x50);
175.                                              while (!(USART3->ISR & USART_ISR_RXNE));
176.                                              measurement = USART3->RDR;
177.                                              USART2_write(measurement,uart_buffer);
178.                                    }
179.                                    else if (uart_buffer[0] == 'p'){
180.                                              USART3_write(0x55);
181.                                              while (!(USART3->ISR & USART_ISR_RXNE));
182.                                              measurement = USART3->RDR;
183.                                              measurement *= 256;
184.                                              while (!(USART3->ISR & USART_ISR_RXNE));
185.                                              measurement += USART3->RDR;
186.                                              USART2_write(measurement,uart_buffer);
187.                                    }
188.                                    vTaskPrioritySet(prox_handle,2);
189.                          }
190.                }
191.      }
192. }
193.
194. void USART3_write(uint8_t measure_type){
195.      //Send command to measure the temperature or proximity distance
196.      while(!(USART3->ISR & USART_ISR_TXE)); //Wait until hardware sets TXE
197.      USART3->TDR = measure_type & 0xFF; //Writing to TDR clears TXE Flag
198.
199.      //Wait until TC bit is set. TC is set by hardware and cleared by software
200.      while (!(USART3->ISR & USART_ISR_TC)); //TC: Transmission complete flag
201.
202.      //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR
203.      USART3->ICR |= USART_ICR_TCCF; //TCCF: Transmission complete clear flag
204. }
217.
218. void USART2_write(uint16_t measurement,uint8_t uart_buffer[1]){
219.      int nBytes = 18;
220.      char serial_message[18] = {0};
221.      float inches = 0;
222.
223.      //Produce appropriate string for temp or proximity measurement received
224.      if (uart_buffer[0] == 't'){
```

```
225.                sprintf(serial_message,"%i deg F\n\r",measurement);
226.      }
227.      else if (uart_buffer[0] == 'p'){
228.                inches = (float)(measurement / 25.4);
229.                sprintf(serial_message,"%.2f inches\n\r",(double)inches);
230.      }
231.      //Send Serial message to USART2 to send to serial terminal
232.      for (int i=0; i < nBytes; i++){
233.                while(!(USART2->ISR & USART_ISR_TXE)); //Wait until hardware sets TXE
234.                USART2->TDR = serial_message[i] & 0xFF; //Writing to TDR clears TXE Flag
235.      }
236.      //Wait until TC bit is set. TC is set by hardware and cleared by software
237.      while (!(USART2->ISR & USART_ISR_TC)); //TC: Transmission complete flag
238.
239.      //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR
240.      USART2->ICR |= USART_ICR_TCCF; //TCCF: Transmission complete clear flag
241. }
```

Init.c code

```
 1. #include "FreeRTOS.h"
 2. #include "stm32l476xx.h"
 3. #include "system_stm32l4xx.h"
 4. #include "task.h"
 5. #include "timers.h"
 6. #include "stdint.h"
 7. #include "queue.h"
 8.
 9. #include "init.h"
10.
11. void clock_Config(void){
12.      //Change System Clock from MSI to HSI
13.      RCC->CR |= RCC_CR_HSION;          // enable HSI (internal 16 MHz clock)
14.      while ((RCC->CR & RCC_CR_HSIRDY) == 0);
15.      RCC->CFGR |= RCC_CFGR_SW_HSI;     // make HSI the system clock
16.      SystemCoreClockUpdate();
17.
18.      //Turn Clock on for GPIOs
19.      RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
20.      //RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
21.      RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
22. }
23.
24. void gpio_Config(void){
25.      //Set PA5 to output mode for LED
26.      GPIOA->MODER &= ~GPIO_MODER_MODE5_1;
27.      GPIOA->MODER |= GPIO_MODER_MODE5_0;
28.      //Turn LED on
29.      GPIOA->BSRR |= GPIO_BSRR_BS5;
30.      //Set PC13 to input mode for Button
31.      GPIOC->MODER &= ~GPIO_MODER_MODE13; //0xf3ffffff
32. }
33.
34. void timer_Config(void){
35.      //Turn on Clock for TIM4
36.      RCC -> APB1ENR1 |= RCC_APB1ENR1_TIM4EN;
37.
38.      //Enable interrupts for TIM4
39.      NVIC->ISER[0] |= 1 << 30;
40.      NVIC_EnableIRQ(TIM4_IRQn);
41.
42.      TIM4->CR1 &= ~TIM_CR1_CMS;     // Edge-aligned mode
43.      TIM4->CR1 &= ~TIM_CR1_DIR;     // Up-counting
44.
```

```
45.      TIM4->CR2 &= ~TIM_CR2_MMS;    // Select master mode
46.      TIM4->CR2 |= TIM_CR2_MMS_2;    // 100 = OC1REF as TRGO
47.
48.      TIM4->DIER |= TIM_DIER_TIE;    // Trigger interrupt enable
49.      TIM4->DIER |= TIM_DIER_UIE;    // Update interrupt enable
50.
51.      TIM4->CCMR1 &= ~TIM_CCMR1_OC1M;
52.      TIM4->CCMR1 |= (TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2);  // 0110 = PWM mode 1
53.
54.      TIM4->PSC = 0x7;                        // 16 MHz / (7+1) = 2 MHz timer ticks
55.      TIM4->ARR = 0xFFFF008E;      // 2 MHz / (70+1) = 14.080 kHz interrupt rate; 64 entry
look-up table = 220 Hz sine wave
56.      TIM4->CCR1 = 0x23;           // 50% duty cycle (35)
57.      TIM4->CCER |= TIM_CCER_CC1E;
58.
59.      //Enable Control Register 1 for Counting
60.      TIM4->CR1 |= TIM_CR1_CEN;
61. }
62.
63. void DAC_Config(void){
64.      //Turn on Clock for DAC1
65.      RCC -> APB1ENR1 |= RCC_APB1ENR1_DAC1EN;
66.      //Configure DAC1 GPIO in Analog Mode 0x3
67.      GPIOA->MODER |= GPIO_MODER_MODE4;
68.      //Enable DAC1 Channel 1
69.      DAC->CR |= DAC_CR_EN1;
70. }
71.
72. void USART2_config(void){
73.      //Enable PA2 (TX) and PA3 (RX) to alternate function mode
74.      GPIOA->MODER &= ~(0xF << (2*2));
75.      GPIOA->MODER |= (0xA << (2*2));
76.      //Enable alternate function for USART2 for the GPIO pins
77.      GPIOA->AFR[0] |= 0x77 << (4*2); //set pin 2 and 3 to AF7
78.      //High Speed mode
79.      GPIOA->OSPEEDR |= 0xF<<(2*2);
80.      //Pull up mode for PA3 RX
81.      GPIOA->PUPDR &= ~(0xF<<(2*2));
82.      GPIOA->PUPDR |= 0x5<<(2*2); //Select pull-up
83.      //GPIO Output type: 0 = push-pull
84.      GPIOA->OTYPER &= ~(0x3<<2);
85.
86.      //Enable clk for USART2
87.      RCC->APB1ENR1 |= RCC_APB1ENR1_USART2EN;
88.      //Select system clock for USART2
89.    RCC->CCIPR &= ~RCC_CCIPR_USART2SEL_0;
90.    RCC->CCIPR |= RCC_CCIPR_USART2SEL_1;
91.
92.      //Disable USART2
93.      USART2->CR1 &= ~USART_CR1_UE;
94.      //set data length to 8 bits
95.      USART2->CR1 &= ~USART_CR1_M;
96.      //select 1 stop bit
97.      USART2->CR2 &= ~USART_CR2_STOP;
98.      //Set parity control as no parity
99.      USART2->CR1 &= ~USART_CR1_PCE;
100.     //Oversampling to 16
101.     USART2->CR1 &= ~USART_CR1_OVER8;
102.     //Set up Baud rate for USART to 9600 Baud
103.     USART2->BRR = 0x683;                    //1D4C
104.     //USART2 Enable Receiver and transmitter
105.     USART2->CR1 |= (USART_CR1_TE | USART_CR1_RE);
106.
107.     //Enable interrupt for USART2
108.     NVIC_EnableIRQ(USART2_IRQn);
```

```
109.      //Enables interrupts for USART RX
110.      USART2->CR1 |= USART_CR1_RXNEIE;
111.
112.      //Enable USART2
113.      USART2->CR1 |= USART_CR1_UE;
114.
115.      //Verify that USART2 is ready for transmission
116.      while ((USART2->ISR & USART_ISR_TEACK) == 0);
117.      //Verify that USART2 is ready for reception
118.      while ((USART2->ISR & USART_ISR_REACK) == 0);
119. }
120.
121. void USART3_config(void){
122.      //Enable PC4 (TX) and PC5 (RX) to alternate function mode
123.      GPIOC->MODER &= ~GPIO_MODER_MODE4_0;
124.      GPIOC->MODER |= GPIO_MODER_MODE4_1;
125.      GPIOC->MODER &= ~GPIO_MODER_MODE5_0;
126.      GPIOC->MODER |= GPIO_MODER_MODE5_1;
127.      //Enable alternate function for USART3 for the GPIO pins
128.      GPIOC->AFR[0] |= GPIO_AFRL_AFSEL4;              //set pin 4 to AF7
129.      GPIOC->AFR[0] &= ~GPIO_AFRL_AFSEL4_3;
130.      GPIOC->AFR[0] |= GPIO_AFRL_AFSEL5;              //set pin 5 to AF7
131.      GPIOC->AFR[0] &= ~GPIO_AFRL_AFSEL5_3;
132.      //High Speed mode
133.      GPIOC->OSPEEDR |= 0xF << (2*4);
134.      //Pull up mode for PC5 RX
135.      GPIOC->PUPDR &= ~(0xF<<(2*4));
136.      GPIOC->PUPDR |= 0x5<<(2*4); //Select pull-up
137.
138.      //Enable clk for USART3
139.      RCC->APB1ENR1 |= RCC_APB1ENR1_USART3EN;
140.      //Select system clock for USART3
141.   RCC->CCIPR &= ~RCC_CCIPR_USART3SEL_0;
142.   RCC->CCIPR |= RCC_CCIPR_USART3SEL_1;
143.
144.      //Disable USART3
145.      USART3->CR1 &= ~USART_CR1_UE;
146.      //set data length to 8 bits
147.      USART3->CR1 &= ~USART_CR1_M;
148.      //select 1 stop bit
149.      USART3->CR2 &= ~USART_CR2_STOP;
150.      //Set parity control as no parity
151.      USART3->CR1 &= ~USART_CR1_PCE;
152.      //Oversampling to 16
153.      USART3->CR1 &= ~USART_CR1_OVER8;
154.      //Set up Baud rate for USART3 to 9600 Baud
155.      USART3->BRR = 0x683;
156.      //USART3 Enable Receiver and transmitter
157.      USART3->CR1 |= (USART_CR1_TE | USART_CR1_RE);
158.
159.      //Enable USART3
160.      USART3->CR1 |= USART_CR1_UE;
161.
162.      //Verify that USART3 is ready for transmission
163.      while ((USART3->ISR & USART_ISR_TEACK) == 0);
164.      //Verify that USART3 is ready for reception
165.      while ((USART3->ISR & USART_ISR_REACK) == 0);
166. }
```