dona1865-443-p6

Ryan Donhaue

October 31, 2018

1 Introduction

For this project, I designed a DC motor control application using a Cerebot MX7cK and FreeRTOS. The main learning objective was to implement a controller area network (CAN) communication protocol between a control unit and IO unit. Figure 1 shows the control unit functionality, which controls CAN 1. Figure 2 shows the IO functionality, which controls CAN 2.

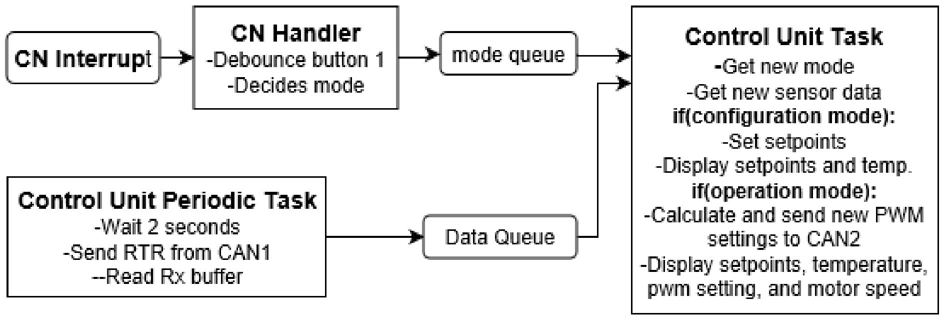

Figure 1: Control Unit

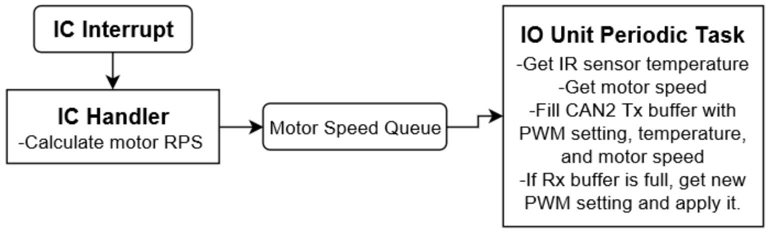
IC Interrupt
IC Handler
-Calculate motor RPS
Motor Speed Queue
IO Unit Periodic Task
-Get IR sensor temperature
-Get motor speed
-Fill CAN2 Tx buffer with PWM setting, temperature, and motor speed
-If Rx buffer is full, get new PWM setting and apply it.

Figure 2: IO Unit

2 What Works

I implemented the following major systems:

- FreeRTOS, for system timing, and control.
- SMBus and IR sensor, for temperature measurement readings.
- CAN, for communication between the two units.
- Output Compare, for PWM output to control motor speed.
- Input Capture, for motor speed calculations.
- LCD, for displaying system status information.

Each system works as specified in the project 6 handout. Figure 3, on the following page, shows the system's CAN transaction. LEDA toggling indicates a request for transmission (RTR) frame from the Control unit has been sent. LEDB toggles when the Control unit receives a sensor message from the IO unit. LEDC indicates the Control Unit sending a PWM message, and LEDD toggles when the IO unit applies the the received PWM setting.

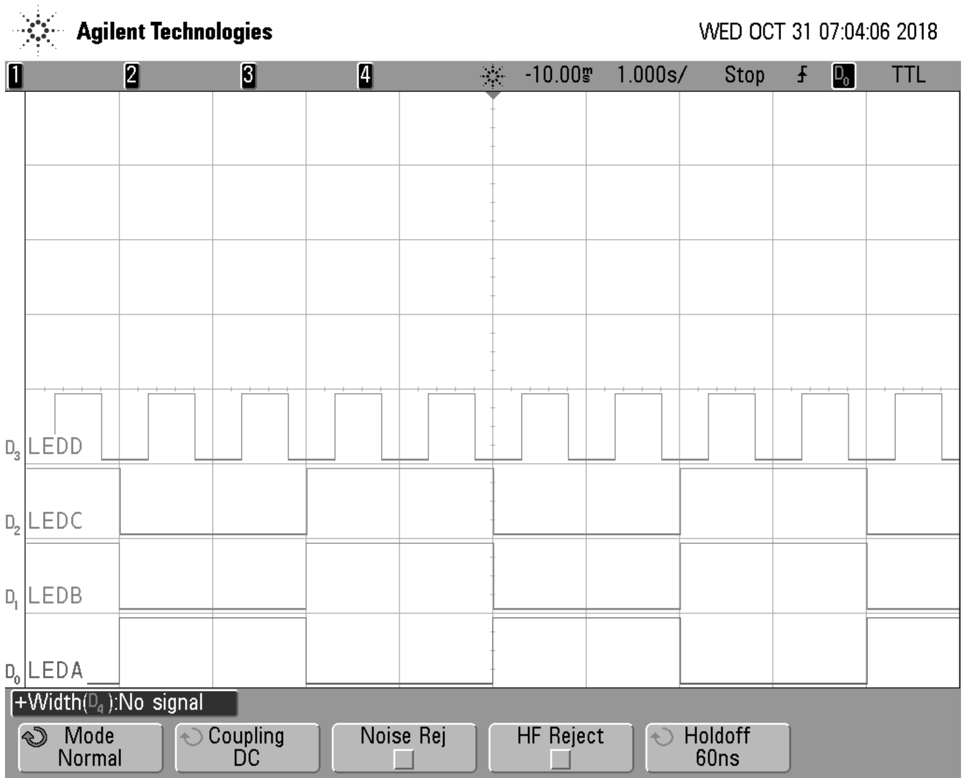

Figure 3: LED Behavior

3 What Doesn't Work

I was able to make every peripheral in the system function. However, due to timing specifications (2 seconds between RTRs) and FIFO buffer latency I had several extra seconds between changes in motor speed and observed RPS change.

I expected round-trip latency to be 4 seconds (two RTRs). After a PWM setting is received and applied, RPS will change and recalculate. The following RTR should return the new motor speed. In my system, this takes additional cycles. I believe this is due to how I attempted to find the most recent message in a Rx buffer. As we discussed after lecture on Wednesday, I looped through the channel's messages, updating the pointer after retrieving each data payload until a null was returned. Supposedly this should have left me with the most recent data sent into the FIFO. However, I debugged and watched the transaction and found the data available was two or three buffers old. I tried several similar strategies (reseting the FIFO, using one buffer, writing over the same message in a Tx buffer) and was unable to reliably retrieve the most recent IO unit data.

4 RTOS Task Partitioning

I did a great job of minimizing global variables by using RTOS queues, tasks, and semaphores. If I executed my code on two devices, I am very confident that the functionality accomplished on the single board would still work.

Control Unit

The control unit had three tasks. In priority order these were:

1. CNHandlerTask
2. ControlUnitPeriodicTask
3. ControlTask

I gave the highest priority to the change notice handler. This was only logical, since operation mode should be changed before running the control task. Between button presses and delays, all tasks yield. Every two seconds the periodic task sends a RTR, a message is received and processed and sent to a queue. This queue unblocks the control task, creating no chance of priority issues.

IO Unit

The IO unit had two tasks. In priority order these were:

1. IC5HandlerTask
2. IOUnitPeriodicTask

The IO unit was responsible for the precise measuring of motor RPS. For this reason, the input compare handler task was set to higher priority than the periodic task.

5 Conclusion

Fun project. I had a heck of a time using CAN at first but Dr. Wall's examples were very helpful and I soon got over the learning curve. Luckily, I was shown the help menu in MPLAB. I can't believe I have never opened this menu! It has all the relevant syntax, explanations, and more for each PLIB function.