CAN Lab 2

2016-10-03

Aim

Advanced CAN communication, detailed experience with the MCP2515 external CAN controller. Introduction to distributed systems programming.

Literature

Document "Tutorial - getting started with CAN", data sheet for MCP2515, can.h, can.c

Brief summary

This exercise follows directly upon CAN Lab 1 and extends the tasks to include CAN message receiving and distributed systems.

We will simulate a distributed temperature monitoring system suitable for a process industry or a process plant, with a dynamically configured set of distributed temperature measuring stations.

Reporting

After completion of the lab, demonstrate all programs for the teaching assistants. The exercises should also be submitted to the teaching team via upload on KTH Social – as instructed later in this document.

1. Introduction and overview

This exercise is based on the following setup and philosophy:

1) Each student group measures temperature and light via the temperature and light sensors mounted on the EVK1100 board.
2) One fixed reference station placed underneath the Scania dashboard does the same measurement and provides reference values.
3) Each student group broadcasts the measured values on the RED CAN bus with a specified ID and format.
4) All student groups therefore have access to all measured values in the room.
5) Depending on the variations of the measured values in the room, various actions are taken on each station.

2. Read CAN messages

First, you need to verify that the reference station is correctly configured and sends values of temperature and light.

The reference station is connected to the RED bus, with the standard bus settings: **250kbs, 75%, SJW2.**

The reference station also broadcasts the value of its potentiometer, for your convenience.

The reference station broadcasts temperature and light with the following configuration:

ID = 0x00001000
Byte 0 = Temperature, 2 most significant bits
Byte 1 = Temperature, 8 least significant bits
Byte 2 = Light, 2 most significant bits
Byte 3 = Light, 8 least significant bits
Byte 4 = Value of potentiometer, varying from 0 to 255.

Connect the MCP2515 to the EVK1100.

Task

a. Verify that the above is correct with the CAN dongle and CANking
b. Estimate the frequency of the received reference messages
c. Write a program that reads the values of the reference temperature, light and potentiometer value and presents this on the display, both as absolute values and also as percentage of max (255).

Report
Submit your c-file(s) on KTH Social after completion of the entire lab.

3. Distributed temperature control, part 1

Now, you need to measure temperature and light on your own station. Use the exact same format as the reference station. You may also broadcast the potentiometer value for debugging.

In this exercise, you should keep using the YELLOW bus with the same settings as above.

Your student team should be assigned a team number (from 1 to 15). This number should be marked on your computer desk.

Now, your station shall broadcast your measurements with an ID that is unique to your station. Use the following:

ID = 0x0000120z where z is the number of your station (1-15). Broadcast with the same format as the reference station.

Your measurements should be broadcasted at least once every second (1 Hz).

In the following tasks, you may include the values from the reference station or not.

Task

a. Write a program that, besides broadcasting your measurements, reads the measured values from all other teams currently working in the lab. Here you should do a filter, so that it only reads the IDs stated above. (The filter should be implemented on the CAN-hardware)
b. On the display, you should present the following details:
    a. The current number of online teams in the lab
    b. Your measured local temperature
    c. The average of all temperature measurements
    d. Your measured local light value
    e. The average of all light measurements

Report
Submit your c-file(s) on KTH Social after completion of the entire lab.

4. Distributed temperature control, part 2

While continuously broadcasting your measured values, you should now program your local station to perform a number of tasks depending on the status at the various stations in the lab.

Definitions and assumptions

1. We now pretend that all stations are spread in different rooms/buildings/countries

2. A light that is *well below average* defines that it's night where the station is located.

3. A temperature measurement that exceeds 15% *above average* means that this station is *too warm.* Average should be in the area of 20 degrees Celsius, and *too warm* can be around 23 degrees and more, which can be achieved by placing a thumb on the sensor.

4. A temperature measurement that is *very much below average* means that there is something wrong with the station/sensor/station. Defined as *faulty station.* This can be achieved in the lab by use of a tube of cooling spray, provided by the assistants.

5. LEDs and push buttons should be used to simulate inputs and outputs.

6. Relevant information to the user should be shown on the display

Specification of demands

The following sets of specification should be realized on your node.

1. At all times, the display should present the number of nodes that's currently online on the bus and broadcasting data.

2. At all times, the display should present the number of nodes that are in daylight respective not daylight (night).

3. At all times, the average value of all temperature and light measurements should be presented on the display.

4. If no node is *too warm* or is *faulty*, this represents a *normal status* and should be indicated with a constant green LED on the node.

5. If one (and exactly one) node is *too warm*, your node should move into a *warning state*. This should be indicated with information on the display and a flashing red LED.

6. When the one too warm node returns to normal temperature, the system should return to the *normal status* mode automatically.

7. If two nodes are *too warm*, the system (your node) should move into an *emergency mode*. This should also be indicated on the display and with two flashing red LEDs.

8. The node should be kept in the *emergency* state until two conditions apply; first no more than one node can be too warm, and, secondly, a user needs to acknowledge the emergency alarm by pressing a push button on the node.

9. If your station receives a faulty measurement, this should be indicated on the node with a flashing LED (that clearly distinguishes this state from the above states) and the value should be disregarded from in the calculations of the average value.

10. Your node should not pay any difference if it's your own node that's too warm or another node.

Report
Submit your c-file(s) on KTH Social after completion of the entire lab.

5. Interrupt-driven CAN

So far, you have most likely regularly polled for any newly received CAN messages to act upon. With such a setup, the processor needs to perform unnecessary processing when no messages are being received. Moreover, a received message may unnecessarily wait (too long) in case your program is performing additional tasks between polls.

Modify your program such that an interrupt is triggered on the microcontroller once a CAN message is received on the PICtail. The interrupt handler should in turn process the message accordingly.

Tips

1. Make a physical connection between the INT-pin on the PICtail and one of the digital input pins on the EVK1100. It's easiest to use either the PA29 or PA30. Both of these are also used for the TWI interface (labeled SDA or SCL in the TWI interface), and therefore a connector can easily be connected same way as the PICtail is connected to the EVK1100.

2. Check the AVR32 software framework for how to enable interrupt on individual pins (such as PA29 or PA30). You can and should simulate the behavior first by detecting interrupt from the push buttons.

Report
Demonstrate your programs to the teaching assistants and submit your c-file(s) on KTH Social.