CG2271 Project Report

Team

Ng Wei Jie, Brandon (e0315868@u.nus.edu)

Joseph Wong Yefeng (e0311252@u.nus.edu)

Nishanth Elango (e0311168@u.nus.edu)

RTOS Architecture

The RTOS architecture on FRDM-KL25Z is shown in Figure 1. There are a total of 7 threads, 1 interrupt, 1 message queue, 1 global variable, 1 event flag, and 1 mutex. The arrows indicate how the data is communicated and flowed from the interrupt to the threads.

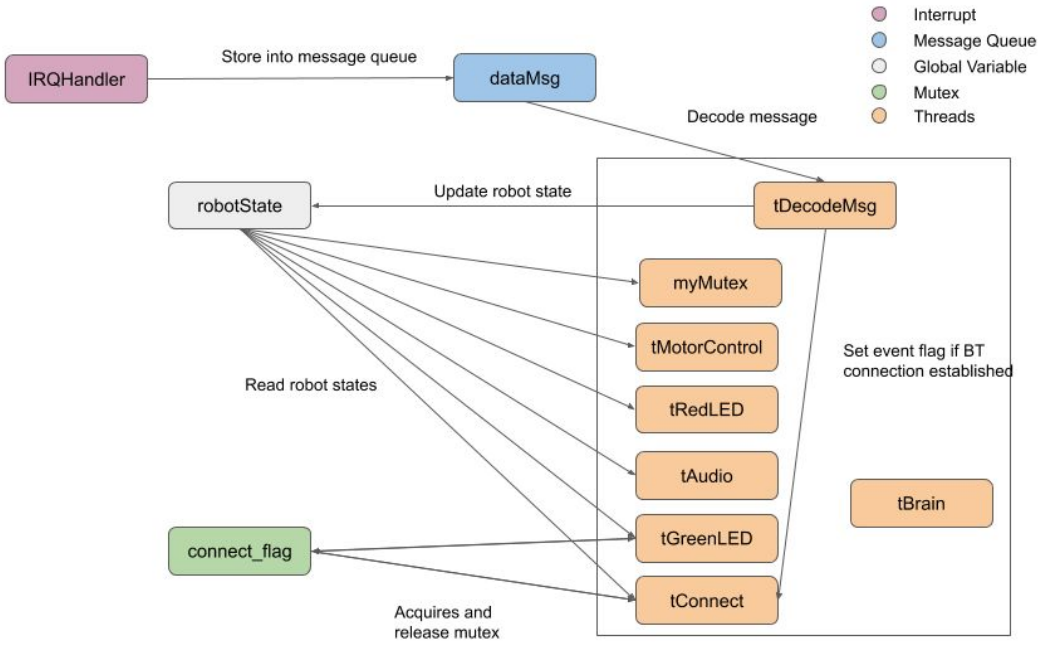

Figure 1: RTOS Architecture

Message Structure

The software created a struct myDataPkt that contains the attributes cmd and data. Both cmd and data indicate the state of the robot as described in the table below.

myDataPkt		cmd				
		0x00	0x01	0x02	0x03	0x04
data	-	INIT	START	STOP	MOVE	END
	0x00	-	-	-	Stop	-
	0x01	-	-	-	Forward Straight	-
	0x02	-	-	-	Forward Left	-
	0x03	-	-	-	Forward Right	-
	0x04	-	-	-	Backward Straight	-
	0x05	-	-	-	Backward Left	-

	0x06	-	-	-	Backward Right	-
	0x07	-	-	-	Rotate Left	-
	0x08	-	-	-	Rotate Right	-

Interrupts

Interrupts	Explanation
UART2_IRQHandler	Incoming data from bluetooth triggers an uart interrupt. The interrupt handler is UART2_IRQHandler. It receives 8 bit data from the mobile app. The data encodes the user actions to perform on the robot. An interrupt is triggered whenever it receives data, and this is preferred over polling that could affect real time performance of the robot.

Message Queue

Message Queue	Explanation
dataMsg	Incoming data from bluetooth triggers an interrupt. The data from the interrupt is converted into a struct type myDataPkt and stored into the message queue. A message queue is an intermediary to help to transfer data from the interrupt to threads. This design is preferred over global variables to communicate between threads and interrupts directly.

Global Variable

Global Variables	Explanation
robotState	New data in the message queue is decoded inside a thread and updated in the variable robotState that has the struct type myDataPkt. Other threads read the updated robotState and change behaviour accordingly. The variable robotState is only updated from the thread that decodes the data in the message queue. This design is preferred to prevent race conditions where multiple threads update the same variable.

Event Flags

Event Flags	Explanation
connect_flag	An event flag is set whenever a command is sent to acknowledge that bluetooth connection is successfully established. This event flag allows the thread to make the green LEDS blink twice. An event flag allows the thread to run a set of actions and then return to the block state. This does not require updates in global variables to control the actions. Thus, this design reduces global variables.

Mutex

Mutex	Explanation
myMutex	The mutex flag is required because of the design and implementation. There are two threads that affect the state of the green LEDs. One thread causes the green LEDs to blink twice upon bluetooth connection, and the

	other causes the green LEDs to light up depending on the robot's moving state. A mutex is preferred to control the two threads instead of a control logic. This reduces the need for more global variables, or the possibility of more complicated mutex.

Threads

Threads	Explanation
tBrain	The task tBrain is responsible for setting up uart to receive data from the bluetooth module through uart interrupts. Data received from the bluetooth module triggers an interrupt. The received data is an 8 bit data that is converted into a struct type myDataPkt and stored into a message queue.
tMotorControl	The task tMotorControl is responsible for reading the decoded data from robotState. It controls the PWM that allows the robot to stop, curve, or move straight in forward and reverse directions. The robot moves only when the robot is in the state MOVE as specified in the attribute cmd of robotState. The directions are specified in the attribute data of robotState.
tRedLED	The task tRedLED is responsible for reading the decoded data from robotState. It controls how fast the red LEDs blink depending on whether the robot is moving or not.
tGreenLED	The task tGreenLED is responsible for reading the decoded data from robotState. It controls whether the greenLEDs are all lit up or in a running state. It also acquires and releases a mutex due to a race condition with the tConnect thread.
tAudio	The task tAudio is responsible for reading the decoded data from robotState. It controls the music that is played. The starting theme is Super Mario Bros. Underground Theme, running theme is Super Mario Bros Main Theme, and completed theme is Super Mario Bros. Music - Game Over.
tConnect	The task tConnect is responsible for making the green LEDs flash twice. It is in a blocked state by default. Only when a bluetooth connection is made, an event flag is set and the task goes to a running state. It also acquires and releases a mutex due to a race condition with the tGreenLED.
tDecodeMsg	The task tDecodeMsg is responsible for getting the data in the message queue and updating the global variable robotState. The other threads do not communicate directly with the interrupt through a global variable as it is considered bad practice. Therefore, the received data from the interrupt is put into a message queue as an intermediary step, decoded in the msg queue, updates a global variables robotState such that other threads read the data but not update it. If the attribute of cmd in robotState is of INIT, the event flag is set to change the tConnect from the blocked to running state.

Design Considerations

- A message queue is an intermediary that holds data from interrupts. The data is decoded in a task tDecodeMsg to update a global variable robotState so that all the other tasks read from the global variable instead of communicating directly with the interrupt via a global variable.

- The green LEDs are controlled by two threads tConnect and tGreenLED. The tConnect is running when an event flag is set. Both threads are not combined because the code to make the green LEDs blink twice occurs infrequently. There is a racing condition between the two threads and is resolved through mutex.
- Only the tDecodeMsg updates the global variable, and the other threads read from the global variable. This prevents racing conditions where multiple threads are updating the same global variable.
- Semaphores are not used in the project as all the project requirements are made. Adding semaphores will only add unnecessary code and make the design and rtos architecture bloated and messier.

Sequence Diagram

Figure 2 shows a sequence diagram on the flow of control on various tasks and how the tasks communicate with each other.

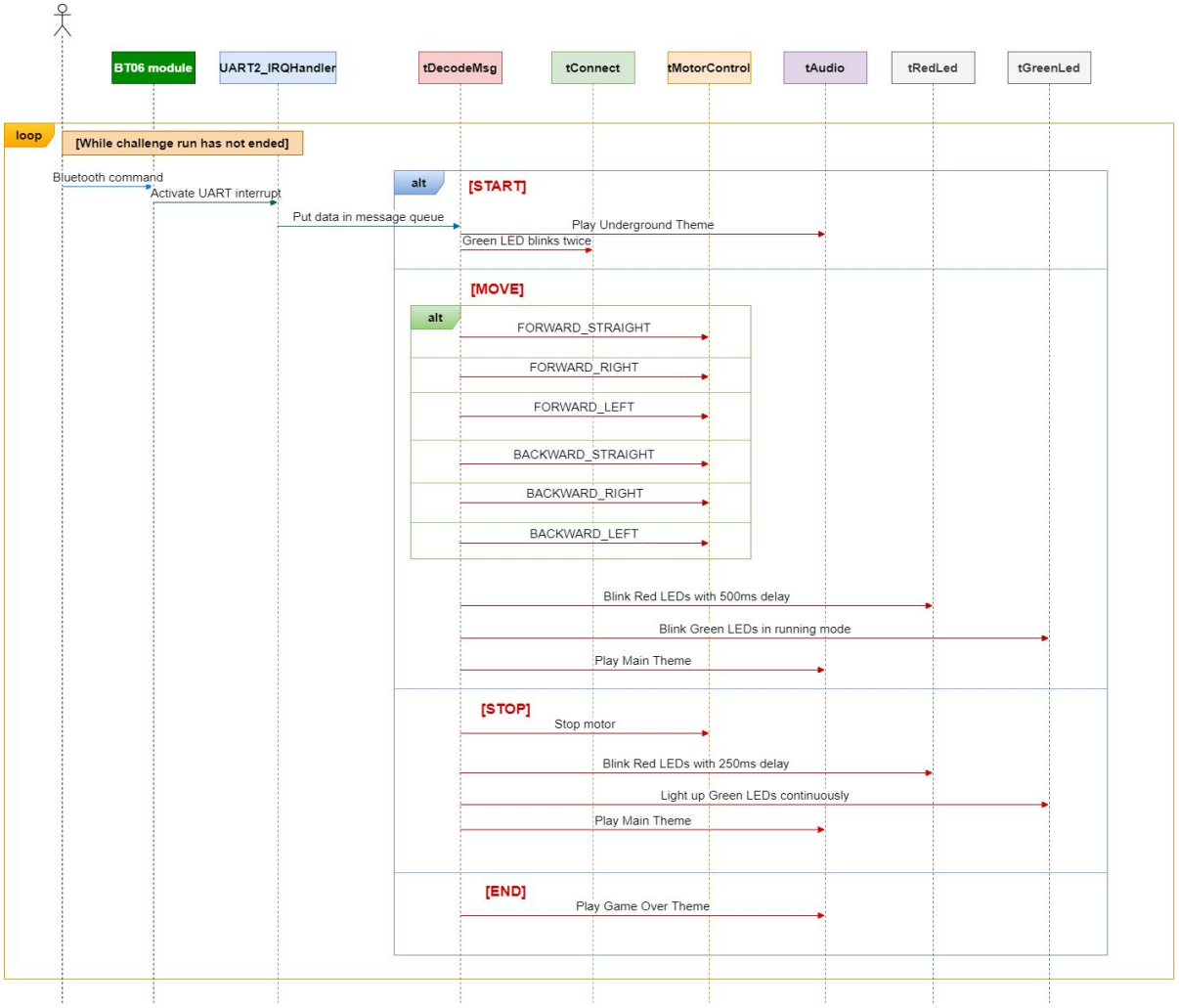

Figure 2: Flow sequence diagram