**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 and the importance of each task.

**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() |

A simplified diagram of the architecture is shown below.

babySharkMusicTask()

motorTask()

q\_motordata

Application

data

Resume/Suspend

greenLedTask()

q\_greenleddata

blueToothTask()

redLedTask()

q\_redleddata

endChallengeMusicTask()

buzzerTask()

q\_buzzerdata

**Bluetooth Task**

The Bluetooth Task checks if Serial is available every 25ms, which is the signal sending period of the Smartphone controller application. If Serial is available, the task sends data input from the application to the appropriate queues. Each data input from the application comes in bytes. Any instruction, other than an instruction for the robot to move will come in one byte of data. If the received byte is the moving instruction, the Bluetooth Task is configured to wait and receive 2 more bytes by polling (with the polling deadline sets to be 10ms) before sending the data to the appropriate queue(s). The breakdown of the bytes is shown below.

|  |  |  |
| --- | --- | --- |
| Byte | Data | Range of data |
| 1st byte | Instruction | BLUETOOTH\_CONNECTED, START\_CHALLENGE, END\_CHALLENGE, STOP, MOVING |
| 2nd byte | Forward/backward power % | -100 => +100 (only when 1st byte is MOVING) |
| 3rd byte | Left/right turning power % | -100 => +100 (only when 1st byte is MOVING) |

**Motor Task**

The Motor Task handles the movement of the robot. It is set to be blocked to wait until it can retrieve new data from the q\_motordata queue. After receiving the data, it will make calculations to determine the PWM duty cycle for each pair of motors on both sides. The task will then do a one-time analogWrite() to all required pins to control the movement of the robot before returning to the blocked state and repeat the cycle of waiting for new data. This allows the motor to run only when the button is held down on the Smartphone controller application, instead of continuously running with one press of the button.

**Green LED Task**

After initializing all 8 LEDs to turn on, the Green LED Task is blocked while waiting for the Bluetooth connection signal from the q\_greenleddata queue. When it receives the signal, it will control the 8 LEDs to flash twice before setting all of them on once again and set the internal state to stationary. From then on, it will run periodically with a cycle of 100ms. At the start of each cycle, it will check for new data from the queue without waiting. If there is new data, it will dequeue the data and change the internal state accordingly. After that, it will control the output of the LEDs according to the current internal state by writing digital signals to a Shift Register. It will either only set one of the LEDs to turn on (which is determined by the LED running mode pattern) when the robot is moving or light up all LEDs when the robot is stationary.

**Red LED Task**

The Red LED Task run in cycles of 250ms. Like the Green LED Task, it will check new data from the q\_redleddata queue at the start of the cycle and change the internal state accordingly. It will then decide how it would toggle its output signal to the 8 red LEDs (toggle once every cycle when the robot is stationary, once every two cycle when the robot is moving), depending on its current internal state.

**Buzzer Task**

At the start, the Buzzer Task waits for the Bluetooth connection signal in the same way as the Green LED Task. When the signal comes, it will play the Bluetooth connection tone once. After that, it begins its cycle of waiting for new data (in blocked state) similar to how the Motor Task works. When new data comes, it will dequeue from q\_buzzerdata queue and see whether it should resume/suspend the Baby Shark Music Task if it is off/on and suspend the End Challenge Music Task (when the robot begins the challenge run) or resume/suspend the End Challenge Music Task if it is off/on and suspend the Baby Shark Music Task (when the robot ends the challenge run).

**Baby Shark & End Challenge Music Task**

The Baby Shark & End Buzzer Task utilizes the Arduino’s Tone library to play different frequency tones with varying delay times to produce music that closely resembles the famous baby shark song and Harry Potter™ theme. At the start, both tasks are suspended until one of them is resumed by the Buzzer Task. Once a task starts, it will continuously play its piece of music in infinite loop until it is suspended by the Buzzer Task. At all time, there will be at least one of the two tasks suspended. The cycle of both tasks is one-eighth of the time duration of a basic music note duration (400ms).

**Conclusion**

One problem we faced at the end of the project is the green LEDs not being able to have full brightness while all 8 of them are turned on at the same time.

For this problem, 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 at any one time.

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