CG2271 Report

Group: 19
Hafidz(A0184179R), Lincoln(A0184208E)

Project architecture

The project contains 6 threads, tBrain, left_tMotor, right_tMotor, red_tLed, green_tLed, tAudio. It also uses a circular queue to hold all the received data from the android application. All the threads are designed to be of the same priority. We used 'osThreadFlagsSet' and 'osThreadFlagsWait' to control both the left motor as well as the right motor. 1 interrupt handler, 'UART2_IRQHandler', to receive the data from the android app which will then be enqueued. It will be dequeued in tBrain every 250ms which will be decoded in the thread. The decoded commands will control which thread or function to activate/deactivate.

Global variables

'volatile bool stationary': to keep track of whether the robot is moving or not. It will then act as a flag for 'green_tLED' and 'red_tLED'. This controls all green LEDs either in running mode or all lighted up. This also helps to control the flashing rate of the red LEDs.

'bool connected': act as a 1-time flag to check if the android app is connected to the robot. The android app will send a byte number once it is connected to the Bluetooth module. The 'UART2_IRQHandler' will get this data and run an if statement to the connected flag. It will then control the green led to blink as well as play a unique audio tone once.

Threads

Thread tBrain:
Controls the functions of other threads via a switch case. Dequeue the data that was captured from 'UART2_IRQHandler' into the variable rx_data which is then decoded in this thread. We use mask values to decode which component to control [COMPONENT_MASK(x) (x & 0x7C)].

Eg. masking the rx_data to decode

Rx_data = 0d10, 0b0000 1010	Mask with 0x7C, 0b0111 1100	Masked value = 0d08, 0b0000 1000, (control right motor)
Rx_data = 0d17, 0b0001 0001	Mask with 0x7C, 0b0111 1100	Masked value = 0d16, 0b0001 0000, (control left motor)
Rx_data = 0d64, 0b0100 0000	Mask with 0x7C, 0b0111 1100	Masked value = 0d64, 0b0100 0000, (reverse motor)

Case 0:
If there is no longer any command from the android application, it means the queue is filled with 0. Thus, the robot will be stationary.

- Set statioaney to be true.

Case 4:

A command from the android application which sets both the osThreadFlagsSet() for left and right motors
Case 8 and case 16:
Set the flag for left motor and right motor if the masked value is 8 or 16, and set the global variable stationary to false.

- osThreadFlagsSet(right_motor_flag, 0x0001), set the flag for right_tMotor with the value of the flag 0x0001 (masked value of 16)
- osThreadFlagsSet(left_motor_flag, 0x0001), set the flag for left_tMotor with the value of the flag 0x0001 (masked value of 8)

Case 32:
Enables the PWM for tAudio

- If the masked value of bit 0 is 1, activate the TPM0 else deactivate

Case 64:
Reverse control for both motors

- Enable both TPM1 and TPM2
- Give 0% (value of 0) duty cycle to CH0 of TPM1 and TPM2
- Give 20% (value of 1500) duty cycle to CH1 of TPM1 and TPM2,
- This will result in the motor going in the opposite direction
- Set stationary to false

Default case:
Stop the motor

- Disable the PWM (TPM1 and TPM2)
- Set stationary to true

Thread left_tMotor:
This thread waits for the flag to be set from tBrain. Once the flag is set, it will continue to decode the rx_data. However, it will mask bit 0 and bit 1 (BIT00_MASK(x) (x & 0x03)). For this thread, the app will send a value of 10 (1010) and 9 (1001). Therefore, masking 10 would result in a value of 2, masking 9 will result in a value of 1. In the case statements, the value C0V and C1V control the duty cycle (voltage) supplied to the motor as well as the polarity.
Case 1:

- Set 100% duty cycle to TPM1 CH0
- Set 0% duty cycle to TPM1 CH1
- Set stationary to false

Case 2:

- Set 20% duty cycle to TPM1 CH0
- Set 0% duty cycle to TPM1 CH1
- Set stationary to false

Thread right_tMotor
This thread waits for the flag to be set from tBrain. Once the flag is set, it will continue to decode the rx_data. However, it will mask bit 0 and bit 1 (BIT00_MASK(x) (x & 0x03)). For this thread, the app will send a value of 17 (1 0001) and 18 (1 0010). Therefore, masking 18 would result in a value of 2, masking 17 will result in a value of 1. In the case statements, the value C0V and C1V control the duty cycle (voltage) supplied to the motor as well as the polarity.
Case 1:

- Set 100% duty cycle to TPM2 CH0
- Set 0% duty cycle to TPM2 CH1
- Set stationary to false

Case 2:

- Set 20% duty cycle to TPM2 CH0
- Set 0% duty cycle to TPM2 CH1
- Set stationary to false

Thread red_tLed

If the stationary flag is true, it will blink at a rate of 500ms. If the robot is moving, it will blink at a rate of 250ms. We used ‘osDelay’ to control the blinking rate, this allows other threads to run by blocking itself when not required.

Thread green_tLed

Has to wait for the connected flag to be true to run the LEDs. Has an array which controls PortC pins 11, 10, 6, 5, 4, 3, 0, 7. If the robot is moving it will be in running mode which sets each pin to high individually. If the robot is stationary, it will set all the pins to high (0b110011111001) which turns all the green LEDs on. We chose these pins because they line up perfectly at a corner on the FRMD board.

Thread tAudio

Uses ‘TPM0 CH0’ to control the frequency. Plays the values that are stored in a header file called ‘song.h’. In ‘song.h’, we can adjust the frequency and duration of the notes individually. This is because every note has a corresponding frequency, and the timing of notes, such as eighths, quarters, are also defined here.

Interrupts

UART2_IRQHandler, receive the data from the android app and enqueue the data into a queue. When the android application first connects to the BT module, the application will send a byte number, 200 (this number does not affect the threads). If the global boolean variable, connected, is false, it will set variable to true and blink the green LED once as well as play a unique tone.

Priorities

All the threads have the same (normal) priorities. We used the default (normal) priority with round-robin enabled, as this will give every thread a chance to run. The two motor threads will only run when their corresponding flag is set. For the audio and two LED threads, we used ‘osDelay’ mainly for timing purposes. It also blocks itself when it is no longer in use. This helps to speed up the thread switching process, making others more responsive.

Component connection