CG2271 Real-Time Operating Systems

Semester 1 2020/2021

Architecture Report

Table 1: Role Delegation

Name	Student ID	Sub-Team	Role
Walter Kong		Software	Software Engineer
Wira Azmoon Ahmed		Software / Hardware	Software / Hardware Engineer
R Ramana		Hardware / Application	Hardware Engineer / Application Developer

Introduction:

Our robot can be controlled via the android application (herein referred as *app*) and can carry out the following tasks:

1. Connect to Bluetooth
2. Flash LED Light
3. Move/Stop
4. Play/Change Audio

In the subsequent sections, we will describe the system architecture, hardware, firmware and software design. A video overview of the robot is available **here**.

System Architecture:

In this section, figure 1 is used as an overview to show how all of the components like the hardware, firmware and software work together to make sure the robot works properly.

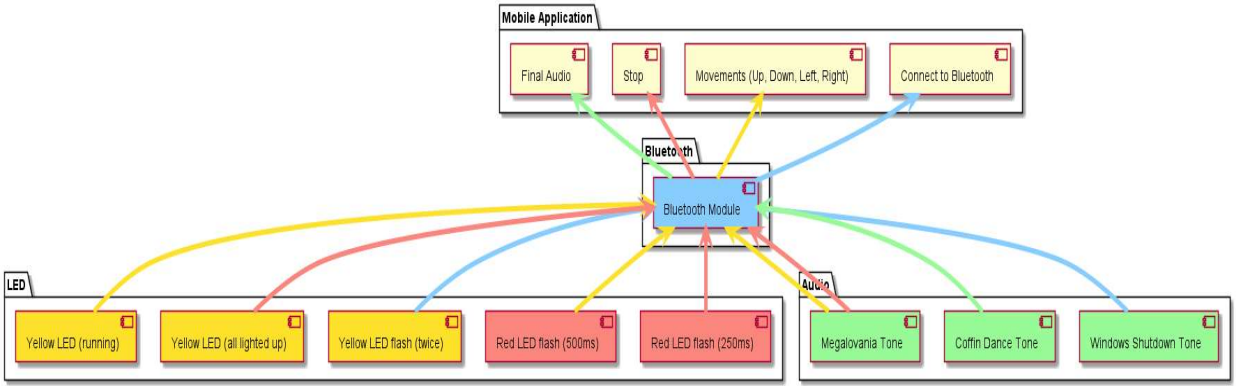


Figure 1: Architecture Overview

Referring to figure 1 above, when the user presses the '*Connect Bluetooth*' button, the app attempts to connect to the Bluetooth module (BT-06). Upon successful connection, the BT-06 sends a data value to the FRDM KL25 board using serial communication. Upon receiving this data, the yellow LED will flash twice and a unique startup tone[1] will play. Similarly, all other LED flash and tones will be dependent on the data received from the Bluetooth module (figure 2). The specifics will be elaborated on in the

```
#define STOP 83
#define UP 85
#define UP1_5 20
#define UP2 25
#define DOWN 68
#define LEFT 82
#define RIGHT 76
#define NW 45
#define NE 57
#define SE 55
#define SW 53
#define AUDIO 65
#define BT 5
```

Figure 2: bleNum values

[1] Windows shutdown tone

following sections. The developers decided to have the application merely sending data to prevent over-reliance on the application. The sequence diagrams can be found below (figures 3 and 4).

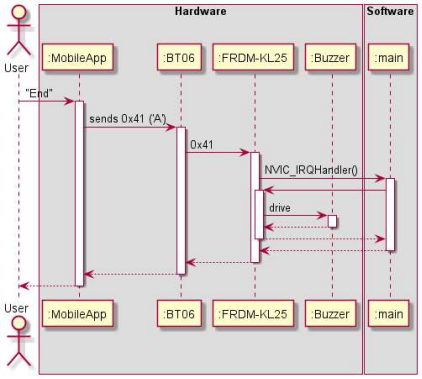


Figure 3: Hardware Sequence Diagram

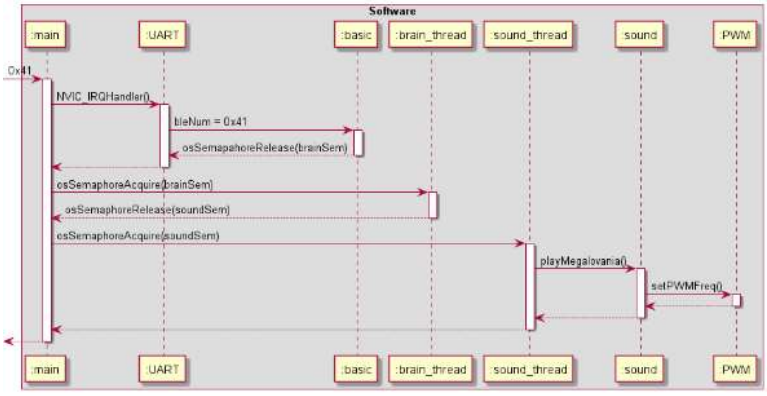


Figure 4: Software Sequence Diagram

RTOS Architecture:

There are 5 RTOS threads in use, each in a blocking state until their respective semaphores are released. Only 1 global variable *bleNum* is used. It is updated by bluetooth interrupt from BT-06 and read in these threads.

Thread	Priority	Semaphore	Task
brain_thread	Highest (Realtime)	Acquires *brainSem*, which is released only by bluetooth interrupt upon receiving new commands	Highest priority to decide whether to release *moveSem* or play the ending song according to *bleNum*
sound_thread	High	Every note acquires, plays, then releases *musicSem*, which can be acquired by the ending song until the song is over.	High priority to play music without any breaks, regardless of the user input
motor_thread	Above Normal	Acquires *moveSem*, which is released only by *brain_thread*	Above normal priority to ensure priority over *led_COLOR_thread*s but not affecting music. Moves bot according to *bleNum.*
led_green_thread	Lowest (Normal)	NA	Trigger LED at a maximum of 4hz according to *bleNum.*
led_red_thread			

With all threads are running with fixed priority, we can analyse the Worst Case Execution Tine (C), Period (P) and Deadline (D) as follows:

Thread	C	D = P
brain_thread	At most 1 ms, since threads edit timer and PWM registers which sets/resets GPIO pins to produce sound, movement, and light.	6.4ms = (48MHz / 2 / 16 / 9600)
sound_thread		155.45ms = 150ms + 1.45ms
motor_thread		100ms
led_green_thread		100ms
led_red_thread		250ms

$$Utilization(U) = \frac{1}{64} + \frac{1}{155.45} + \frac{1}{100} + \frac{1}{100} + \frac{1}{250} = 0.02206 < B(5) = 0.743 < 1$$

Thus, our threads are schedulable as they satisfy sufficient (utilization bound) and necessary conditions.

Hardware Design:

The FRDM-KL25 board receives data from the BT06 module and drives the LED and motor driver. Figure 5 shows the overall hardware architecture and how the components interact with one another, while figure 6 shows the placement of the components on the robot.

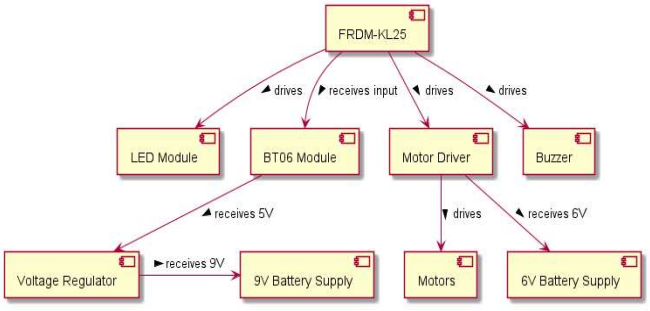
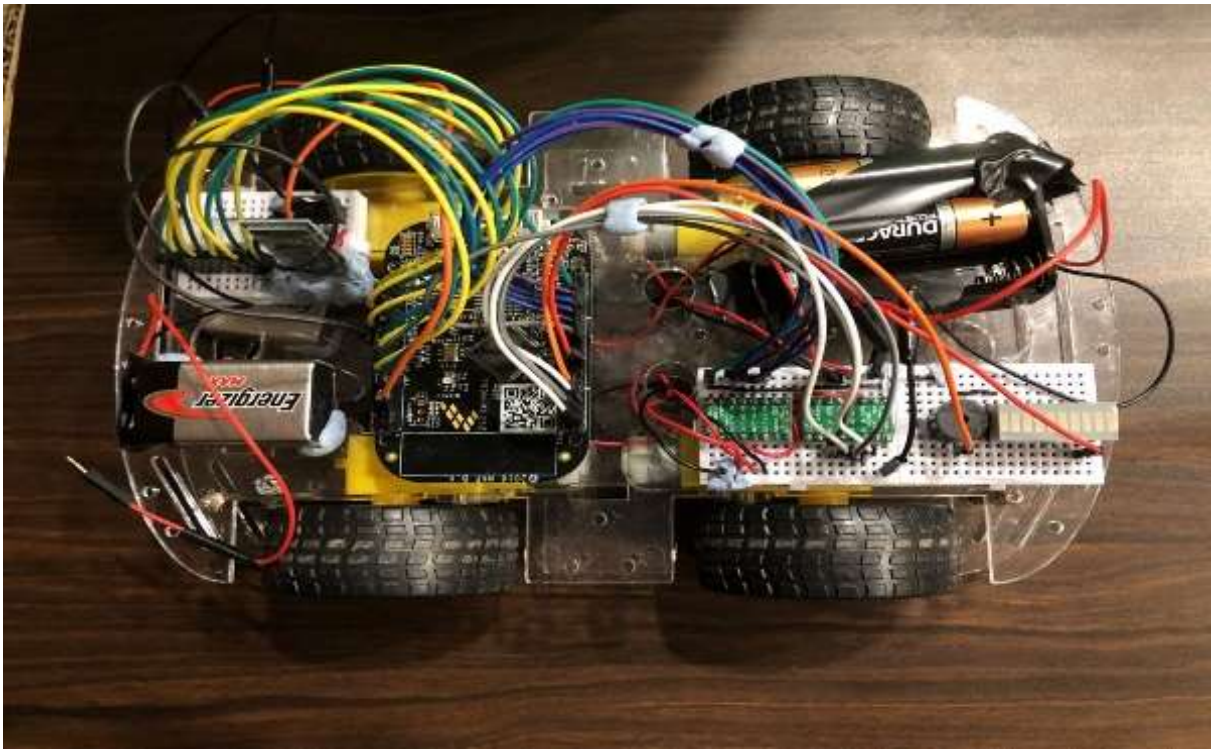


Figure 5: Overall Hardware Architecture

Figure 6: Placement of Components



LED Architecture:

LED module is run periodically by using osDelay. It calls *isMoving()* function, which checks if *bleNum* bluetooth command translates to moving or stopping. Only UP, DOWN, LEFT, RIGHT, NE, NS, SE, SW, NW indicate movement as seen in figure 2 above. If the robot is moving, the yellow LEDs will be in running mode and red LEDs will be flashing with a period of 500ms. When the robot is stationary, the yellow LEDs will all be lighted up (not flashing), while the red LEDs will be flashing with a period of 250ms.

Audio Architecture:

After the signal of a successful connection, regardless if the robot is in motion or stationary, '*Megalovania*' audio will be played. '*Coffin Dance*' will be played as the victory tone. This victory tone will be enabled if *bleNum == 65* as seen in figure 4 above.

Software Design:

To make the code cleaner and to improve readability, many of the functions were abstracted. Functions to initialize input/output, UART and PWM pins were all abstracted. Codes can be found in the '*/Header*' directory of the source code. The main code can be found in *myProject.c*. Figure 7 shows the overall software architecture. The '*basic*' file contains the delay functions and the port mask as well as the semaphores.

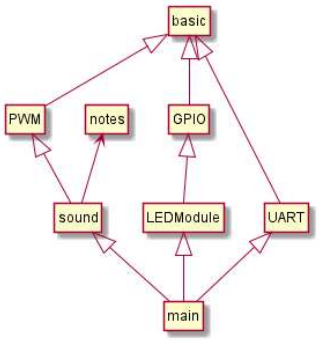


Figure 7: Overall Software Architecture

Firmware Design:

The firmware has been abstracted out and mostly use what was defined in the *system_MKL25Z4* header file. They consist predefined codes such as the NVIC for interrupts and the SIM_SCGCn to enable clocking in relevant ports.