**CG2271 Mini-Project Report**

**Pham Quang Minh, A0170723L | Huu Nguyen, A0170745A | Putra Danish, A0164802J**

In this report, we will elaborate on the tasks that we have implemented for the project, as well as provide an in-depth explanation for the overall RTOS architecture.

**Tasks**

For this project, we used a total of 7 tasks to be scheduled appropriately for the robot. The tasks are as follows, according to priority (Higher number corresponds to higher priority):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Priority | Task | Task Code Name | Period (ms) | |
| 6 | Bluetooth | blueToothTask() | 25 | |
| 5 | Motor | motorTask() | 25 | |
| 4 | Buzzer | buzzerTask() | 25 | |
| 3 | Green LED | greenLedTask() | 25 | 100 |
| 2 | Red LED | redLedTask() | 250 | |
| 1 | Baby Shark Buzzer | babySharkMusicTask() | 400 | |
| 1 | End Buzzer | endChallengeMusicTask() | 400 | |

We schedule the tasks based on RMS priority scheduling policy.

**Overall Architecture**

In order to handle all the data inputs, we designed the architecture to have 4 queues. Each of the 4 queues handles data received from the Bluetooth Task and carries out the appropriate action, according to the task(s) it handles. The table below summarizes each queue.

|  |  |  |
| --- | --- | --- |
| Queue Code Name | Data Handled | Task(s) Handled |
| q\_motordata | Motor | motorTask() |
| q\_greenleddata | Green LED | greenLedTask() |
| q\_redleddata | Red LED | redLedTask() |
| q\_buzzerdata | Buzzer | buzzerTask() |
| babySharkMusicTask() |
| endChallengeMusicTask() |

A simplified diagram of the architecture is shown below.

motorTask()

Application

q\_motordata

babySharkMusicTask()

data

Resume/suspend

greenLedTask()

q\_greenleddata

Bluetooth Task

redLedTask()

q\_redleddata

buzzerTask()

endChallengeMusicTask()

q\_buzzerdata

**Bluetooth Task**

The Bluetooth Task checks if Serial is available every 25ms. If Serial is available, the task sends data input from the Android application to the appropriate queues. Each data input from the Android application comes in bytes. Any instruction, other than an instruction for the robot to move will come in one byte of data. As for the moving instruction, the Bluetooth Task is configured to wait for 2 more bytes before sending the data to the appropriate queue(s). The breakdown of the bytes is shown below.

|  |  |  |
| --- | --- | --- |
| Byte 1 | Byte 2 | Byte 3 |
| Instruction | PWM voltage (% for vertical direction) | PWM voltage (% for horizontal direction) |

A timed threshold is configured to wait for those 2 other bytes, thus receiving the whole packet and sending it to the appropriate queue(s).

Any other instruction only comes in one byte and is fed to the appropriate queue(s).

**Motor Task**

The Motor Task handles the power supplied to the left and right motors by using analogWrite() to control the amount of voltage supplied to each side. The Motor task is blocked until new data is dequeued from q\_motor. This allows the motor to run only when the button is held down on the Android application, instead of continuously running with one press of the button.

**Green LED Task**

The Green LED Task is blocked while waiting for initialization. When the Bluetooth device is initially connected, the Bluetooth sequence for the Green LED will run once. Then, the Bluetooth task will toggle all 8 Green LEDs to “*all on mode*”. At every 150ms cycle, q\_greenleddata will dequeue and the data will be read by the Green LED Task. The Green LED Task then toggles the green LEDs to either “*running mode*” or “*all on mode*” depending on the current state of the green LEDs and what the data from q\_greenleddata is.

**Red LED Task**

The Red LED Task is identical to the Green LED Task. The Red LED Task also has two modes, that is to blink once every cycle or once every two cycles, depending on the durrent state of the Red LEDs and what the data from the q\_redleddata is.

**Buzzer Task**

Similar to the Green LED Task, the Buzzer Task is blocked while waiting for initialization. When the Bluetooth device is initially connected, the Bluetooth Tone will run once. Upon initializing the start of the challenge through the Android application, q\_buzzer will dequeue data that corresponds to running babySharkMusicTask(). At every cycle, q\_buzzer dequeues data that will then toggle between either babySharkMusicTask() or endChallengeMusicTask(), depending if the input by the user is “start” or “end” respectively. While one tone is playing, the other is suspended.

**Baby Shark & End Buzzer Task**

The Baby Shark & End Buzzer Task utilizes the Tone library in C to play different frequency tones with varying delay times to produce music that closely resembles the famous baby shark song and Harry Potter™ theme. The loop is run perpetually, thus either Task only needs to run once to run forever. This is of course until the Task is suspended manually by toggling it to “*end*” or “*start*”, depending on the current tone playing.

**Conclusion**

A few of the challenges we faced along the way include the Green LEDs not being able to have full brightness while on “*all on mode*” and the Buzzer Tasks running indefinitely.

As for the Green LED obstacle, we have decided that it was a minute compromise as it still completes the criteria of having all 8 Green LEDs light up, albeit dim. This is due to the limitation of the driver chip not being able to supply more power when all Green LEDs are lit at the same time. Fortunately, this is not a problem when the Green LEDs are in “*running mode*” as each Green LED is able to receive full power from the chip.

The Buzzer Tasks will always run after initialization. We currently have no way of stopping the music through the Android application. The only way to truly turn off the sound is by manually resetting the Arduino on the robot itself. This is inefficient.

Overall, we have deemed these problems to be non-issues and may be fixed, given more time. Despite these flaws, the robot meets all the criteria for this mini-project and hence was a success.