

Final Project Theremin – ECE 5780

Nate Sheffield

A02268057

Nathan Critchfield

A02283426

Objective

The purpose of this lab is to implement a program using FreeRTOS to create a digital theremin instrument that can play 25 discrete notes from C3 to C5 and adjust between 8 discrete volumes. Then to use this digital theremin to play a simple song like “Mary had a Little Lamb.”

Procedure

For this lab we started by testing the previous code that we had used in Lab 4. This included code that would run the distance sensor and code that would be able to drive the speaker. Once we were able to confirm that we had things set up correctly for that lab to run we began working on the features we would need to add to the new lab.

The first feature that we added was the additional notes that are needed to fulfil the lab requirements. This included two additional octaves. Once we were able to confirm that these notes functioned correctly, we removed the control of the pitch from the keyboard/UART connection and moved it into the distance sensor control. We would poll the distance sensor for a reading every 100ms then use that reading to determine the pitch to be played through the speaker.

Once we had confirmed that each note was playable by controlling the temperature sensor, we moved on to the volume control with the second distance sensor. This included adding an additional UART connection. The UART that we chose to use was UART 4. We connected the second distance sensor and once we had the UART set up we were able to get readings from that sensor as well.

With the second UART connection setup we next had to find a way to control the volume. We decided to control the amplitude of the sine wave that we send to the DAC. This was done by simply multiplying the outputs to the DAC by a scalar. The scalar that we used was based on the 8 volume levels that were required by the lab. With a scalar of 1 being the highest volume and a scalar of 0 being the lowest volume.

The final step that we had to take was making sure that everything got reported to the serial monitor correctly. We made sure that each volume and note change was displayed on the serial monitor. This required use of a mutex to control access to UART 2, which the serial monitor used, because the volume and note changing functions were contained in different tasks.

Once all of this was finished the lab was basically done. The only thing that we had to do was practice and perform a song. The song that we chose to perform was “Mary had a Little Lamb”. We chose this song for its simplicity and beauty.

Results

For our lab to produce the desired results we had to modify our existing code to incorporate the new feature of a second proximity sensor and to use the proximity sensors to adjust the volume and frequency of the sound produced from the audio amplifier circuit. In order to do this we had to implement a third UART connection that wrote to and read from the volume controlling proximity sensor. While initializing the third UART we had an issue that our UART would not produce any data from the proximity sensor even though we had initialized it the exact same as the previous UARTs. We then found that because we used UART4 that it used a different alternative function mode AF8 instead of AF7 which we had previously used on our other two USARTs. After changing this setting the proximity sensor worked great.

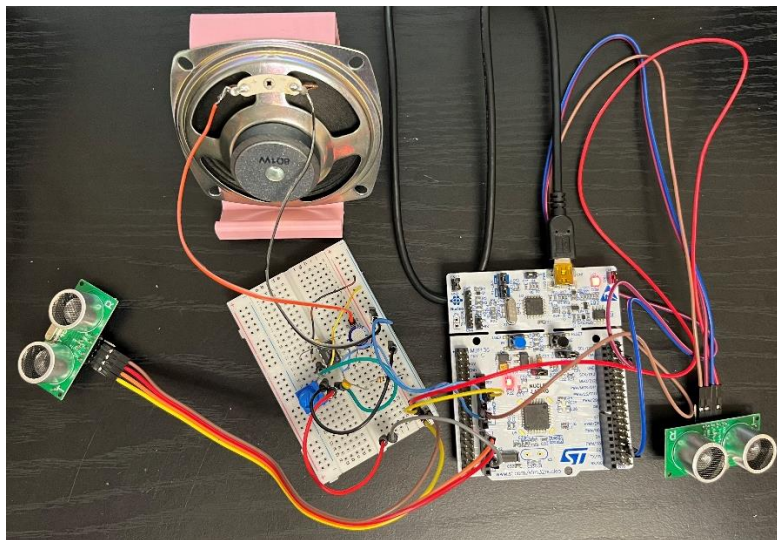
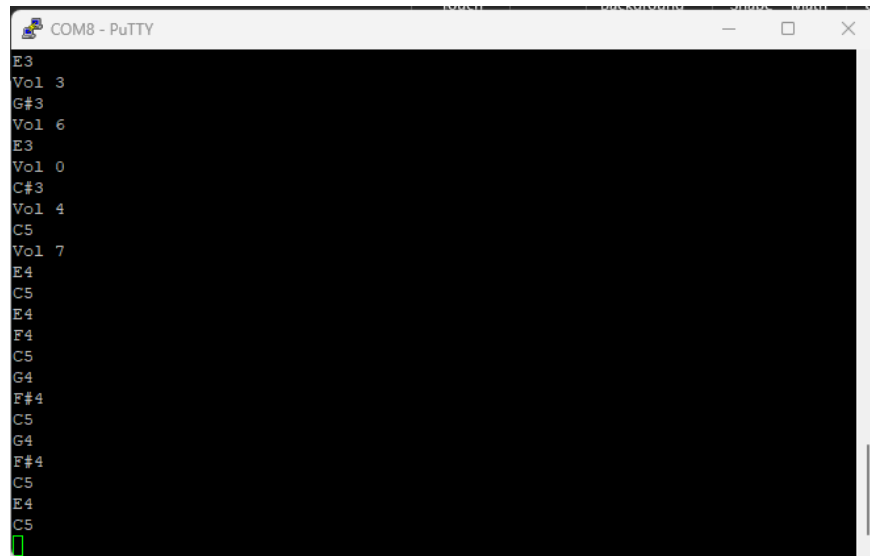


Figure 1. Proximity Sensors and Audio Amplifier Circuit connected to STM32 Nucleo Board

After getting both proximity sensors working, we then modified the number of frequencies we could produce to include all the half step notes from C3 to C5, 25 frequencies in total. We implemented a system to make the distance range for each note to be 1 inch tall. We then implemented a scaling variable inside of our timer Interrupt handler that would scale the output of the interrupt based on the sensor data we got to raise or lower the volume of the audio circuit. This allowed us to produce 8 different volume levels from off at 0 to 7 as full volume. We set the volume to change each two inches for each volume except for the first volume level 0 which was 6 inches because it was the most important to use with the instrument and made it easier to play. We then modified our UART2 output to the serial terminal to send updates with the current volume and note being played if there was a change from the previous note or volume.

After testing the playability of the theremin it was determined that it was very difficult to play with the circuit capable of changing a note 100 times a second. We therefore slowed down the rate of the proximity sensors to run at a 100 ms rate so we could get 10 different notes per second. This made it easier to play and allowed the player to skip in between notes that they did not want to play. After getting the theremin working we then practiced playing the song "Mary had a little lamb" to pass of the final project lab. We then passed it off (With a few mistakes but it least it was kind of recognizable).



```
COM8 - PuTTY
E3
Vol 3
G#3
Vol 6
E3
Vol 0
C#3
Vol 4
C5
Vol 7
E4
C5
E4
F4
C5
G4
F#4
C5
G4
F#4
C5
E4
C5
```

Figure 2. Pitch and Volume Proximity Sensors Outputs on Serial Terminal

Conclusion

In conclusion, we were able to get our final project theremin instrument working. The theremin is able to play 25 distinct notes from C3 to C5 using the measurements from the first proximity sensor. We are also able to change the volume level of the audio circuit using the second proximity sensor to eight distinct volumes from off to fully on. Our theremin also outputs any changes in the current note or volume to a serial terminal through UART2. In the end our theremin was playable, could produce a recognizable song and met all of the final project requirements. However, it was a bit difficult to play even with practice but this came down to user error instead of a design problem.

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. #include "stdio.h"
9. #include "semphr.h"
10.
11. #include "init.h"
12.
13. int main(void) {
14.     //Initialize System
15.     SystemInit();
16.     clock_Config();
17.     but_led_queue = xQueueCreate(2,sizeof(uint8_t));
18.     but_tim_queue = xQueueCreate(2,sizeof(uint8_t));
19.     uart2_to_uart3_queue = xQueueCreate(4,sizeof(uint8_t));
20.     vol_queue = xQueueCreate(2,sizeof(float));
21.
22.     gpio_Config();
23.     USART2_config();
24.     USART3_config();
25.     UART4_config();
26.     timer_Config();
27.     DAC_Config();
28.
29.     //Set the priority for the interrupts
30.     //NVIC_SetPriority(USART2_IRQn,0x07);
31.     NVIC_SetPriority(TIM4_IRQn,0x07);
32.
33.     //Mutex for writing to serial terminal
34.     uart_mutex = xSemaphoreCreateMutex();
35.
36.     //Task for LED
37.     if(xTaskCreate(LED_task, "LED", 32, NULL, 2, NULL) != pdPASS){
38.         while(1);
39.     }
40.     //Task for Button
41.     if(xTaskCreate(Button_task, "Button", 32, NULL, 2, NULL) != pdPASS){
42.         while(1);
43.     }
44.
45.     if(xTaskCreate(note_task, "note_task", 256, NULL, 2, -e_handle) != pdPASS){
46.         while(1);
47.     }
48.
49.     if(xTaskCreate(volume_task, "volume_task", 256, NULL, 2, &vol_handle) != pdPASS){
50.         while(1);
51.     }
52.
53.     //Start Task Scheduler
54.     vTaskStartScheduler();
55.     while(1);
56. }
57.
58. //Function to toggle led_state
59. void LED_task(void *pvParameters){
60.     static uint8_t buffer[1];

```

```

61.     buffer[0] = 0;
62.     while(1){
63.         if(uxQueueMessagesWaiting(but_led_queue) > 0){
64.             if(xQueueReceive(but_led_queue,buffer,50)== pdTRUE){}
65.         }
66.         //If the LED is off turn it on
67.         if(buffer[0] == 1){
68.             GPIOA->BSRR |= GPIO_BSRR_BS5;
69.         }
70.         //If the LED is on turn it off
71.         else if(buffer[0] == 0){
72.             GPIOA->BSRR |= GPIO_BSRR_BR5;
73.         }
74.     }
75. }
76.
77. //Function to read in button state and led_state
78. void Button_task(void *pvParameters){
79.     static uint8_t buffer[1];
80.     buffer[0] = 0;
81.     uint32_t button_in;
82.     while(1){
83.         //Read in the value of the button
84.         button_in = GPIOC->IDR;
85.         button_in &= GPIO_IDR_ID13_Msk;
86.
87.         //If the button is pressed toggle the LED
88.         if(button_in == 0){
89.             while(button_in == 0){
90.                 button_in = GPIOC->IDR;
91.                 button_in &= GPIO_IDR_ID13_Msk;
92.             }
93.             if(buffer[0] == 0){
94.                 buffer[0] = 1;
95.                 //Send led_state to queue for LED Task
96.                 xQueueSendToBack(but_led_queue,buffer,50);
97.                 //Send led_state to queue for TIM4_IRQHandler
98.                 xQueueSendToBack(but_tim_queue,buffer,50);
99.             }
100.            else {
101.                buffer[0] = 0;
102.                //Send led_state to queue for LED Task
103.                xQueueSendToBack(but_led_queue,buffer,50);
104.                //Send led_state to queue for TIM4_IRQHandler
105.                xQueueSendToBack(but_tim_queue,buffer,50);
106.            }
107.        }
108.    }
109. }
110.
111. void TIM4_IRQHandler(void){
112.     static uint32_t sine_count = 0;
113.     static uint8_t buffer[1];
114.     //Valid Volumes: 0 0.1428 0.2856 0.4284 0.5712 0.714 0.8568 1
115.     static float volume[1] = {1.0};
116.
117.     const uint16_t sineLookupTable[] = {
118.         305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,
119.         609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,
120.         365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,
121.         13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,
122.         188, 216, 245, 275};
123.
124.     //if there is a message waiting in the queue from ISR
125.     if(uxQueueMessagesWaitingFromISR(but_tim_queue) > 0){

```

```

126.         xQueueReceiveFromISR(but_tim_queue,buffer,NULL);
127.     }
128.     //if the LED is on
129.     if (buffer[0] == 1){
130.         sine_count++; //Increment to the next value in the table
131.         if (sine_count == 64){
132.             sine_count = 0;
133.         }
134.     }
135.     //if there is a message waiting in the queue from ISR
136.     if(uxQueueMessagesWaitingFromISR(vol_queue) > 0){
137.         xQueueReceiveFromISR(vol_queue,volume,NULL);
138.     }
139.
140.     //Assign DAC to Sine_Wave Table Current Value
141.     DAC->DHR12R1 = (uint16_t)(sineLookupTable[sine_count]* volume[0]) + 45;
142.     TIM4->SR &= ~TIM_SR_UIF; //Clears Interrupt Flag
143. }
144.
145. /*void USART2_IRQHandler(void){
146.     uint8_t uart_buffer[1];
147.     uart_buffer[0] = (uint8_t)(USART2->RDR); //Get serial data
148.     change_note(uart_buffer[0]);
149.     //If uart_buffer = t or p send into the queue
150.     if (uart_buffer[0] == 't' || uart_buffer[0] == 'p'){
151.         xQueueSendToBackFromISR(uart2_to_uart3_queue,uart_buffer,NULL);
152.     }
153. }*/
154.
155. void change_note(uint8_t note){
156.     if (note == 'a'){
157.         TIM4->ARR = 0xFFFF00EF; //239; 130.813 Hz C3
158.     }
159.     else if (note == 'b'){
160.         TIM4->ARR = 0xFFFF00E1; //225; 138.591 Hz C#3
161.     }
162.     else if (note == 'c'){
163.         TIM4->ARR = 0xFFFF00D5; //213; 146.832 Hz D3
164.     }
165.     else if (note == 'd'){
166.         TIM4->ARR = 0xFFFF00C9; //201; 155.563 Hz D#3
167.     }
168.     else if (note == 'e'){
169.         TIM4->ARR = 0xFFFF00BE; //190; 164.814 Hz E3
170.     }
171.     else if (note == 'f'){
172.         TIM4->ARR = 0xFFFF00B3; //179; 174.614 Hz F3
173.     }
174.     else if (note == 'g'){
175.         TIM4->ARR = 0xFFFF00A9; //169; 184.997 Hz F#3
176.     }
177.     else if (note == 'h'){
178.         TIM4->ARR = 0xFFFF009F; //159; 195.998 Hz G3
179.     }
180.     else if (note == 'i'){
181.         TIM4->ARR = 0xFFFF0096; //150; 207.652 Hz G#3
182.     }
183.     else if (note == 'j'){
184.         TIM4->ARR = 0xFFFF008E; //2 MHz/(142) = 14.080 kHz interrupt rate; 220 Hz sine
wave A3
185.     }
186.     else if (note == 'k'){
187.         TIM4->ARR = 0xFFFF0086; //134; 233.082 Hz sine wave A#3
188.     }
189.     else if (note == 'l'){

```

```

190.         TIM4->ARR = 0xFFFF007E; //126; 246.94 Hz B3
191.     }
192.     else if (note == 'm'){
193.         TIM4->ARR = 0xFFFF0077; //119; 261.626 Hz C4
194.     }
195.     else if (note == 'n'){
196.         TIM4->ARR = 0xFFFF0071; //113; 277.183 Hz C#4
197.     }
198.     else if (note == 'o'){
199.         TIM4->ARR = 0xFFFF006A; //106; 293.66 Hz D4
200.     }
201.     else if (note == 'p'){
202.         TIM4->ARR = 0xFFFF0064; //100; 311.127 Hz D#4
203.     }
204.     else if (note == 'q'){
205.         TIM4->ARR = 0xFFFF005E; //94; 329.63 Hz E4
206.     }
207.     else if (note == 'r'){
208.         TIM4->ARR = 0xFFFF0059; //89; 349.23 Hz F4
209.     }
210.     else if (note == 's'){
211.         TIM4->ARR = 0xFFFF0054; //84; 369.994 Hz F#4
212.     }
213.     else if (note == 't'){
214.         TIM4->ARR = 0xFFFF004F; //80; 392.00 Hz G4
215.     }
216.     else if (note == 'u'){
217.         TIM4->ARR = 0xFFFF004B; //75; 415.305 Hz G#4
218.     }
219.     else if (note == 'v'){
220.         TIM4->ARR = 0xFFFF0046; //71; 440 Hz A4
221.     }
222.     else if (note == 'w'){
223.         TIM4->ARR = 0xFFFF0043; //67; 466.164 Hz A#4
224.     }
225.     else if (note == 'x'){
226.         TIM4->ARR = 0xFFFF003F; //63; 493.883 Hz B4
227.     }
228.     else if (note == 'y'){
229.         TIM4->ARR = 0xFFFF003C; //60; 523.251 Hz C5
230.     }
231. }
232.
233. void note_task(void *pvParameters){
234.     uint16_t measurement;
235.     float inches = 0;
236.     uint32_t change = 0;
237.
238.     //Track what is the previous and current note being played
239.     uint8_t note = 'a';
240.     uint8_t prev_note = 'a';
241.     float volume[1] = {1};
242.     //float prev_volume[1] = {1};
243.
244.     while(1){
245.         vTaskPrioritySet(note_handle,3);
246.         USART3_write(0x55);
247.         while (!(USART3->ISR & USART_ISR_RXNE)); //Wait until hardware sets RXNE
248.         measurement = USART3->RDR;
249.         measurement *= 256;
250.         while (!(USART3->ISR & USART_ISR_RXNE)); //Wait until hardware sets RXNE
251.         measurement += USART3->RDR;
252.         inches = (float)(measurement / 25.4);
253.
254.         //Based on the distance from the sensor change the note being played

```

```

255.         if (inches <= 2){
256.             //C3
257.             note = 'a';
258.         }
259.         else if (inches > 2 && inches <= 3){ //C#3
260.             note = 'b';
261.         }
262.         else if (inches > 3 && inches <= 4){ //D3
263.             note = 'c';
264.         }
265.         else if (inches > 4 && inches <= 5){ //D#3
266.             note = 'd';
267.         }
268.         else if (inches > 5 && inches <= 6){ //E3
269.             note = 'e';
270.         }
271.         else if (inches > 6 && inches <= 7){ //F3
272.             note = 'f';
273.         }
274.         else if (inches > 7 && inches <= 8){ //F#3
275.             note = 'g';
276.         }
277.         else if (inches > 8 && inches <= 9){ //G3
278.             note = 'h';
279.         }
280.         else if (inches > 9 && inches <= 10){ //G#3
281.             note = 'i';
282.         }
283.         else if (inches > 10 && inches <= 11){ //A3
284.             note = 'j';
285.         }
286.         else if (inches > 11 && inches <= 12){ //A#3
287.             note = 'k';
288.         }
289.         else if (inches > 12 && inches <= 13){ //B3
290.             note = 'l';
291.         }
292.         else if (inches > 13 && inches <= 16){ //C4 --Mary 13 - 16
293.             note = 'm';
294.         }
295.         else if (inches > 16 && inches <= 17){ //C#4
296.             note = 'n';
297.         }
298.         else if (inches > 17 && inches <= 20){ //D4 --Mary 17 - 20
299.             note = 'o';
300.         }
301.         else if (inches > 20 && inches <= 21){ //D#4
302.             note = 'p';
303.         }
304.         else if (inches > 21 && inches <= 24){ //E4 --Mary 21 - 24
305.             note = 'q';
306.         }
307.         else if (inches > 24 && inches <= 25){ //F4
308.             note = 'r';
309.         }
310.         else if (inches > 25 && inches <= 26){ //F#4
311.             note = 's';
312.         }
313.         else if (inches > 26 && inches <= 29){ //G4 --Mary 26 - 29
314.             note = 't';
315.         }
316.         else if (inches > 29 && inches <= 30){ //G#4
317.             note = 'u';
318.         }
319.         else if (inches > 30 && inches <= 31){ //A4

```



```

319.         note = 'v';
320.     }
321.     else if (inches > 31 && inches <= 32){ //A#4
322.         note = 'w';
323.     }
324.     else if (inches > 32 && inches <= 33){ //B4
325.         note = 'x';
326.     }
327.     else if (inches > 34){
328.         //C5
329.         note = 'y';
330.     }
331.
332.     //Change the note frequency based on distance from sensor
333.     change_note(note);
334.     //Only update output on serial terminal if the note changed
335.     if (note != prev_note ){
336.         change = 0;
337.         xSemaphoreTake(uart_mutex,(TickType_t)50);
338.         USART2_write(note,volume,change);
339.         xSemaphoreGive(uart_mutex);
340.     }
341.     prev_note = note;
342.     vTaskPrioritySet(note_handle,2);
343.     vTaskDelay(100 / portTICK_PERIOD_MS);
344. }
345.
346. void volume_task(void *pvParameters){
347.     uint16_t measurement;
348.     float inches = 0;
349.     uint32_t change = 0;
350.
351.     uint8_t note = 'a';
352.     //uint8_t prev_note = 'a';
353.     float volume[1] = {1};
354.     float prev_volume[1] = {1};
355.
356.     while(1){
357.         vTaskPrioritySet(vol_handle,3);
358.         //Pull distance value from 2nd proximity sensor for volume
359.         UART4_write(0x55);
360.         while (!(UART4->ISR & USART_ISR_RXNE)); //Wait until hardware sets RXNE
361.         measurement = UART4->RDR;
362.         measurement *= 256;
363.         while (!(UART4->ISR & USART_ISR_RXNE)); //Wait until hardware sets RXNE
364.         measurement += UART4->RDR;
365.         inches = (float)(measurement / 25.4);
366.
367.         //Determine what the volume scaler should be based on 2nd proximity sensor
368.         if (inches <= 6){ //volume off
369.             volume[0] = 0;
370.         }
371.         else if (inches > 6 && inches <= 7){
372.             volume[0] = 0.1428;
373.         }
374.         else if (inches > 7 && inches <= 8){
375.             volume[0] = 0.2856;
376.         }
377.         else if (inches > 8 && inches <= 9){
378.             volume[0] = 0.4284;
379.         }
380.         else if (inches > 9 && inches <= 10){
381.             volume[0] = 0.5712;
382.         }

```

```

383.         else if (inches > 10 && inches <= 11){
384.             volume[0] = 0.714;
385.         }
386.         else if (inches > 11 && inches <= 12){
387.             volume[0] = 0.8568;
388.         }
389.         else if (inches > 12){ //Volume on fully
390.             volume[0] = 1;
391.         }
392.
393.         //Send volume multiplier to Timer Interrupt handler
394.         xQueueSendToBack(vol_queue, volume, 50);
395.
396.         //Only update output on serial terminal if the volume changed
397.         if (volume[0] != prev_volume[0]){
398.             change = 1;
399.             xSemaphoreTake(uart_mutex, (TickType_t)50);
400.             USART2_write(note, volume, change);
401.             xSemaphoreGive(uart_mutex);
402.         }
403.         prev_volume[0] = volume[0];
404.         vTaskPrioritySet(vol_handle, 2);
405.         vTaskDelay(100 / portTICK_PERIOD_MS);
406.     }
407. }
408.
409. void USART3_write(uint8_t measure_type){
410.     //Send command to measure the temperature or proximity distance
411.     while(!(USART3->ISR & USART_ISR_TXE)); //Wait until hardware sets TXE
412.     USART3->TDR = measure_type & 0xFF; //Writing to TDR clears TXE Flag
413.
414.     //Wait until TC bit is set. TC is set by hardware and cleared by software
415.     while (!(USART3->ISR & USART_ISR_TC)); //TC: Transmission complete flag
416.
417.     //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR
418.     USART3->ICR |= USART_ICR_TCCF; //TCCF: Transmission complete clear flag
419. }
420.
421. void UART4_write(uint8_t measure_type){
422.     //Send command to measure the temperature or proximity distance
423.     while(!(UART4->ISR & USART_ISR_TXE)); //Wait until hardware sets TXE
424.     UART4->TDR = measure_type & 0xFF; //Writing to TDR clears TXE Flag
425.
426.     //Wait until TC bit is set. TC is set by hardware and cleared by software
427.     while (!(UART4->ISR & USART_ISR_TC)); //TC: Transmission complete flag
428.
429.     //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR
430.     UART4->ICR |= USART_ICR_TCCF; //TCCF: Transmission complete clear flag
431. }
432.
433. void USART2_write(uint8_t note, float volume[1], uint32_t change){
434.     int nBytes = 30;
435.     char serial_message[30] = {0};
436.
437.     //Send update to serial terminal if the note changed
438.     if (change == 0){
439.         switch(note){
440.             case 'a':
441.                 sprintf(serial_message, "C3\n\r");
442.                 break;
443.             case 'b':
444.                 sprintf(serial_message, "C#3\n\r");
445.                 break;
446.             case 'c':
447.                 sprintf(serial_message, "D3\n\r");

```

```
448.         break;
449.     case 'd':
450.         sprintf(serial_message, "D#3\n\r");
451.     break;
452.     case 'e':
453.         sprintf(serial_message, "E3\n\r");
454.         break;
455.     case 'f':
456.         sprintf(serial_message, "F3\n\r");
457.     break;
458.     case 'g':
459.         sprintf(serial_message, "F#3\n\r");
460.         break;
461.     case 'h':
462.         sprintf(serial_message, "G3\n\r");
463.     break;
464.     case 'i':
465.         sprintf(serial_message, "G#3\n\r");
466.         break;
467.     case 'j':
468.         sprintf(serial_message, "A3\n\r");
469.     break;
470.     case 'k':
471.         sprintf(serial_message, "A#3\n\r");
472.         break;
473.     case 'l':
474.         sprintf(serial_message, "B3\n\r");
475.     break;
476.     case 'm':
477.         sprintf(serial_message, "C4\n\r");
478.         break;
479.     case 'n':
480.         sprintf(serial_message, "C#4\n\r");
481.     break;
482.     case 'o':
483.         sprintf(serial_message, "D4\n\r");
484.         break;
485.     case 'p':
486.         sprintf(serial_message, "D#4\n\r");
487.     break;
488.     case 'q':
489.         sprintf(serial_message, "E4\n\r");
490.         break;
491.     case 'r':
492.         sprintf(serial_message, "F4\n\r");
493.     break;
494.     case 's':
495.         sprintf(serial_message, "F#4\n\r");
496.         break;
497.     case 't':
498.         sprintf(serial_message, "G4\n\r");
499.     break;
500.     case 'u':
501.         sprintf(serial_message, "G#4\n\r");
502.         break;
503.     case 'v':
504.         sprintf(serial_message, "A4\n\r");
505.     break;
506.     case 'w':
507.         sprintf(serial_message, "A#4\n\r");
508.         break;
509.     case 'x':
510.         sprintf(serial_message, "B4\n\r");
511.     break;
512.     case 'y':
```

```

513.             sprintf(serial_message, "C5\n\r");
514.             break;
515.         }
516.     }
517.     //Send update to serial terminal if the volume changed
518.     else if (change == 1){
519.         //Print out current volume
520.         if((double)volume[0] < 0.1){
521.             sprintf(serial_message, "Vol 0\n\r");
522.         }
523.         else if((double)volume[0] > 0.1 && (double)volume[0] < .25){
524.             sprintf(serial_message, "Vol 1\n\r");
525.         }
526.         else if((double)volume[0] > 0.25 && (double)volume[0] < .4){
527.             sprintf(serial_message, "Vol 2\n\r");
528.         }
529.         else if((double)volume[0] > 0.4 && (double)volume[0] < .55){
530.             sprintf(serial_message, "Vol 3\n\r");
531.         }
532.         else if((double)volume[0] > 0.55 && (double)volume[0] < .7){
533.             sprintf(serial_message, "Vol 4\n\r");
534.         }
535.         else if((double)volume[0] > 0.7 && (double)volume[0] < .85){
536.             sprintf(serial_message, "Vol 5\n\r");
537.         }
538.         else if((double)volume[0] > 0.85 && (double)volume[0] < 1.0){
539.             sprintf(serial_message, "Vol 6\n\r");
540.         }
541.         else if((double)volume[0] >= 1.0){
542.             sprintf(serial_message, "Vol 7\n\r");
543.         }
544.     }
545.
546.     //Send Serial message to USART2 to send to serial terminal
547.     for (int i=0; i < nBytes; i++){
548.         while(!(USART2->ISR & USART_ISR_TXE)); //Wait until hardware sets TXE
549.         USART2->TDR = serial_message[i] & 0xFF; //Writing to TDR clears TXE Flag
550.     }
551.     //Wait until TC bit is set. TC is set by hardware and cleared by software
552.     while (!(USART2->ISR & USART_ISR_TC)); //TC: Transmission complete flag
553.
554.     //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR
555.     USART2->ICR |= USART_ICR_TCCF; //TCCF: Transmission complete clear flag
556. }
557.

```

Init.c

```

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. #include "semphr.h"
9.
10. #include "init.h"
11.
12. void clock_Config(void){
13.     //Change System Clock from MSI to HSI
14.     RCC->CR |= RCC_CR_HSION; // enable HSI (internal 16 MHz clock)
15.     while ((RCC->CR & RCC_CR_HSIRDY) == 0);
16.     RCC->CFGR |= RCC_CFGR_SW_HSI; // make HSI the system clock

```

```

17.     SystemCoreClockUpdate();
18.
19.     //Turn Clock on for GPIOs
20.     RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOAEN;
21.     //RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
22.     RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
23. }
24.
25. void gpio_Config(void){
26.     //Set PA5 to output mode for LED
27.     GPIOA->MODER &= ~GPIO_MODER_MODE5_1;
28.     GPIOA->MODER |= GPIO_MODER_MODE5_0;
29.     //Turn LED on
30.     GPIOA->BSRR |= GPIO_BSRR_BS5;
31.     //Set PC13 to input mode for Button
32.     GPIOC->MODER &= ~GPIO_MODER_MODE13; //0xf3ffffff
33. }
34.
35. void timer_Config(void){
36.     //Turn on Clock for TIM4
37.     RCC -> APB1ENR1 |= RCC_APB1ENR1_TIM4EN;
38.
39.     //Enable interrupts for TIM4
40.     NVIC->ISER[0] |= 1 << 30;
41.     NVIC_EnableIRQ(TIM4_IRQn);
42.
43.     TIM4->CR1 &= ~TIM_CR1_CMS;    // Edge-aligned mode
44.     TIM4->CR1 &= ~TIM_CR1_DIR;    // Up-counting
45.
46.     TIM4->CR2 &= ~TIM_CR2_MMS;    // Select master mode
47.     TIM4->CR2 |= TIM_CR2_MMS_2;    // 100 = OC1REF as TRGO
48.
49.     TIM4->DIER |= TIM_DIER_TIE;    // Trigger interrupt enable
50.     TIM4->DIER |= TIM_DIER_UIE;    // Update interrupt enable
51.
52.     TIM4->CCMR1 &= ~TIM_CCMR1_OC1M;
53.     TIM4->CCMR1 |= (TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2); // 0110 = PWM mode 1
54.
55.     TIM4->PSC = 0x7;                // 16 MHz / (7+1) = 2 MHz timer ticks
56.     TIM4->ARR = 0xFFFF008E; // 2 MHz / (141+1) = 14.080 kHz interrupt rate; 64 entry look-up
table = 220 Hz sine wave
57.     TIM4->CCR1 = 0x23;                // 50% duty cycle (35)
58.     TIM4->CCER |= TIM_CCER_CC1E;
59.
60.     //Enable Control Register 1 for Counting
61.     TIM4->CR1 |= TIM_CR1_CEN;
62. }
63.
64. void DAC_Config(void){
65.     //Turn on Clock for DAC1
66.     RCC -> APB1ENR1 |= RCC_APB1ENR1_DAC1EN;
67.     //Configure DAC1 GPIO in Analog Mode 0x3
68.     GPIOA->MODER |= GPIO_MODER_MODE4;
69.     //Enable DAC1 Channel 1
70.     DAC->CR |= DAC_CR_EN1;
71. }
72.
73. void USART2_config(void){
74.     //Enable PA2 (TX) and PA3 (RX) to alternate function mode
75.     GPIOA->MODER &= ~(0xF << (2*2));
76.     GPIOA->MODER |= (0xA << (2*2));
77.     //Enable alternate function for USART2 for the GPIO pins
78.     GPIOA->AFR[0] |= 0x77 << (4*2); //set pin 2 and 3 to AF7
79.     //High Speed mode
80.     GPIOA->OSPEEDR |= 0xF << (2*2);

```

```

81. //Pull up mode for PA3 RX
82. GPIOA->PUPDR &= ~(0xF<<(2*2));
83. GPIOA->PUPDR |= 0x5<<(2*2); //Select pull-up
84. //GPIO Output type: 0 = push-pull
85. GPIOA->OTYPER &= ~(0x3<<2);
86.
87. //Enable clk for USART2
88. RCC->APB1ENR1 |= RCC_APB1ENR1_USART2EN;
89. //Select system clock for USART2
90. RCC->CCIPR &= ~RCC_CCIPR_USART2SEL_0;
91. RCC->CCIPR |= RCC_CCIPR_USART2SEL_1;
92.
93. //Disable USART2
94. USART2->CR1 &= ~USART_CR1_UE;
95. //set data length to 8 bits
96. USART2->CR1 &= ~USART_CR1_M;
97. //select 1 stop bit
98. USART2->CR2 &= ~USART_CR2_STOP;
99. //Set parity control as no parity
100. USART2->CR1 &= ~USART_CR1_PCE;
101. //Oversampling to 16
102. USART2->CR1 &= ~USART_CR1_OVER8;
103. //Set up Baud rate for USART to 9600 Baud
104. USART2->BRR = 0x683; //1D4C
105. //USART2 Enable Receiver and transmitter
106. USART2->CR1 |= (USART_CR1_TE | USART_CR1_RE);
107.
108. //Enable interrupt for USART2
109. NVIC_EnableIRQ(USART2_IRQn);
110. //Enables interrupts for USART RX
111. USART2->CR1 |= USART_CR1_RXNEIE;
112.
113. //Enable USART2
114. USART2->CR1 |= USART_CR1_UE;
115.
116. //Verify that USART2 is ready for transmission
117. while ((USART2->ISR & USART_ISR_TEACK) == 0);
118. //Verify that USART2 is ready for reception
119. while ((USART2->ISR & USART_ISR_REACK) == 0);
120. }
121.
122. void USART3_config(void){
123. //Enable PC4 (TX) and PC5 (RX) to alternate function mode
124. GPIOC->MODER &= ~GPIO_MODER_MODE4_0;
125. GPIOC->MODER |= GPIO_MODER_MODE4_1;
126. GPIOC->MODER &= ~GPIO_MODER_MODE5_0;
127. GPIOC->MODER |= GPIO_MODER_MODE5_1;
128. //Enable alternate function for USART3 for the GPIO pins
129. GPIOC->AFR[0] |= GPIO_AFRL_AFSEL4; //set pin 4 to AF7
130. GPIOC->AFR[0] &= ~GPIO_AFRL_AFSEL4_3;
131. GPIOC->AFR[0] |= GPIO_AFRL_AFSEL5; //set pin 5 to AF7
132. GPIOC->AFR[0] &= ~GPIO_AFRL_AFSEL5_3;
133. //High Speed mode
134. GPIOC->OSPEEDR |= 0xF << (2*4);
135. //Pull up mode for PC5 RX
136. GPIOC->PUPDR &= ~(0xF<<(2*4));
137. GPIOC->PUPDR |= 0x5<<(2*4); //Select pull-up
138.
139. //Enable clk for USART3
140. RCC->APB1ENR1 |= RCC_APB1ENR1_USART3EN;
141. //Select system clock for USART3
142. RCC->CCIPR &= ~RCC_CCIPR_USART3SEL_0;
143. RCC->CCIPR |= RCC_CCIPR_USART3SEL_1;
144.
145. //Disable USART3

```

```

146.     USART3->CR1 &= ~USART_CR1_UE;
147.     //set data length to 8 bits
148.     USART3->CR1 &= ~USART_CR1_M;
149.     //select 1 stop bit
150.     USART3->CR2 &= ~USART_CR2_STOP;
151.     //Set parity control as no parity
152.     USART3->CR1 &= ~USART_CR1_PCE;
153.     //Oversampling to 16
154.     USART3->CR1 &= ~USART_CR1_OVER8;
155.     //Set up Baud rate for USART3 to 9600 Baud
156.     USART3->BRR = 0x683;
157.     //USART3 Enable Receiver and transmitter
158.     USART3->CR1 |= (USART_CR1_TE | USART_CR1_RE);
159.
160.     //Enable USART3
161.     USART3->CR1 |= USART_CR1_UE;
162.
163.     //Verify that USART3 is ready for transmission
164.     while ((USART3->ISR & USART_ISR_TEACK) == 0);
165.     //Verify that USART3 is ready for reception
166.     while ((USART3->ISR & USART_ISR_REACK) == 0);
167. }
168.
169. void UART4_config(void){
170.     //Enable PA0 (TX) and PA1 (RX) to alternate function mode
171.     GPIOA->MODER &= ~GPIO_MODER_MODE0_0;
172.     GPIOA->MODER |= GPIO_MODER_MODE0_1;
173.     GPIOA->MODER &= ~GPIO_MODER_MODE1_0;
174.     GPIOA->MODER |= GPIO_MODER_MODE1_1;
175.     //Enable alternate function for UART4 for the GPIO pins
176.     GPIOA->AFR[0] &= ~GPIO_AFRL_AFSEL0;           //set pin 0 to AF8
177.     GPIOA->AFR[0] |= GPIO_AFRL_AFSEL0_3;
178.     GPIOA->AFR[0] &= ~GPIO_AFRL_AFSEL1;           //set pin 1 to AF8
179.     GPIOA->AFR[0] |= GPIO_AFRL_AFSEL1_3;
180.     //High Speed mode
181.     GPIOA->OSPEEDR |= 0xF; //<<(2*0);
182.     //Pull up mode for PA1 RX
183.     GPIOA->PUPDR &= ~(0xF); //<<(2*0));
184.     GPIOA->PUPDR |= 0x5; //<<(2*0); //Select pull-up
185.     //Enable clk for UART4
186.     RCC->APB1ENR1 |= RCC_APB1ENR1_UART4EN;
187.     //Select system clock for UART4
188.     RCC->CCIPR &= ~RCC_CCIPR_UART4SEL_0;
189.     RCC->CCIPR |= RCC_CCIPR_UART4SEL_1;
190.
191.     //Disable UART4 -----
192.     UART4->CR1 &= ~USART_CR1_UE;
193.     //set data length to 8 bits
194.     UART4->CR1 &= ~USART_CR1_M;
195.     //select 1 stop bit
196.     UART4->CR2 &= ~USART_CR2_STOP;
197.     //Set parity control as no parity
198.     UART4->CR1 &= ~USART_CR1_PCE;
199.     //Oversampling to 16
200.     UART4->CR1 &= ~USART_CR1_OVER8;
201.     //Set up Baud rate for UART4 to 9600 Baud
202.     UART4->BRR = 0x683;
203.     //USART3 Enable Receiver and transmitter
204.     UART4->CR1 |= (USART_CR1_TE | USART_CR1_RE);
205.
206.     //Enable USART4
207.     UART4->CR1 |= USART_CR1_UE;
208.
209.     //Verify that UART4 is ready for transmission
210.     while ((UART4->ISR & USART_ISR_TEACK) == 0);
211.

```

```

212.     //Verify that UART4 is ready for reception
213.     while ((UART4->ISR & USART_ISR_REACK) == 0);
214. }

```

Init.h

```

1. #ifndef INIT_H
2. #define INIT_H
3.
4. #include "FreeRTOS.h"
5. #include "stm32l476xx.h"
6. #include "system_stm32l4xx.h"
7. #include "task.h"
8. #include "timers.h"
9. #include "stdint.h"
10. #include "queue.h"
11.
12. #define BufferSize 8
13.
14. static QueueHandle_t but_led_queue;
15. static QueueHandle_t but_tim_queue;
16. static QueueHandle_t uart2_to_uart3_queue;
17. static QueueHandle_t vol_queue;
18. static TaskHandle_t note_handle;
19. static TaskHandle_t vol_handle;
20. static SemaphoreHandle_t uart_mutex;
21.
22. void LED_task(void *pvParameters);
23. void Button_task(void *pvParameters);
24. void note_task(void *pvParameters);
25. void volume_task(void *pvParameters);
26.
27. void clock_Config(void);
28. void gpio_Config(void);
29.
30. void timer_Config(void);
31. void DAC_Config(void);
32. void TIM4_IRQHandler(void);
33.
34. void USART2_config(void);
35. //void USART2_IRQHandler(void);
36. void USART2_write(uint8_t note, float volume[1], uint32_t change);
37. void change_note(uint8_t uart_buffer);
38.
39. void USART3_config(void);
40. void USART3_write(uint8_t measure_type);
41.
42. void UART4_config(void);
43. void UART4_write(uint8_t measure_type);
44.
45. #endif

```