Using a Controller Area Network (CAN) API

1 Introduction

This lab is concerned with using a CAN API to develop both event-triggered (interrupt-driven) and time-triggered (polling) solutions for the reception and transmission of CAN messages.

2 In the lab

1. Clone the `git` repository for the lab:

   ```
   $ git clone http://github.com/DavidKendall/cm0605_lab09
   ```

   Start up EWARM and load the workspace `cm0605_lab09/workspace.eww`. The workspace contains two projects: `ET` and `TT`. `ET` is an event-triggered (interrupt-driven) program. `TT` is a time-triggered program. Connect a `LPC-2378-STK` board to a USB port on your computer. Connect CAN port 1 to CAN port 2 on the `LPC-2378-STK` board, using a straight-through CAT-5/6 twisted-pair cable. Download and debug the project `ET`. Run it and observe its behaviour. This program transmits messages on CAN port 1 and receives them on CAN port 2. The data sent is a simple TX count that is displayed when the message is received. Make sure that you can see a count incrementing rapidly on the LCD. Now do the same for the `TT` project. Once you are happy that the software is working for both projects, you should carry on with the exercises below.

2. Work with a partner and connect two `LPC-2378-STK` boards together via their CAN ports. Connect CAN port 1 on board 1 to CAN port 1 on board 2. Use the CAN bus on the production cell. Ensure that the jumpers are in place so that the bus is properly terminated.

3. Modify both projects so that they both send and receive messages on the same port (port 1). You'll need to study the header file `can.h` to see what is expected by the API. N.B. Both projects already send on port 1 so all you need to change is how messages are received.

4. Download and debug the `TT` project to board 1 and the `ET` project to board 2. Run the programs. You should see the counts incrementing on the LCDs of both boards.

5. Currently, the task that receives messages in the `ET` program runs at 10 Hz and checks a boolean variable `can2RxDone` to see if a message has been received. The `can2RxDone` variable is set in the interrupt handler `canHandler`. It would be better if the receiving task were controlled by a semaphore. It should pend on the semaphore at the top of its loop. The semaphore should be posted by `canhandler` when a message arrives.

   Modify the `ET` program so that message reception is handled in this way. Download and debug your program. Run it to check that it is behaving as you think it should.

6. Modify the `TT` program so that it only sends a message when `BUT_1` is pressed. It should still display messages received from the `ET` program. Modify the `ET` program so that it only sends a message when it receives a message from the `TT` program.

   Download and debug your programs. Run them to check that they are behaving as you think they should.

7. Using the interval timer functions that you worked with in `lab07`, measure the time that it takes from the sending of a message by the `TT` program to the receiving of a response at the `TT` program. Display the time taken on the LCD.

   Can you deduce anything about the transmission rate of the CAN network from this result? What factors do you need to consider?

8. Modify the data rate of the CAN bus so that it runs at maximum speed (1 Mbps). Repeat the measurement of the message response time. Is it clear that the response time has been affected by the change in the data rate?

Remember if there's anything you don't understand in the labs, please ask your tutor for help.

Git documentation can be found at http://git-scm.com/doc.