Wireless Mobile Robots: A Tool for Undergraduate Electrical Engineering Instruction

PI: Dr. Richard S. Wolff
Montana State University
Department of Electrical and Computer Engineering
rwolff@montana.edu
(406) 994-7172

Final Report
April 10, 2006

Wireless Mobile Robots: A Tool for Undergraduate Electrical Engineering Instruction

PI: Dr. Richard S. Wolff
Montana State University
Department of Electrical and Computer Engineering
rwolff@montana.edu
(406) 994-7172

Final Report
April 10, 2006

Project goals

The goals of this NWACC grant are to develop an 802.11 b/g wireless interface for the Montana State University mobile robot, to install the complementary wireless infrastructure in our teaching labs, and to provide a platform that can be used to enhance the curriculum of our microcomputer software engineering course, EE371— Microprocessor Hardware and Software Systems—by incorporating robot-centered instruction. This will enable the students to program and control the mobile robot in real time over the wireless network and to measure outcomes in real time. These additions will be leveraged by other courses in communications and control systems in the future.

The mobile robot is currently used in our basic first year introductory engineering course, EE101, where students learn engineering skills by constructing and testing the roving robots. Our intention is to actively engage promising young engineering students in the fields of electronics, robotics and embedded computer systems—areas of ever-increasing importance to this nation's goals in communications, transportation, and space exploration. Because each of our students will now have a mobile and programmable robot, we can capitalize on their experience by utilizing the robots as a teaching platform in subsequent courses. Specifically, we are incorporating wireless communication technology between the robots and laptop or tablet computers to support student learning in communications, embedded systems, and control. Our academic leadership effort is producing mechanical drawings, schematics, laboratory experiments, and supplementary materials sufficient to allow other universities to adapt the course to their own needs. Further details on the EE101 robot project including lab manuals, class assignments and assembly guides are posted on our Web site http://www.coe.montana.edu/ee/rmaher/ee101/ecebot/.

Summary of accomplishments

- Senior design team organized and project initiated: Three Electrical and Computer Engineering students (Brad Benjamin, Carson Drew, Chris Stephani) participated in this project and developed the prototype robot wireless interface as an assignment in their senior design capstone project course, EE391/EE492. (April – December 2005)

- Summer internship: One student (Brad Benjamin) participated in a summer internship assignment under the direction of the PI to determine project requirements, develop a initial design, select components and develop the project web site. (June-August 2005

- Initial hardware design: The initial functional design for the robot wireless interface was completed and was successfully implemented using a commercially available development board that provided the required functionality. This design was used as the basis for the detailed hardware prototype. (November 2005).

- Initial software design. The initial software to program the wireless board and control the robot over the wireless network was successfully completed and tested. (November 2005).

- Demonstration using campus wireless network: The robot equipped with the development board implementation of the wireless board was successfully demonstrated using the MSU campus wireless network at the Engineering Senior Design Fair. (December 2005)

- Prototype hardware design: The first hardware prototype of the wireless board that meets the robot form factor was completed in December 2005 by the first senior design team (Benjamin, Drew and Stephani). A second EE391/EE492 student team (Ivan VanDessel, Tim O'Neil, Conrad Donavan) began work in Fall 2005 and tested the prototype board. (January 2006).

- Final hardware design: The final hardware design was completed and validated by the second senior design project team (VanDessel, O'Neil, Donavan). (April 2006).

- Final software design: The final robot control software and GUI were completed and demonstrated by the second senior design team (VanDessel, O'Neil, Donavan). April 2006).

- First instructional use of wireless robot: The wireless robot is being used by a third EE391/EE492 senior design team (Daniel Colson, Andrew Edwards, Loo Chee Yang, Ali Alniemi) to program and test the robot running a maze conforming the IEEE MicroMouse robotic navigation competition. (February 2006 - December 2006).

- Operational wireless robots: Components were acquired to provide twelve wireless robots for instructional use in Fall 2006.

- Project schedule: The project was completed on schedule.

- Project budget: The project was completed with the allocated budget.

- Project web site: The web site for the project was developed and made available August12, 2005. The link is http://www.coe.montana.edu/ee/seniordesign/archive/fl05/robot_radio/

Senior design teams

This project was performed largely by Electrical and Computer Engineering (ECE) undergraduate students as assignments for the capstone engineering design course EE391/EE492. Three project teams, consisting of a total of nine students, participated in the project. Additional, one undergraduate worked on the project in the summer 2005 as an undergraduate research assistant.

Three Electrical and Computer Engineering students (Brad Benjamin, Carson Drew, Chris Stefani) were selected to participate in the initial phase of this project to develop the robot wireless interface as an assignment in their senior design capstone project course, EE391/EE492. The team was formed in spring 2005 and completed a preliminary assessment of the functional components for the project. These were presented as a deliverable for EE391. The basic concept of using a Wi-Fi wireless link was explored and the feasibility of interconnecting the 68HC912C32 microcontroller on the robot with a second processor controlling a radio interface was demonstrated. Figure 1shows the project display used at the ECE department design fair held in Spring 2005.

Figure 1. Spring 2005 Wireless Robot Project Engineering Design Fair

One member of this team, Brad Benjamin, worked on the project over the summer as a research assistant, funded by the NWACC grant. Brad developed CAN bus software, as well as the initial website design, described below, The team continued work on the project in Fall 2005 and completed a working prototype of the wireless system, using a commercially available development board, and successfully integrated it with the robot. .Figure 2 shows a proof of concept example of the robot with the prototype wireless module added as a daughter board. This initial effort demonstrated the feasibility of adding a wireless interface to the robot and also identified several design issues that needed to be addressed to assure that the additional functionality could be incorporated without disturbing the basic robot operation and control circuits.

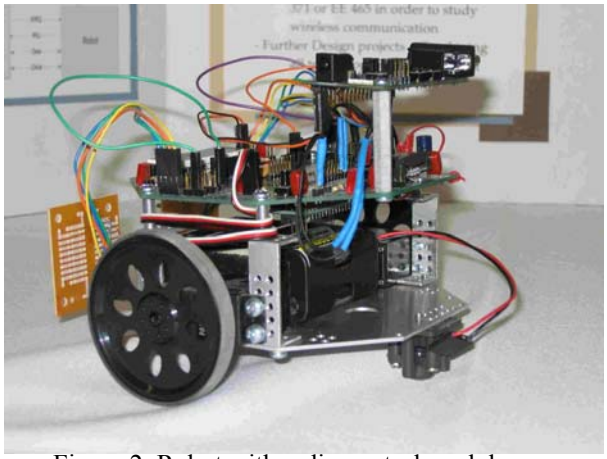


Figure 2. Robot with radio control module

The second senior design team, consisting of Ivan VanDessel, Tim O'Neil, Conrad Donavan continued the project beginning in Fall 2005. Their primary task was to develop a test a printed circuit board implementation of the wireless system and integrate it to the robot. This team began with a design proposed by the first team, implemented and tested it, and used the results to design and build a second generation wireless printed circuit board. The first board completed is shown in figure 3.

Figure 3. First generation wireless printed circuit board, which attaches to the robot as a daughter board.

The third senior design team, consisting of Daniel Colson, Andrew Edwards, Loo Chee Yang, Ali Alniemi, has begun work with the wireless robot to develop navigation and maze-solving capability, using the wireless communications functionality and programmability of the CSM12C32. The team began work in January 2006. This will be the first operational application of the wireless robot.

Two processor design

Selection of a separate processor to handle communications functions was a key design decision in the early phase of the project. It was more effective to run the robot control code, resident

in the primary processor, as a separate program, and to put all the communications and data management functions in a second program running on a separate processor. This enabled the primary robot motor control functions to be implemented without interruption. Decoupling these functions assured that further development of the robot motor drive control program, planned for the coming year, could proceed independently.

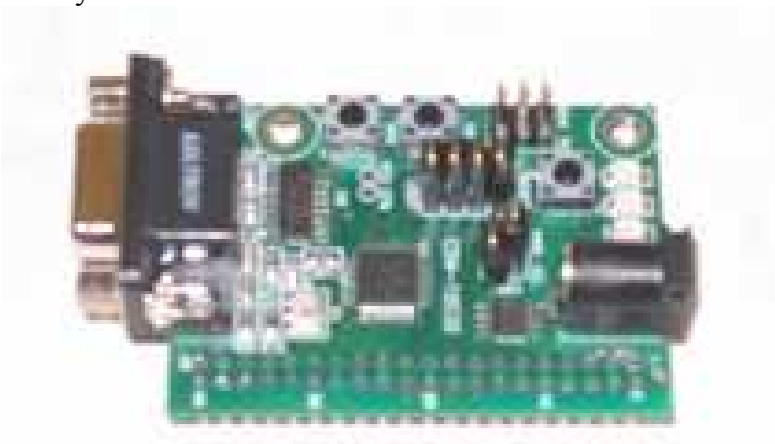


Figure 4 Motorola CSM12C32

A second key design decision involved the selection of the second processor. Plans were already underway to use the Motorola CSM12C32 module in the curriculum of our microcomputer software engineering course, EE371 Microprocessor Hardware and Software Systems to incorporate robot-centered instruction. This module has extensive input/output and processing capability and the team, after careful evaluation and prototyping, elected to use it for the wireless communications and data management functions. The CSM board, shown in figure 3, provides serial RS232 and Controller Area Network (CAN) interfaces as well as 32K Byte Flash EEPROM, 2K Bytes RAM, 31 I/O lines and a rich variety of other features listed below:

♦ Timer/PWM
♦ SCI and SPI Communications Ports
♦ Key Wake-up Port
♦ BDM DEBUG Port
♦ CAN 2.0 Module
♦ Analog Comparator
♦ 8 Mhz Internal Bus Operation Default
♦ 25 MHz Bus Operation using internal PLL
♦ +3.3VDC to +5VDC operation
♦ 40 pin connector provides access to most MCU I/O signals
♦ Power Input Selection Jumper
♦ On-board, regulated +5V power supply
♦ Optional power input from Connector J1
♦ Optional power output through Connector J1
♦ 16 MHz Ceramic Resonator
♦ RS-232 Serial Port w/ DB9 Connector
♦ 8-Ch, 10-bit, Analog Comparator with full rail-to-rail operation and external trigger capability
♦ 8-Channel, 16-bit Timer with Input Capture, Output Compare, and PWM capabilities
♦ 40-pin MCU I/O Connector
♦ 2.0mm Barrel Connector Power Input

♦ DEBUG BDM Connector
♦ DB9 Communications Connector

Physical and electrical interfaces enabling the CSM module to inter-work with the robot were identified after consultation with Dr. Fred Cady, instructor for EE371, and Dr. Rob Maher, responsible for the ECE robot development project. To avoid major modifications to the basic robot, the best approach was to develop a daughter board incorporating the CSM and additional components, which mounts on the robot, sharing power from the primary batteries.

Inter-processor communication

A third key design issue was the selection of the communication path between the robot processor and the CSM module. While the CSM offers a rich set of input/output options, the 68HC912C32 microcontroller on the robot has a limited set of I/O pins (forty), must of which are already dedicated to primary robot operations functions. Several options were explored resulting in the decision to use the CAN bus as the communication path between the CSM and robot 68HC912C32 microcontroller. The advantages of this approach are that the basic I/O architecture of the robot is not disturbed, as the CAN interface is not used for the robot control, and that implementation of the CAN functionality offers the potential for adding other features to the robot in the future, such as additional sensors, a GPS receiver, and actuators, with minimal redesign or programming.

The CAN bus is a well established technology, developed initially to automotive and other control applications is well supported by microcontroller manufacturers and component vendors. The bus provides a high degree of reliability and flexibility, Key features include

- High-integrity serial data communications bus for real-time control applications
- Data rates of up to 1 Mega bits per second
- Excellent error detection and confinement capabilities
- Being used in many other industrial automation and control applications
- OSI session layer known as Time Triggered CAN (TTCAN)
- Documented in ISO 11898 (for applications up to 1 Mega bits per second) and ISO 11519 (for applications up to 125 K bits per second)

Data messages transmitted from any node on a CAN bus do not contain addresses of either the transmitting node, or of any intended receiving node. Instead, the content of the message (e.g. wheel revolutions per second, location, bumper actuated, robot stopped, etc.) is labeled by an identifier that is unique throughout the network. All other nodes on the network receive the message and each performs an acceptance test on the identifier to determine if the message, and thus its content, is relevant to that particular node. If the message is relevant, it will be processed; otherwise it is ignored. The unique identifier also determines the priority of the message. The lower the numerical value of the identifier, the higher the priority. In situations where two or more nodes attempt to transmit at the same time, a non-destructive arbitration technique guarantees that messages are sent in order of priority and that no messages are lost. CAN uses Non Return to Zero (NRZ) encoding (with bit-stuffing) for data communication on a differential two wire bus. The use of NRZ encoding ensures compact messages with a minimum number of transitions and high resilience to external disturbance. The two-wire bus is usually a twisted pair (shielded or unshielded). CAN will operate in extremely harsh environments and the extensive error checking mechanisms ensure that any transmission errors are detected. By using the CAN bus the robot microcontroller can potentially connect to 10 or more devices while only requiring 1 input line, assuring future growth without additional I/O pins.

The content-oriented nature of the CAN messaging scheme delivers a high degree of flexibility for system configuration. New nodes that are purely receivers, and which need only existing

transmitted data, can be added to the network without the need to make any changes to existing hardware or software. Measurements needed by several controllers can be transmitted via the bus, thereby removing the need for each controller to have its own individual sensor. To determine the priority of messages, CAN uses the established method known as Carrier Sense, Multiple Access with Collision Detect (CSMA/CD) but with the enhanced capability of non-destructive bitwise arbitration to provide collision resolution, and to deliver maximum use of the available capacity of the bus. Non-destructive bitwise arbitration provides bus allocation on the basis of need, and delivers efficiency benefits that can not be gained from either fixed time schedule allocation (e.g. Token ring) or destructive bus allocation (e.g. Ethernet.) With only the maximum capacity of the bus as a speed limiting factor, the CAN bus will not collapse or lock up. Outstanding transmission requests are dealt with in their order of priority, with minimum delay, and with maximum possible utilization of the available capacity of the bus.

As CAN interfaces are included in the CSM module and the robot 68HC912C32 microcontroller, these features make CAN an excellent design choice. A transceiver is needed to interface a CAN enabled device to the bus. The Freescale MCP2551 high-speed transceiver was selected, as its small size and low power consumption make it ideal for the robot application. Other key features of this chip include

Supports 1 Mb/s operation

- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent dominant) on TXD input
- Power-on reset and voltage brown-out protection
- An un-powered node or brown-out event will not disturb the CAN bus
- Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
- Protection against high-voltage transients
- Automatic thermal shutdown protection
- Up to 112 nodes can be connected
- High noise immunity due to differential bus implementation

Further information on the architecture and use of the CAN bus in provided in Appendix 1, written by Brad Benjamin to acquaint students in EE371 and other courses with the key features in the bus implementation.

Wireless interface

The goal of the project is to enable communication with the robot using an 802.11 (W-Fi) wireless link. The initial phase of the project explored the availability of Wi-Fi chip sets, embeddable modules and packaged, self-contained transceivers. The first feasibility demonstration was accomplished using an embeddable module mounted on a development board. This approach facilitated interfacing to the CSM as an RS-232 port was available on both modules. However, the cost of the embeddable wireless module, in the range of $200, made this approach unattractive. Furthermore, the size of the unit, with its RS232 interface, was not acceptable.

Alternative wireless products were examined and a more standard, and widely available wireless network interface card (NIC) in a PCMCIA configuration was selected. This approach offers several advantages. First, WiFi PCMCIA NICs are widely available at extremely attractive prices – as low as $10. Second, the modularity affords the option of plugging the wireless NIC in as needed, rather than having it permanently attached to the robot or daughterboard. A PCMCIA compatible wireless

interface module with a serial TTL interface that could easily be interfaced to the CSM was chosen, as the module uses TTL rather than RS232 lowers the cost and power consumption. The Sollae EZL-80, shown in figure 5 was selected. This device runs on 3.3V and draws 10 mA of current. The serial level is TTL and it supports a wide range of data transfer protocols, including UDP, TCP, and IP. Having a built in IP protocol stack on the EZL-80 is a major advantage as it obviates the need to develop this data handling capability on the CSM module. Hence the CSM puts data into IP packets and transfers the packets transparently to the EZL-80, as though it were communicating with an external node. This makes the wireless link "invisible" to the robot.

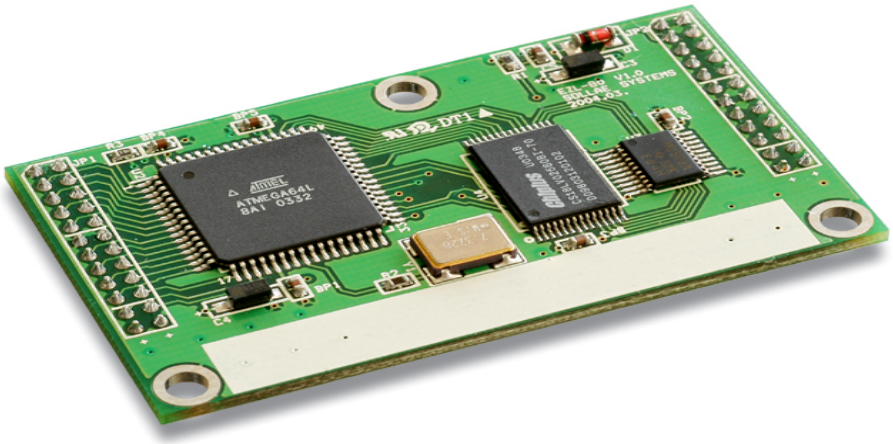


Figure 5 EZL-80 wireless adapter

A further advantage of the EZL-80 is that the PCMCIA card socket is separate from the circuit board. This allows a physical layout where the wireless PCMCIA NIC can be located more conveniently on the daughter board. Figure 6 shows the overall relationship between these components.

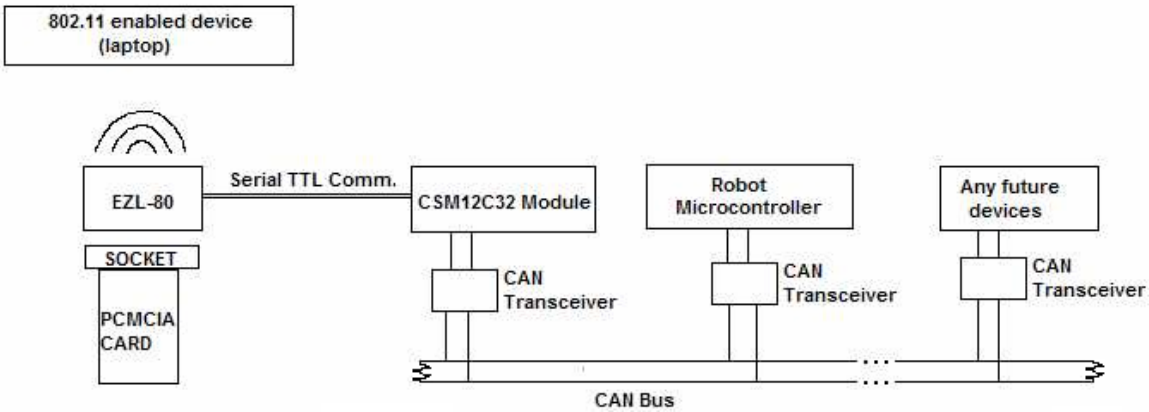


Figure 6 EZL 80 and related components

Figure 6 also shows an 802.11 enabled laptop, which connects to the robot using the MSU campus wireless network, as described in more detail below. The campus wireless network is interconnected to

the campus LAN, enabling students to use PCs or laptops on the campus network to communicate over the wireless link to the robot.

Wireless system implementation

The wireless interface was first implemented as a functional prototype of the selected design using a commercially available EZL-80 development board and other selected components. This approach enabled a full end-to-end demonstration of control of the robot over a wireless network without requiring the development of a custom printed circuit board. This facilitated rapid and low-cost debugging. Figure 7 shows the robot with the first prototype wireless daughter board installed.

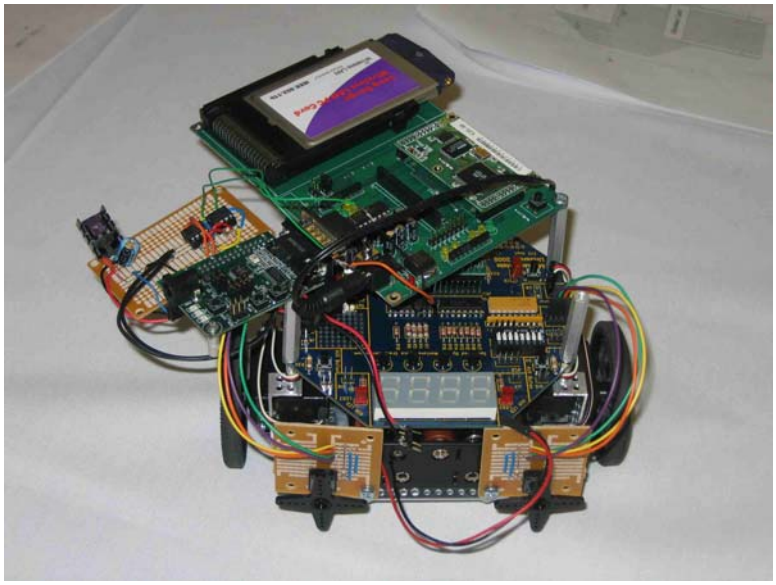


Figure 7. Robot with initial functional prototype wireless board installed.

This configuration was used to debug the hardware and software and led to the initial wireless interface printed circuit board design.

A printed circuit board (PCB) was then designed consisting of the EZL-80, PCMCIA socket, CSM module and CAN transceiver shown functionally in figure 6, to interface to the robot as a daughter board. The board also has a power regulator to convert 5 volts obtained from the robot power regulator to 3.3V needed for the EZL-80. A CAN transceiver was added to the robot board to enable the use of the CAN bus for inter-processor communication. This board design was completed in December 2005 and tested by the second senior project design team. The daughter board layout is shown if figure 8.

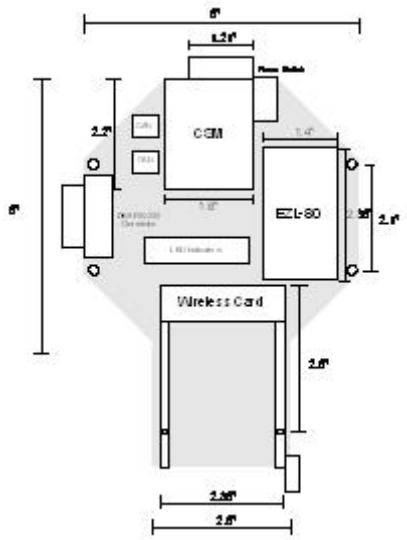


Figure 8. Mechanical layout of first wireless daughter board

Figure 9 shows the first PCB implementation with the PCMCIA card, EZL-80 and CSM modules installed. This board was successfully tested with the robot and several deficiencies were identified, leading to a more compact design for the final board.

Figure 9. First generation wireless daughter board with PCMCIA card, EZL-80 and CSM modules installed.

The final version of the wireless daughter board was designed to fit the robot more compactly and included several wiring changes to support an on/off switch and LEDs to indicate data flow and other functions. The mechanical layout is shown in figure 10. Further electrical details including the circuit design are given in Appendix 2.

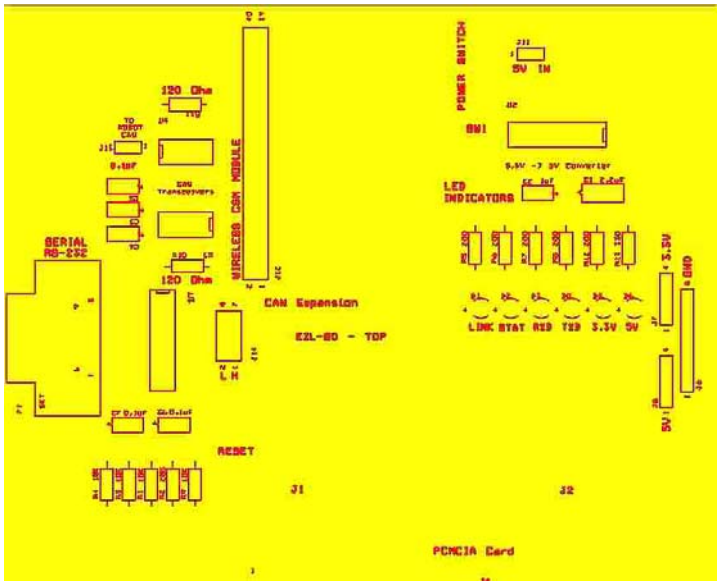


Figure 10. Mechanical layout of final wireless daughter board.

Figure 11 shows the electrical layout for the final version of the wireless daughter board The components to assemble twelve daughter boards for use in EE371 in Fall 2006 have been purchased and will be installed on the boards during summer 2006.

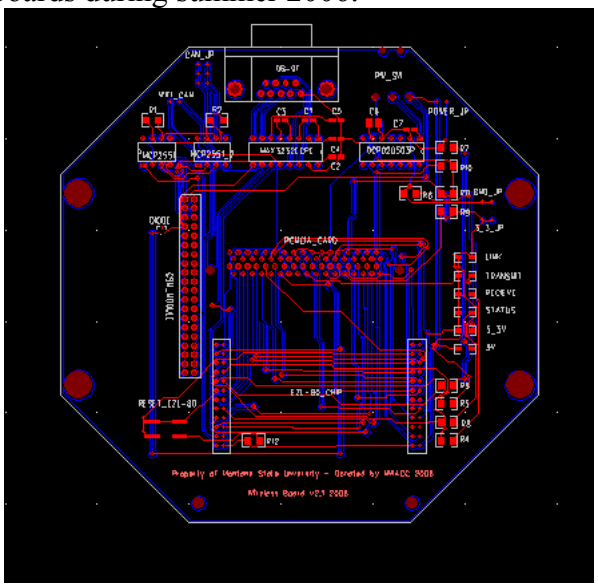


Figure 11. Electrical layout of final wireless daughter board

Software design

There were software requirements for the PC, the CSM-12C32, and the robot microcontroller. A modular approach was used to develop each piece of software, enabling effective debugging. Figure 12 shows the software modules and data flow.

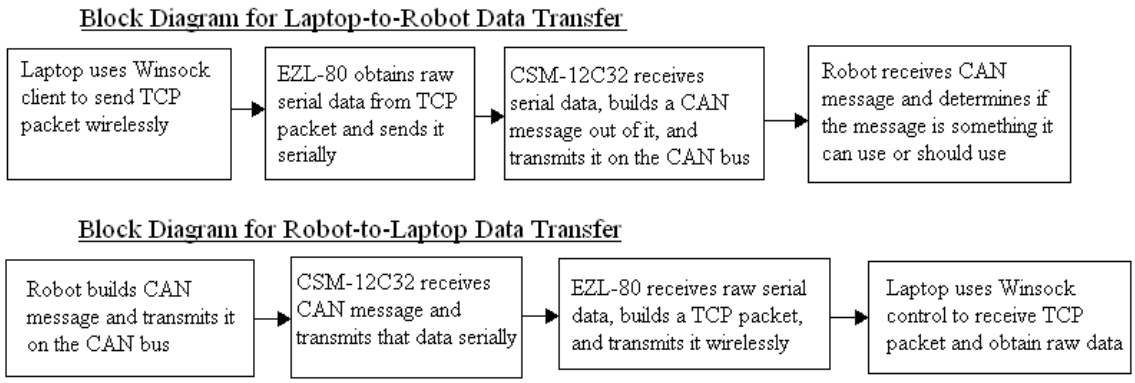


Figure 12. Functional description of data flow and software design.

The existing code on the ECE Robot was not changed in any way. For this project, a 'Mode 8' was added to enable use of the CAN bus.. This mode is basically interrupt-driven from the CAN transmission system. If the robot receives a command, it will execute motor control changes by calling subroutines. If an unrecognized message is received, the robot will stop. It also transmits a signal when a bumper is pressed.

The software module for the CSM and EZL-80 serial link was constructed to use the SCI interface on the CSM to communicate with the built-in serial capability of the EZL-80. The CSM is also driven entirely by interrupts. A CAN interrupt will trigger a serial transmission to the EZL-80, effectively forwarding anything it receives from the robot. A serial transmission triggers the CSM to decode the motor control messages and send the correct values to the robot on the CAN bus.

The software for the PC used to control and communicate with the wireless robot was developed using Visual Basic 6.0 (VB6). VB6 allows for pre-built controls to be implemented easily and graphical user interface is extremely simple as well. The Winsock control was used to create a TCP Client that can connect to the EZL-80. The PC software can connect to the EZL-80 if the EZL-80 is in T2S mode. It can also accept a connection when it is requested by the EZL-80 in COD mode. A GUI was developed for the PC to allow the user to send commands to the robot and send and transmit data. The GUI is shown in figure 13. More information about the EZL-80 modes can be found in the PC software documentation in appendix 3.

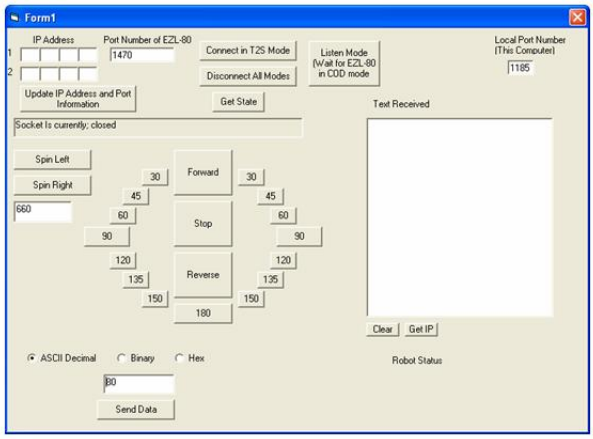


Figure 13. GUI used to control robot, send and receive data over wireless network.

Wireless robot demonstrations and use

The first full demonstration of the wireless robot was carried out using this prototype at the MSU volley ball game in October 2005, where the ECE robots were featured as part of the halftime activities. The Initial phase of the project was completed in December 2005 and demonstrated at the Engineering Design Fair, shown in figure 14.

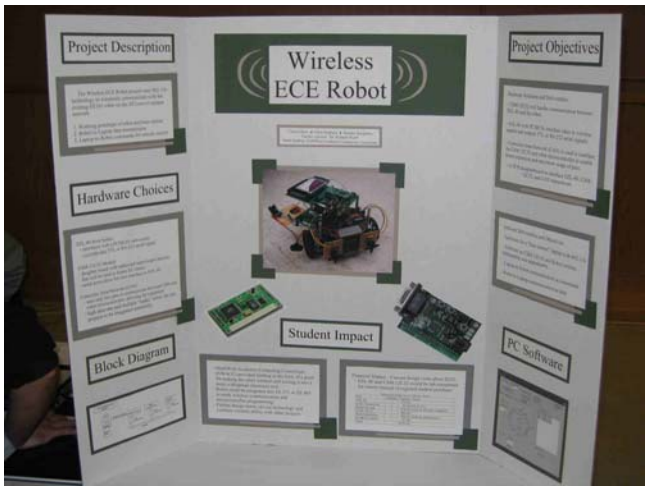


Figure 14. Wireless project featured at the Engineering Design Fair, December 2005.

A user manual, providing step-by-step instructions on configuring and using the robot with the wireless interface was written for use by students in EE371 and other courses. The instructions are attached as appendix 4.

The first operational use of the wireless robot began in Spring 2006, with the initiation of a new senior design project to add position location sensors and navigation software to the robot. This project will use the wireless interface to send data from the robot to a PC where the robot location, trajectory and maze pattern will be plotted. The navigation and maze solving functionality is being designed to enable students to enter the IEEE MicroMouse robot contest. The Montana section of the IEEE has purchased a maze for student use.

Plans to use the wireless robots in the EE371 curriculum are under way. The robot was designed with the CSM module already in use in EE371, enabling a common microcontroller hardware and software environment. Twelve robot wireless daughter boards will be ready for use in the EE371 microcontroller programming lab in Fall 2006.

Interface to the MSU campus wireless network

In summer 2005 a Wi-Fi network was installed throughout the College of Engineering buildings complex for instructional use. The network coverage includes all classrooms and student labs and is interfaced, through a gateway, to the campus wired LAN. Authentication, using a web interface and entry of a login and password is usually required for wireless devices to access the campus network. The student team worked with the campus Information Technology Center to develop a procedure for registering the wireless robots to the campus network without using the normal login method. This enables students to control their robots using any computer on the campus network using the wireless infrastructure.

Project budget

$10,000 has been expended on this project. Principle expenditures have been

- Summer internship: $3152
- Faculty supervision: $2400
- Benefits: $600.39
- Materials: $3847.61

A detailed financial report is provided in appendix 5.

Project web site

The project web site was established in August 2005 and has bee updated as the project proceeded. Figure 15 shows a screen shot of the web page, which can be found at http://www.coe.montana.edu/ee/seniordesign/archive/fl05/robot_radio/. The web page contains an overall project description, as well as technical details and links to pertinent material include specifications, data sheets, and background information.

Figure 15. Home page of project web page

Appendix 1.

CAN (Controller Area Network) Documentation

for groups using CAN with the ECE robot with the Motorola HCS12 processor

Why is CAN important to future groups dealing with the robot?

Connecting the ECE processor to a single device would be simple. A simple serial communication standard would suffice. However, with many ECE senior design projects dealing with the robot, many devices need to interface with the processor. These projects will build on previous technology to add features and improve older features. However, the processor cannot simply perform serial communication with multiple devices as it has limited I/O pins available. CAN is a standard that offers a solution to this problem. CAN only uses two pins (TX and RX) to connect a device to the CAN bus. The features of CAN allow multiple devices to connect to that CAN bus without worry for loss of data. This makes it an ideal choice for the ECE robot projects.

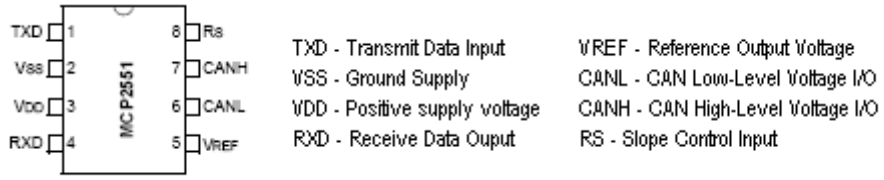
General CAN description

Data messages transmitted from any node on a CAN bus do not contain addresses of either the transmitting node, or of any intended receiving node. Instead, the content of the message (e.g. wheel revolutions per second, location, bumper actuated, robot stopped, etc.) is labeled by an identifier that is unique throughout the network. All other nodes on the network receive the message and each performs an acceptance test on the identifier to determine if the message, and thus its content, is relevant to that particular node. If the message is relevant, it will be processed; otherwise it is ignored. The unique identifier also determines the priority of the message. The lower the numerical value of the identifier, the higher the priority. In situations where two or more nodes attempt to transmit at the same time, a non-destructive arbitration technique guarantees that messages are sent in order of priority and that no messages are lost. Identifiers for small systems are usually 11 bits. If there were enough messages to require a greater number, messages can be sent in 29-bit identifier mode.

CAN works because each transceiver can be put into *dominant* or *recessive* state by the hardware it is connected to. If three devices wanted to send a message simultaneously, they couldn't all put the message on the data bus at the same time – it would be destroyed. There is a type of acknowledge system where it reads the state of the bus to determine if another processor has control of it. A dominant state is characterized by a certain difference in voltage between the bus lines. A recessive state means that that voltage difference is low, and when the transceiver sees that, the processor is allowed to put its message onto the bus. When multiple processors see a recessive bus state, it uses the priority of each message to determine which is allowed to send the bus into a dominant state. Because the bus line in the robot system is so short, there is not much propagation delay so this is not an issue to worry about.

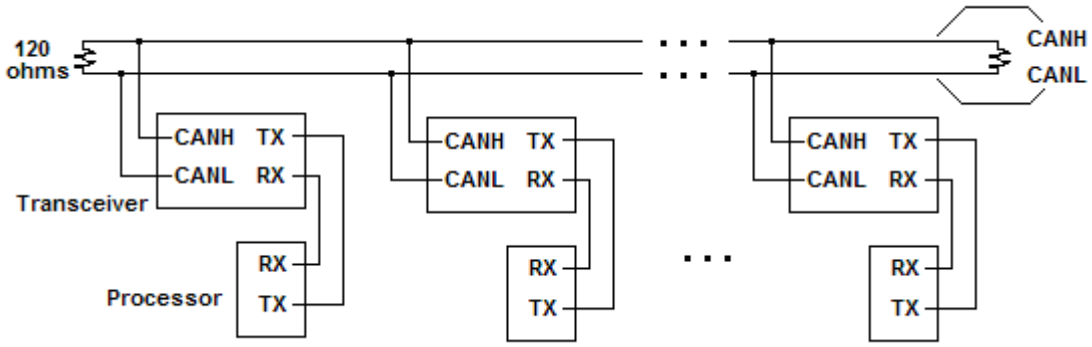
Connecting multiple devices using CAN (hardware)

When connecting multiple devices using CAN, each device must have a CAN transceiver. We used a common one, the MCP2551. It is an 8-pin DIP that offers basic functionality. The following is a MCP2551 pinout description.



In our case, we connected RS to VSS to select high speed mode. It is possible to control the slope of the signal by connecting a resistor proportional to the current from RS to GND. We found that there is minimal electromagnetic interference or residual capacitance at 250 kHz (the frequency we used for testing) so it wasn't necessary to use a slope-control mode. If electromagnetic interference becomes an issue because of higher speeds, a current-controlling resistor must be inserted between RS and VSS/GND. VREF is simply a test for the entire chip. If its voltage is ½ of VDD, the device is working properly.

That leaves the most important pins – TX, RX, CANH, and CANL. TX and RX must go to the corresponding lines on the processor. CANH and CANL must be connected to the bus. The bus must be physically created with two long wires. CANH and CANL should be terminated with a 120-ohm resistor at each end of those wires. The CANH and CANL connections from various transceivers should connect in between those two resistors. The following diagram shows the connectivity:



The diagram is a general description. The processors in the picture could be the robot microcontroller, the CSM Module, or any other processor that can produce valid CAN signals.

CAN Software

Source code is attached at the end of the document. Important documents to have in hand while programming using CAN include:

- MSCAN Block Guide 2.0
- Detailed Register Addresses by Motorola

In the MSCAN Block Guide description, information is given for each bit of each register. Within this guide, there is information about bits that require writing a '1' to them to clear them. It is important to reference this guide when debugging code. Stepping through each command in the simulator and looking at the addresses where the CAN registers are located is the best way to make sure commands are doing what is expected. However, some register values do not always stay set. For example, when the CAN Transmit is scheduled to run, it may actually send the message and clear the register. That can make things confusing while looking at the simulator, so it's something to be aware of. Also, certain registers cannot be written to while Initialization mode is activated, and others cannot be written to while Initialization is disabled.

There are two additional things about writing code that I would like to discuss in more detail. The first is the timing of CAN messages and how to change them using the registers provided. The second discussion involves the ID and data registers and what these mean for the development of the overall project.

CAN Timing

The two registers used for controlling CAN timing are CANBTR0 ($0142) and CANBTR1 ($0143). These registers are separated into components of various length to give the programmer control over the CAN timing. The following is an example calculation of 125 kHz.

$$\text{CAN Frequency} = \frac{\text{crystal frequency}}{2 * (\text{syncseg} + \text{propseg} + \text{p1seg} + \text{p2seg}) * (\text{prescalar} + 1)}$$

There is a great amount of detail involved in explaining how the processor uses the values, but the main purpose is to maintain synchronization. Even when a dominant-to-recessive state occurs early or late, the processor can eliminate Syncseg if necessary and either P1Seg or P2Seg can be shortened or lengthened. The following is a description of each component of CAN synchronization.

Propseg - the propagation time caused by physical delays such as very long wires. Our system is small, so is not considered in programming.
Syncseg – the synchronization segments the processor is allowed to insert to match the timing of the system clock. A typical value for this is 1.
Prescalar (also referred to as BRP or Baud Rate Prescalar) – defines the length of the Time quanta Tq. The values of Syncseg, p1seg, and p2seg are multipliers of Tq.
P1seg & P2Seg – These are phase segments. Typically Phase 1 is longer than Phase 2 and the HCS12 devotes one extra bit to Phase 1 for that reason. Usually these registers and Syncseg add to form a common multiplier (such as 16)
SJW – Syncronization Jump Width – this sets the number of Tq that the phase segments can be altered by to achieve synchronization.

For the 125kHz case, with a 16 MHz crystal oscillator frequency, p1seg and p2seg need to add to 15, so p1seg=8 and p2seg=7 are used. This is arbitrary, 9 and 6 could also be used with similar results. The limitation involved is that both phase segments must be greater than the SJW. The prescalar is then 3 to complete the equation:

$$\text{CAN Frequency} = \frac{16\ \text{MHz}}{2 * (1 + 8 + 7) * (3 + 1)}$$

After obtaining the values needed, then it is important to get them into the registers properly.

CANBTR0	SJW1	SJW2	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
CANBTR1	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10

It is helpful to create the exact binary string and then convert it into a hex digit for programming. CANBTR0 is comprised of SJW and BRP. Keep in mind that the hex values start at 1, so $00 is the decimal number 1. If we use 2 (%01) for SJW and 3 (%10) for BRP, then CANBTR0 = %01000010 in binary. That corresponds to $42 hex. CANBTR1 is comprised of P1Seg and P2Seg as well as a

variable called SAMP that differentiates between 1 sample taken and 3 samples taken every bit. We found 1 sample to be sufficient. The necessary information to create CANBTR1 is SAMP = 0, TSEG2 = 7 (%110) and TSEG1 = 8 (%0111). CANBTR = %01100111. That corresponds to $67 hex. In the sample code at the end of this document, different values are used, but this is a description of how to write to the timing registers and manipulate the CAN timing frequency.

CAN ID's and message table

Theoretically, each message is sent with an ID that identifies what type of message is contained. With a small system, CAN ID's can simply be ignored. In testing, we set them before we transmit a message, but currently in the CANRX interrupt these values are not checked. In the robot system, there are a limited number of possible messages, so the message itself can contain the necessary information. At this point, the robot only receives commands that were processed by the CSM module. Therefore, any data message received can be put through a case-statement to operate a robot command. However, if multiple devices were connected to the robot by CAN, it may be useful to have a table with ID's describing which device produced which message. A sample table is given below:

Identifier	Identifier Source	Type of Data	Notes	Who uses it?
$00	Robot	Critical Robot status	Robot error - feedback to other systems	all other nodes
$03	CSM Module	Commands for Robot	wireless steering	Robot
$0F	Robot	Bumper information	for wireless bumper logging	CSM Module
$3F	GPS	Position coordinates	When $02 is seen, read DLR to know how many bytes to receive	CSM Module and Robot

The identifier itself is a priority. In the CANRX interrupt handling routine, the programmer would check for the highest priority (lowest hex ID) items and handle them first.

Masking or filtering is also possible with the CAN IDs. For instance, if the CSM module was not programmed to be able to use any GPS coordinates, one could set the filtering registers so that the CSM module did not even receive messages with an ID of $03F. In this way, CAN IDs can be used to improve the efficiency of the entire system by reducing the demands on individual CAN systems. When devices receive fewer messages, they have less processing to do.

Masking CAN IDs

There are three register sets to deal with when setting the masks up:
A) CIDAC – determines the configuration of the mask and acceptance banks into filter sizes and the number of filters. The available options are:

- 2 x 32-bit filters
- 1 x 32-bit filter, 2 x 16-bit filters
- 1 x 32-bit filter, 4 x 8-bit filters
- 2 x 16-bit filters, 4 x 8-bit filters
- 4 x 16-bit filters
- 8 x 8-bit filters

The MSCAN 2.0 block guide should be referred to when deciding how to achieve each possible configuration.

B) CIDMR[0-3] – This is a mask on the acceptance filter. Its function is to basically determine which bits to look at for the actual filter itself. A '0' written to a bit means the processor will check an incoming message to see that it matches that bit of the filter. A '1' here is like a "don't care" in that it doesn't matter whether the incoming message is a '0' or '1' because the processor doesn't filter that bit.

C) CIDAR[0-3] – This is the filter that bits of incoming messages are compared to. When a message is rejected because of a filter, it simply allows the next message to overwrite it in the input buffer.

An example would be helpful here. If the desired filter is 0001x1001x0, then CIDMR[0-3] would be configured as 00001000010, making bits 1 and 6 don't-cares. In CIDAR[0-3], bits are ignored if the corresponding bits in CIDMR[0-3] are '1'. CIDAR[0-3] would then be set to 0001<u>0</u>10010<u>0</u> or 0001<u>1</u>10011<u>0</u>. Either value is correct since bits 1 and 6 are ignored.

Transceiver Type

It should not be a problem to use a different transceiver than the MCP2515. As always when connecting hardware, care needs to be taken to ensure TX, RX, CANH, and CANL are connected properly as well as the connections to select the transceiver mode and supply the correct voltage and current.

Processor Type

Without knowledge of the processors that may be used in the future, it is important to reduce the problem down to the main considerations:

- Timing – With some CAN devices, it will simply not be possible to obtain certain CAN frequencies. This should dictate the choice of hardware and/or any group exploring a different processor must be certain that it can change the current processor software to match.
- Registers – A set of registers to manage CAN and store data and ID values is very helpful in the programming of CAN. If a register setup similar to that of the Motorola HCS12 is used, it may even be possible to just find corresponding assembly instructions that do the same thing as the Motorola code included in this document.

Other useful discussion

There is not a specific section to put these in, but the information here may prove helpful in fixing problems along the way.

- Debugging CAN timing – One of the best ways is to put the TX and/or RX pins on the oscilloscope and check the timing of the device. First, visually identify a single bit within the message. Multiple high bits or low bits could throw your measurement off, so it is important to try to find a single bit in the oscilloscope readout. The CAN frequency is the inverse of the period, which can be measured on the oscilloscope. Then use the CAN timing registers to modify the timing and retest and check on the oscilloscope. The CAN timing section above should prove helpful in this part.

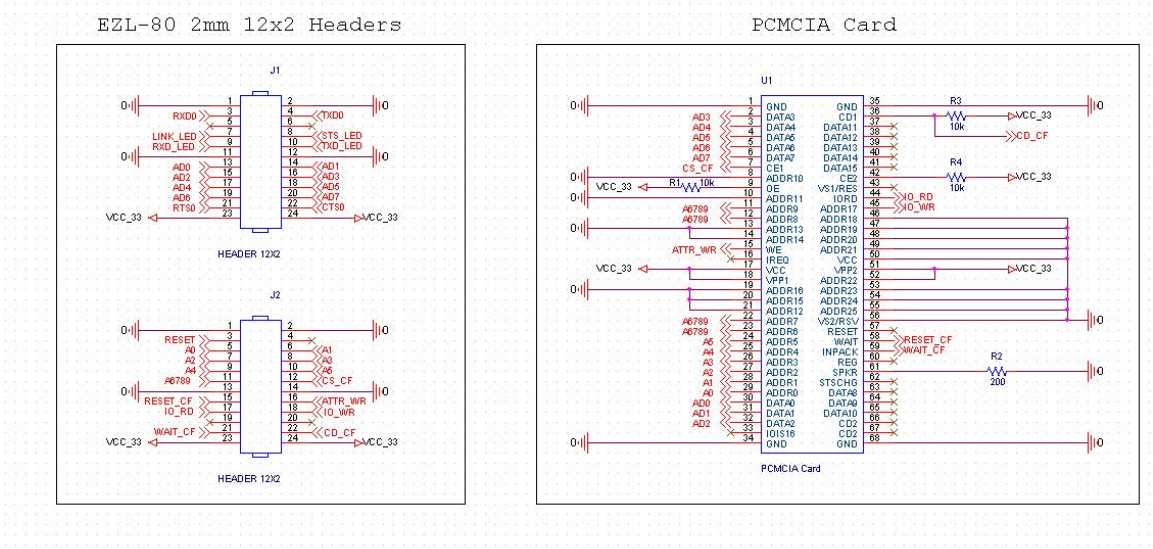
CAN4USB PC device – This adapter enables a CAN bus to be connected to a computer with a USB port. It enables debugging using application software that allows the capture and visualization of CAN messages. This software is provided with the adapter and must be installed on the computer. The CAN4USB adapter is provided by Zanthic.

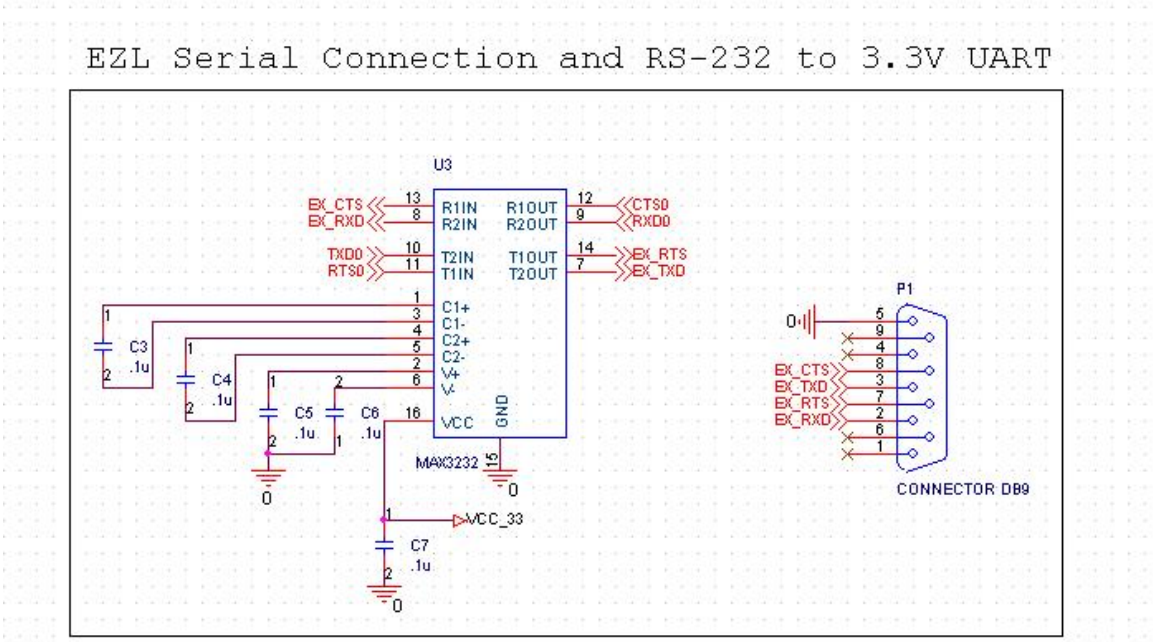
CAN Software Source Code
(Applies to Motorola HCS12 Processor)

```
//******************
//*CANTx – This function takes in a character from a main loop,
// performs all of the message setup, and then transmits the
// message (simply the one character) onto the CAN bus.
// Inputs:  Two characters, at the moment only the first is used.
// Returns:  None
//********************
void CANTx(char var1, char var2) {
  CANCTL0 = (CANCTL0 | 0x01);       // Enter Initialization Mode
  CANCTL1 = (CANCTL1 & 0xEF);     // Exit Listen-only Mode
  CANCTL0 = (CANCTL0 & 0xFE);     // Leave Initialization Mode

  CANTBSEL = (CANTBSEL | 0x01); // Choose Message Buffer 0

  CANTXIDR0 = 0x00;  // Write to Identifier Register 0
  CANTXIDR1 = 0x00;  // Write to Identifer Register 1
  CANTXDSR0 = var1; // Write to Data Byte 0
  CANTXDSR1 = var2; // Write to Data Byte 1
  CANTXDLR = 0x01;  // Write to Data Length Register

  //CANTIER = (CANTIER | 0x01);
  CANTFLG = 0x01;
  _asm("nop");

  _asm("nop");
  CANRFLG = 0x03;   // write a 1 to clear the overrun and receive flags
  CANRIER = (CANRIER | 0x01); // Enable CAN Receive Interrupt
}

//******************
//*CANInit() – This function should be called just before
// the main loop to initialize all of the CAN registers so that they
// work with a processor-to-processor system.
// Inputs:  None   Returns:  None
//********************
void CANInit(){
            // MSCAN Initialization
  CANCTL0 = (CANCTL0 | 0x01); // Go into Initialization Mode
  CANCTL1 = (CANCTL1 | 0x80); // Assert CANE
  CANBTR0 = 0x82; // Timing registers
  CANBTR1 = 0x44; // Set for 250k CAN Bus clock

  CANIDAC = 0x10;                        // Enable ID registers
  CANIDAR0 = 0x00;                                                                    // Set up Identifiers
  CANIDAR1 = 0x00;            // Accept everything
  CANIDAR2 = 0x00;
  CANIDAR3 = 0x00;
  CANIDMR0 = 0xFF;          // Ignore CAN Acceptance Registers
  CANIDMR1 = 0xFF;
  CANIDMR2 = 0xFF;
  CANIDMR3 = 0xFF;

  CANCTL1 = (CANCTL1 & 0xEF);     // Exit Listen-only Mode
  CANCTL0 = (CANCTL0 & 0xFE);    // Leave Initialization Mode
  CANRFLG = 0x03;
  CANRIER = (CANRIER | 0x01); // Enable CAN Receive Interrupt

}
```

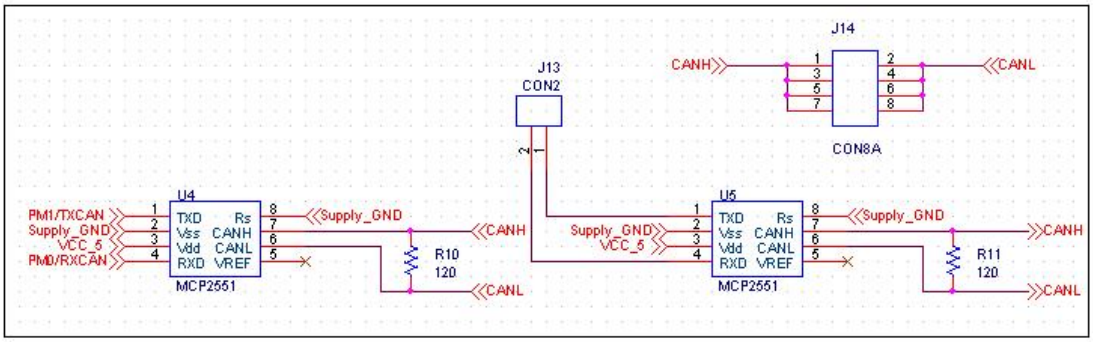
```
//  This can be used as a check to see if a message was correctly
// sent
interrupt 39 void CANTxEmptyInterrupt() {
   //FFB0 is the vector
   _asm("nop");
}

//*****************
//*CANRxInterrupt – This is entered when the processor receives
// a CAN message.  In this example, the message is checked;
// against 3 values that mean something in a sort of look-up table
// for processor-to-processor communication
// Inputs:  CAN Message stored in Receive buffer
// Returns:  None
//******************
interrupt 38 void CANRxInterrupt() {
  if (CANRXDSR0 == 0x66) {
    _asm("nop");
    //PORTB_BIT4 = 0;
    //PORTA_BIT0 = 0;
    PTAD_PTAD7 = 0;
    PTAD_PTAD6 = 0;
  }
  else if (CANRXDSR0 == 0x64) {
      //PORTB_BIT4 = 0;
      //PORTA_BIT0 = 1;
    PTAD_PTAD7 = 0;
    PTAD_PTAD6 = 1;
  }
  else if (CANRXDSR0 == 0x46) {
      //PORTB_BIT4 = 1;
      //PORTA_BIT0 = 0;
    PTAD_PTAD7 = 1;
    PTAD_PTAD6 = 0;
  }
  else {
    //PORTB_BIT4 = 1;
    //PORTA_BIT0 = 1;
    PTAD_PTAD7 = 1;
    PTAD_PTAD6 = 1;
  }

  //CANCTL0 = 0x80;
  _asm("nop");
  CANRFLG = 0x03;
  _asm("nop");
  _asm("nop");
  _asm("nop");
}

interrupt 37 void CANOverrunInterrupt() {
   // This needs to be an exact copy of CANRxInterrupt.
   // We've never received this interrupt, but if so, I believe this is
   // what should be here.
}
```

Appendix 2. Final daughter board circuit design

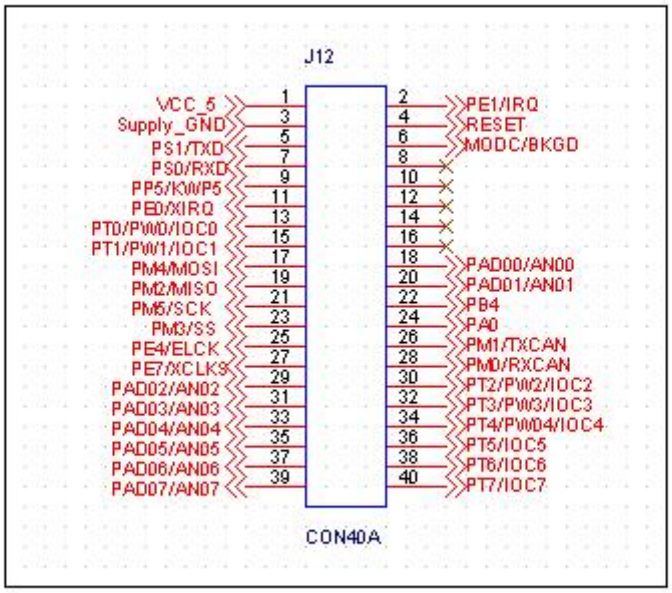


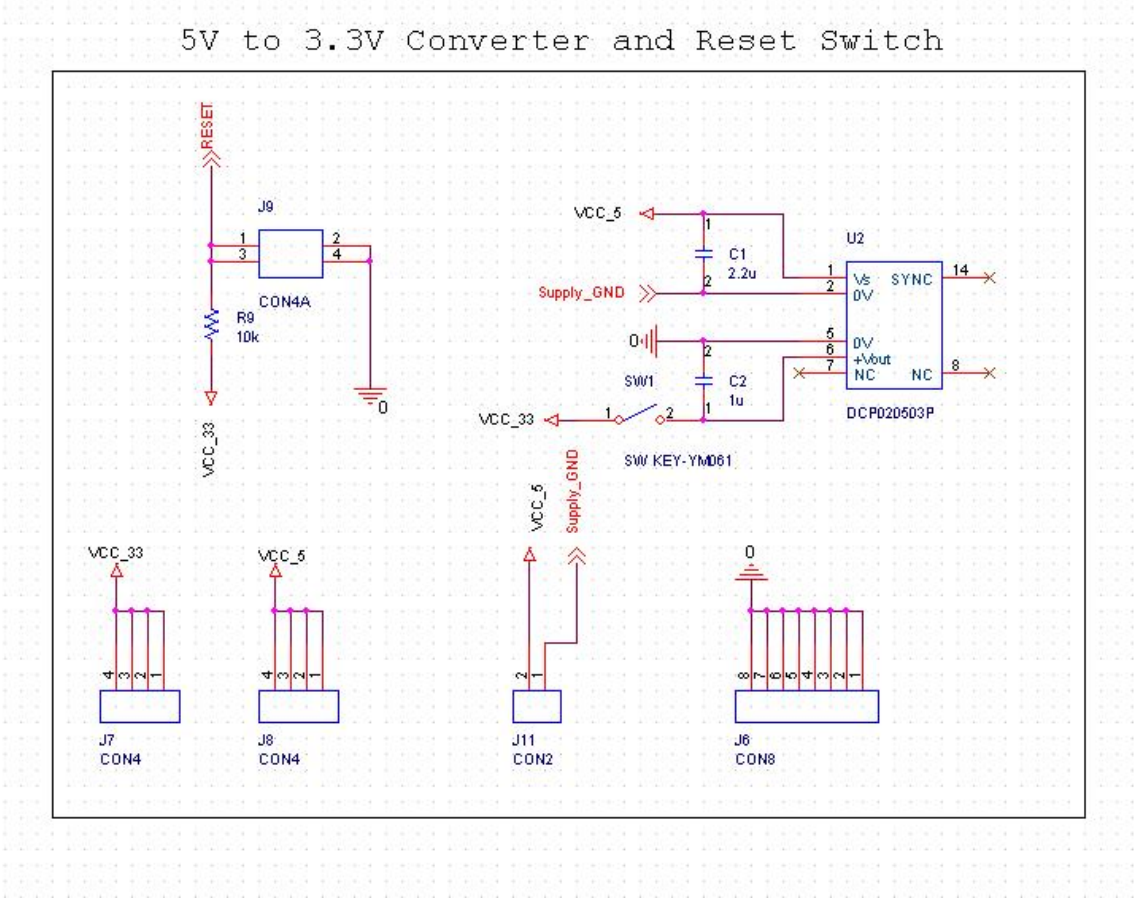


Wireless HCS12 to Robot HCS12 CAN Communication Bus



Wireless CSM Module (HCS12)





Robot CAN Transmit	- pin 17 on robot to pin 1 of Transceiver 1
Robot CAN Receive	- pin 18 on robot to pin 4 of Transceiver 1
CSM CAN Transmit	- pin 26 on CSM to pin 1 of Transceiver 2
CSM CAN Receive	- pin 28 on CSM to pin 4 of Transceiver 2
CAN Bus	- CANH of all transceivers connected to one line
	- CANL of all transceivers connected to another line
	- 120-ohm resistors at each end connecting the two
CSM SCI Transmit	- pin 5 on CSM connected to Pin 3 on JP1 on EZL-80
CSM SCI Receive	- pin 7 on CSM connected to Pin 4 on JP1 on EZL-80

When using TTL level SCI signals, the traces CT-4 and CT-5 must be cut. This disables the UART on the CSM and allows it to output the correct voltage levels on pins 5 and 7. When using RS232 communication, these traces must be intact.

Appendix 3. Software

PC Program Description

This is a description of the program from the point that a connection has been established onward. That means the Status Bar shows 'Socket has connected to remote computer'.

Receiving data from EZL-80

When the EZL-80 sends any information, it will show up in the text received textbox with a timestamp associated with it. This is currently there just to show that we successfully received. Other things can be done with this information by getting into the *tcpClient_DataArrival* function and using the data differently.

Sending data to the EZL-80

There are many different ways to do this. Currently, most of them are associated with button presses for purposes of remote control. The following is a quick description of the button presses:
Forward – Sends 0x01 to EZL-80
Reverse – Sends 0x02 to EZL-80
Spin Right – Sends 0x03 to EZL-80
Spin Left – Sends 0x04 to EZL-80
Stop – Sends 0x05 to EZL-80
Degree Turns – Sends 0x03 or 0x04 for turning, waits a programmed amount of time, and then sends 0x05 to stop it.

The time length corresponds to an integer value stored in a textbox underneath the Spin Left and Spin Right buttons. That value is the timer interval that corresponds to a 90 degree turn. For a 30 degree turn, the timer interval is set to 1/3 of the value in that textbox.

There is also the option to send data in binary and ASCII decimal format. These are not really an integral part of the program, just trying to prove that it can be done. The hex option does not work, because I need a good VB function for converting hex values to ASCII characters. The important part of the function is within the button press handling subroutine where it uses the command 'tcpClient.SendData chr(127)'.

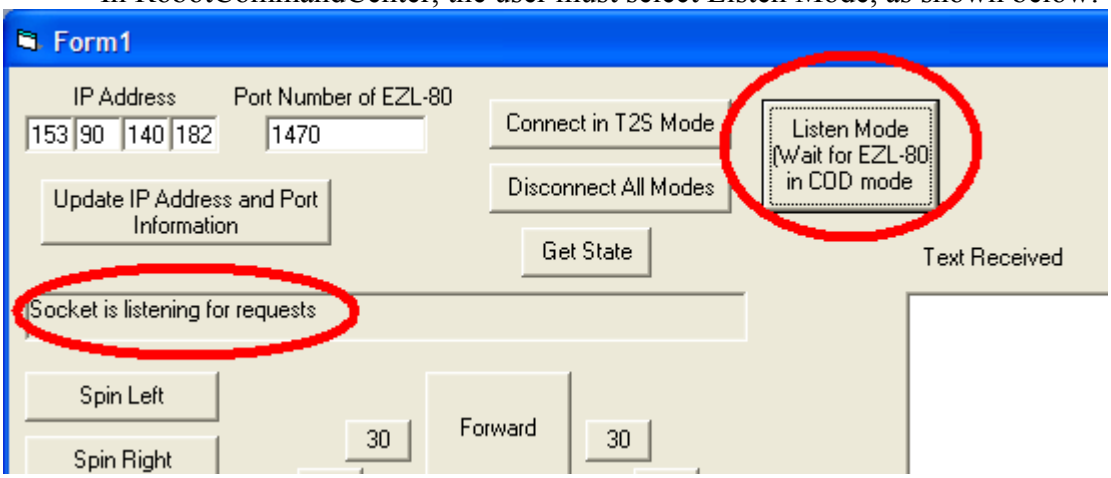
PC and EZL-80 Mode Description

There are two modes of operation that we were able to use to successfully communicate between the EZL-80 and the PC Software that is named 'RobotCommandCenter'. This document will describe how to connect in both modes from the PC side.

Connect on Demand (COD) Mode / TCP Client Mode

This mode is used when you want the Robot and EZL-80 to initialize the connection to the computer. This method was preferred by us, because the Robot sends a one-byte initialization string on startup. The 'Conn. Byte' variable is set to 1 in the ezConfig program. That requires the EZL-80 to send one byte of information to request the connection to the computer.

In RobotCommandCenter, the user must select Listen Mode, as shown below:



When Listen Mode is selected, the status will read "Socket is listening for requests". As soon as the EZL-80 receives a byte of information and transmits it, the status will change to show that the PC and EZL-80 are connected.

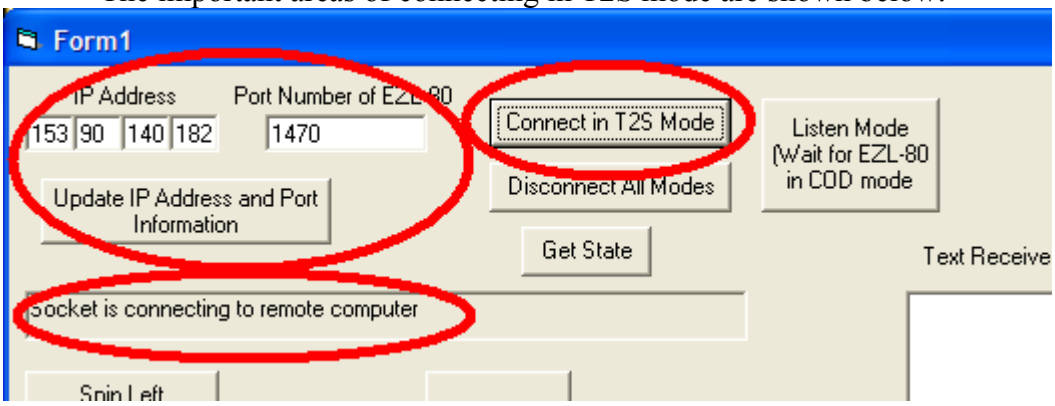
To Disconnect the EZL-80 and the PC, you simply press the 'Disconnect All Modes' button. This can be done not only after a connection has been established, but also when it is currently listening for requests.

It is important to note that you do not need to set the IP address and port number of the EZL-80 in COD mode. For the EZL-80 to connect in COD mode, it needs to have the IP address of the PC running RobotCommandCenter hard-coded into it, but there is nothing extra within RobotCommandCenter. Any computer can use ezConfig to change the IP address that the EZL-80 will try to connect to. Therefore, another computer could take over control. There is an option to add a password to the EZL-80 you are connecting to, so it would be a good idea to do that if you are worried about control.

TCP to Serial (T2S) Mode / TCP Server

This mode is used when you want RobotCommandCenter to initialize the connection between PC and EZL-80. To use this mode, you have to know the IP address and the port of the EZL-80 and update this with the 'Update IP Address and Port Information' button. Once this is done, you must press the 'Connect in T2S Mode button.

The important areas of connecting in T2S mode are shown below:



When Listen Mode is selected, the status will read "Socket is listening for requests". As soon as the EZL-80 receives a byte of information and transmits it, the status will change to show that the PC and EZL-80 are connected.

To Disconnect the EZL-80 and the PC, you simply press the 'Disconnect All Modes' button. This can be done not only after a connection has been established, but also when it is currently waiting for the EZL-80 to accept the connection request.

Other Modes

We did not do any testing with the other modes. UDP mode for UDP packets may be interesting, but RobotCommandCenter uses a Winsock control that had example code dealing with TCP packets. I am not sure what support there is for UDP in Visual Basic 6.0, but it probably exists and is easily implemented. UDP may be interesting because it doesn't have to go through a connection request-and-accept process before data can be transferred.

Data transfer considerations.

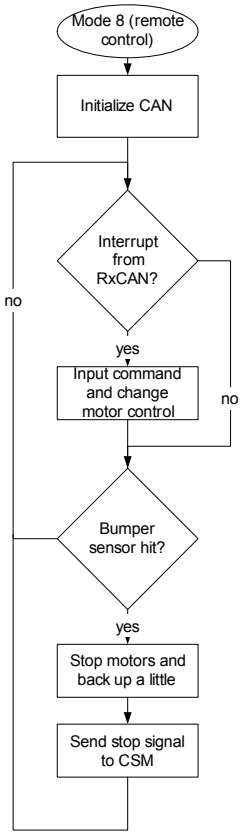
The actual byte data that is transferred is not critical on its own, but when every device in the system knows what the value means, it is useful. We have a table of values defining what each command means and what value it corresponds to. This table should be updated as the project gets more complicated. If more than 255 commands or data messages are needed, you can use identifiers in the CAN system or use more than one byte. Our system is so small that one byte was all that was necessary. The final table is as follows:

	Forward	Reverse	Right	Left	Stop
PC Sends/CSM Receives	0x01	0x02	0x03	0x04	all others (0x05 usually)
CSM Sends/Robot Receives	0x66	0x44	0x46	0x64	0x67
	Bumper 1	Bumper 2	Startup	CSM Btn1	CSM Btn2
Robot Sends/CSM Receives	0x01	0x02	0xFF	N/A	N/A
CSM Sends/PC Receives	0x01	0x02	0xFF	0x03	0x62
CSM Sends/Robot Receives	N/A	N/A	N/A	0x64	0x46

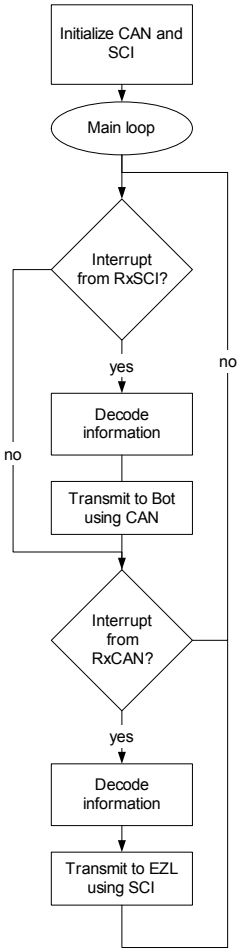
Table 1: Commands and data messages in the entire Wireless ECE Robot data flow

Wireless Robot firmware Flow Charts

Robot code for a new remote control mode



CSM-12C32 code



Carson Drew 12/10/05

Appendix 4. Step-by-step user instructions
Connecting the Wireless ECE Robot – Start to Finish

Throughout this document, I'll be under the assumption that the daughterboard that combines the CSM-12C32, the EZL-80 and PCMCIA socket, and CAN hardware has been completed.

Connecting the Hardware

First, create the connections for the CAN communication between the CSM-12C32 and the Robot. Materials include two CAN Transceivers (MCP2551 from Microchip), the current ECE Robot, a CSM-12C32, and two wires connected into a socket on one end. The CSM-12C32 plugs into the 40-pin socket on the daughterboard. There is a built-in connection from the CAN pins (26 and 28) on the CSM-12C32 to the first CAN Transceiver. A CAN transceiver (the MCP is an 8-pin DIP) must be inserted into each of the 8-pin sockets on the daughterboard. Then, connect the two wires to the robot microcontroller on pins 17 and 18. It is helpful to use a socket to keep this enclosed.

It should be noted that the CSM-12C32 can be connected through the RS-232 connection on the daughterboard, but the design allows for the CSM-12C32 to be connected on the 40-pin header. This requires cutting cut traces 4 and 5, but the 40-pin header handles power, so this is ideal for the project. Power must be wired manually to the CSM-12C32 if it is connected using RS-232.

Second, you must insert the EZL-80 into the two 20-pin headers. Make sure that the side of the EZL-80 that is labeled 'JP1' is inserted into the header labeled 'JP1'. If not, you may have the EZL-80 upside-down. The serial connections between the EZL-80 and the CSM-12C32 are already internally wired. The connections between the EZL-80 and the PCMCIA socket are also internally wired. At this point, you can move on to the software setup steps.

Third, you must use four screws to attach the daughterboard to the robot above or below the main robot PC board. Also, power connections must be made according to specifications. 5V and ground must be wired from the 2-pin power header on the robot to the 2-pin power header on the daughterboard.

Software Setup

The first thing to do is to use ezSerialConfig to configure the EZL-80. It is possible to connect a serial cable directly from the computer running ezSerialConfig to the serial port on the daughterboard. Once this is done, set the communication port to the one you are using on your computer (COM[1-4]). Then press 'Read'. If there is no response, there is an error in hardware connections.

The program will display the MAC address of the EZL-80 when a connection is established and many textboxes will become available. By setting these and then writing

them to the EZL-80, you are configuring the EZL-80. Set the EZL-80 to COD mode in the drop-down list ezTCP Mode, Set 'Conn. Byte' to 1, set the Peer IP and Peer Port to your computer's IP and desired port. Make sure Parity is set to 'NONE' with 8 data bits. Baudrate should be set to 9600, and flow control should be disabled. The last thing to do is to set the Target SSID. This is the name of the network that you want to use for the connection. We used RFconnect, the campus network. If the EZL-80s are newly ordered, the MAC addresses will not be on file with ITC. They may not work, so you should call Bob Underkofler at ITC (x5100) to help with that. Note that the PC running RobotCommandCenter must be on the same network that the EZL-80 is configured to.

Starting the Connection

Once the EZL-80 has been configured, start RobotCommandCenter on the networked PC. For COD mode, you simply have to press the 'Listen Mode (for COD mode)' button. The robot will request a connection when powered up for the first time. To establish this connection, you must power on the systems by flipping the switch on the robot. The robot will go through initialization that lasts around 10 seconds. This is enough time for the EZL-80 to get set up on the network. This initialization does not do anything for this project, it is just remaining code from Dr. Maher's basic class setup. It can be removed, but it may not allow enough time for setup of the EZL-80. If a connection is not established after 15 seconds or so, you may just press a bumper on the robot and it will send a different byte and establish a connection. Then, send and receive commands using RobotCommandCenter.

Appendix 5. Financial Report

Budget Summary

4W0445 Wolff, Richard — Reconciled 04/10/06 — FY 2006
Wireless Mobile Robots: A Tool for Undergraduate E — End 04/14/06 — G&C
-

Account	Description	Budget	Spent	Outstanding	Remaining
Expense					
61123	Contract Faculty	5,400.00	0.00	0.00	5,400.00
61130	Summer Salaries	0.00	2,400.00	0.00	(2,400.00)
61224	Temp Hrly	0.00	1,704.00	0.00	(1,704.00)
61225	Student Labor	0.00	1,448.00	0.00	(1,448.00)
	Total Acct Type: Salaries	5,400.00	5,552.00	0.00	(152.00)
	Total Salary	5,400.00	5,552.00	0.00	(152.00)
61499	Benefits - General	690.00	600.39	0.00	89.61
	Total Acct Type: Benefits	690.00	600.39	0.00	89.61
	Total Benefits	690.00	600.39	0.00	89.61
	Total Personnel Services	6,090.00	6,152.39	0.00	(62.39)
62208	Laboratory Supplies	0.00	2,448.33	0.00	(2,448.33)
62299	General Sup/Materi	3,910.00	0.00	0.00	3,910.00
	Total Acct Type: Supplies	3,910.00	2,448.33	0.00	1,461.67
	Total Operations	3,910.00	2,448.33	0.00	1,461.67
	TOTAL EXPENSE	10,000.00	8,600.72	0.00	1,399.28
	BALANCE	10,000.00	8,600.72	0.00	1,399.28

Additional materials purchased (not included in above budget summary):

Item	cost
4/7/06 BALANCE	$1,399.28
DATA FLOW SYSTEMS	$206.80
DIGI-KEY CORP	$366.63
DIGI-KEY CORP	$420.99
PCB EXPRESS	$404.86
FINAL BALANCE	$0.00

Budget Detail

4W0445 Wolff, Richard Reconciled 04/10/06 FY 2006
Wireless Mobile Robots: A Tool for U End 04/14/06 G&C

			Amount	Encumbered	Clr	DPO
Contract Faculty - 61123						
07/18/05 - 1625	Carry Over for FY 2006	Original budget	-5,400.00	0.00	99	
Total Contract Faculty - 61123			-5,400.00	0.00		
Summer Salaries - 61130						
09/09/05 - 1630	Banner Salary	Wolff, Richard 9.9.2005 9 F0042475	2,400.00	0.00	9	
Total Summer Salaries - 61130			2,400.00	0.00		
Temp Hrly - 61224						
07/18/05 - 1622	Carry Over for FY 2006	Carryover Balance	672.00	0.00	13	
08/11/05 - 1628	Banner Salary	Benjamin, Bradley 8.9.2005 8 F0041781	1,032.00	0.00	8	
Total Temp Hrly - 61224			1,704.00	0.00		
Student Labor - 61225						
07/18/05 - 1623	Carry Over for FY 2006	Carryover Balance	184.00	0.00	13	
09/09/05 - 1631	Banner Salary	Benjamin, Bradley 9.9.2005 9 F0042475	1,264.00	0.00	9	
Total Student Labor - 61225			1,448.00	0.00		
Benefits - General - 61499						
07/18/05 - 1626	Carry Over for FY 2006	Original budget	-690.00	0.00	99	
07/18/05 - 1624	Carry Over for FY 2006	Carryover Balance	57.05	0.00	13	
08/11/05 - 1629	Payroll	Benefits 8.2005	86.19	0.00	8	
09/09/05 - 1632	Payroll	Benefits 9.2005	409.15	0.00	9	
09/14/05 - 1514	Banner External Feeds	Term Pool Benefits	48.00	0.00	9	
Total Benefits - General - 61499			-89.61	0.00		
Laboratory Supplies - 62208						
07/07/05 - 463	Banner External Feeds	P>ZANTHIC TECHNOLOGIES-Lab supplies	234.76	0.00	7	
07/08/05 - 462	Banner External Feeds	P>DKC*DIGI KEY CORP-Lab supplies	21.76	0.00	7	
08/04/05 - 572	Data Flow Systems, Inc.	DATA FLOW SYSTEMS, INC-Converter	78.75	0.00	8	
08/11/05 - 975	Banner External Feeds	P>DKC*DIGI KEY CORP-Mem cards	38.90	0.00	8	
10/13/05 - 2148	Banner External Feeds	P>DKC*DIGI KEY -Research lab supplies	42.35	0.00	10	
10/14/05 - 2185	Banner External Feeds	P>SUNSTONE PCB-Wireless robot parts	140.00	0.00	10	
10/22/05 - 2300	Banner External Feeds	P>SUPERDROID-Type1 Serial Converter	43.15	0.00	11	
11/01/05 - 2398	Banner External Feeds	P>PAYPAL *ARIDZ Research lab supplies	34.97	0.00	11	
11/14/05 - 2489	Richard S. Wolff	WOLFF, RICHARD-PayPal Purchase	44.99	0.00	12	
01/24/06 - 3502	Banner External Feeds	P>ZANTHIC TECHN Research lab supply	267.39	0.00	1	
01/24/06 - 3512	Banner External Feeds	P>SUNSTONE PCB Research lab suppl	175.00	0.00	1	
03/30/06 - 4583	Banner External Feeds	P>AXIOM MANUFAC Research lab suppl	1,075.00	0.00	4	
04/04/06 - 4615	Banner External Feeds	P>MICROCOM TECH Research lab suppl	251.31	0.00	4	
Total Laboratory Supplies - 62208			2,448.33	0.00		
General Sup/Materi - 62299						
07/18/05 - 1627	Carry Over for FY 2006	Original budget	-3,910.00	0.00	99	
Total General Sup/Materi - 62299			-3,910.00	0.00		
Expense			-1,399.28	0.00		
Balance			-1,399.28	0.00		

Item	cost
4/7/06 BALANCE	$1,399.28
DATA FLOW SYSTEMS	$206.80
DIGI-KEY CORP	$366.63
DIGI-KEY CORP	$420.99
PCB EXPRESS	$404.86
FINAL BALANCE	$0.00