Implementing CANopen in Autonomous Underwater Vehicles

Elizabeth Yin, Stanford University
Mentors: Mark Sibenac, Bill Kirkwood
Summer 2003

Keywords: CAN, CANopen, AUV, CC01, Controlled Area Network

ABSTRACT

The current communications system used for operating data collecting devices on the AUVs at MBARI are based on RS-232 and ethernet connection. While this system is functional, the network is incredibly complicated, as each data collecting device is directly connected to the main computer, which can potentially cause a myriad of cabling problems. The computer sends requests for data from all the devices over each individual cable connected to each device; once it receives data, it subsequently processes it. In addition to the communications being inefficient, because of the complicated cabling, it becomes important to strategize where each device ought to be placed within an AUV. However, this can be problematic when trying to add new devices to the AUV, often causing all components of the network to be re-shuffled within the vehicle. These complications could be avoided entirely if the Communications Area Network (CAN) system replaced the current AUV communications system. The CAN system utilizes a "plug-and-play" model, which allows each device to simply plug into a bus, called a CANbus, which connects to the computer. All devices plugged into the CANbus receive the same message from the computer simultaneously along the bus. As a result, it is not necessary to strategically place a device in the vehicle, as long as it is placed closed enough to the CANbus. Such a communications system makes the vehicle more organized, and in addition to generally being a better communications system, the network can also mitigate the risks of other problems that typically plague underwater vehicles, namely leaks and intense pressure.

INTRODUCTION

With the development of technology, marine scientists can discover the ocean environment in a less expensive and more efficient manner. At advanced research centers, such as the Monterey Bay Aquarium Research Institute, autonomous underwater vehicles (AUVs) have become a great resource through which scientists can collect data. An ideal AUV requires no human support and would be able to explore the seas, navigating, sending data back to the laboratories via satellite, and recharging itself. Such an autonomous machine would reduce the cost of labor and the boating expenses that are currently necessary for transporting scientists on the ocean. Because AUVs are a relatively recent development, they have not yet reached their fullest technological potential, as they still require initial deployment and retrieval assistance and cannot travel long distances due to energy storage limitations. However, in the future, AUVs will be able to collect much more ocean data than any human team at a significantly lower cost.

Although artificial intelligence is progressing rapidly, robotic technology is still a fairly new field. In addition, like other underwater robots, such as remotely operated vehicles (ROVs), AUVs must also contend with the difficulties of operating underwater. The controls system must overcome choppy waves, harsh currents, and swells; the entire robot itself must be tolerant to pressure and must be leak-proof.

In focusing solely upon underwater-based problems, pressure and potential leaks are significant problems that slow down the development of AUVs. Pressure increases linearly with depth, as the total pressure is approximately 1.5 times the depth in meters. Thus, in building underwater robots, it is critical that all components must withstand extreme pressure levels, often on the order of 10k psi, equivalent to approximately a depth of 6670 meters. Depending on the AUV, the most advance AUV can probably explore a maximum depth of approximately 6000 meters. To cope with these drastic conditions, parts must themselves be housed in an epoxy-based mold, or housed in another pressure resistant material, such as glass or titanium. To test whether a housing is resistant to pressure changes, each component often undergoes testing in a pressure chamber on land.

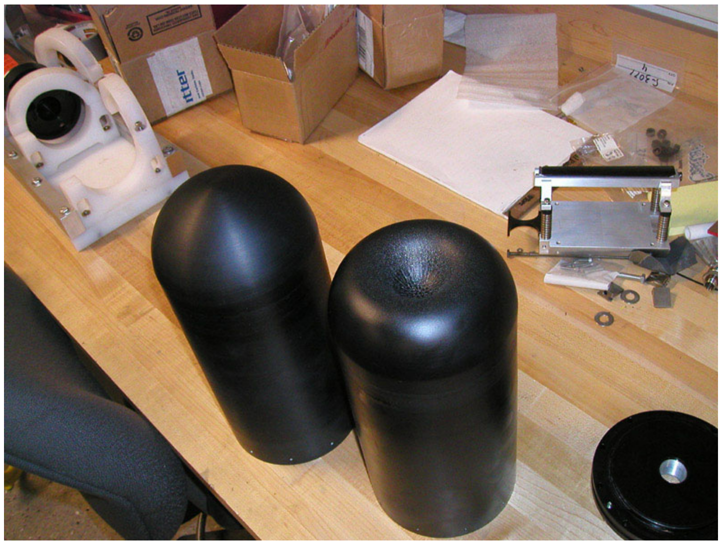


Figure 1 Originally Identical Aluminum Objects; the right one failed pressure tests

As shown in Figure 1, the results of pressure tests are often visually apparent. Though originally identical, these two aluminum objects cannot be used in any AUV, as the one on the right underwent and failed the pressure tests. However, sometimes, the results of such tests are not visible to the naked eye. In particular, quartz crystal oscillating clocks are prone to failing the pressure tests, because an air pocket must exist within the clock, to allow the crystal to oscillate. Thus, the pressure often causes the component to collapse onto itself.

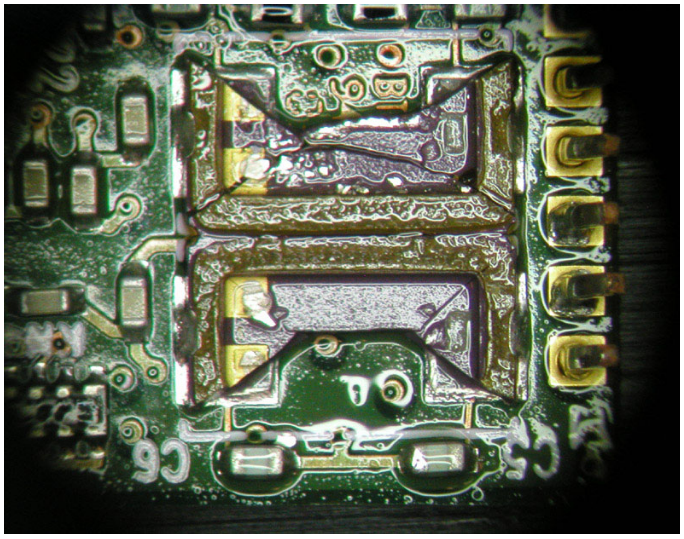


Figure 2 A Cracked Oscillating Crystal

Figure 2 is a photograph taken through a microscope, which allows one to clearly see that the oscillating crystal of this chip cracked and broke apart. This failed test only became noticeable after removing several soldered chips from a circuit board and after pulling tiny components apart.

Once the AUV is solely comprised of pressure tolerant components, mitigation the number of leaks is another difficulty. Contrary to popular belief, the AUV takes on water; thus, all components housed within must be waterproof, and the cable connectors for communications and power must form water tight seals. Leaks cause corrosion and electrical shorts within the AUV's electronics, and because a given AUV may house a myriad of cables, finding a leaky connector is next-to-impossible. Ultimately, using fewer connectors would significantly reduce the risk of leaks.

The current communications system is based on RS-232 serial and ethernet connectors, a fairly standard network. However, the cabling for this system ca often become muddled, as each component connects directly to others, as indicated in the following schematic.

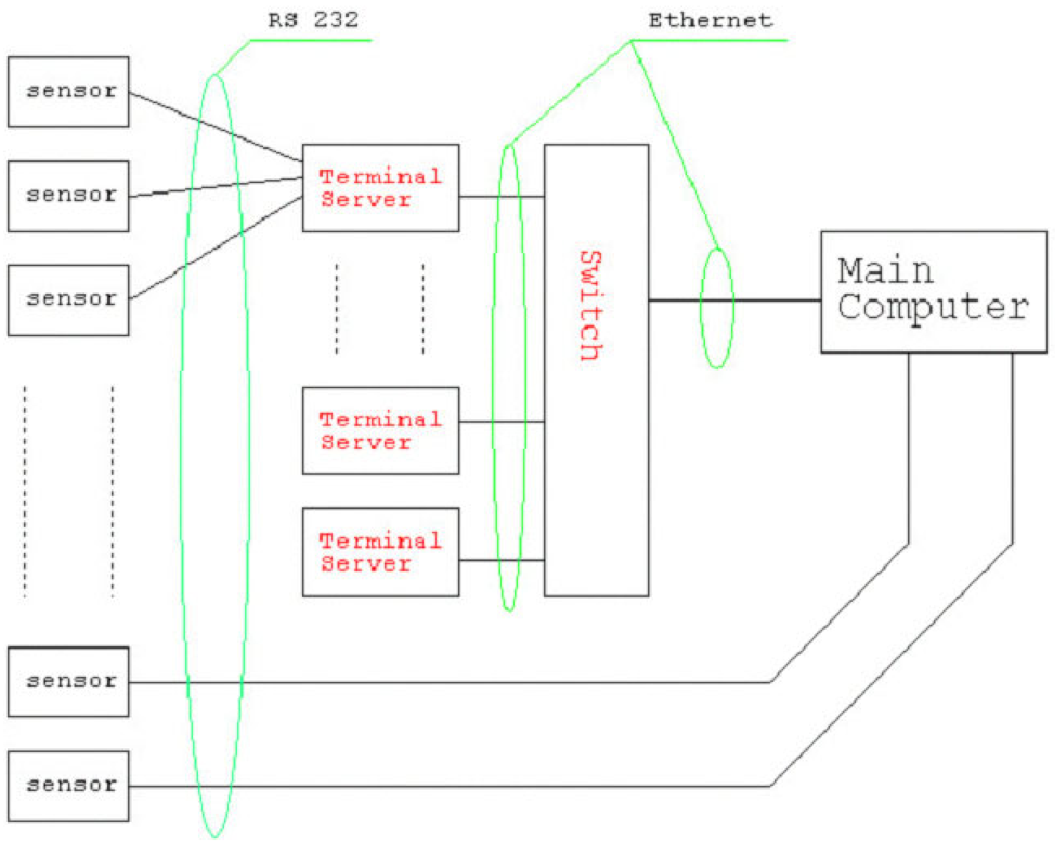


Figure 3 Current ethernet & RS-232 communications system

Because each component has its own line of communication with the main computer, strategic placement of a given component is critical. This becomes particularly problematic when one tries to add new devices to the network later. Often reorganization of the devices onboard is required in order to add new components, and this often leads to really convoluted cabling, as parts need to be disconnected, moved, and then reconnected. Because leaks are particularly prevalent around connectors, when parts are disconnected, moved, and reconnected, the chance of leakage in an AUV increases. It would also be ideal to have a communications system which has parts that are inherently pressure tolerant without special housing.

Thus, to cope with pressure and avoid leaks, introducing the Communications Area Network (CAN) system seemed like a better solution than the current communications system for reasons explained in the subsequent sections. The CAN system, though widely used in Europe, is hardly found in America. It has numerous salient qualities, which include a relatively low cost and a good error-detection and error-handling system. Additionally, it is fairly resistant to electromagnetic interference and can transmit up to eight byte messages at 1 Mbit per second. Most importantly, however, messages can be sent to multiple devices at the same time.

This project focuses on developing a prototype of the CAN system, which requires programming drivers for various data collection tools such as a Sailcomp 103AC Digital

Compass, a Datasonics PSA-916 Sonar (altimeter), a Parosci 8CB Pressure Sensor, and a Seabird FastCat CTD. This prototype demonstrates the efficiency and value of the CAN system, and if successful, in the near future, perhaps this system can replace the current communications system on the AUVs.

MATERIALS AND METHODS

In order to set up a working CAN system prototype, this project was broken down into several parts and milestones. The following resources were used for the execution of this project: Phycore T89C51CC01 Microcontroller (CC01), ATMEL Flip, CANCARD X, PCANOpen Magic, CANSetter and the listed data collecting tools mentioned in the introduction. It is assumed that the reader of this report is familiar with the general functionality of these tools.

OVERVIEW OF THE CAN SYSTEM

The basic premise of the CAN system is that all devices are relatively independent and are connected to a bus, called a CANbus, which is attached to the main computer. When a message is transmitted from the computer across the CANbus, all the connected devices receive this message instantaneously. Thereafter, because each device has its own CAN chip, it's own microcontroller, and accompanying driver software, it can process its own data itself; in the current system, the main computer has to process all the data of each of the devices connected to the network. Thus, the CAN system is highly organized, as shown in the schematic below, and very efficient.

Devices can attach to a microcontroller if it they have an RS-232 serial connector. The computer transmits messages along the CANbus to each CC01 microcontroller of all the devices along specific unique nodes. Each microcontroller looks to see if a device driver is connected to its assigned node. if the CC01 can connect with a device along its appropriate node, then a message can be transmitted to the device along that path, and a response from the device driver will be sent via the CC01 back to the computer along that same node. From a software perspective, code called a CANbootloader resides on the CC01. The bootloader serves as the software that operates the microcontroller. Individual driver programs for the devices connected to the serial port on the network then get programmed into the flash memory of the CAN chip via the bootloader. The bootloader code must first be uploaded onto the CC01 via the computer. Next, the driver code for the compass is then uploaded onto the flash memory via the bootloader. When a sync is emitted from the computer, the bootloader receives the sync, interfaces with the driver code, and then the driver code instructs the bootloader to return a response to the computer. The bootloader code serves as the communicator for the driver code and does not need to be specially configured for each device and accompanying microcontroller.

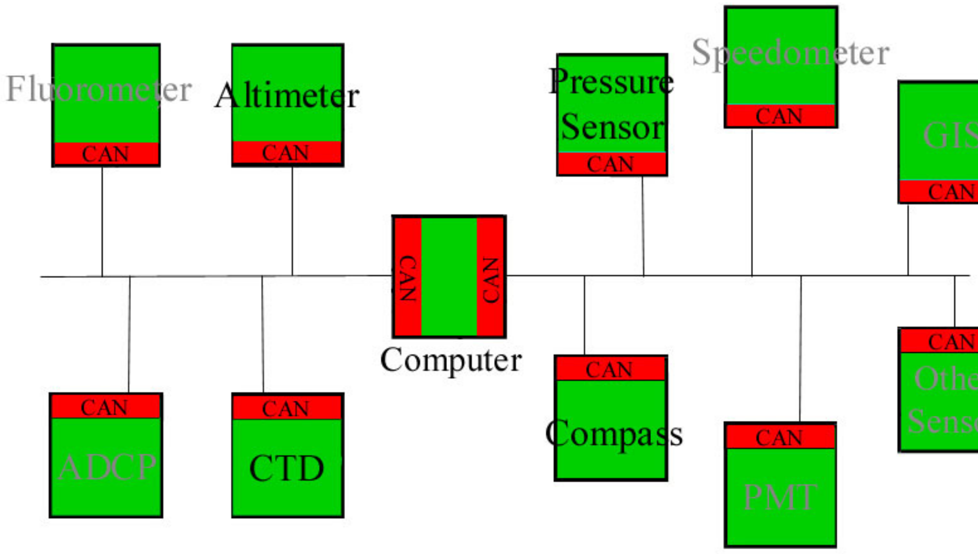


Figure 4 Proposed Communications Area Network System

THE PROCEDURE

To produce a working prototype, this project was divided into two parts—the hardware and software. Telecommunications engineering student Jean-Christophe Bouvier of the Institut des Sciences de l'Ingenieur de Toulon et du Var developed the hardware for the CAN prototype, while I wrote the software drivers for various devices. Integrating the two parts towards the end of the summer allowed for a successful prototype of the CAN system. This paper, however, will focus primarily on the details of the software development.

The first step was to successfully use the Keil CC01 simulator, which could act like a CC01 microcontroller, allowing one to test driver software on the simulator without actually using the CC01 chip or CAN. Once "Hello World," a standard testing program worked on the simulator, the next step was to put that code on the actual CC01 chip. Because "Hello world" does not require any communicating capabilities, it was not necessary to upload and utilize the CAN bootloader. To upload code onto the microcontroller, ATMEL Flip was used. (See documentation for details) There were initial difficulties in uploading "Hello World" with Flip. Ultimately, the problem lay in the incorrect setup of the Flip software. After re-installing it fully and correctly, the CC01 could be programmed, and "Hello World" ran successfully off the microcontroller. Other programs such as Microsoft Flash Tools can sometimes be used to program microcontrollers. However, in this case, ATMEL Flip proved to be the better option; it was easier to use, and FlashTools often has special restrictions that must be heeded in order to operate properly. To use ATMEL Flip, the computer was connected to the CC01 via the Peak Dongle, not with CANCARD X. Both, however, work equally well and connect via the parallel port. After "Hello World" successfully proved itself, driver software could now be written. The first written driver software for an electronic compass. The program would continuously read in the outputted strings of the compass, and the driver software would parse and return the compass heading. Using HyperTerminal on the computer, the compass was tested for functionality first; the DB-9 connector plugged into the serial port of the computer for this test. Thereafter, the completed driver was tested with the compass on the Keil CC01 simulator with the compass still attached directly to the computer. After debugging and refining the driver code, it was time to write driver software for the other data collecting tools onboard the AUV. After all driver software was operational using the Keil Compiler, which mimicked the CC01, it was time to try placing this software on the actual CC01 chip using the CAN software. However, it was first important to test the CAN system with a simple piece of software. Thus, once again, "Hello World" was utilized; first, the CANbootloader was uploaded onto the chip using ATMEL Flip, and then "Hello World" was to be uploaded onto the flash memory of the CC01 using the bootloader in PCANOpen Magic or CANSetter. However, several difficulties impeded the success of the latter. After days of troubleshooting, it appeared that the CAN chip in use was not compatible with the CANbootloader code. Special kudos go to my mentor Mark Sibenac, who edited the CANbootloader code so that it was compatible with the chip. Thereafter, both PCANOpen Magic and CANSetter worked equally well. (see appendices for detailed analysis). Thus, "Hello World," could be uploaded via the bootloader code, and it worked! However, prior to this, special CAN-related commands were inserted into the "Hello World" program, and this extra code was copied directly from former MBARI Summer Intern Rado Bortel's CAN programs, written in 2002. (see programs) All the driver software written for the AUV data collection tools was meshed with Rado's CAN-related code, allowing all devices to successfully operate on a CAN system.

THE RELATED CAN HARDWARE

Several boards, replicating the CAN hardware were created, so that each device would connect to a serial port on a separate board. With multiple boards, in theory, it was possible for all of the devices to connect to the CAN system simultaneously, making our CAN system prototype quite accurate to how the system would be implemented in the AUVs. All of this hardware created by Jean-Christophe underwent testing in the pressure chamber on August 14, 2003 in the morning, to see if the CAN hardware could withstand high pressures without any special housing or epoxy.

RESULTS

Indeed, the CAN system prototype—including the hardware and the software drivers--consisting of an electronic compass, an altimeter, a pressure sensor, and a CTD, was accomplished. All the data tools communicate successfully over the CAN system.

According the Rado Bortel, the CAN hardware was expected to tolerate high pressures, and this prediction was essentially true. The CAN chip and related hardware underwent testing in the pressure chamber, and overall, it survived up to approximately 6400psi. However, once past 6400psi, the CAN system was inhibited unless the pressure was increased very slowly. In fact, the CAN prototype would continue to work, even past

10,000 psi if the pressure increased no faster than 400 psi per minute. If the pressure was increased more quickly, the CAN system would product errors, though not break.

DISCUSSION

With regards to lessening the likelihood of leaks and dealing with pressure issues, the CAN system is somewhat a better choice than the current ethernet-based communications system. The buses involved in the CAN system will dramatically simplify the cabling and perhaps reduce the chance of leakage. However, it appears that Bortel and perhaps mentor Sibenac had hoped the CAN hardware would require no special housing or epoxy for the CAN hardware. However, based upon the results of the tests conducted in the pressure chamber, it appears that it is necessary to encase the CAN hardware in epoxy, in case an AUV with the CAN system descends rapidly.

After the procedures of this project were analyzed, it appears that almost 70% of the time spent on this project involved re-discovering how to use CAN software or re-discovering how the software interacts with the hardware. Additionally, the time spent in determining that the CANbootloader was not compatible with the newest CAN chip was of long duration. In fact, if one used the CAN software, such as PCANopen Magic or CANsetter, on a regular basis, most likely programming all the drivers and writing the necessary documentation would require approximately two weeks. CAN is a relatively un-user friendly system with little documentation, but once one has discovered how it works, it is relatively simple to use. Thus, to increase the efficiency of completing future projects involving the CAN system, extensive documentation on CAN has been provided in the appendices.

CONCLUSIONS & RECOMMENDATIONS

A new communications system for the AUV will most likely improve troubleshooting efficiency during AUV operations and should reduce the number of problems that typically arise. The CAN system is an organized, clean, “plug & play” method for attaching data tools to the AUV computer. The device tools can be controlled by the AUV’s computer in a more synchronized, organized fashion and will not require crazy inefficient wiring schematics. This project has proven that the basic CAN system is practical and feasible and is a good investment to improve MBARI’s AUVs. Initial work will be required to create hardware and program drivers for all data collecting tools on the AUV, and the resources required to swap the current communications network with the CAN system will be somewhat tedious. However, in the long-run, this initial investment pales in comparison to the expected amount of drained resources in troubleshooting AUV problems. It is recommended that the CAN system be implemented in one of MBARI’s AUVs for further evaluation of the network’s effectiveness and value. If few or no resources can be spared for converting this system, it is believed that an intern studying electrical engineering and/or computer science can accomplish this as a summer project in 2004.

ACKNOWLEDGEMENTS

I would like to specially thank my mentor Mark Sibenac and Bill Kirkwood for all the fantastic support and help on this project. Furthermore, Mark spent so much time helping me troubleshoot and many late hours trying to set up much of the complicated CAN software required for this project. I would also like to specially thank George Matsumoto for a terrific internship program and for taking care of everything for the interns. The MBARI internship program has given me so many opportunities to learn and explore various aspects of marine science, the Monterey Bay area, and meet numerous kind-hearted people. Other kudos go to Jean-Christophe Bouvier for complementing my part of the project with the CAN hardware and for taking digital photos of everything. Also, kudos go to Rado Bortel, MBARI intern 2002, for his already written CAN compatible software.

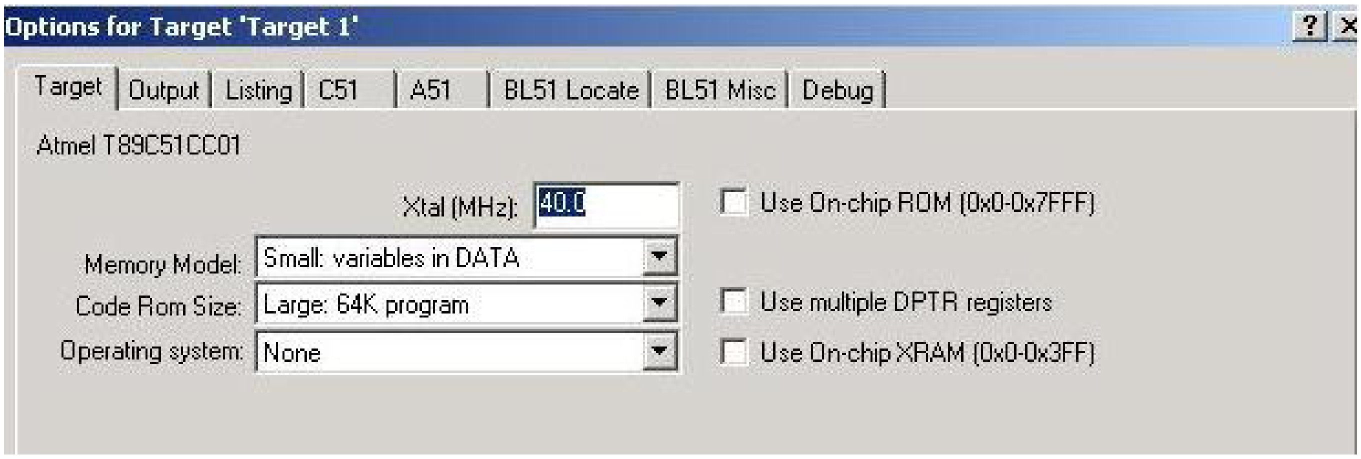
APPENDICES

DOCUMENTATION
This documentation section should hopefully help with using the CAN software in the future. There are a lot of unintuitive settings for many of the programs, so this section is fairly crucial to this report.

<u>Keil Compiler for CC01</u>
Overall, the Keil Compiler is incredibly straightforward and easy to use, like any other compiler. There are a few parameters that need to be set when first creating a project.

-Go to "Project"->"Select Device for Target 'Target 1.'"
-On the left column, you'll see a series of folders. Choose "T89C51CC01."

-Go to "Project"->Options for Target 'Target 1.'"
-Make sure that the following settings look like the following:

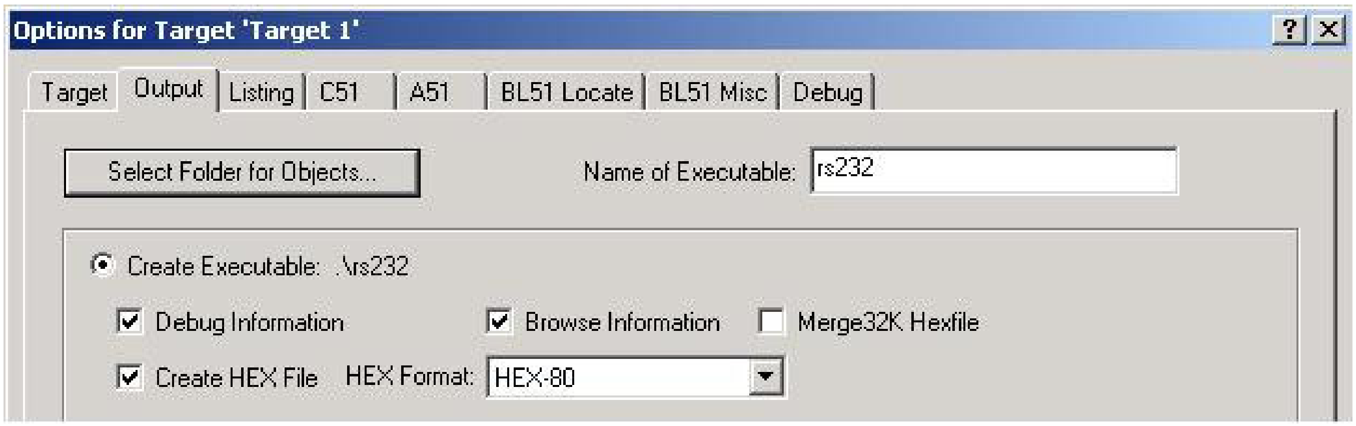


Atmel T89C51CC01

Xtal (MHz): 40.0
Memory Model: Small: variables in DATA
Code Rom Size: Large: 64K program
Operating system: None
☐ Use On-chip ROM (0x0-0x7FFF)
☐ Use multiple DPTR registers
☐ Use On-chip XRAM (0x0-0x3FF)

Figure 5 View of Keil Compiler Options, Target

-Then, click on the "Output" tab, and make sure that the settings look like the following:



Name of Executable: rs232

Create Executable: .\rs232

[✓] Debug Information [✓] Browse Information [ ] Merge32K Hexfile

[✓] Create HEX File HEX Format: HEX-80

Figure 6 View of Keil Compiler Options, Output

-All other features directly related to the compilation features can be found in the "Debug," "Project," and "View" columns in the main menu. These are very straightforward and easy to figure out.

-With regards to streaming I/O through the serial port, the following two commands need to be typed into the prompt when viewing the serial window. (The serial window can be found in the "View" category.)

```
>MODE COM1, 4800, 0, 8, 1
>ASSIGN COM1 <SIN >SOUT
```

The first line opens up com1 at 4800 baud. To change the baud, replace 4800 with a different value. The second line allows streaming data to enter com1 and also allows the user to send data to be sent out the serial port using the printf command.

Phycore CC01 (Hardware Manual and QuickStart)

-According to section 3.1 of the Hardware Phycore CC01 Manual, Flip is used when the jumpers for J1 are on pins 2 & 3. Make sure the jumpers are on the correct pins.

ATMEL Flip

Loading the bootloader

Using ATMEL's FLIP 1.8.8, one can program the CAN bootloader onto the CC01 chip. Under "Device"->"Select," the correct device can be chosen, namely the T89C51CC01. Thereafter, select "Settings"->"Communication"->"CAN"->PEAK PCAN Dongle. The CAN system is implemented via the parallel port of the computer, which is connected to the CC01. Click "Init," then reset the microcontroller by pressing the bootloader button first followed by the reset button. Release the buttons in the opposite order. Thereafter, press the "Node Connect" button. If you receive a "Time Out" message, try pressing the bootloader and reset buttons again in the correct order but hold down the buttons for a longer duration of time. Sometimes, it takes several tries to press the two buttons correctly, even after knowing how to press them. Once you successfully connect to ATMEL, program the bootloader code onto the chip by selecting "File"->"Load Hex" and then choosing the code entitled “cobootloader.hex.” **It is important to note that this file was edited from the original cobootloader.hex file, because the original bootloader file was not compatible with the CAN chip.** Thus, finding the **correct** cobootloader.hex file is imperative.

Executing the bootloader with ATMEL Flip
Once the bootloader hex file has been chosen, click all the checkboxes that read “Erase,” “Blank Check,” “Program,” and “Verify.” Make sure that the BLJB checkbox on the right side of the window is not clicked. If it is, unmark it and click the “Set” button to save the changes. Thereafter, click on the “Run” button, and hopefully the program will successfully load the hex file onto the chip. Assuming that this is successful, you no longer need Flip to load additional programs when programming onto the CANbootloader..

PCANopen Magic
Loading programs onto the bootloader with PCANopen Magic
Settings for PCANopen magic include the following:

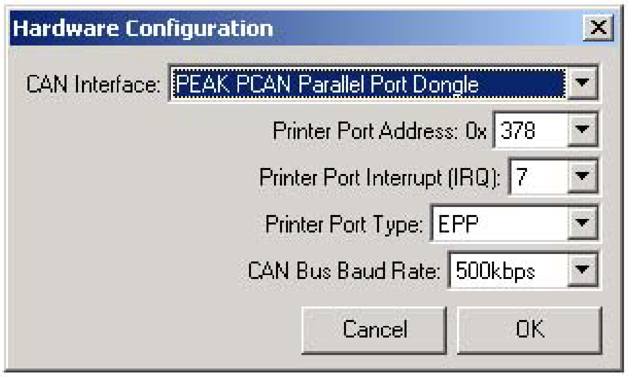
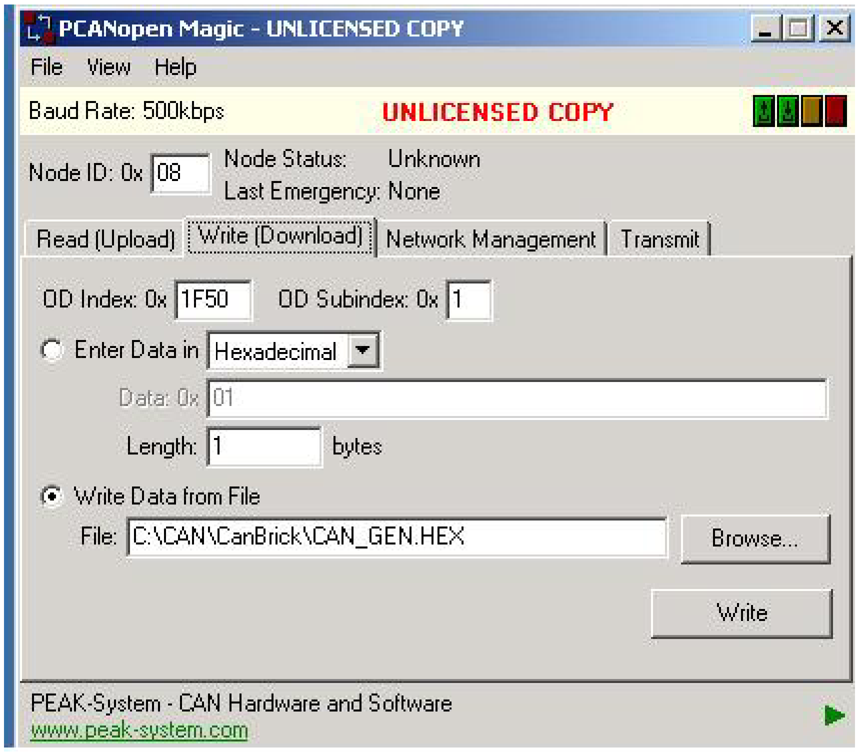

Hardware Configuration
CAN Interface: PEAK PCAN Parallel Port Dongle
Printer Port Address: 0x 378
Printer Port Interrupt (IRQ): 7
Printer Port Type: EPP
CAN Bus Baud Rate: 500kbps

Figure 7 View of PCANopen Magic Settings

Once the correct parameters have been set, one can begin reading and writing programs to the flash memory of the CC01 using the bootloader. Unfortunately, PCAN open magic does not have an object dictionary unlike CANsetter. Both CANsetter and PCAN open magic are programs that will allow you to program the CC01 using the bootloader. The sole advantage to using PCANopen Magic is that MBARI has a special connection to Vector’s Chris Kadel, the author of PCAN open magic, who works in Vector’s San Jose office.

Once you’re in the main menu, you will see a screen much like this:


PCANopen Magic - UNLICENSED COPY

File View Help

Baud Rate: 500kbps UNLICENSED COPY

Node ID: 0x 08 Node Status: Unknown
Last Emergency: None

Read (Upload) | Write (Download) | Network Management | Transmit

OD Index: 0x 1F50 OD Subindex: 0x 1

( ) Enter Data in Hexadecimal

Data: 0x 01

Length: 1 bytes

(•) Write Data from File

File: C:\CAN\CanBrick\CAN_GEN.HEX

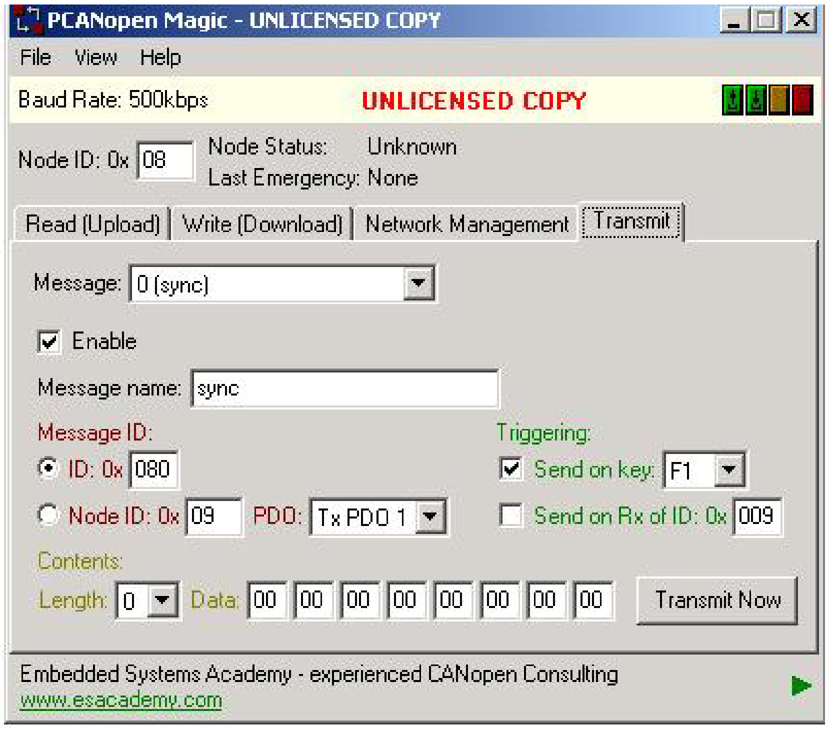
Figure 8 View of PCANopen Magic Main Menu

This program is incredibly complicated to use, but once you know how to use it, everything will work smoothly. To check what node you are connecting to, go to "View"->"Network Overview." Click the "Scan Network" button, and the window should show which node is currently in use. However, if nothing shows up, then there is a problem in finding a node, with which to connect; this suggests that something in your code or bootloader code is incorrect.

To view the action, go to "View"->"Trace." This will help you debug in PCANopen Magic; I guarantee it.

If you connect to a node, make sure that the same number appears in the "Node ID: 0x ___" section on the main window (as shown above as 08 in this example). This will allow you to communicate on the same node. The bootloader code operates in node 08. In order to program files onto the flash memory using the bootloader code, you must make sure that you are operating in node 08. Once you are in node 08, click on the "Write (Download)" tab, and click on the radio button that reads "Write Data from File." Make sure that the file you desire to write onto the CC01 memory is specified with the correct path. To write this file, make sure that the "OD Index: 0x ___" reads "1F50" and the "OD Subindex: 0x __" reads "1." Click on the "Write" button, and you should have successfully written your file. Press the reset button on the CC01; sometimes you need to press the reset button multiple times and rescan the the network. A different node should be written into the window this time. If this is not the case, there is most likely something wrong with your code. Usually, if there is an error with PCANopen Magic, it will not even perform the desired action and will time out instead. If you decide to program a different piece of code onto the CC01, you need to connect to the bootloader's node (08), and you can do this by selecting the radio button that reads "Enter Data in Hexadecimal." Make sure the "Data: 0x__" reads "01" and the "Length" is 1 byte. Now type in the new node number that was found during your most recent the scan, and click on the "Write" button. You should now be connected to the node 08 again. At this point, you are back on the "bootloader mode" and can program something else using the bootloader.

I/O with PCANopen Magic
To interact with the devices connected to the CAN system, you will need to transmit syncs from the computer and receive responses from the devices. To do this, make sure that the PCANOpen Magic settings look like the following:


PCANopen Magic - UNLICENSED COPY

Baud Rate: 500kbps
UNLICENSED COPY

Node ID: 0x 08
Node Status: Unknown
Last Emergency: None

Message: 0 (sync)
[✓] Enable
Message name: sync
Message ID:
(●) ID: 0x 080
( ) Node ID: 0x 09 PDO: Tx PDO 1
Triggering:
[✓] Send on key: F1
[ ] Send on Rx of ID: 0x 009
Contents:
Length: 0 Data: 00 00 00 00 00 00 00 00

Embedded Systems Academy - experienced CANopen Consulting
www.esacademy.com

Figure 9 View of PCANopen Magic Transmit Menu

The "Message:" should be set on "0 (sync)." This should be enabled. The "Message name" should be sync. The "Message ID:" should be in "ID" mode, and should read "0x__" where it equals 80. Make sure that the triggering is checked to send out a sync for a given key; if you press that key, then a sync should be transmitted. You can double check this by looking at the traffic shown in the Trace window mentioned above. If your code is working properly, the connected device should return a message in response to your transmission.

SPECIAL CODE SETTINGS

Timers & Baudrates:
There are 3 timers: timer0, timer1, and timer2. Timer 0 is used to generate interrupts, while 1 and 2 are used to generate the baud rate. This documentation will only focus upon timers 1 and 2. More information on timers, baud settings can be found in the General C51 Users Manual (pdf) and detbaudrate.c (code).

To use any of the timers
Use within main() :

```
#ifndef MONITOR51
   SCON  = 0x50;            /* SCON: mode 1, 8-bit UART, enable rcvr      */
   TMOD |= 0x20;            /* TMOD: timer 1, mode 2, 8-bit reload        */

/* If you'd like to use timer 1 to set the baud, use the following commands: */
/* COPY FROM HERE */
   TR1   = 1;               /* TR1:  timer 1 run       */
   TH1   = 245;             /* TH1:  reload value for 4800 baud @ 20MHz   */
/* UNTIL HERE */

/* If you'd like to use timer 2, use the following commands: */
/* COPY FROM HERE */
         TR2 = 1;           /* TR2: timer 2 run */
         RCAP2H = 255;      /* These settings are for 20Mhz, 9600 baud */
         RCAP2L = 191;
         RCLK = 1;
         TCLK = 1;
/* UNTIL HERE */

         TI    = 1;       /* TI:   set TI to send first char of UART    */
#endif
```

Copy this entire piece of code to set the necessary baud rates with the timers. Both timer 1 and timer 2 can be used simultaneously. You can change the desired baud rate by changing the TH1 value for timer 1. You can calculate the new value for the desired baud rate by using the formula listed below. To change the baud rate for timer 2, you must recalculate RCAP2H and RCAP2L. This, too, can be calculated from the formulas below. Values for the baudrates can be adjusted by using the following formulas:

$$\text{Baud Rate} = \frac{K \times \text{Oscillator freq.}}{32 \times 12 \times [256-(TH1)]}$$

if SMOD = 0, then K = 1.

If SMOD = 1, then K = 2. (SMOD is the PCON register). Most of the time the user knows the baud rate and needs to know the reload value for TH1. Therefore, the equation to calculate TH1 can be written as :

$$TH1 = 256 - \frac{K \times \text{Oscillator freq.}}{384 \times \text{baud rate}}$$

TH1 must be integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.

Figure 10 Timer 1 Equations

And if it being clocked internally the baud rate is :

$$\text{Baud Rate} = \frac{\text{Osc. Freq}}{32 \times [65536 - (RCAP2H,\ RCAP2L)]}$$

To obtain the reload value for RCAP2H and RCAP2L the above equation can be written as :

$$RCAP2H,\ RCAP2L = 65536 - \frac{\text{Osc. Freq}}{32 \times \text{Baud rate}}$$

Figure 11 Timer 2 Equations

RCAP2H and RCAP2L are the high and low components of the RCAP byte. For more details on timers and calculating the baud rate, consult the "General C51 Users Manual".

Toggling

Toggling the microcontroller's light is often a good way to indicate whether a program is properly loaded onto the CC01. The toggling code should be placed in main() and is as follows:

```
while (1) {
  P1 ^= 0x01;                    /* Toggle P1.0 each time we print */
}
```

An example of a simple, working code is in the HELLO project. The code itself is called test2.c. IT IS NOT hello.c.

It becomes particularly **important to use the toggling light code when using a CAN chip**. After uploading code onto a CAN chip, printing to the monitor will be impossible.

Modified Libraries & Code

The following files are modified code and libraries within the intsio2 project:

-sio2.c
-main2.c

Assigned Node IDs for the CAN prototype:
-CAN bootloader = 8
-Compass code = 9
-Altimeter = 10
-CTD = 11
-Parosci = 12

CAN commands:
It is helpful to use the CAN-friendly code written by Rado Bortel. If you incorporate his lines of code into your program, you can be assured that your program will be CAN-friendly too. I have taken the liberty of highlighting his CAN-friendly code in red on the first program of the "PROGRAMS" section below. If you copy all the "red" lines, and place them appropriately around your code, there should be no problem.