02/09/24

Lab 3 Audio UART – ECE 5780

Nate Sheffield
A02268057
Nathan Critchfield
A02283426

Objective

The purpose of this lab is to implement a program using FreeRTOS to produce 8 different notes from 220 Hz to 440 Hz when an LED is activated by a button push on our STM32 Nucleo Board. These will be produced as sine waves that are then put through an audio amplifier circuit and an 8-ohm speaker to produce the sound of the sine wave.

Procedure

The first step in this lab was to get the code from the previous lab and import it into a new project. With the old code setup, the next step was to get the UART peripheral functioning on the board. This would allow communication between the board and a UART terminal on a PC. The UART setup included setting up the clock and other initialization registers for the UART. Once UART was set up we began setting up queues to allow communication between the tasks and interrupts. This allowed our system to recognize button presses to turn on and off the sound and LED. Once all of that was set up, we had a functioning Lab 3.

Results

For our lab to produce the desired results we had to modify our existing Lab 2 code to be able to produce 8 sine wave frequencies (notes) from 220 Hz to 440 Hz. These sine wave frequencies are outputed through our audio amplifier circuit when the LED is activated with a button press. In order to switch which note is produced our labs accepts UART serial input from a PuTTY terminal. If a character from ‘a’-‘h’ is sent from the terminal it will change what note is currently being played through the speaker. These notes that are played represent the notes in one octave of a musical scale: low a, b, c, d, e, f, g, and high a. As per the lab requirements our code does not contain global variables and instead uses two different queues to send data between tasks and interrupts. We have a queue to send data between two tasks and then a different queue to send data between a task and an interrupt. This allows our interrupts to be aware if the LED is active or not and whether or not the TIM4 should be outputing the sine wave frequency to the audio amplifier through the DAC.

While writing this lab we had a lot of issues getting our code to function properly. When we first implemented our UART we had several issues with getting all of the necessary intialization steps implemented with the right values and in the right order. We had to refer back to how we initialized our lab in the microcontrollers class and then modify it to implement interrupts and use a different clock. At first we were unable to receive any input from the UART and we could not figure out what was wrong with our initialization. After a lot of looking through the STM32

datasheet we were able to figure out that we had one bit wrong on our clock selection and that our gpio pin was not getting put in the correct mode which was also only off by one bit. After this we were finally able to accept serial input from a PuTTY terminal.

We additionally had issues with getting our queues implemented correctly to eliminate the global variables in our codes. We implemented the correct queue receive and send functions in the proper locations dependant on if it was in an interrupt or a task. However, we were trying to use our queue in the timer interrupt before we had created our queues. After discovering this our queues were able to work but we ran into another issue. After this we were hitting the config assert function which meant our priorities of our tasks and interrupts were off as well as were using too much memory in the creation of our tasks and interrupts. After adjusting our values to make them smaller we were finally able to get our LED, Button, Timer and UART working again.

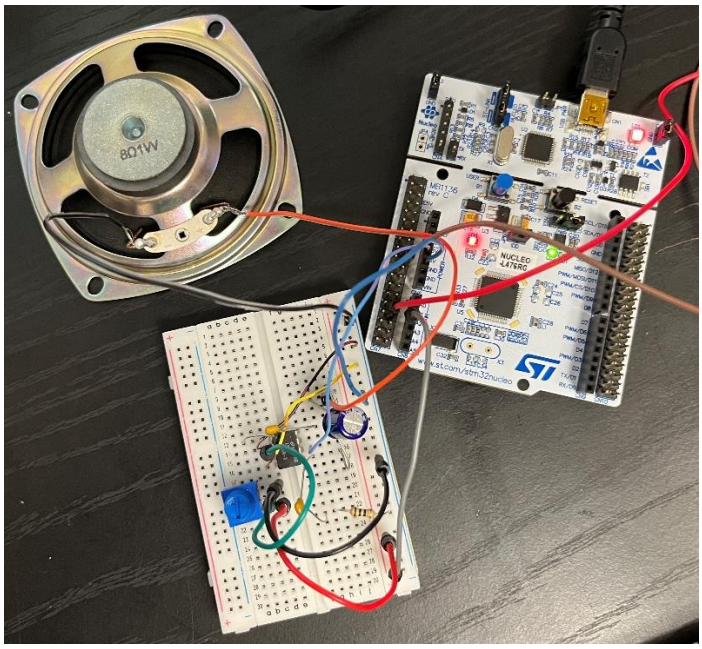

Figure 1. Audio Amplifier Circuit connected to our STM32 Nucleo Board

```
uart_buffer = (uint8_t)(USART2->RDR);
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
}
else if (uart_buffer == 'd'){
  TIM4->ARR = 0xFFFF006A; //106; 293.66 Hz
}
else if (uart_buffer == 'e'){
  TIM4->ARR = 0xFFFF005E; //94; 329.63 Hz
}
else if (uart_buffer == 'f'){
  TIM4->ARR = 0xFFFF0059; //89; 349.23 Hz
}
else if (uart_buffer == 'g'){
  TIM4->ARR = 0xFFFF004F; //79; 392.00 Hz
}
else if (uart_buffer == 'h'){
  TIM4->ARR = 0xFFFF0046; //71; 440 Hz (High A)
}
```

Figure 2. Code Snippet of the ARR values that produce the different note frequencies

Conclusion

In conclusion, we learned a lot from the issues of this lab. Though it was frustrating to run into issue after issue it helped us understand more about the structure of queues and how the UART works using the STM32 and FreeRTOS. We learned that the order of how things are initialized matters and that trying to use a queue before it is created causes your code to get stuck in an infinite loop. We also learned that one or two bits can completely change the initialization of a module and cause it to not work properly. Likewise, it is important to understand the scope of your variables and to make sure that things are defined in the right location and in the right way so their data does not get overwritten or deallocated. Overall, this was a challenging lab for us but in the end, we were able to accomplish all the lab requirements successfully.

Appendix

Main.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. int main(void) {
12.     //Initialize System
13.     SystemInit();
14.     clock_Config();
15.     but_led_queue = xQueueCreate(2,sizeof(uint8_t));
16.     but_tim_queue = xQueueCreate(2,sizeof(uint8_t));
17.     gpio_Config();
18.     UART_config();
```

```
19.      timer_Config();
20.      DAC_Config();
21.
22.               //Set the priority for the interrupts
23.      NVIC_SetPriority(USART2_IRQn,0x07);
24.      NVIC_SetPriority(TIM4_IRQn,0x07);
25.
26.      //Task for LED
27.      if(xTaskCreate(LED_task, "LED", 32, NULL, 2, NULL) != pdPASS){
28.               while(1);
29.      }
30.      //Task for Button
31.   if(xTaskCreate(Button_task, "Button", 32, NULL, 2, NULL) != pdPASS){
32.               while(1);
33.      }
34.
35.      //Start Task Scheduler
36.      vTaskStartScheduler();
37.      while(1);
38. }
39.
40. //Function to toggle led_state
41. void LED_task(void *pvParameters){
42.      static uint8_t buffer[1];
43.      buffer[0] = 0;
44.      while(1){
45.               if(uxQueueMessagesWaiting(but_led_queue) > 0){
46.                        if(xQueueReceive(but_led_queue,buffer,50)== pdTRUE){}
47.               }
48.               //If the LED is off turn it on
49.               if(buffer[0] == 1){
50.                        GPIOA->BSRR |= GPIO_BSRR_BS5;
51.               }
52.               //If the LED is on turn it off
53.               else if(buffer[0] == 0){
54.                        GPIOA->BSRR |= GPIO_BSRR_BR5;
55.               }
56.      }
57. }
58.
59. //Function to read in button state and led_state
60. void Button_task(void *pvParameters){
61.      static uint8_t buffer[1];
62.      buffer[0] = 0;
63.      uint32_t button_in;
64.      while(1){
65.               //Read in the value of the button
66.               button_in = GPIOC->IDR;
67.               button_in &= GPIO_IDR_ID13_Msk;
68.
69.               //If the button is pressed toggle the LED
70.               if(button_in == 0){
71.                        while(button_in == 0){
72.                                 button_in = GPIOC->IDR;
73.                                 button_in &= GPIO_IDR_ID13_Msk;
74.                        }
75.                        if(buffer[0] == 0){
76.                                 buffer[0] = 1;
77.                                 //Send led_state to queue for LED Task
78.                                 xQueueSendToBack(but_led_queue,buffer,50);
79.                                 //Send led_state to queue for TIM4_IRQHandler
80.                                 xQueueSendToBack(but_tim_queue,buffer,50);
81.                        }
82.                        else {
83.                                 buffer[0] = 0;
```

```
 84.                                     //Send led_state to queue for LED Task
 85.                                     xQueueSendToBack(but_led_queue,buffer,50);
 86.                                     //Send led_state to queue for TIM4_IRQHandler
 87.                                     xQueueSendToBack(but_tim_queue,buffer,50);
 88.                             }
 89.                     }
 90.             }
 91. }
 92.
 93. void TIM4_IRQHandler(void){
 94.             static uint32_t sine_count = 0;
 95.             static uint8_t buffer[1];
 96.
 97.             const uint16_t sineLookupTable[] = {
 98.             305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,
 99.             609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,
100.             365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,
101.             13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,
102.             188, 216, 245, 275};
103.
104.             //if there is a message waiting in the queue from ISR
105.             if(uxQueueMessagesWaitingFromISR(but_tim_queue) > 0){
106.                     xQueueReceiveFromISR(but_tim_queue,buffer,NULL);
107.             }
108.             //if the LED is on
109.             if (buffer[0] == 1){
110.                     sine_count++; //Increment to the next value in the table
111.                     if (sine_count == 64){
112.                             sine_count = 0;
113.                     }
114.             }
115.             //Assign DAC to Sine_Wave Table Current Value
116.             DAC->DHR12R1 = sineLookupTable[sine_count] + 45;
117.             TIM4->SR &= ~TIM_SR_UIF; //Clears Interrupt Flag
118. }
119.
120. void USART2_IRQHandler(void){
121.             uint8_t uart_buffer = 0;
122.             //xQueueSendToBackFromISR(uart_tim_queue,&uart_buffer,NULL);
123.             change_note(uart_buffer);
124. }
125.
126. void change_note(uint8_t uart_buffer){
127.             uart_buffer = (uint8_t)(USART2->RDR);
128.             if(uart_buffer == 'a'){
129.             TIM4->ARR = 0xFFFF008E; //2 MHz/(142) = 14.080 kHz interrupt rate; 220 Hz sine wave
130.             }
131.             else if (uart_buffer == 'b'){
132.                     TIM4->ARR = 0xFFFF007E; //126; 246.94 Hz
133.             }
134.             else if (uart_buffer == 'c'){
135.                     TIM4->ARR = 0xFFFF0077; //119; 261.63 Hz
136.             }
137.             else if (uart_buffer == 'd'){
138.                     TIM4->ARR = 0xFFFF006A; //106; 293.66 Hz
139.             }
140.             else if (uart_buffer == 'e'){
141.                     TIM4->ARR = 0xFFFF005E;       //94; 329.63 Hz
142.             }
143.             else if (uart_buffer == 'f'){
144.                     TIM4->ARR = 0xFFFF0059; //89; 349.23 Hz
145.             }
146.             else if (uart_buffer == 'g'){
147.                     TIM4->ARR = 0xFFFF004F; //79; 392.00 Hz
148.             }
```

```
149.      else if (uart_buffer == 'h'){
150.                TIM4->ARR = 0xFFFF0046; //71; 440 Hz (High A)
151.      }
152. }
```

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;       //(70+1) = 14.080 kHz rate; 64 entry = 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 UART_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. }
```

Init.h

```
#ifndef INIT_H
#define INIT_H

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

#define BufferSize 8

static QueueHandle_t but_led_queue;
static QueueHandle_t but_tim_queue;
//static uint8_t buffer[1];

void LED_task(void *pvParameters);
void Button_task(void *pvParameters);

void clock_Config(void);
void gpio_Config(void);

void timer_Config(void);
void DAC_Config(void);
void TIM4_IRQHandler(void);

void UART_config(void);
void USART_read(uint8_t uart_buffer);
void USART2_IRQHandler(void);
void change_note(uint8_t uart_buffer);

#endif
```