

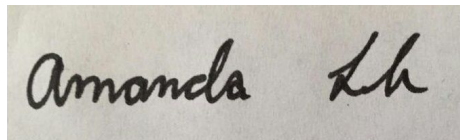
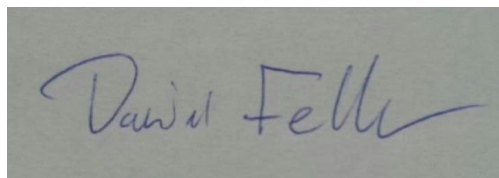
EE 472 Lab 5

RoboTank Final Project

Amanda Loh
Ryan Cysewski
Daniel Feller

June 12, 2015

Team Signatures

A handwritten signature in black ink that reads "Amanda Loh".A handwritten signature in black ink that reads "Ryan Cysewski". Below the signature is a horizontal line, and the word "Signature" is printed to the left of the line.A handwritten signature in blue ink that reads "Daniel Feller".

Introduction

The RoboTank was built to learn the fundamentals of embedded programming. This robotic tank runs on top of FreeRTOS, which is an open source real time operating system. A real time operating system allows functionality to occur instantaneously because data processed without buffering delays. The RoboTank can be controlled remotely from an android app to move forward, back, left, and right. The RoboTank also has the ability to move autonomously based on the environment around it. The tank can sense its environment through four distance sensors that help the tank determine if it will hit obstacles. Two square wave pulses power two motors that rotate the wheels of left and right side of the tank. Changing the width of the wave pulses can change the RoboTank's speed and direction. Also, a square wave pulse is connected to the speaker of the tank to generate music, while the tank drives in the reverse and to the right. The RoboTank is composed of pre built hardware and programmed in C on IAR Workbench. The microcontroller chip used is a Stellaris ARM LM3S8962 microcontroller with peripherals such as speakers, GPIO ports, etc.

Features of the Tank

The tank has many features. Some are part of specification and some were added on our own as designers. The major components of the tank are the ADC Sensor Distance Measuring System, Remote Control Mode, Autonomous Mode, Bluetooth/Keypad Mode, PWM0 Speaker Enable, RoboTank Build on FreeRTOS, and Headlights. See Table 1 for features.

Table 1. Features of the RoboTank

ADC Sensor Distance Measuring System	Four distance sensors are used to detect the distance directions and avoid obstacles. The values for each distance sensor are averaged each time 1,024 samples are taken, in order to obtain a stable value. These values are used in the control logic of the RoboTank system.
Remote Control Mode	The tank is controlled remotely via Bluetooth. We used an Android application to send signals using the microcontroller's UART functionality, which allows us to drive the RoboTank remotely.
Autonomous Mode	The RoboTank has a self driving mode that allows it to drive around without user input, while still avoiding obstacles. There is a button on the Android app reserved for toggling between autonomous and regular mode.
Bluetooth/Keypad Mode	The RoboTank can be controlled via Bluetooth over the UART, or by using the keypad on the Stellaris board. There is a button reserved on the Android app that toggles between the two modes when pressed.
PWM0 Speaker Enable	The system utilized PWM0 for speaker functionality. The speaker plays a melody from the Harry Potter movies when the tank is backing up and the song the

	Entertainer when it is turning right..
RoboTank Build on FreeRTOS	We utilized a real time operating system (FreeRTOS) for our tank. The foundation of this operating system is a priority scheduler that executes tasks based on their importance.
Headlights	The headlights alert things behind it when it backs up by lighting up two LEDs as the car moves backwards.

OLED Display

The OLED display is what is used to display the state of the tank to the user. The OLED is mounted to the Stellaris board that is attached to the tank. The menu looks as shown in Figure 1. The first line is the name of the tank that we called the “Power Weasel.” The next three lines display the sensor distances in mm for all four sensors. The top left number shows the front sensor reading, the bottom left number shows the back sensor reading, the top right number shows the right sensor reading, and the bottom right number shows the left sensor reading. The next line down is the the status for if the car is in remote control mode, “Manual Mode”, or “Autonomous Mode.” The next line down determines if the tank is being controlled by Bluetooth or by the keypad located directly on the Stellaris board. The next line reports the current direction of the car. Finally, the last line displays the current speed for the car, which can be a 1 “slow” or 2 “fast.”



Figure 1. This is the OLED display for the tank. This OLED display shows the current state of the tank.

Software Implementation Details

The design of the RoboTank system is complex, so it is easiest to understand the system by splitting it into five modules. The five modules are: the distance measuring system, the PWM module, the motor control module, the wireless communication system, and the operating system: FreeRTOS. The open source real time operating system called FreeRTOS, is used with the LM3S8962 microcontroller. The microcontroller is programmed to provide several outputs and input data in order to operate the car. The microcontroller takes in input from the sensors to measure distance and wireless commands from an Android phone in remote control mode. Then the microcontroller processes this information to tell the PWMs and, ultimately the motor, how the car should be driven. Figure 2 shows the relationship between the five components that make up the RoboTank.

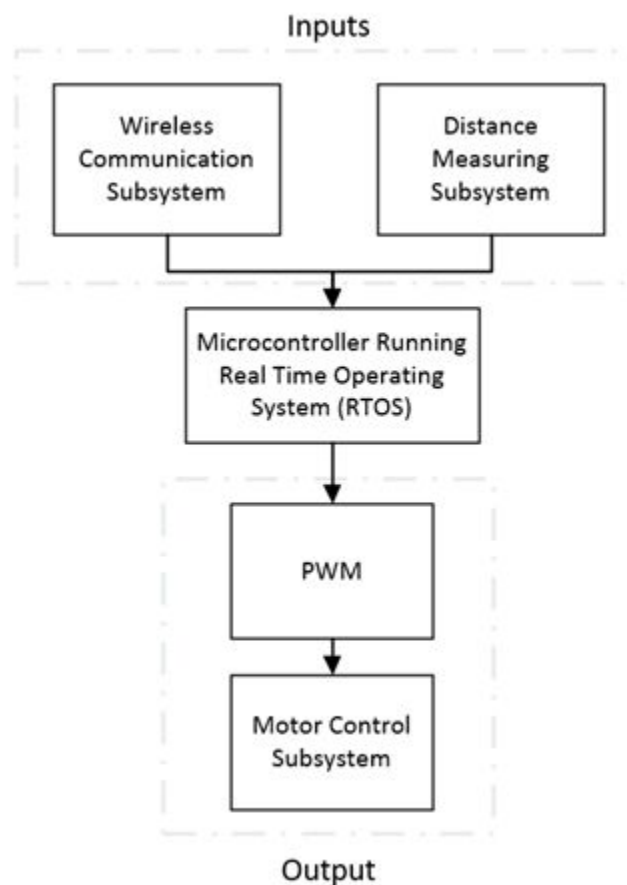


Figure 2. The flowchart shows the relationship between the five fundamental components of the RoboTank system. Notice that the wireless communication subsystem and the distance measuring subsystem are inputs that cause the microcontroller to generate an output to control the PWM and, ultimately, the motor that drives the RoboTank.

The main components of the project that required an immense amount of design, on behalf of the engineers, are listed below. The six components are the ADC Sensor Distance Measuring System, the Remote Control Mode, the Autonomous Mode, and Bluetooth/Keypad Mode, the PWM0 speaker enable, and the FreeRTOS system .

1. ADC Sensor Distance Measuring System

Four Sharp GP2Y0A21YK0F distance sensors are used to detect the distance directions and avoid obstacles. The sensors are placed in a way that they receive data from

the right, left, forward, and back positions of the tank. Analog to digital converters inside the microcontroller convert analog data from the chip from continuous voltage to discrete digital ones and zeros so that the data can be read and analyzed by the microcontroller through software means. The analog to digital converters (ADC) are sampled 1,024 times and then averaged.

The sampling is done because the system clock runs at 8MHz. This clock will cause over eight million samples to be taken for each sensor if a sampling technique is not implemented. By having four ADC sensors implemented, with eight million samples being taken for each ADC, this takes up a lot of the system's time and resources, which is not an efficient use of the system. This lack of efficiency is why a the smaller sampling rate 1,024 is chosen. This was changed from the default lab specification of 4,096 sample requirement because taking less samples made our system more responsive. However, with a smaller sampling rate, there is more inaccuracy, so a good averaging function has to be implemented to correct and restore the accuracy of the data collected from the ADC.

A timer interrupt service routine (ISR) controls how frequently the ADC gathers data from the four sensors. Each time the timer interrupt is fired, a sample is taken from each ADC sensor. As each sample is collected, it is added to a running sum for each sensor. Once 1,024 samples are collected through timer interrupts, the ISR will initiate a task to average the 1,024 samples to improve the accuracy of what has been collected. The ADC sensors convert an analog input of 0 to 3.3 volts to a digital output between 0 and 1023 to represent distance.

Storage space is sacrificed over speed by implementing a large capacity lookup table that converts the digital output received, a number 0 to 1023, to a distance in millimeters. For the 0x2222-0xFFFF case, the millimeter value has to be interpolated. The interpolation was done by constructing a lookup array of length 806 that maps all 56, 797 numbers in-between 0x2222 and 0xFFFF is to a number in millimeters in the range 70 to 800. This algorithm was tedious but efficient in generating fast access from digital output to a millimeter conversion. Figure 3 shows how the sensors were placed on the RoboTank. Our sensors were very responsive due to our fast access array algorithm.

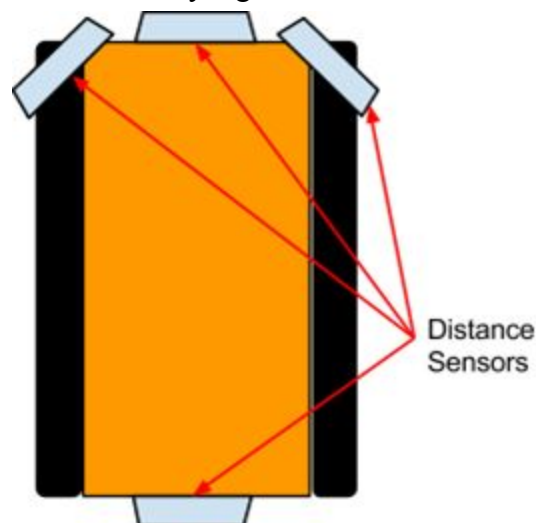


Figure 3. Placing the right and left sensors closer to the front of the car enables the RoboTank to be more effective in autodrive mode. Forward and back sensors are also important to make sure the car does not run into obstacles immediately in front or behind it.

2. Remote control mode:

Remote control mode, or manual mode as it is called on the OLED screen, is the mode for controlling the RoboTank from the Android phone. The remote control mode has eight buttons that can be pressed to control the car. Forward moves the car forward, back moves the car backward, right rotates the car right, left rotates the car left, and forward right turns the car gradual to the right while going forward. Forward left, reverse left, and reverse right are similar to forward right except in different directions. Figure 4 shows a visual image of the android app that can be used to control the car remotely. More about the toggle Bluetooth, toggle speed, and toggle Auto functions will be discussed later.



Figure 4. This is the the android app layout for remote control mode.

3. Autonomous mode:

In autonomous mode, the tank starts by checking to see if it can move forward. If there is room to move forward, it will do so until the sensor detects an obstacle in front. If there is no room to move forward, the tank will stop, and on the next cycle it will check the left and right distance sensors. If the left distance sensor detects that there is room to move, the tank will rotate to the left, and on the next cycle it will do a second check to verify that there is room to move before it begins to go forward. However, if the left distance sensor detects an obstacle, then the tank will check the right distance sensor for obstacles. If both the left, right, and front sensors detect obstacles, then the tank will move in reverse for a set distance, and then check the left and right sensors again. See Figure 5 for a visual flow diagram. Autonomous mode can be toggled by pressing the Toggle Auto button. The current status of the tank will show on the OLED screen.

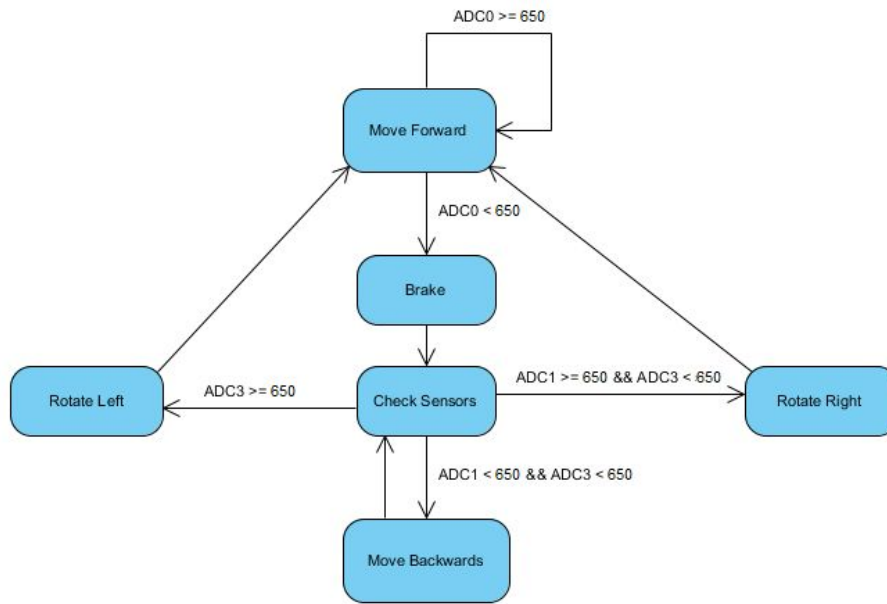


Figure 5: Diagram of autonomous mode behavior

4. Bluetooth/keypad mode:

The system uses a Bluetooth module to communicate between the RoboTank and the controller. The Universal asynchronous receive/transmitter (UART) pin of the microcontroller receives serial data from the Bluetooth modem. Data is then converted from bytes to ASCII characters. These ASCII characters are converted to directional and speed values, which are used by the motor controller task to move the tank

Universal asynchronous receive/transmitter is a piece of hardware that translates data between parallel and serial forms. Serial form involves transferring bits are sent one after the other in a sequence such as 10001111. Parallel form is when bits are read together, like in a byte (8 bits), so that the bits are all read all at once.

In order to initialize the Bluetooth module, we first set it to Command Mode, then Slave Mode. After this, we enabled SPP protocol, turned off encryption, rebooted the device, and exited Command Mode. After we performed this initialization sequence, we were able to pair an Android phone to the Bluetooth using the “Bluetooth SPP Pro” application.

The Bluetooth application allowed us to program keys to send data to the controller in order to operate the tank. The buttons we programmed the keypad to respond to are shown in Table 2. An ASCII value was sent and then translated into a code number that is used in if statements to control the motors.

Table 2. ASCII Value Table for Tank Directions and Code Numbers

Direction/Button	ASCII Character	Code Number
Forward	a	1
Reverse	b	2
Left	c	3
Right	d	4
Forward Left	e	5
Forward Right	f	6
Reverse Right	g	7
Reverse Left	h	8
Standby	i	9
Change Speed	j	10
Bluetooth Toggle	k	11
Auto Mode Toggle	l	12

As seen in Table 2, there are eight different directions the tank can go, as well as buttons to toggle between keypad and Bluetooth mode, and between autonomous and remote control/manual mode. There is also a button to control the speed setting of the RoboTank. Figure 6 below shows the overall block diagram of how the Bluetooth works with the UART on the Stellaris board.

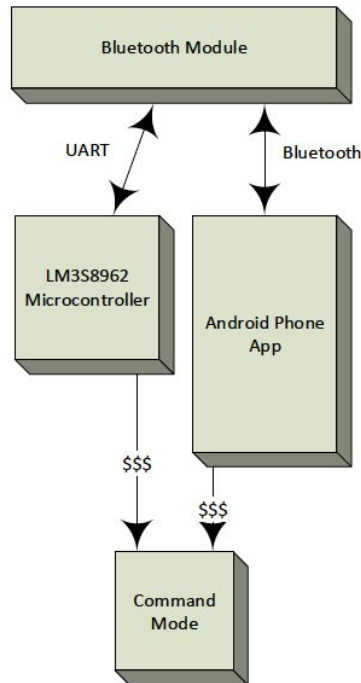


Figure 6. This flow diagram shows a how the microcontroller, Bluetooth chip, and the android application work together. Notice three dollar signs have to be sent out in order to enter command mode to change Bluetooth configurations.

5. PWM 0 Speaker enable

PWM 0 (GPIO Port G, Pin 1) was used for its speaker functionality, in order to play a melody whenever the tank is in reverse. We created an array of frequency values corresponding to the notes of the Harry Potter theme song, and an array of duration values corresponding to the length of each note in the song. We then used the `vTaskDelay` function and our array of duration values to make each note play for the correct amount of time. We did the same thing with the Entertainer song except we programmed it onto the right remote control direction. To play both songs, we had to balance tasks so they would not use the PWM0 at the same time. Therefore, we had to suspend one song while the other one is playing.

6. RoboTank built on RTOS

For using the RTOS, the design mainly involved how to organize the many tasks that the RoboTank had. The way that tasks were organized is that all tasks were set to the same priority. However, tasks that were not needed right away, such as the averaging function for the distance sensor, were suspended until they were required. This way, tasks are called only when they are needed. Also, tasks can share the same priority, which allows two or more tasks to operate at once. For example, it is important that the motor control keeps running to keep the tank running while the ADC averages its samples for accuracy to keep making sure the car does not run into any obstacles. Also, having more than two tasks share a priority is not ideal so making sure only two tasks were running at a time was a primary focus.

7. Headlights

Headlights were implemented so that people can be alerted when the car is in reverse. The lights are LEDs that are red to signify danger. The LEDs were implemented by being connected to Port C. This port was chosen because it did not have any alternate functionality.

Hardware Schematics

Contained in this section are all the hardware pinouts for setting up the Robotank.

ADC Pinout

The ADC pinout for the sensors to the Stellaris board pins are shown below: See Figure 7.

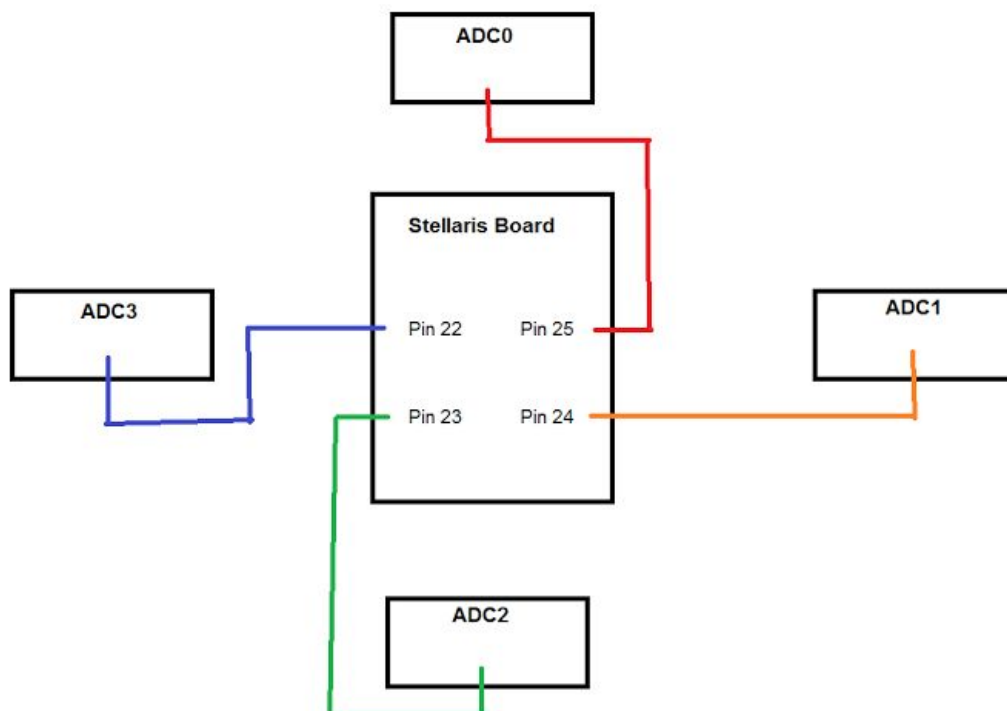


Figure 7: Pinout for the ADCs. ADC0 represents the front of the car. ADC2 is the back of the car. ADC1 is the right side of the car and ADC3 is the left side of the car.

Bluetooth Pinout

This is the hardware schematic for how the Bluetooth module was connected to the Rx, Tx, VCC, and Gnd pins. Notice that the Bluetooth's transmit pin goes to the receive on the UART for the Stellaris board and the Bluetooth's receive pin goes to the transmit on the UART for the Stellaris board. See Figure 8 for the pinout.

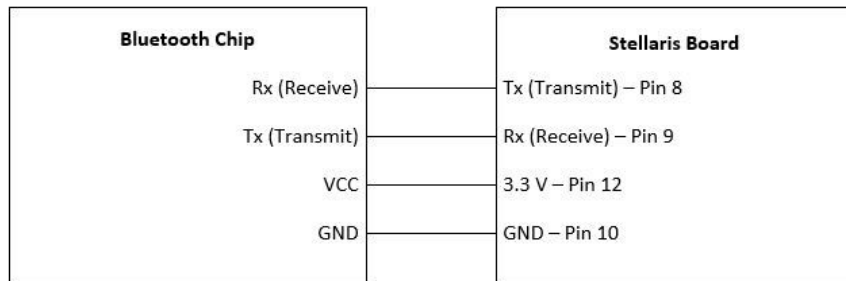


Figure 8. This is the Bluetooth pinout.

H-bridge Pinout

This is the hardware pinout from the Stellaris board to the H-bridge TB6612FNG. This is the driver that communicates to the motors what direction to turn or how fast to rotate given inputs from the Stellaris board. See Figure 9 for a visual of the pinout of the H-bridge TB6612FNG itself. See Figure 10 for a pin connections from the Stellaris board to the H-bridge.

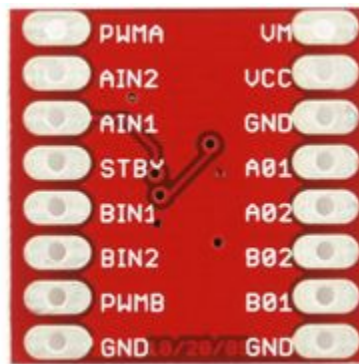


Figure 9. Pinout for Stellaris to H-Bridge TB6612FNG.

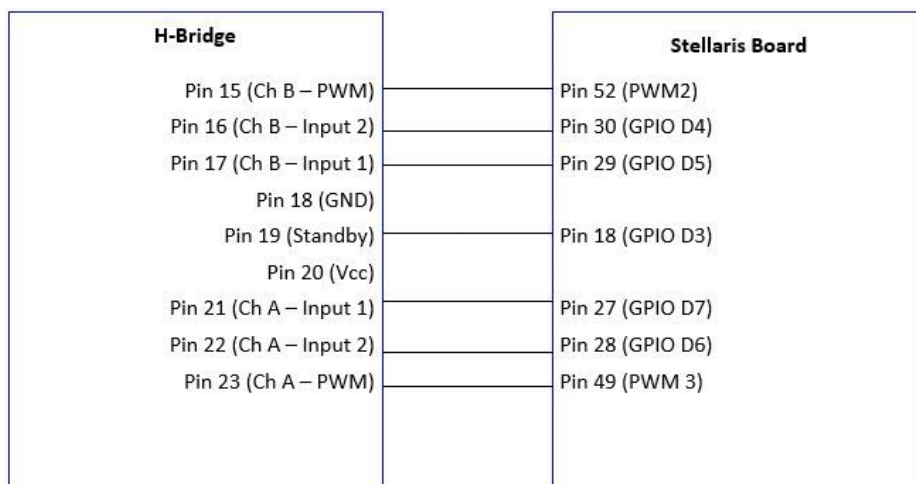


Figure 10. Pinout from Stellaris board to the H-Bridge TB6612FNG.

Team Member Contributions

Amanda Loh

Hours Dedicated to Project: Too many to count (~65)

Design: 40

Coding: 20

Debug/Test: 3

Documentation: 2

What she completed: implementation and design of the headlights, logic for the entertainer song: duration note array, tweaking autonomous mode so that it would work correctly, commented all the code, put the tank together, soldered parts, formatted, edited, and wrote most sections of report, debugged various issues regarding task conflicts, read datasheets to figure out how to configure Bluetooth module, read datasheets in general to design system effectively, and designed most of extra credit car dancing material.

Daniel Feller

Hours Dedicated to Project: 35

Design: 4

Coding: 20

Test/Debug: 10

Documentation: 1

What he completed: worked on autonomous/manual and Bluetooth/button controls, other miscellaneous things, made controls work, helped debug, did some logic for various controls and OLED printing, random debugging stuff, made sure controls worked, and fixed debouncing error with auto mode.

Ryan Cysewski

Hours Dedicated to Project: 35

Design: 10

Coding: 20

Test/Debug: 10

Documentation: 5

What he completed: transcribed the musical notation for the Entertainer song, in code wrote ADC sensor checking portion of the code, wrote some of lab document, wrote portion of the README document, corrected the problem of our tank running into objects before altering course, debugged the Bluetooth initialization until it worked, and wrote code for Bluetooth initialization.

Appendix

Implementing 'safe' inter-task or (inter-process/inner-thread) communication and synchronization is an important concept in multi-tasking (or multiprocessing or multithreading) systems.

‘safe’ inter-task (or inter-process/inner-thread) communication and synchronization in some detail

For our tasks, we had the motor control that drives the motor be of the smallest priority. In RTOS, smaller priorities like 1 or 2 mean the task is not that important. The higher the number task, the more important it is. This is the only task that is not suspended when the FreeRTOS system boots up. All other tasks share a priority of 3: the ADC averaging, as well as the two songs. These tasks are suspended initially, and are run when needed. For example, when the 1,024 samples are taken. The motor control task is but aside briefly so that the ADC sampling task can resume to average and print the distances the car is sensing. This task takes such a short amount of time that the motor control can be but aside for a little while while this task runs. There are many issues that arise if too many tasks are running at the same time so made sure one task was running at a time to decrease possibility of resource competition.

Why it is Important

Since each task in our RoboTank system shares the same resources, we need a way for these tasks to safely and efficiently share resources and communicate between each other while avoiding conflicts. Essentially, the highest priority task that is currently able to execute (not suspended) is the one that the scheduler will execute first. If multiple tasks with equal priority are able to execute, then they will split resources. Also, the inter task communication used queues to share data between tasks. These requirements, as well as a few others that we did not need to implement in our system, form the basis of safe inter-task communication. Without this inter-task communication, tasks would execute without regard for each other, causing conflicts, and they could also use outdated data if there was no queue setup, which could lead to unsafe system behavior.

Examples

One example of inter-task communication is in our RoboTank system. Our RoboTank has 3 different tasks that are all at the same priority level, meaning that they must be able to run at the same time. In order for them to work correctly, it is important that they do not conflict with each other. One example of a possible conflict would be with writing to the OLED display: if more than one task is able to write to the OLED display at the same time, they can interfere with each other and cause the display to malfunction. When designing our software, we made sure that only one task was able to write to the OLED display at any given time, so that we could be sure that the system would work correctly.

Another example is the two songs that are played at different times when the tank moves. These tasks must be carefully handled so both tasks are not sharing the PWM at the same time. This can be handled by creating a global variable. When song one is needed, the

global variable will be set to a 1. Then, the 1 indicates to some logic that it must turn off the second song and resume the task of song one, if the second song is currently playing.