A Software Package for Enhanced Student-Learning in Real-Time Embedded-Systems Networking

Abstract

Today, embedded systems networking is common in manufacturing, automobiles, medical equipment, and home appliances, but few undergraduate engineering and technology curricula teach courses devoted to it. Not having appropriate educational experiences risks a decline in US technical expertise. Various companies have developed commercial software packages for analysis, diagnosis and simulation of real-time embedded-systems networking protocols. However, these commercial software packages are very expensive and complex to be used for undergraduate courses. We have developed a user-friendly and easy-to-use software package for the PIC microcontroller to study the Controller Area Network (CAN) protocol and its use for various real-time applications. The software can be used in an undergraduate Electrical and Computer Engineering design class. The use of the software allows the students to understand the CAN protocol. The students can also modify different modules of the software based on the hardware setup of their real-time systems, and then do various types of experiments with their hardware setup. The use of our software enhances student-learning of embedded-systems networking. The paper presents a detailed description of our software architecture and its use for various real-time applications.

Introduction

We are living in a world today in which rapid technological change is occurring faster than even our universities can keep up with. If we look back 50 years ago, we see an automobile industry in which almost all systems are mechanically based. Fast-forward to today, and industry projections are showing that by 2010, 40% of the total cost of a car will be dedicated to electronics and distributed electronic control systems. During this time span, individual manufacturers have developed their own processes and their own proprietary electronic architectures and protocols. Many of these standards did not make it out of their company's doors, but we finally have a situation where certain protocols are used globally. This global industry standardization did not come about until around 10 years ago. After a decade of preferred usage across Europe, Bosch's CAN protocol[1] finally won widespread acceptance in the US auto industry during the late 1990s. Worldwide usage brings certain advantages with it. Standardization of components drives down manufacturing costs; it also reduces maintenance costs when replacements are easy to obtain. In addition, auto mechanics and repair personnel only have to learn one electronics communication protocol to diagnose and repair faulty systems. In this entire situation, the only weak link is the educational system.

Many schools around the country teach a course in embedded design, but very few[2, 3, 4] focus on the networking of embedded systems. With the globalization of our workplace, and jobs moving to other countries, it is important to train US workers on these new technologies to remain competitive with the rest of the world.

The main issue with teaching a course in embedded systems networking is that new software and hardware tools are necessary. Vector Group[5] and Dearborn Group[6] do produce CAN tools.

However, these tools are distinctly designed for industrial usage. Although very sophisticated, they are very complex, expensive, and time consuming to learn and use. For the purposes of a one-semester undergraduate course, these tools are cost-prohibitive, and are complex enough that they would consume far too much class time in just learning how to use them. We need a software and hardware environment that is simpler and more user-friendly while still displaying the functionality of CAN to the student.

In this paper, we present the architecture of our software package that meets these goals, along with its usage in real-time applications.

Background

Today, CAN is the standard for mission critical real-time electronic control networks. Developed in the 1980s by Robert Bosch GmbH, with first silicon from Intel Corporation, CAN was created to meet a specific need in the new fleet of automobiles on the road. Microprocessors started to become economical enough and small enough to be implemented in cars. Fuel was now being electronically injected, sensor data was being collected for temperatures, fluid levels, and emissions, and we still had all of the wiring from power windows, doors, and other systems. In this scenario, just as shown in Figure 1, the Electronic Control Unit (ECU) in the car would need to be directly connected to all devices it would interface with. This would include analog or digital connections to every sensor it would be processing. With dedicated links between every processing control center and their external devices, massive amounts of wiring were needed in addition to ever growing wiring harness sizes to accommodate these wires. This not only increases the cost of the vehicle, but adds a noticeable amount of weight to the vehicle.

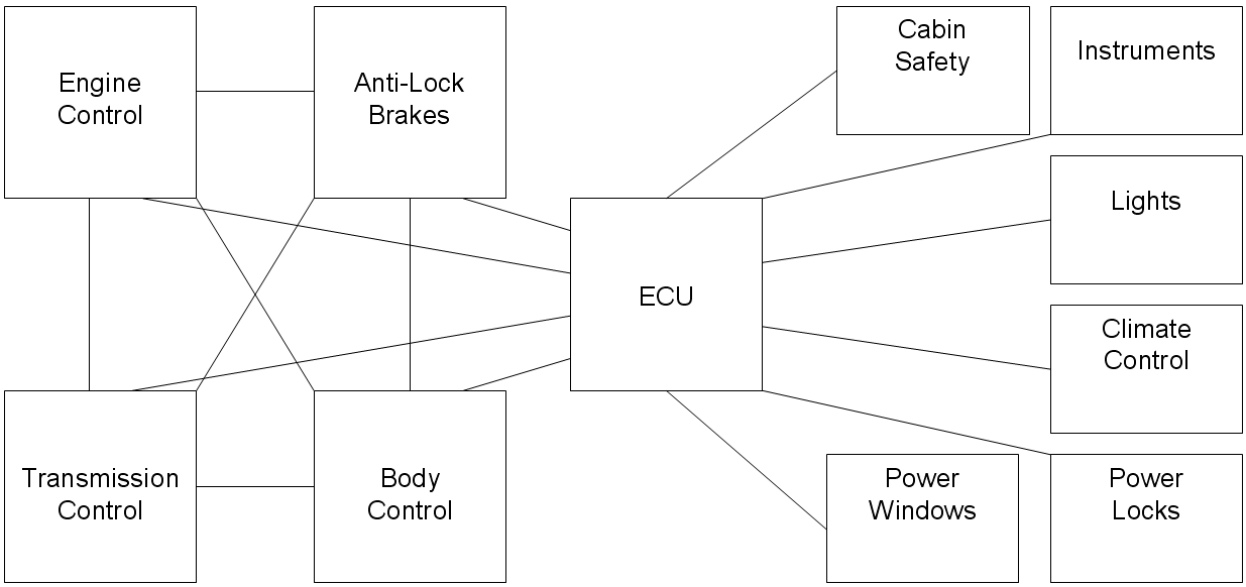


Figure 1: Simplified control network for average automobile without CAN.

CAN eschews this structure and replaces it with a serial communication bus between all devices in the vehicle. For the cost of adding a CAN processing chip on each node in the network we are gaining many advantages. We now have to send only one set of wires (most likely two)

throughout the vehicle and every device can tap into the same bus. As can be seen in Figure 2, instead of needing numerous analog and digital ports on the ECU, we can now reduce it down to one CAN connection. This severely reduces the number of wires in the car, and the number of inputs needed on the ECU, but it can also increase the intelligence of each node on the network. Every node is connected to the same bus, and therefore, every node sees all of messages sent on the bus. If multiple nodes need a specific sensor's data, they can pull that data out simultaneously from the same sensor when the data travels over the bus. This is a large improvement versus having to make multiple requests to the ECU for the sensor data, or having to build in local redundant sensors for each node if the ECU cannot handle the traffic. Another great feature is expandability. If the vehicle manufacturer wanted to add a new system to the car, the electronics would have to be placed, the wiring laid out, the harnesses changed to accept the new wiring, and the ECU changed to one that had enough ports. In a CAN network, they would only need to add the electronics and tap into the CAN bus. All other changes would be made to the software firmware of the ECU so that the unit knew the new system's CAN messages and what to do with them.

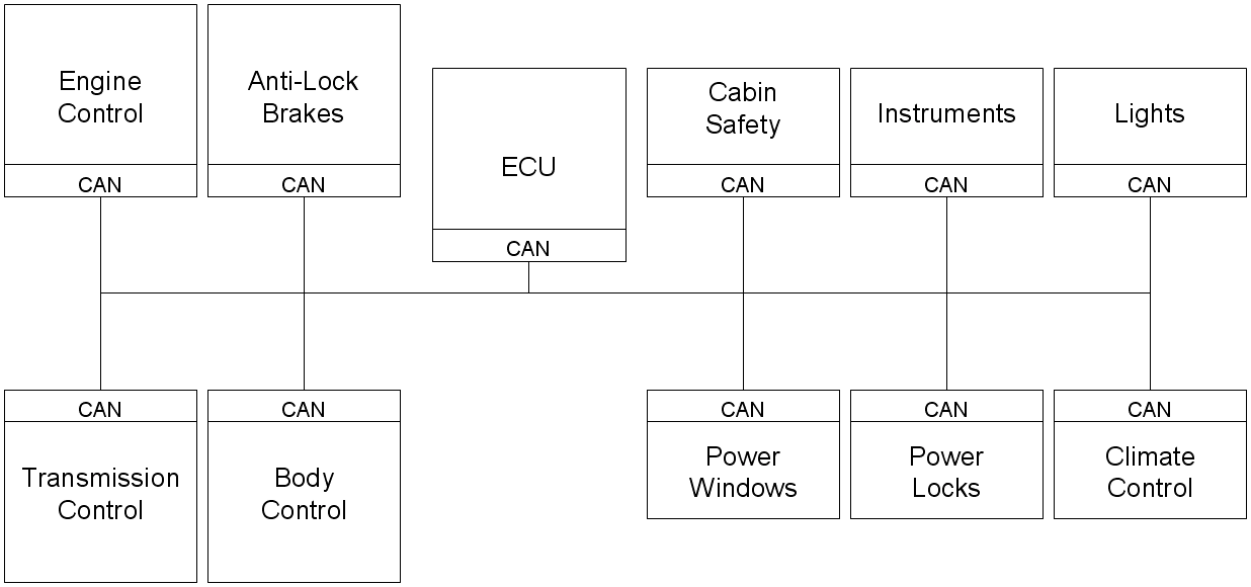


Figure 2: Simplified control network for average automobile using CAN.

From a protocol level, CAN is already a mature standard[7, 8]. CAN became an ISO standard in the early 1990s, and has seen 20 years worth of testing and applications so far. The protocol was implemented in silicon very soon after introduction, so the CAN node controllers in use can operate at very high speed, and low power consumption with the error handling and fault confinement capabilities handled automatically. The message format and communication methods are also designed with error detection in mind. The robust format allows easy detection of invalid message strings, and quick and easy notification of the transmitter that there was a problem. The CAN protocol also has very good handling of faulty nodes. Instead of a faulty node being able to consume all bandwidth on the network and block all transmissions, the CAN protocol has features built in that allow the network to disable that faulty node. Since we are not 100% reliant on a single node in the network, any transactions that do not involve the faulty node can continue as if nothing bad happened.

Even though it was originally applied to automobiles, the dependability and advantages of CAN have attracted other industries to utilize the protocol. The textiles industry was the first to make a major move toward redesigning all of their machines to utilize CAN networks. CAN based weaving, knitting, and sewing machines are in widespread use today. CAN networks can also be found in train systems, airplanes, hospital room controls, and even large and small home appliances[9]. Learning this protocol for embedded networking will help prepare students for a large number of possible industries.

On the development side of CAN, Vector Group and Dearborn Group both make tools to aid in design and production. Vector's CANoe[10] is a software package that allows you to design, simulate, analyze, and test distributed systems with ECUs. Dearborn Group[6] has a few software and hardware options that can allow you to simulate and analyze many different automotive networks. Although they are well respected and capable software packages used in industry, for the purposes of teaching an undergraduate class in embedded systems networking, they are too costly and too complex. With only one semester in which to teach students about this topic, the time penalty for learning how to use these software packages is too great. That time could be better utilized covering more topics in this large field.

Architecture

Everything with the software and hardware system is designed to make the setup and usage of the system as easy as possible to allow the student to focus on learning the characteristics of the CAN protocol that are being displayed at that time. The hardware that is being utilized with the software package is based on the Microchip PIC microcontroller. Individual hardware boards, one of which is shown in Figure 3, are set up with two CAN nodes on each and an external bus connector to tie boards together. Headers are set up on the board in order to facilitate an easy

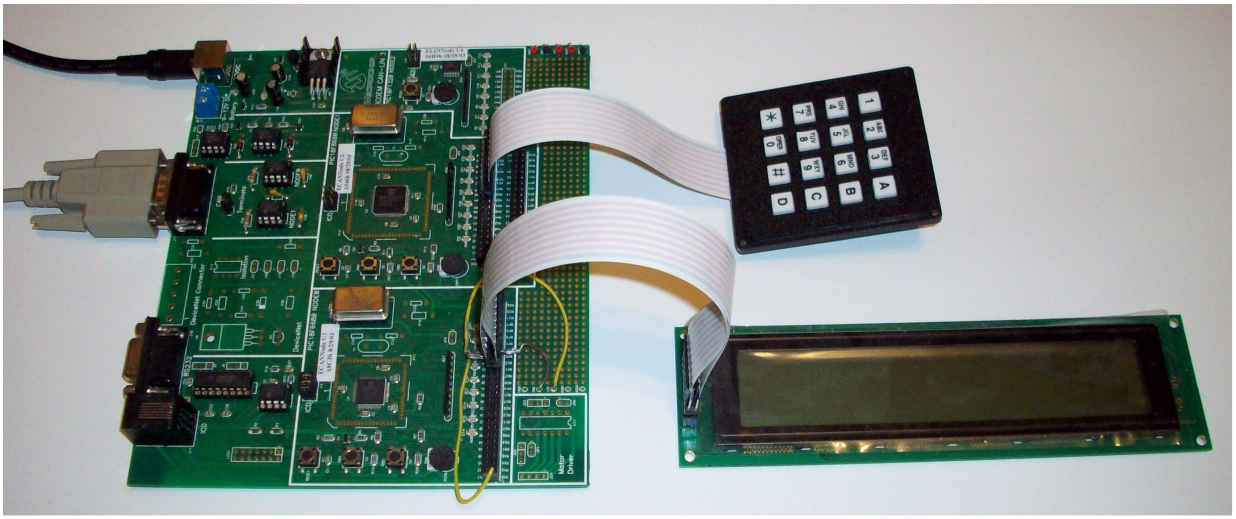


Figure 3: Single CAN board with two nodes – Connected to power, external CAN bus, keypad, and LCD.

plug-and-play experience for the student. Instead of soldering components on, LCD displays, analog and digital sensors, and various actuators can be plugged into their respective positions and are ready to use.

The software package is broken up into segments for lab experiments, all written in PIC assembly. Each experiment builds upon the previous to add more functionality and expose new features. As shown in Figure 4, the functions for each action are organized into modules or just written straight in the main source code based on their utility. Hardware devices such as the LCD unit have their own separate libraries with individual function calls to utilize the components. Devices that are used less frequently in the course are integrated directly into the main source code. However, their code is sectioned off in the source and commented for easy identification and understanding if the student wishes to modify the code environment for testing purposes.

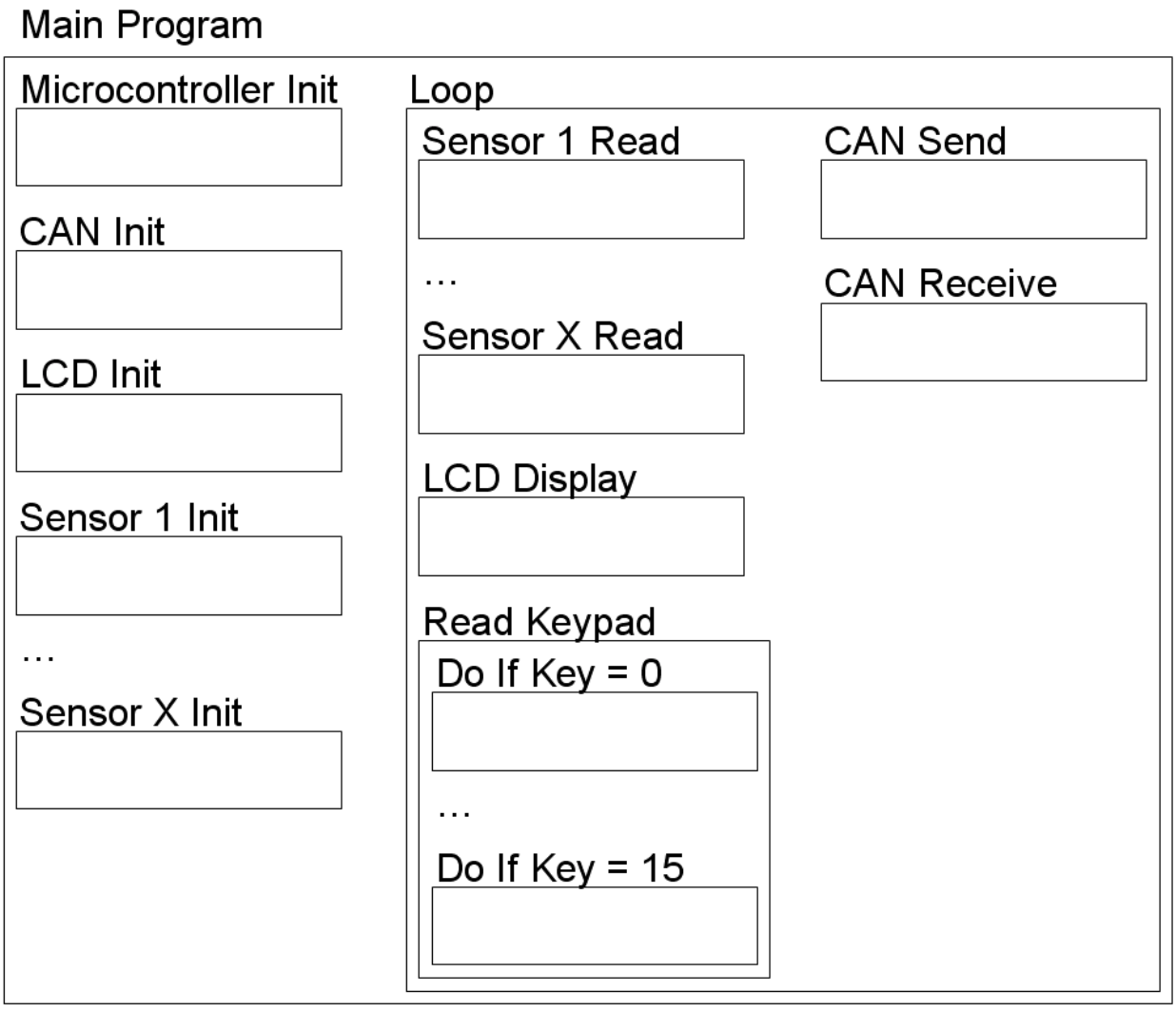


Figure 4: Example of module structure of code base.

Learning Outcomes of the Laboratory

After completing this course, students should be able to do the following:

1. Explain various features of the CAN-LIN 3 board.
2. Illustrate basic concepts of a real-time system.
3. Explain working principles of various sensors and actuators.
4. Install sensors and actuators in a real-time system.
5. Write codes to read data from sensors.
6. Write codes to operate and control actuators.
7. Design distributed real-time system using microcontrollers, sensors and actuators.
8. Write codes to send data using a CAN bus.

Brief Descriptions of Labs

So far we have developed five laboratory experiments. Brief descriptions of these five labs are presented below. Detailed step by step activities of Lab1 is presented in the Appendix. Detailed step by step activities for other labs are similar to that of Lab1. All these five labs together will help the students to accomplish the above learning objectives.

Lab1: Introduction to Hardware, Software and Basics of CAN protocol.

The goals of Lab1 are: 1) to become familiar with the usage of CAN-LIN 3 Development Board while touching on some of the basics of CAN communication, 2) to become familiar with the hardware requirements for setting up the board for this and future CAN labs, 3) to become familiar with the procedures for programming the multiple microcontrollers on each board, 4) to become familiar with the usage of MPLAB software, 5) to become familiar with the software running on the boards and make modifications to it for different system behavior, and 6) to be able to create different CAN messages for different nodes and send them through the CAN bus.

Lab1 allows the student to build the most important device of all of the labs, a CAN node that snoops the bus. In addition to building a device that will be able to show the student what messages are passing across the CAN bus, this lab is also introducing them to using external hardware devices, in this case, an LCD. As an example of all of the hardware devices, we have written software libraries to interface with the LCD. The student will only have to call functions such as “LCDInit” to initialize the LCD, or “LCDWrite” to write the data in the working register to the screen. Functions are even provided to output data in binary or hexadecimal formats. As with every device, all of the source code is readily available, readable, and commented. If the student wants to experiment on their own with devices, or wants to try something slightly different, they can easily edit the code directly. Lab1 deals with the topic of bus arbitration. The student will utilize two CAN boards with four nodes. The two boards are connected by a CAN bus as shown in Figure 5. Each board has one node with a keypad and one node with a display.

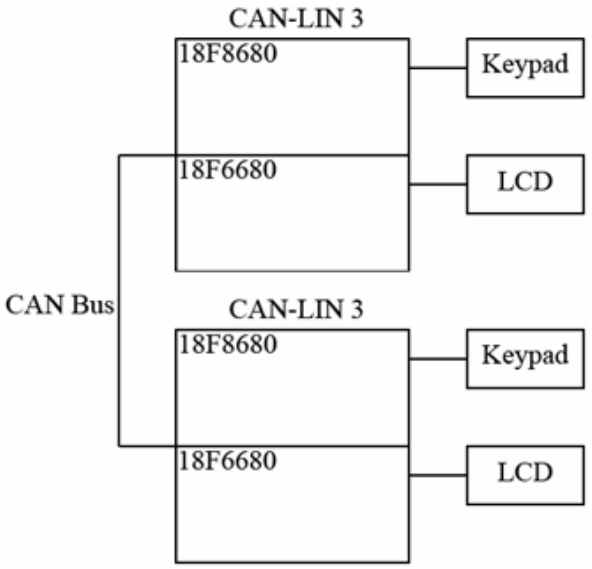


Figure 5: A simple CAN system with two CAN-LIN 3 boards containing four CAN nodes.

Each keypad press on the nodes will cause that node to send a CAN message with a specific ID. By default, the IDs are based off of the numbers on the keypad, but one CAN board's IDs are all higher priority than the other board's IDs. The LCD node is set up to recognize which node is on its own board and which node is not. The screen node will parse messages for an ID, remote transmission request bit, data length code, and data payload. If the message was sent by the local node, it will display in the top half of the LCD, and if it was sent by the remote node, it will display in the bottom half of the LCD. The system is set up so that if a keypad button is held down, the CAN node will saturate the bus with continuous sending of a message with the same ID. The students will see that if they hold down the high priority key and then press the low priority key on the other board, nothing will happen. However, if they hold down the low priority key and then press the high priority key once, they will see that the message was still able to make it through the bus. The code is cleanly broken up into what steps are performed at the press of each button, so that the student will be able to change the ID or data payload that is transmitted with each button press in order to investigate which messages will get priority on the bus. This lab will enable the students to learn the basics of CAN protocol with hands-on experience.

Lab2: Collecting Data from Analog and Digital Sensors

Since a real-life embedded networking system deals with sensors, actuators and distributed control, Lab2 has been designed to train students about how to deal with various types of analog and digital sensors. The students will do this lab based upon the lessons learned from Lab1. Since the board and software set up are ready, the students will expand the system by connecting various types of sensors.

The goals of Lab2 are: 1) to become familiar with various types of analog and digital sensors, 2) to become familiar with various types of input ports of the microcontroller, 3) to be able to initialize the A/D converter of the microcontroller to read data from analog sensors, and 4) to be able to edit and develop codes for reading data from sensors and then display them on the LCD.

Lab2 deals with two analog and two digital sensors. The analog sensors are *Light* and *Temperature* sensors. These analog sensors are connected to the microcontroller via the analog ports. The digital sensors are *Digital Compass* and *Touch Sensor*. The digital compass has an I^2C interface. Thus, it is connected to the microcontroller through the I^2C port. The Touch Sensor is connected to the microcontroller through the standard digital input pins. This lab uses only one CAN-LIN 3 development board that contains two microcontrollers: PIC18F8680 and PIC18F6680. The two microcontrollers on the board are connected by an internal CAN bus. All the sensors are connected to PIC18F8680 microcontroller and the LCD is connected to PIC18F6680 microcontroller as shown in Figure 6. Thus, the sensor data goes from one microcontroller to another microcontroller via the CAN bus before it is displayed on the LCD.

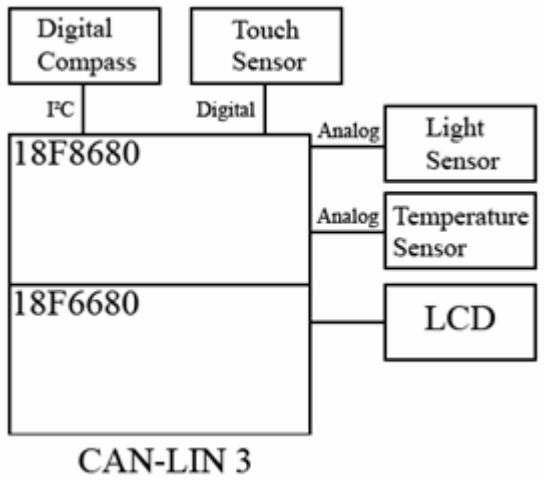


Figure 6: A CAN system with two nodes connected to various sensors and an LCD unit.

Procedures (subroutines) for reading sensors via various input ports are available in our software package. The students can connect default sensors and run the software to collect data from sensors and display them on the LCD. If the students connect other sensors, then they need to edit codes in the subroutines to make the codes compatible with the dynamic range of the sensors. From this lab, the students gain experience in dealing with various types of sensors.

Lab3: Controlling Actuators based on Sensor Data
Students will do this lab based upon the lessons learned and experience gained from Lab1 and Lab2.

The goals of Lab3 are: 1) to become familiar with various types of actuators, 2) to become familiar with various types of output ports of the microcontroller, and 3) to be able to edit and develop codes for activating and controlling various actuators.

Since after completing Lab1 and Lab2 we have a system that can collect data from the environment, we will now expand the system to act on that data. The system will be expanded to include three new features: 1) lights will be turned on if the ambient light level goes too low, 2)

an LED array will emulate the operation of a handheld compass by reading and displaying data from the digital compass, and 3) a fan will engage if the temperature climbs too high.

Like Lab2, this lab also uses only one board with two microcontrollers (two nodes). All the sensors are attached to the PIC18F8680 microcontroller of the board. The PIC18F6680 microcontroller will handle an LCD, fan, compass display, and its own integrated LED lights as shown in Figure 7. Both microcontrollers will communicate with each other using the board's internal CAN bus. The sensor equipped node will poll all of its sensors and send out the data on a regular basis. The actuator equipped node will continuously accept that message and will display the data on the LCD, turn on the lights if the light intensity gets too low, turn on the fan if the temperature gets too high, and will display the correct needle orientation on the compass face.

Procedures (subroutines) for controlling actuators via various output ports are available in our software package. The students can edit codes in the subroutines to control actuators based on sensor data. From this lab, the students gain experience in dealing with various types of actuators.

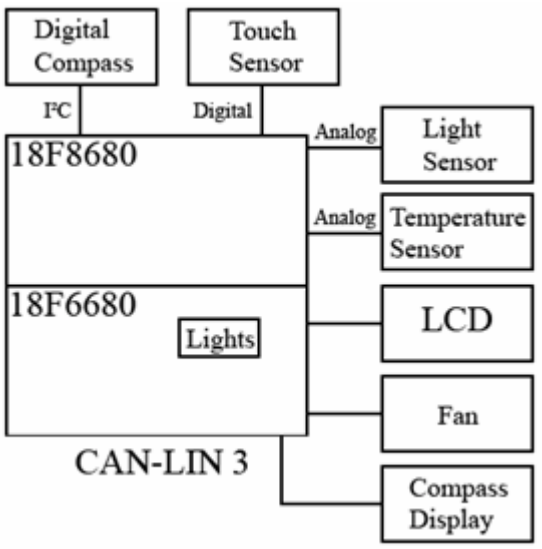


Figure 7: A CAN system with two nodes connected to various sensors and actuators.

Lab4: A Distributed Real-Time System

This lab serves as the culmination of the previous three labs. We will expand upon the system from Lab2 and Lab3, and integrate the keypad hardware from Lab1.

The goals of Lab4 are: 1) to become familiar with a distributed real-time system, 2) to be able to design a small distributed real-time system by integrating various components, and 3) to be able to edit codes based on the requirements.

Like Lab1, this lab consists of two boards connected by a CAN bus. Each board has two microcontrollers (two nodes) connected by an internal CAN bus. One board will continue

functioning as in Lab3, with sensors and other devices. The second board contains the keypad and LCD hardware from Lab1. Figure 8 shows the schematic diagram of the hardware setup of Lab4. The second board will take the sensor data from the first board and allow you to specify what sensor you want the current value from by using the keypad.

The keypad node will send out specific on-demand data requests. Depending on the key pressed, you can ask for light, temperature, touch, or compass data at that exact moment. This will then display that single value requested on the remote LCD node. The student will be able to see how all of the sensor inputs on one node do not need to be replicated on the other nodes because the CAN bus can carry all of that information. They will see that multiple nodes can accept data at the same time, or a single node can request specific data just for itself. They will also see how real-time monitoring of sensors can lead to actions and changes across the entire network.

Procedures (subroutines) for reading sensor data and controlling actuators via various output ports are available in our software package. The students can edit codes in the subroutine to design the system based on certain needs and specifications.

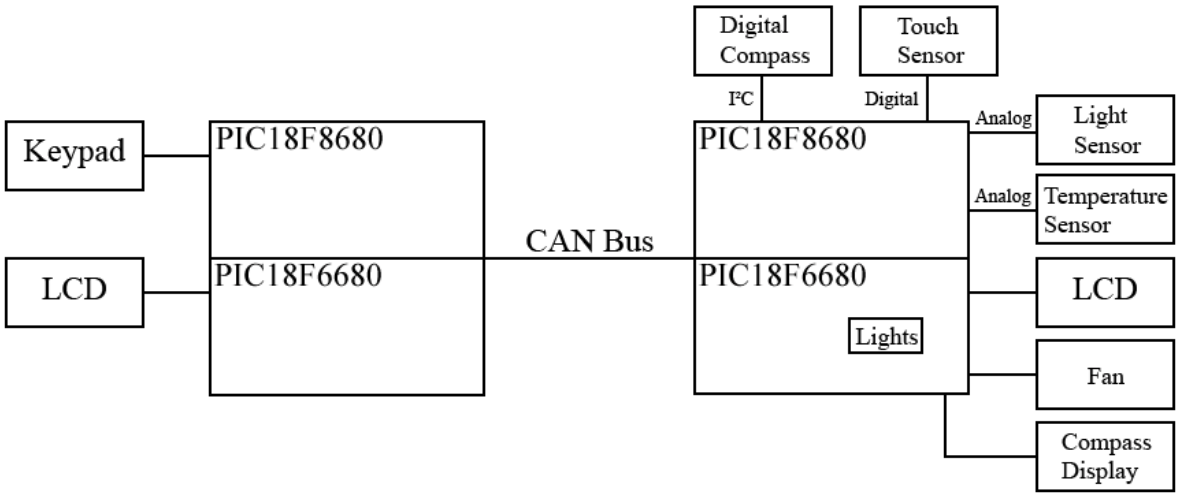


Figure 8: A Distributed Real-Time System.

Lab5: Error Conditions on the CAN Bus

In Lab5, we investigate forced errors in the system and how the CAN protocol handles them.

The goals of Lab5 are: 1) to become familiar with the transmit and receive error counters of a CAN node, 2) to introduce errors in a CAN system and observe the behavior of the system, and 3) to monitor how the contents of the error counters change depending upon different types of errors.

In this lab, only two nodes on one board are utilized as shown in Figure 9. One node has an LCD display that shows that node's transmit error count, receive error count, transmit bus state, receive bus state, and error flags. It can also send a message at the push of a button. The other node has a keypad that allows you to configure the node. You can turn it on or completely off, set the data transmission rate, or send a message. The student can investigate what happens when only one node on the network is active by disabling the keypad node. When they send a

message, an acknowledgment error comes back and the node continuously retries sending the message, incrementing the transmit error count each time. This continues until the error count reaches high enough and the bus state changes on the display. The student can then turn the keypad node back on and see that the message will be sent and the transmit error count will reduce. It will go down with each successful message, and move the bus back to an "on" state. The student can also play around with mismatched bus speeds to start seeing transmit and receive errors from both ends depending on the timing, and how the protocol handles these situations.

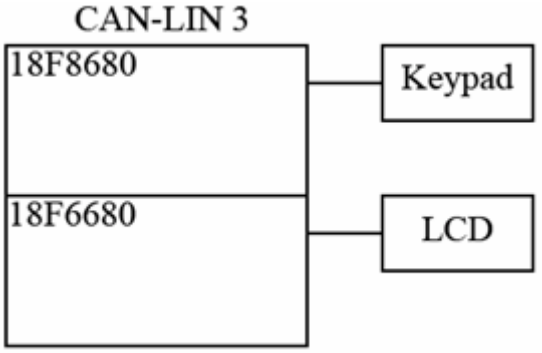


Figure 9: A simple CAN System with two nodes.

In the end, the software package for all of the labs is written in order to make the identification of the program's function as easy as possible in addition to being sectioned off and commented enough to be easy for the student to change the code based on their own curiosity.

Testing of the Software by a Small Group of Students

The software has been made available to a group of 5 undergraduate senior students for beta testing. The students worked on this software as a part of their directed study. One of the goals of the students' directed study was to detect any possible software bugs that may exist in our package. Another goal was to determine the effectiveness of learning using this software. The students had prior knowledge in CAN protocol, assembly language programming and PIC microcontroller. The students found the software was very user friendly and easy to use. It took them very little time to become familiar with the software and the associated hardware setup. The students were able to change codes very easily based on their need for different types of sensors and actuators. All students felt that within a short time they received very good hands-on experience in real-time embedded networking systems. The five students were asked to participate in a survey to assess the learning outcomes of the laboratory. The survey result is shown in Table I.

Table I. Survey results of the learning outcomes.
(Numerical values in the cells of the following table indicate the number of students.)

Learning Outcomes	No Ability	Some Ability	Adequate Ability	More than Adequate Ability	High Ability
1. Explain various features of the CAN-LIN 3 board.			1	2	2
2. Illustrate basic concepts of a real-time system.			1	1	3
3. Explain working principles of various sensors and actuators.			1	2	2
4. Install sensors and actuators in a real-time system.			2	2	1
5. Write codes to read data from sensors.			2	2	1
6. Write codes to operate and control actuators.			1	2	2
7. Design distributed real-time system using microcontrollers, sensors and actuators.		1	1	2	1
8. Write codes to send data using a CAN bus.		1	1	2	1

The above survey results indicate that we should be able to use the software package in a regular undergraduate course in embedded system. Starting from Fall 2008 semester we would like to use this software package in our regular course on embedded system. In future, we intend to add couple of more labs which will cover closed-loop distributed real-time systems.

Conclusion

This paper presented the description of a software package along with its associated hardware setup which has been developed to teach CAN (Controller Area Network) protocol for real-time embedded networking applications. The software was tested by a group of 5 students to determine its effectiveness in student learning. Using this software, the students quickly gained hands-on experience in real-time embedded networking systems. Thus, we believe student learning was enhanced due to the use of this software. The source code of the software along with a detailed schematic diagram of the hardware setup can be made available (free of charge) to any interested instructors for teaching embedded networking systems at their institutions.

Bibliography

1. Bosch, "CAN Specification Version 2.0," Robert Bosch GmbH, Stuttgart, Germany, 1991.
2. Oklahoma State University – Advanced Embedded Systems Design – http://biosystems.okstate.edu/Home/mstone/5030_04/5030_04index.htm
3. Carnegie Melon University – Embedded Systems Design - http://www.ece.cmu.edu/~ece549/index.html
4. Wayne State University – Capstone Design – http://ece.eng.wayne.edu/~smahmud/ECECourses/ECE4600/ECE4600.htm
5. Vector Group Worldwide – http://www.vector-worldwide.com/
6. Dearborn Group - http://www.dgtech.com/

7. International Standards Organization, “Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling,” ISO 11898-1, 1993.
8. International Standards Organization, “Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit,” ISO 11898-2, 1993.
9. CANopen Application Examples – http://www.canopen.us/applications.htm
10. Vector CANoe – http://www.vector-worldwide.com/vi_canoe_en,,223.html

Appendix

Detailed step by step activities of Lab-1

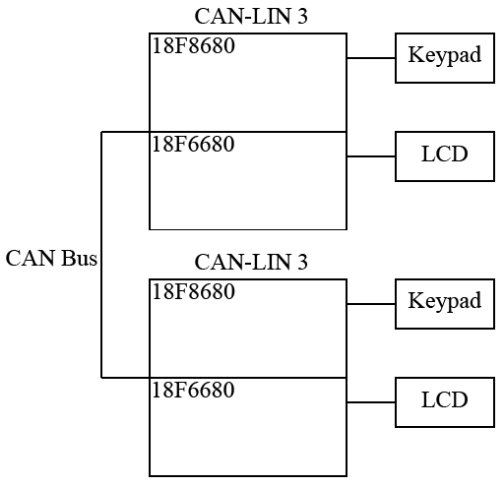
Lab1: Introduction to Hardware, Software and Basics of CAN protocol.

The goals of Lab-1 are as follows:

- To become familiar with the usage of CANLIN-3 Development Board while touching on some of the basics of CAN communication.
- To become familiar with the hardware requirements for setting up the board for this and future CAN labs along with running through the procedures for programming the multiple microcontrollers on each board.
- To become familiar with the usage of MPLAB software.
- To become familiar with the software running on the boards and make modifications to it for different system behavior.

Block Diagram

This lab utilizes four microcontrollers sitting on two CANLIN-3 boards. Each board has its own keypad connected to a PIC18F8680 along with an LCD connected to a PIC18F6680, as shown in the figure below. The two boards are connected with a serial cable joining the CAN bus on each board.

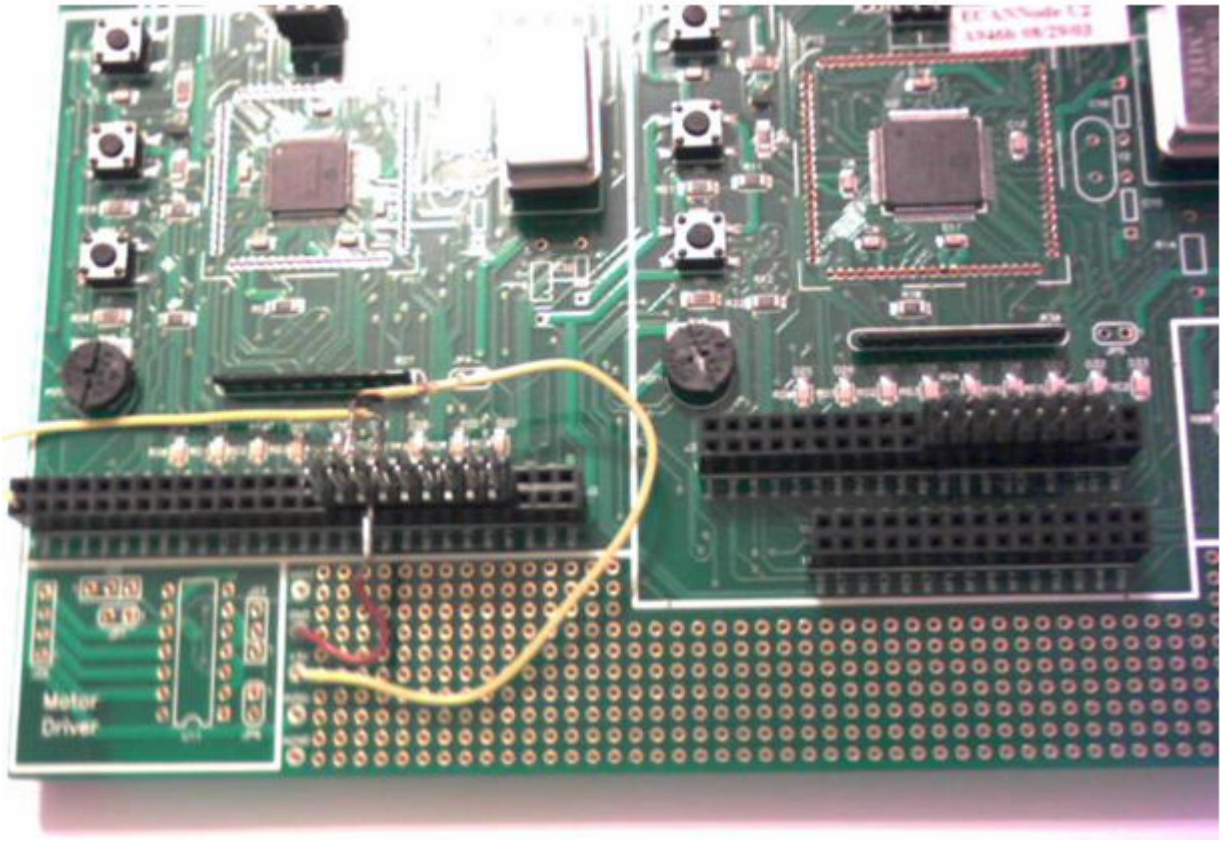


Parts List

Quan	Manufacturer	Item Name	Part Number
1	Microchip	MPLAB ICD2	DV164005
2	Microchip	CAN-LIN3	DM163015
2	Grayhill	4x4 Matrix Keypad With Cable	96BB2-006-R
2	Hantronix	LCD Character Display 4x40 (Grey, Reflective)	HDM40416H-5-S00S
2	Sullins Electronics (Break Into 2x9, 2x9, 2x8)	.1" 52 Position Male Header	PEC26DFCN
2	Sullins Electronics	.1" 18 Position Female Receptacle	PPPC092LFBN-RC
2	Sullins Electronics	.1" 30 Position Female Receptacle	PPPC152LFBN-RC
2	Sullins Electronics	.1" 40 Position Female Receptacle	PPPC202LFBN-RC
2	Sullins Electronics	.1" 54 Position Female Receptacle	PPPC272LFBN-RC
4	Tyco Electronics (AMP)	.1" 8 Contact Female To Female Flat Flex Cable (8" Long)	A9BBA-0808F (DigiKey)
4		3" Wire	
2		1" Wire	
1		6' Female To Female Serial Cable	

Building Hardware (If Needed)

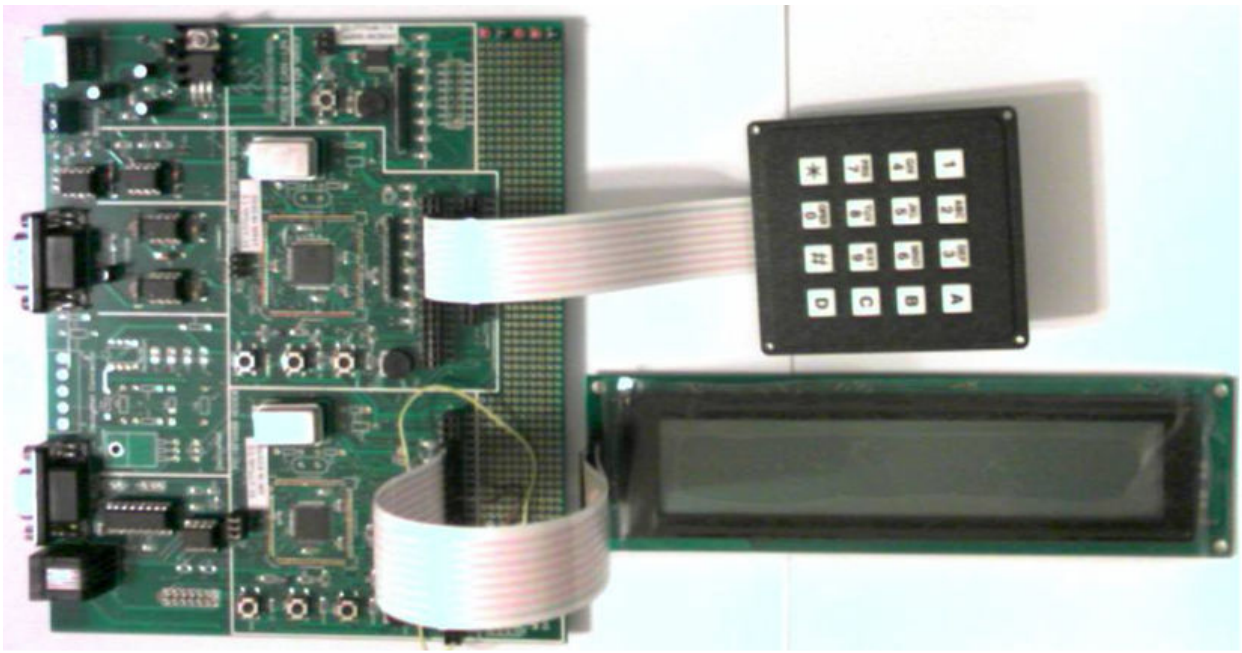
- The first thing that needs to be done is the soldering of all of the female receptacles to the board.
- The 54-Pin, 40-Pin, and 30-Pin receptacles all need to be soldered onto the CANLIN-3 board where the microcontroller pins are pulled out. The 18-Pin receptacle needs to be soldered onto the LCD board where its pins are pulled out.
- With the receptacles in, there is one more complex piece to put in.
- A 2x9 male header needs to be inserted into the receptacle near the 6680. It will be placed from pin RF7 to RD7. The header is taller than the receptacle is deep to allow you to bend the bottom half of three pins out so that they do not go into the receptacle. The pin above RE3 needs to be bent out and connected to GND. The pin above RE2 is bent out and goes to +5V. These power the LCD. Finally, the pin above RE4 is bent out and goes to RA0. This allows the potentiometer on the board to be used to adjust the LCD contrast.
- Finally, you can placed a 2x8 male header between pins RC7 and RB1 under the 8680, and you can place a 2x9 male header in the LCD receptacle. That will give you the board below.



Building Hardware (For The Student)

- Hopefully you didn't have to do all that soldering and it was already done for you. However, it wouldn't be bad to check and make sure the male headers are in the correct location.
- The 6680 should have a 2x9 header from RF7 to RD7 with a pin going to GND instead of RE3, a pin going to +5V instead of RE2, and a pin going to RA0 instead of RE4.
- The 8680 should have a 2x8 header from RC7 to RB1.
- The LCD should have a 2x9 header filling the receptacle.
- Now we need to check the flat flex cables.
- Pin 1 on the keypad should connect to RC7.
- Pin 8 connects to RB1
- Cable 1: Pin 1 on the LCD connects to RF7, Pin 15 connects to RE1.
- Cable 2: Pin 2 on the LCD connects to RF6, Pin 16 connects to RE0.

- This leaves the following pins open on the headers:
- LCD: 17, 18
- Board: 6680 – RD7, RG0 8680 – RB2, RB4, RB6, RC0, RC2, RC4, RC6, RD0
- It should leave you with a board looking like one shown below



Software Preparation

In the file package with this lab, you should have two sets of files

CANTx.asm, Keypad.asm
CANRx.asm, LCD.asm

The CANTx files are for the input gathering and data transmitting PIC18F8680s
The CANRx files are for the data receiving and displaying PIC18F6680s
The programming will be broken into two different projects, one for the 8680s and one for the 6680s.
Both boards will be programmed using the same files, albeit with changes made to the source files.
Start by creating two directories for your projects. This lab will use:

C:\PIC18\CANLab1\CANTx – For the CANTx files
C:\PIC18\CANLab1\CANRx – For the CANRx files

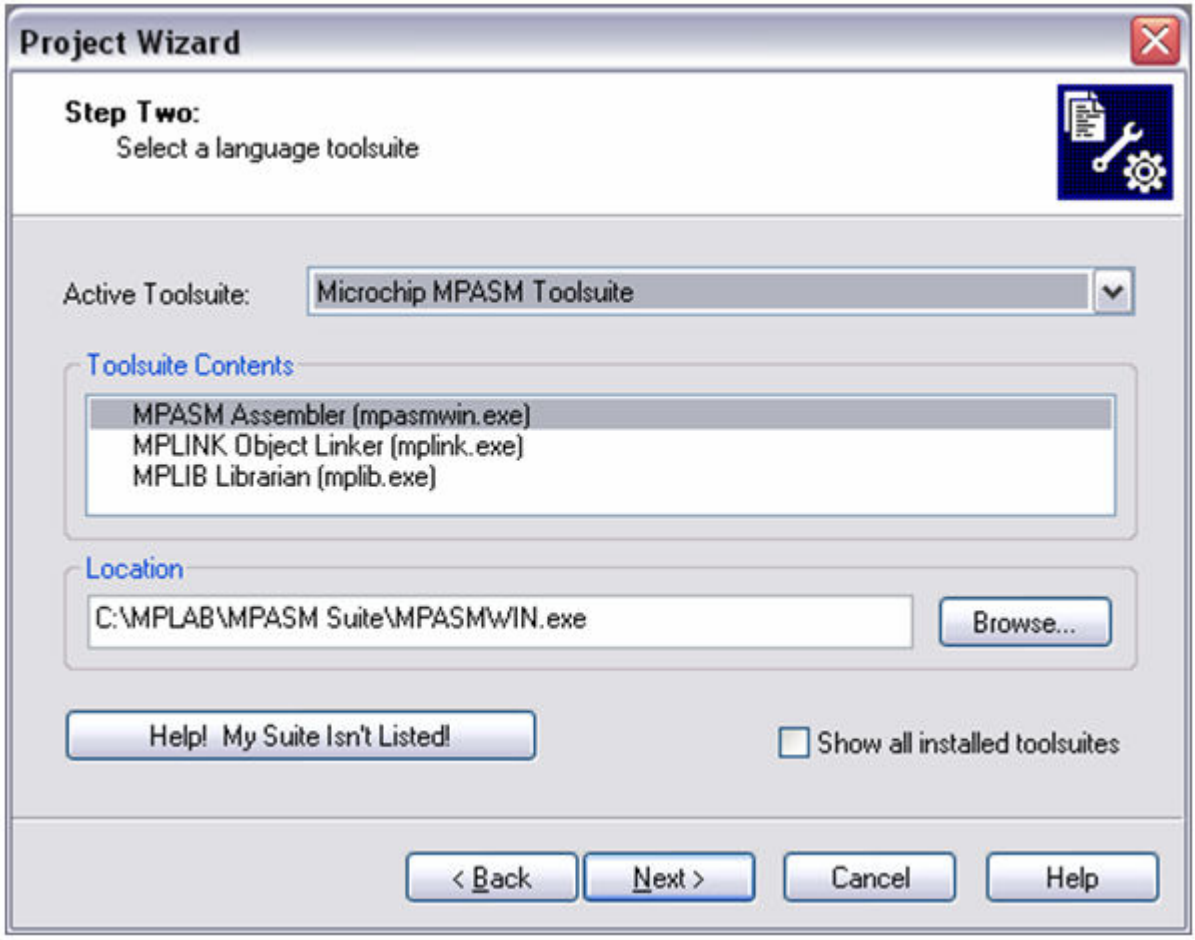
Copy CANTx.asm and Keypad.asm into the CANTx directory.
Copy CANRx.asm and LCD.asm into the CANRx directory.
Open up MPLAB IDE on your PC. This guide will be using version 7.60, so some steps may be slightly different depending on your IDE revision.

New Project

We are going to start with the CAN Transmitters, so get the CANTx.asm and Keypad.asm files ready.
Start by going to Project >> Project Wizard. Hit next on the "Welcome" screen.

Choose Device
Since we will be using the 8680 microcontroller, choose PIC18F8680 from the device selection drop-down box. Hit next.

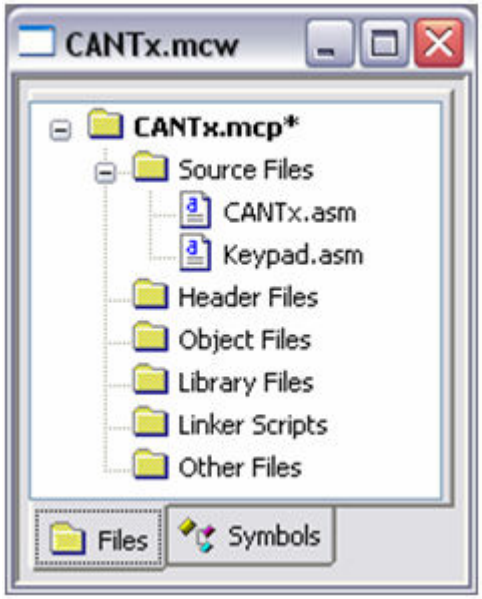
Choose Toolsuite
Select the "Microchip MPASM Toolsuite" in the Active Toolsuite drop-down box. The three choices should all be in the \MPASM Suite\ directory. Hit next.



Create Project
Under "Create New Project File" put in the directory you created earlier to CANTx. Hit next. Skip past adding project files by hitting next again. Hit Finish.

Add Source Files

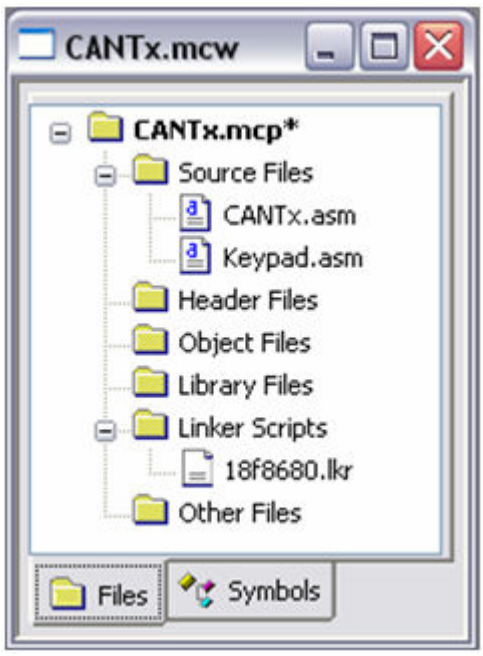
Open your CANTx directory in Windows Explorer and drag-and-drop the files into the project window shown below.



Add Linker Scripts

In the project window, right click on "Linker Scripts" and choose "Add Files…". Linker files are stored in \MPLAB\MPASM Suite\LKR We need to choose the non-debug 8680 file: 18f8680.lkr and hit "Open"

With the two source files and one linker file, your project window should now look like the one below.



Check / Alter Code

Open up the CANTx.asm file in your preferred editor.
Search for the lines:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
BOARD = 0                    ;Using Board 0
```

;;
One board has to be chosen as Board 0 and the other as Board 1.
If you are programming board 0, then leave it at BOARD = 0
If you are programming board 1, change the middle line to BOARD = 1
Save the file after the changes are done.

Build Code
With the code saved, go to Project >> Build All to assemble and link the project. If everything went fine, the output window should appear and come up with zero errors along with a "BUILD SUCCEEDED" message.

Hardware Connection
Plug the ICD2 device into the USB Port on your PC.
Plug the Power Brick into the CANLIN 3 board at (18).
Plug the ICD2 device into the RJ-11 jack on the CANLIN 3 board at (17).
Make sure the 2x3 Jumper is on the ICD Header above the 8680 microcontroller.

Select Programmer
Now we need to select the programmer.
Go to Programmer >> Select Programmer >> 2 MPLAB ICD 2

Connect
If MPLAB is not set up to automatically connect to the ICD 2, go to Programmer >> Connect to do so.

Successful Connection?
If you had a successful connection, the output window will show that the ICD connected, found the target device, ran a self test, and is ready.

Program It
If everything checks out, we can program the device by going to Programmer >> Program.

Programmed?
If the output window shows "Programming succeeded" we are done for this chip.

Program The 8680 On The Other Board
Now we can retrace a few steps to program the PIC18F8680 on the other board.
DO NOT Close MPLAB or your current project. We can continue using it.
Shortened Steps To Program New Board:
Open CANTx.asm in your editor.
Change the BOARD = 0 line to BOARD = 1 and save file.
Go to Project >> Build All to build code again.
Connect your second board to power and the ICD 2. Move the ICD jumper to above the 8680 microcontroller on that board.
Go to Programmer >> Connect to verify you are connected to the new microcontroller.
Go to Programmer >> Program to program the chip on the second board.

Check / Alter Code
Open up the CANRx.asm file in your preferred editor.
Search for the lines:
;;
BOARD = 0 ;Using Board 0
;;
If you are programming board 0, then leave it at BOARD = 0
If you are programming board 1, change the middle line to BOARD = 1
Save the file after the changes are done.

Build Code

Go to Project >> Build All to build the code.
If everything went fine, the output window should appear and come up with zero errors along with a "BUILD SUCCEEDED" message.

Hardware Ready?
Again we have to deal with the hardware connections. Make sure the following has occurred.
Plug the ICD2 device into the USB Port on your PC.
Plug the Power Brick into the CANLIN-3 board at (18).
Plug the ICD2 device into the RJ-11 jack on the CANLIN-3 board at (17).
Make sure the 2x3 Jumper is on the ICD Header above the 6680 microcontroller.

Connect And Program
If all of the hardware connections are made:
Go to Programmer >> Select Programmer >> 2 MPLAB ICD 2
Go to Programmer >> Connect if MPLAB does not auto-connect to the device
Go to Programmer >> Program to program the chip
If the output shows a successful program as shown below, you are done with this chip.

Program The 6680 On The Other Board
Now we can retrace a few steps to program the PIC18F6680 on the other board.
DO NOT Close MPLAB or your current project. We can continue using it.
Shortened Steps To Program New Board:
Open CANRx.asm in your editor.
Change the BOARD = 0 line to BOARD = 1 (or the other way around) and save file.
Go to Project >> Build All to build code again.
Connect your second board to power, and the ICD 2. Move the ICD jumper to above the 6680 microcontroller on that board.
Go to Programmer >> Connect to verify you are connected to the new microcontroller.
Go to Programmer >> Program to program the chip on the second board.

Running The Programs
With that, all of the microcontrollers should be ready to run.
Remove the ICD 2 connection from the CANLIN 3 board.
Ensure both boards are powered from the plug at (18) and are connected together using a female-to-female serial cable at plug (15).

Program Operation
CANTx
Board 0
The keypad is polled and all keys except 1-8 are ignored.
Pressing keys 1-8 will cause the 8680 to send CAN messages.
    Pressing Key X:
        SID = 100 + X
        X Data Bytes: Value = 0 + X
Board 1
The keypad is polled and all keys except 1-8 are ignored.
Pressing keys 1-8 will cause the 8680 to send CAN messages.
    Pressing Key X:
        SID = 228 + X
        X Data Bytes: Value = 0 + X

CANRx
CAN receive buffer is polled while waiting for message.
When message is received, it is parsed for Standard ID (Binary), Remote Transmission Request Bit (Binary), Data Length Code (Binary), and Data Bytes (Hexadecimal).

Message sent from local board is displayed on top half of LCD and message sent from remote board is displayed on the bottom half of the LCD.

sssssssssss: 11-Bit Binary Standard ID
r: 1-Bit Binary Remote Transmission Request
dddd: 4-Bit Binary Data Length Code
yy: 8-Bit Hexadecimal Data Byte (Display Only Shows Valid Bytes)

```
SID: sssssssssss
RTR: r DLC: dddd yy yy yy yy yy yy yy yy
SID: sssssssssss
RTR: r DLC: dddd yy yy yy yy yy yy yy yy
```

Items Of Note

While we know that you cannot have two CAN nodes sending messages with the same ID, we can see that more than one node can receive and process a message with the same ID. When you press a key on one keypad, the message will appear on the local LCD on top and on the remote LCD on the bottom.

Because the keypad is being polled and because the processor is faster than the CAN peripheral, if you hold down a key on the keypad, the bus can be saturated by the continuous sending of that message.

If both keypads have a button pushed down, we can see CAN Arbitration at work. Try holding down one button on one keypad and pressing individual buttons on the other keypad. Then switch which keypads you are doing it to. One keypad will respond while the other is held down, but it won't work the other way around. That is because one board's messages will always win arbitration over the other. Which messages are dominant over which other messages?

Exercise

Instead of always sending data that is the same as the DLC, let's try something different. Since key 9 isn't doing anything, we can use that.

Find the following code in CANTx.asm

```
        ;Keypad 9 [W = 10]          ;Not Valid
key10   decf    WREG
        bnz     key11
        goto    Loop
```

Change It To

```
        ;Keypad 9 [W = 10]
key10   decf    WREG
        bnz     key11
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        movlw   b'00000001'          ;STD ID = 109, 237
        iorwf   BOARDID, 0
        movff   WREG, TXB0SIDH
        movlw   b'10100000'
        movff   WREG, TXB0SIDL
        movlw   b'11011110'
        movff   WREG, TXB0D0;Load Data Regs
        movlw   b'11001010'
        movff   WREG, TXB0D1
        movlw   b'11011110'
        movff   WREG, TXB0D2
        movlw   b'00000011'
        movff   WREG, TXB0DLC        ;Sending Three Bytes
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
        goto    SendMsg
```

The new code will set a new CAN message to key 9 with the ID = 109. This message will be sent containing three bytes of data. Remember to set BOARD = 0 at the beginning of the code, and to put the ICD jumper above the 8680 chip on the board. Save, build, and program the 8680 on Board 0. Verify that the code sends three bytes.