

Lab 4: FREERTOS Scheduling and Multitasking

Connor Lowe, 1573616
Leonard Dul, 2706640
Assignment: ECE474 Lab 4

09-June-2021

Demo Link

[demo.mp4](#)

Introduction

In this laboratory experiment the purpose was to complete four different tasks and a personal project through the use of the FreeRTOS scheduling library. The lab was divided into two sections with the first main goal to complete the RT1-4 tasks, which include blinking an LED, playing music, and computing the average time for 5 FFTs to compute and reporting it to the serial line to the computer. The second section involved implementing a personal project. We chose to make a driving game that would be played on the 8x8 led matrix. Users could use either the rotary encoder or two push buttons to dodge incoming obstacles. The time spent driving is displayed on a 4 digit seven segment display, and when the user crashes their best time spent driving is displayed on the screen. Once users crash, they have the option of performing a soft reset with another push button to restart the game, but save their best kept score. Throughout the game the theme from mario is played on the speaker and when the user crashes a short clip from the mario underworld melody plays. Utilizing FreeRTOS to structure these tasks allowed us to correctly multithread, suspend, and resume tasks.

Methods and Techniques

In order to achieve the learning objectives, the following instruments were used:

- 1) Arduino Mega 2560
- 2) Breadboard
- 3) 330-ohm resistor (2)
- 4) 10-Kohm resistor (3)
- 5) 8ohm speaker
- 5) LED
- 6) USB/Wall adapter
- 7) Arduino IDE
- 8) 5641AS 4 digit 7 segment display

- 9) MAX7291 LED Matrix
- 10) Push buttons (3)
- 11) Quadrature rotary encoder
- 12) jumper wires

Using the mentioned instruments the four learning objectives are described as follows

External Led / RT1(void *pvParameters)

After we finished wiring the external led we identified the correct port (PORTL), data direction register (DDRL), and bit to manipulate (2) in order to flash the led. We then utilized a method of writing to the register at the correct time in order to flash the led, while utilizing the taskDelay FreeRTOS function to achieve proper timing.

16-Bit Timer/Counter and sound / RT2(void *pvParameters) / playMusic(void *pvParameters) / playMusicEnd(void *pvParameters)

To begin this portion of the lab we first needed to identify all the components we would need in our code to use the 16-Bit Timer/Counter. The first step in this process was to choose our timer and then enable it. Using PRTIM4 we set the bit to 0 to enable it as per datasheet instructions. Following this we identified that our 16-bit Timer/Counter used two 8 bit control registers called TCCR4A and TCCR4B. By manipulating the specific bits in our register (WGM40/41/42/43) we could select the desired waveform, in this case a SquareWave output with a 50% duty cycle. Using bits 2:0 on the TCCR4B we also defined the clock select as clkI/O/1 (no prescaling). We also set COM4A0 to control the output pin OCR4A. Using the datasheet we ascertained that we needed to use Pin 6 and set its respective port (PORT H) to have its register (DDRH) output data. With the 16-Bit Timer/Counter established the remaining steps involved creating a function that would loop through the defined melody for the different mario tunes.

Our functions playMusic() and playMusicEnd() loop through their respective melodies and tempo arrays to find the correct duration for each note. Once this duration is found the function will set the timer compare for the appropriate frequency to be played and then delay for the note duration. When the function starts again it will play the next note in the sequence. This is done only once for the music when a crash happens, but the notes loop continuously when driving so that music is always playing.

RT3(void *pvParameters) / RT4(void *pvParameters)

RT3 and RT4 compute FFTs and report the average time every 5 takes back on the serial link. The functions are implemented using two queues. One queue sends the pointer to the data to

perform an FFT on to RT4 and the second queue lets RT3 know when RT4 is done. This allows us to use the millis() command to track the time every 5 takes.

Rotary Encoder / readRotaryEncoder(void *pvParameters)

The implementation of a rotary encoder was new and exciting for this lab.. We did this by writing a task readRotaryEncoder() that would scan the rotary encoder's two inputs. It checks to see if the state is equal to the other, and makes a determination if the knob was spun clockwise or counterclockwise. This is done by checking to see if the clk signal is equal or not to the state of the dt signal, and keeping track of the previous state of the DT signal. This allows us to manipulate a global int to keep track of the car position very accurately.

Buttons / readButtons(void *pvParameters) / restart(void *pvParameters)

3 different push buttons were used in this project. 2 buttons are used to provide an alternative input to the rotary encoder movement, in other words two of the buttons also control the car movement. The third button implements a soft reset for the game, that keeps track of the best score, so that the best score is always displayed when you crash. The function readButtons() that also manipulates the car movements checks for the posedge signal of the pin and moves the car whenever it is received. The restart utilizes the same logic for reading the pin but instead starts up the suspended tasks that keep the game running and reset their initial conditions. A restart is only valid after a crash happens, meaning the tasks must be suspended for them to be resumed again.

4 digit 7 segment display / Task_Counter(void *pvParameters)

The implementation of the 4 digit 7 segment display in our project is taken from Blake Hannaford's implementation given in ECE474 Spring 2021. The task utilizes an ISR to keep track of time accurately. A few key functions are used such as i2ds(int i) which converts an int into a struct containing 4 ints that represent the different decimal places. This makes it easy for the segLightUp(volatile int segs[8]) function to appropriately turn on the correct segment digits based on the number from i2ds(int i).

8x8 led matrix / setLeds(void *pvParameters)

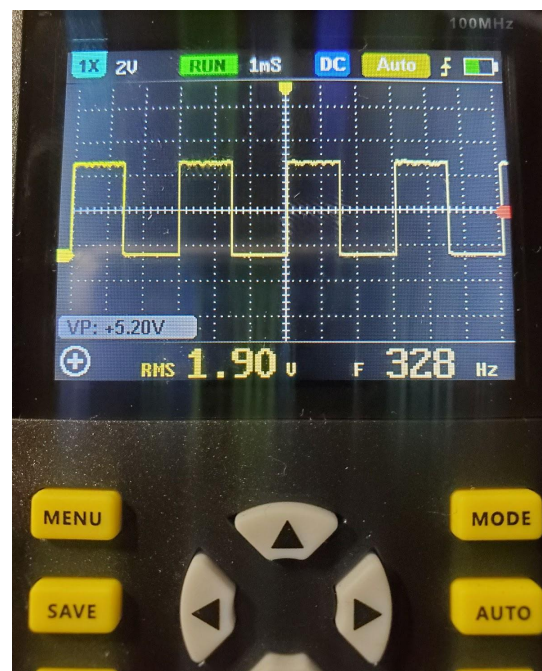
Our 8x8 led matrix used the previously given spiTransfer function written by Ishaan Bhimani. We wrote one new task and two functions to control the led matrix throughout the game. This consists of a clearLeds() function that clears all leds on the matrix. The task to set leds works by shifting the rows of the matrix from right to left, while generating new objects to dodge on the right. Objects are generated one at a time randomly on the right, and the delay is controlled by the time the user has spent driving. The chooseDelay() function takes into account the time spent driving and will delay the task for a shorter period of time, meaning objects will

start coming at the car quicker. If an object in the final row is ever in the same position as the car a crash happens and all tasks that control the game are suspended. When this happens, the best score is saved and displayed on the 4 digit seven segment display and the intro to the underworld theme plays from mario to signal a crash. Once this happens the user has the option to reset the game.

Testing

To ensure we were outputting the correct tones for the tune of Close Encounters we propped the output to the speaker for various tones we wanted to generate to ensure they were correct. After seeing our tones display correctly within 1hz each time on the screen we are confident we can generate sounds accurately.

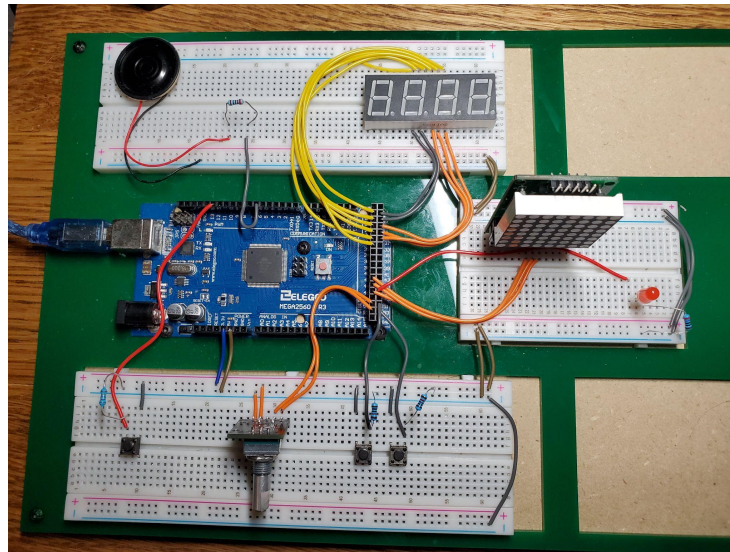
To ensure our tasks kept proper timing one thing we did was probe the signal being generated to the external LED. Verifying that the signal was 1Hz and peaked at around 1.5 volts made us confident our timing and circuit is correct. This value is appropriate because it corresponds to the average output of a 5V signal with a 33.3% duty cycle (100ms on and 200ms off).



Experimental Results

Setup

Figure 1: Breadboard Connection



Power and ground: The power and ground buses are connected on the edges of each breadboard with brown and grey jumper wires. Power and ground are connected to the arduino from the 5 volt and ground connections. A blue wire is used for power and a brown for ground

Speaker: The speaker is connected to ground using its black lead and also connected on the breadboard by its red lead. The red lead is then in series with a resistor that we have been changing out when we want our tone either louder or softer. The other resistor is then connected to the arduino pin 6 with a larger grey jumper.

Led: The red LED is connected in series with 330 ohm resistors that then connect to ground. The anode of the leds are then connected to the arduino for power with the larger grey jumper. The led is connected to pins 47.

5641AS 4 digit 7 segment display: The display makes 11 connections to consecutive pins 22-32 on the arduino. D1 connects to pin 32. A connects to pin 30. F connects to pin 28. D2 connects to pin 26. D3 connects to pin 24. B connects to pin 22. E connects to pin 23. D connects to pin 25. C connects to pin 27. G connects to pin 29. D4 connects to pin 31.

MAX7291 LED Matrix: The LED matrix connects to the breadboard power and ground buses with two small grey jumpers. The three inputs to the matrix (DIN, CS, CLK, from left to

right) are then wired to the arduino with three large orange jumpers. DIN is connected to pin 42, CS is connected to pin 44, and CLK is connected to pin 46.

Buttons: The two buttons for motion are wired to the breadboard power with a small grey jumper and ground with a series 10kOhm resistor. The button is then wired to the arduino right before the resistor with a larger grey jumper that connects to pin 53 for up and 51 for down. The third button for a soft reset is connected the same way but connected to arduino pin 12 with a long red jumper on the left side of the rotary encoder.

Rotary Encoder: The rotary encoder is connected to power and ground with two small orange jumpers. The outputs of the encoder DT and CLK are connected to the arduino with large orange jumpers to pins 50 and 52 respectively.

Figure 2: Arduino Hardware Connections

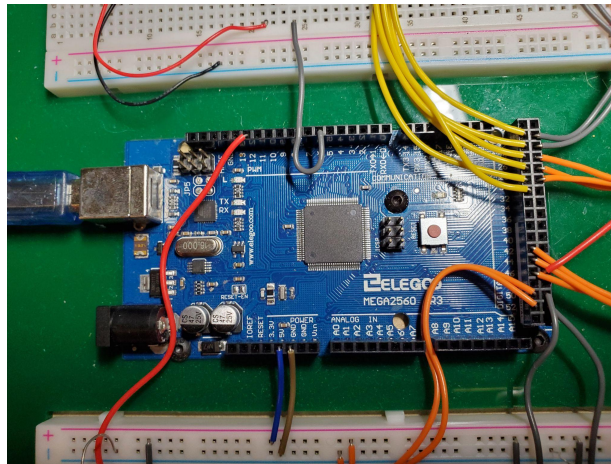
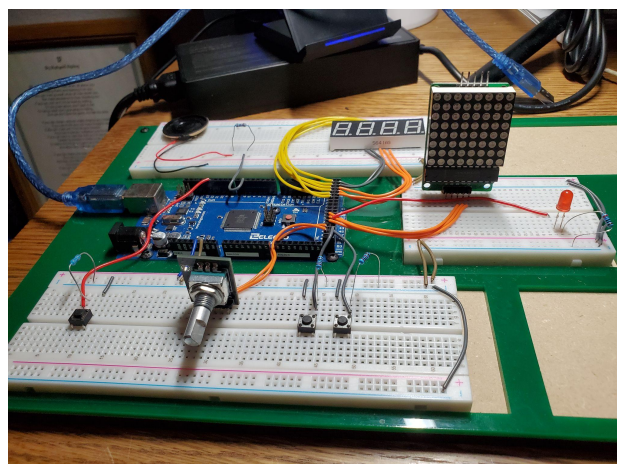


Figure 3: Lab Hardware Setup



Demo 1

Demo 1 had us show 4 different FreeRTOS tasks operating. This includes: RT1 that blinks an LED on and off repeatedly, RT2 that plays the tune from close encounters three times and then stops, RT3 and RT4 which compute the average time for every 5 FFTs and reports that back on the serial line to the computer. We built these tasks up one by one and found RT1 and RT2 to be increasingly easy to implement given our experience in previous labs. We however experienced more trouble when implementing RT3 and RT4 as the size of our FFT was causing stack overflow and we did not realize it for a long time. Once we lowered our FFT size to 256 and used a second queue to notify RT3 that RT4 is done computing we were able to run all 4 tasks simultaneously. For a 256 length FFT our serial line reports back an average time of about 245ms, which is consistent over large periods of time.

Demo 2

Demo 2 had us implement our own personal project and we chose to implement a driving game. This is implemented using a rotary encoder for movement, 2 buttons for movement, a button to reset, a 4 digit seven segment display to show the time driving and keep track of the best time, a speaker to play the music, and a 8x8 red led matrix to play the game on. We found great success in implementing the speaker sound to play the tune from mario as well as the matrix and 4 digit display from our previous implementations in other labs. We did however have trouble when trying to read inputs from the two new devices, the buttons and rotary encoder. We at first tried to just read in the input but quickly realized we need to keep track of the previous state of the button so that we only move on the posedge signal received. This would stop and rapid movement across the screen or multiple button presses the arduino would interpret instead of just one. After we implemented this we were able to successfully interface with the display and the rest of our game relied on logic and FreeRTOS functions. Specifically, we needed to suspend all critical tasks that updated the game when a crash occurred or an obstacle in the last row would overlap with the car. We also needed to resume tasks when the user input reset button is pressed to give the user another shot at the game. This took some trial and error, especially because we tended to have a task suspend itself before its work was done. After these kinks were worked out we had a completely successful implementation of a driving game.

Code Documentation

Please see the included Doxygen HTML folder for the code documentation. Please see the mainpage for a code description and a layout of what functions correspond to what line number.

Overall Performance

We believe we have demonstrated the learning objectives by showing how to implement various tasks with the FreeRTOS scheduling library. This includes high CPU tasks that operate at the maximum FreeRTOS tick of 15ms, which include the tasks that read our input buttons, rotary encoder, and output to our led 4 digit 7 segment display, and 8x8 matrix.

In addition, our demo was executed without any issues and all tasks performed within our expectations. There was precise communication established on how to address each portion of the lab and what our focus needed to be when discussing the code. We believe we have also done a much better job at providing a smooth demo and switching our screen and audio sources when appropriate. We are increasingly happy with how our project turned out and think it would be fun to see it ported to a smaller board and made into a portable game that we could easily show off to other people.

Teamwork Breakdown

For this lab the team worked both together and independently on various parts of the lab. Expectations were set that each member would work on each portion as well as each build their own board with the appropriate connections. For this lab Leonard spent the majority of his time completing the first two portions of the lab and attempting to aid in debugging the FreeRTOS memory issues encountered when trying to compute a lengthy FFT. Connor helped with the successful implementation on all portions of the lab as well as creatively thinking up the solution on how to update the matrix with different periods for the car and incoming obstacles. Because of his efforts both the 8x8 led matrix update task as well as the seven segment update tasks could be completed without any of the previous issues. Connor also was instrumental in helping his teammate understand the code structure and functionality of portions not understood.

Discussion and Conclusion

One of the first challenges faced was implementing the FFT library with the FreeRTOS library. This is because we had an extremely difficult time debugging why there was no output at all and our board kept indicating stack overflow errors. After a lot of trial and error, we realized that we were running out of available space because of the large size of the 512 point FFT, which was simply fixed by lowering this number to 256. This was one of the most time consuming parts of our debugging step as we found it to be a hard error to flag in the code. One of the most interesting things we learned during the lab was how we can utilize the taskSuspend, taskDelete, and taskResume FreeRTOS functions to produce the same behavior as our previous schedulers but much simpler. This allows for not just optimized code because less functions are being called and variables created but also for greater flexibility in stopping and starting tasks when you want. This process was taken even further when we implemented a multitude of tasks that all seem to

run synchronously to produce our game behavior. Another challenging part of this lab was the minimum timing of 15ms that comes with FreeRTOS tasks. This was a little difficult to manage as we needed to take in inputs that were being read at the fastest tick and produce those results on the 8x8 matrix for the car and obstacles. We tried to create a faster tick that would allow the car to blink and be more distinguished from the obstacles, but found the minimum tick of 15ms still much too slow to properly play the game. Overall, we found this lab extremely rewarding, are happy with the success we had in our project, and can confidently implement tasks for the FreeRTOS scheduling library.