cobalt
by Econolite

Econolite Connected Vehicle CoProcessor Linux Support Guide—External

Rev. 00

January 2016

ECONOLITE
Group, Inc.

Documentation

Published: January 20, 2016

Trademarks

683-10557-1215

Table of Contents

Glossary

ARM	Advanced RISC (reduced instruction set computing) Machines
ATC	Advanced Traffic Controller
CVCP	Connected Vehicle CoProcessor
DSRC	Dedicated Short-Range Communications
DIN Connector	Deutsches Institut für Normung (German standard) connector
EIA	Electronic Industries Alliance
ENET	Ethernet
ISOGND	Isolated Ground. This references an external (field) +12 VDC rail per Transportation Electrical Equipment Specifications (TEES)
LED	Light-Emitting Diode
MAC	Media Access Controller
MIB	Management Information Base
NTCIP	National Transportation Communication for ITS Protocol (https://ntcip.org)
OID	Object ID (Identifier). An OID appears as a group of characters that allows a server or end user to retrieve an object without needing to know the physical location of the data.
PDU	Protocol Data Unit for snmp/ntcip
PoE	Power over Ethernet
PSE	Power Service Equipment
RSU	Roadside Unit
RTC	Real Time Clock
RTS-CTS	Ready to Send—Clear to Send
SNMP	Simple Network Management Protocol
SO-DIMM	Small Outline - Dual Inline Memory Module
SoM	System on Module. In this document, we refer to the Nitrogen 6X-SoM module Engine Board as "the SoM."
SP1 and SP2	Serial Ports 1 and 2
SRAM	Static Random Access Memory
TEES	Transportation Electrical Equipment Specifications
V2I	Vehicle to Infrastructure

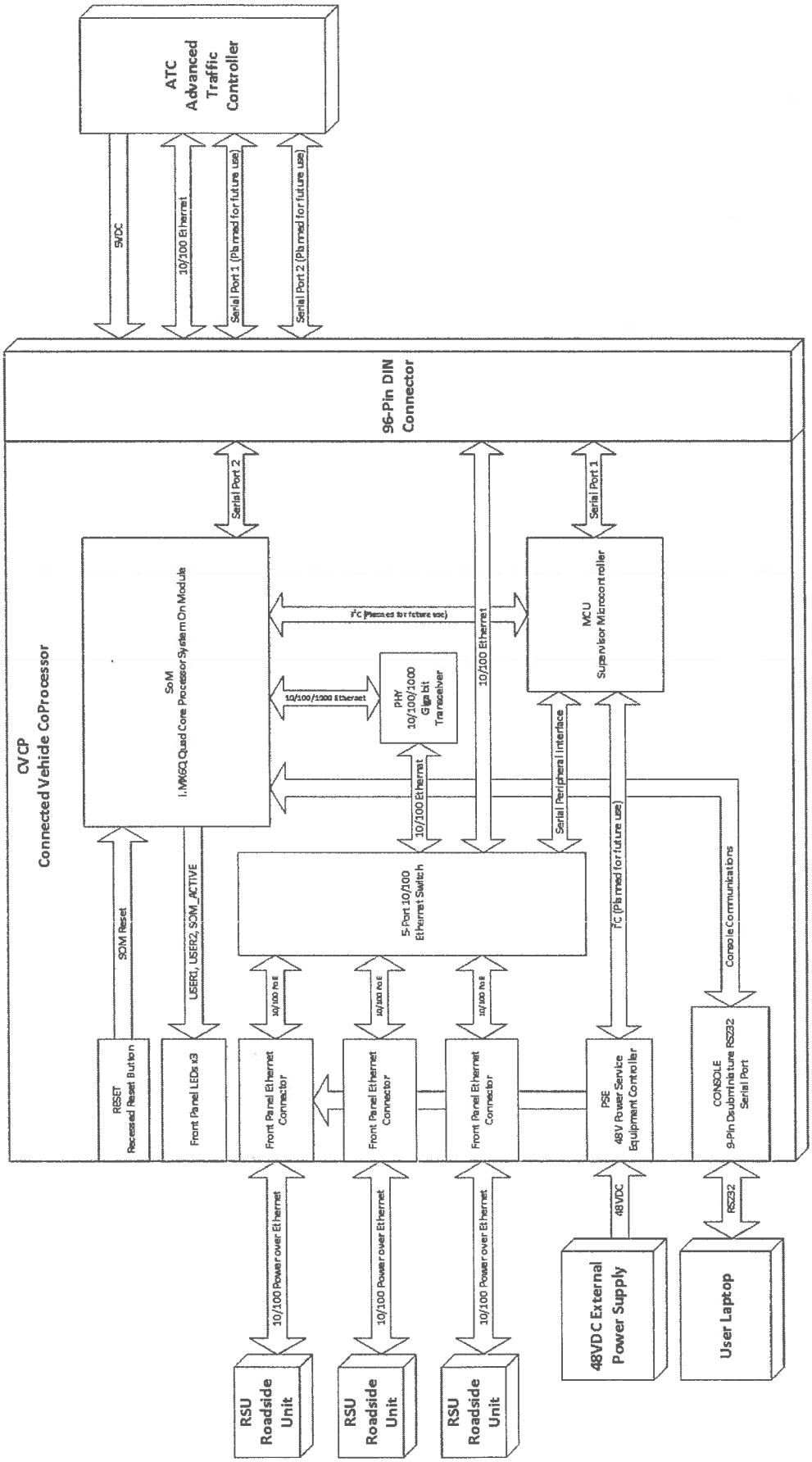
Introduction

This guide describes a Connected Vehicle Co-Processor (CVCP) module intended to allow third-party-developed and processor-intensive applications to be used with an ATC-compliant traffic controller application. On the subsequent page is a block diagram that shows how the CVCP might be used where an application running on the CVCP interfaces with an ATC traffic controller and Dedicated Short-Range Communications (DSRC) Roadside Units (RSUs).

The CVCP runs Linux 3.10.17. To use the CVCP you should be modestly familiar with serial and ssh terminal connections to an embedded product. You should also be familiar with cross compiling on a Linux Host PC and transferring binaries via scp to or from the CVCP.

If you are an advanced user of the CVCP, you should be familiar with remote GDB notions along with strace, top and ps. Also, as an advanced user, you should be conversant with cross-compiling packages including the use of ./configure, make & make install.

This guide is divided into two sections, a Hardware Overview and a Linux Software Overview.



Connected Vehicle CoProcessor (CVCP) Field Application Diagram

Econolite Connected Vehicle CoProcessor Hardware Overview

The CVCP is installed into:

- An A2 communication module slot of a 2070 controller using a 2070-1C CPU module

  - OR -

- The communications slot of an Econolite Cobalt shelf-mount or rack-mount ATC controller

The CVCP uses the Nitrogen 6X-SoM[1] module, from Boundary Devices, which is based on a Freescale, quad core i.MX 6 processor running at a clock rate of up to 1GHz. The i.MX 6 series of application processors can provide a wide array of display and camera type interfaces; however, this design only uses its raw processing power as well as serial and Ethernet communication ports.

The CVCP carrier board includes a 5-port, Ethernet switch. The SoM provides one 10/100/1000 Ethernet communications port. This port is connected to Port 4, the Gigabit Ethernet capable port, of the switch. Port 5 of the switch is routed to the 10/100 Ethernet port of the 96-pin DIN connector facilitating high speed communications between the SoM and the Engine Board of an ATC controller. Ports 1, 2 and 3 of the switch are routed to RJ-45 connectors on the front panel that are capable of Power over Ethernet (PoE). Ports 1 thru 3 include speed and link / active LEDs on the front panel.

The carrier board provides three serial communications ports. Both ATC-specified EIA-485 interface Serial Ports 1 and 2 from the 96-pin DIN connector are used by the module. Serial Port 1 is connected to an optional management microcontroller and Serial Port 2 is connected to the SoM. The carrier board also provides a Linux console port that is isolated then converted to EIA-232 signals and brought to the front panel. The carrier board front panel includes three PoE capable Ethernet ports, one micro-SD card slot, the RS232 Linux console port, an external, 48VDC power input connector, and a recessed, SoM reset button.

To support Power over Ethernet (PoE) requirements of DSRC Road Side Units (RSUs), a Power Service Equipment (PSE) injector and control circuit is implemented on the front panel Ethernet ports. The PSE circuit provides full IEEE 802.3af compliance, however because PoE devices require 48V to operate, you must supply 48V power via a connector on the front panel of the CVCP.

Also, an internal 48 VDC connector is provided to facilitate the use of an internal 48 VDC power supply in stand-alone applications. PoE functionality is provided by the CVCP using a PoE controller that provides IEEE 802.3af compliance without additional software development.

The carrier board includes a microcontroller that facilitates management of the Ethernet switch and PoE controller via the SoM. This microcontroller is for potential future use and switch management is not currently provided.

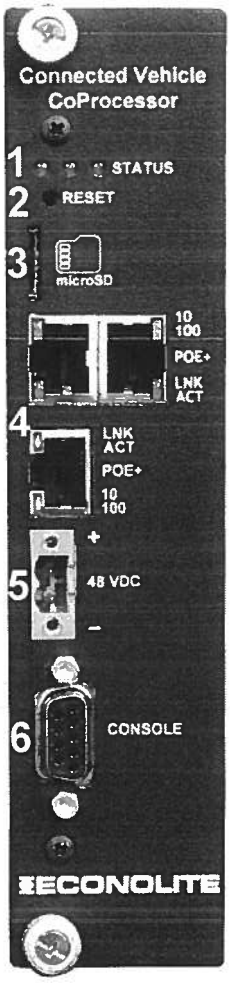
The carrier board includes an additional, internal micro-SD card slot, to house the system boot image, and a real time clock chip powered by the system backup voltage and local super capacitors.

Contents of Connected Vehicle CoProcessor Kit

- CVCP Module (142-1002-501)
- 48 VDC Terminal Block Plug (TE 796859-2)
- Operating System microSD Card (Transcend TS8FMDMEC70IKA)
- CVCP Support Disc (142-1004-501)

---

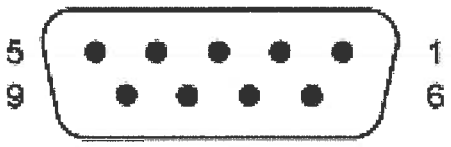
[1] SoM = System on Module. In this document, we refer to the Nitrogen 6X-SoM module Engine Board as "the SoM."



Front Panel Components

1. Front Panel LEDs (The two left-side LEDs are application-controlled[2]. The far-right LED is currently not used.)
2. SoM Reset Button
3. External microSD Card Slot
4. Power Over Ethernet (PoE) Communication Ports
5. External 48VDC Power Supply Connector
6. RS232 SoM Console Port (see below for pinout)

Console Port Pinout

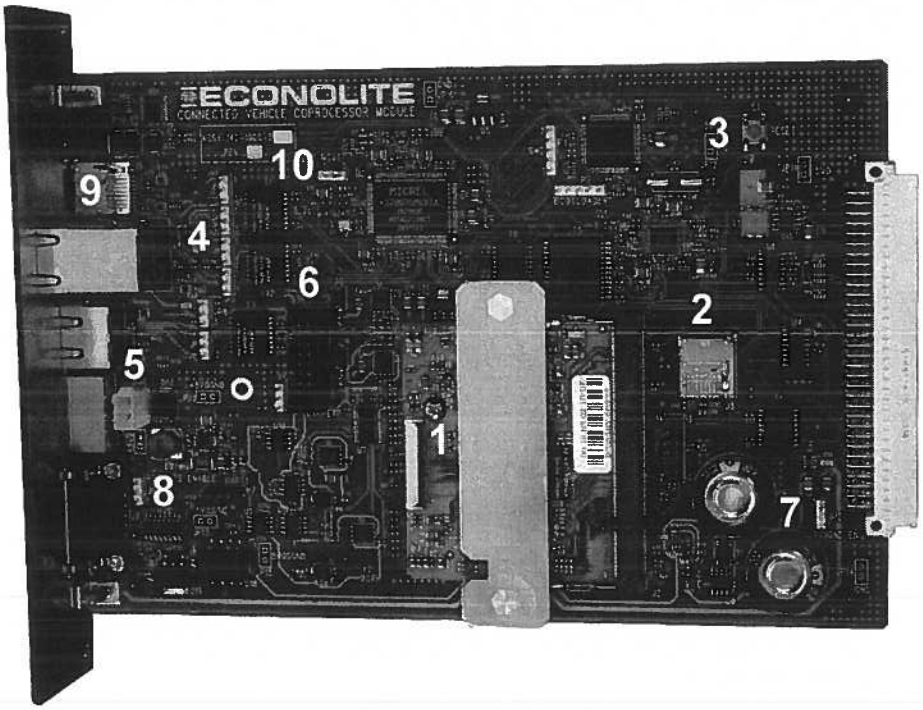


1. No Connect
2. RS232 Receive Data
3. RS232 Transmit Data
4. No Connect
5. Ground
6. No Connect
7. RS232 Ready-To-Send
8. RS232 Clear-To-Send
9. No Connect

The mating 9-pin male D sub-cable assembly is Econolite P/N 35119G20

[2] For programming information, refer to *Programming Front Panel LEDs* on Page 18.

Connected Vehicle CoProcessor Module Components



1. Processor System on Module (SoM)
2. Internal Operating System microSD Card (included)
3. Internal System Reset Button
4. Power Sourcing Equipment Mode Select Jumpers (JP3 thru JP8)
5. Internal 48VDC Power Connector
6. Ethernet Expansion Connector
7. ISOGND Enable Jumper (JP2)
8. Console Port RTS-CTS Enable Jumper (JP1)
9. External microSD Card Slot
10. Ethernet Switch Management Enable Jumper (JP17)

Connected Vehicle CoProcessor Module Jumper Positions and Functions

Jumper	1 and 2	2 and 3 (default)
JP1	Connect RTS to CTS pin on C50S (not ATC/TEES standard compliant)	RTS and CTS not connected (ATC/TEES standard)
JP2	Allow 48V0GND on A2 connector (not ATC/TEES standard compliant)	No 48V0GND on A2 (ATC/TEES standard)

Power delivered by **data lines** is PoE **Mode A** and power delivered by **spare lines** is PoE **Mode B**.

Note: The RSU must support both Mode A and Mode B. In the table below, the jumper pairs shown together (JP3/JP4, JP5/JP6 and JP7/JP8) must be moved together.

Jumper	1 and 2—Mode A	2 and 3 (default)—Mode B
JP3	Connect 48V0 to NET3 TX Center Tap	Connect 48V0 to NET3 spare pins 4 & 5
JP4	Connect PSE_N2 to NET3 RX Center Tap	Connect PSE_N2 to NET3 spare pins 7 & 8
JP5	Connect 48V0 to NET2 TX Center Tap	Connect 48V0 to NET2 spare pins 4 & 5
JP6	Connect PSE_N1 to NET2 RX Center Tap	Connect PSE_N1 to NET2 spare pins 7 & 8
JP7	Connect 48V0 to NET4 TX Center Tap	Connect 48V0 to NET4 spare pins 4 & 5
JP8	Connect PSE_N3 to NET4 RX Center Tap	Connect PSE_N3 to NET4 spare pins 7 & 8

Note: Jumper JP17 is reserved for future use.

Econolite Connected Vehicle CoProcessor Module Specifications

- **Main Processor System on Module (SoM)**—for complete SoM Specifications go to: https://boundarydevices.com/product/nitrogen6x-som/
    - Freescale i.MX6Q Quad Core ARM Cortex™-A9 processor running at 800 MHz (Max 1 GHz)
    - 1 GB DDR3 dynamic memory
    - 2 MB static serial NOR flash memory
    - 10/100/1 GB Ethernet Media Access Controller (MAC)
    - 200-pin SO-DIMM edge connector to carrier board
- ARM Cortex-M3 Microcontroller operating at 100 MHz
- Supercapacitor-backed Real Time Clock with SRAM
    - Guaranteed hold-up of 2 weeks at +74 °C for Real Time Clock (RTC) only
- Two user-programmable LEDs located on front panel

- **Communications**
  - Serial
    - Two internal, EIA-485, serial ports (SP1 and SP2) supporting asynchronous rates of: 1200, 2400, 4800, 9600, 19.2k, 38.4k, 57.6k, and 115.2k bps
      - SP1 provides serial communications between Host Board microcontroller and the ATC Engine Board
      - SP2 provides communications between SoM and the ATC Engine Board
    - One external, EIA-232, 9-pin asynchronous serial console port located on front panel.
      - Provides configuration communications to SoM
      - Default rate of 115200 bps, 8 bits data, no parity, 1 stop bit
      - RTC-CTS jumper to bypass flow control restrictions on serial port console
  - Ethernet
    - Integrated 5-port 10/100 managed Ethernet switch
      - Three 10/100, 802.3 at Type-1, ports capable of Power-Over-Ethernet on front panel
      - One 10/100 port to backplane DIN connector for communications with ATC switch
      - One 10/100/1000 port to SoM
- **Power**
  - Internal: 5 VDC @ 0.5A (typical, ATC/TEES-compliant)
  - External: 48 VDC @ 1.05 A (only required for PoE). This 48 VDC comes from a power supply through the connector provided, plugged into the 48 VDC connector on the front panel.
    - Recommended 48 VDC power supplies:
      - Office/development use: SL Power CENB1080A4803F01
      - Field use: TDK-Lambda DPP120481
- **Mechanical Specifications**
  - Product Dimensions (L x W x D) – 220.0 mm x 177.0 mm 40.4 mm
  - Standard Model 2070 communications slot mechanical form factor
- **Environmental Specifications**
  - ATC/TEES-compliant (-37 °C to +74 °C)

Econolite Connected Vehicle CoProcessor (CVCP) Linux Overview

The Connected Vehicle Co-Processor (CVCP) module runs Linux 3.10.17. A Linux console serial port running at 115200,n,8,1 is available as well as an ssh server with the user "econolite".

This board is intended to operate as an interface co-processor between an ATC or 2070-1C CPU and other external devices such as a DSRC RSU.

The CVCP uses a Freescale iMX.6 and contains an ext2 root file system with normal packages such as OpenSSH/SSL, Python, Sqlite, Net-Snmp, tcpdump, iputils. The rootfile system includes the normal ipv4 tools and in addition includes ipv6 tools to do configuration including ping6, tracepath6 and traceroute6.

The net-snmp-5.7.3 package is used as an application program example. Specifically the snmpwalk, snmpget & snmpset programs are cross compiled and their use demonstrated as an example for communication with a traffic controller such as an Econolite Cobalt or a 2070 running Econolite ASC/3 LX firmware. The use of the net-snmp library as a cross compilation example allows a developer to have the header and shared object libraries available for include and linking steps.

Scope

This Linux overview:

- Shows dhcp & static network configuration
- Describes how to install a cross compile toolchain, compile, load onto board and remote gdb
- Shows examples of using various packages

Change password

Use the standard "passwd" utility on the board to change the password of the root user, the econolite user or both as desired.

Network Configuration

The network can be either dhcp or static ip addresses and is controlled by the file /etc/network/interfaces. By default, static is used but, by commenting out the static lines and uncommenting the dhcp line, the interface can be enabled as dhcp. This operation must be done as root. In the "interfaces" file, both dhcp and static are shown.

```
## CVCP board /etc/network/interfaces file
auto eth0
## uncomment one group or the other, but not both
#iface eth0 inet dhcp
# the 1 line above for dhcp, the 4 lines below for static
iface eth0 inet static
address 10.20.70.52
netmask 255.255.252.0
gateway 10.20.70.254
```

Execute a program from a resource script on startup

When the system boots, all the scripts in /etc/init.d beginning with upper-case S are executed in alphabetic order. Add a script named S99local and edit into it any commands for programs to execute at startup. Ensure the script is executable with chmod. This must be done as root. Here is an example to create an "S99local" script in the /etc/init.d directory and set it executable.

```
## CVCP board
# vi /etc/init.d/S99local
# chmod a+x /etc/init.d/S99local
```

Install Compiler for use

We use the Mentor CodeSourcery pre-compiled toolchain arm-2014.05 which is gcc version 4.8.3. It may be obtained from the Mentor web site at http://sourcery.mentor.com/public/gnu_toolchain/arm-none-linux-gnueabi . This toolchain is a .bz2 compressed file and is installed with tar on a host PC. The argument "xvfj" extracts the .bz2 appropriately to the current working directory. This toolchain executes on an X86 PC and produces binaries which run on the CVCP processor.

The lines below show the steps to install and configure the toolchain for use and represent the commands used on the host PC terminal. Note that the directory being created "cv-tools" can be any name. This name is used as an example.

```
## Host PC
$ cd ~
$ mkdir cv-tools
$ cd cv-tools
$ tar xvfj .../arm-2014.05-29-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
$ ./arm-2015.02/bin/arm-none-linux-gnueabi-gcc -v
//should be gcc version 4.8.3

// Add to path and set CROSS_COMPILE
$ export PATH=$PATH:~/cv-tools/arm-2015.02/bin/
$ export CROSS_COMPILE=arm-none-linux-gnueabi-

// May now be referenced directly from the $PATH either on the command line
// or in a Makefile
$ arm-none-linux-gnueabi-gcc
```

Compile a program

Given an installed toolchain, a $PATH to that toolchain and a $CROSS_COMPILE variable with the compiler prefix, one can then compile a typical C or C++ program by navigating to the appropriate source directory and invoking make.

```
## Host PC
$ export PATH=$PATH:~/cv-tools/arm-2014.05/bin/
$ export CROSS_COMPILE=arm-none-linux-gnueabi-
$ cd ~/appropriate/source/directory
$ make
```

Install Binary on the CVCP

You can either use tftp or ssh to transfer files to the CVCP. The Host PC is typically setup to also have a tftp server running. In the two examples below, replace <ipAddressOfTftpServer> with the actual address of your Host PC, <filename> with the actual name of the file being transferred, <user> with the name of the user on your HostPC and <boardIpAddress> with the actual IP address of the board.

These examples are written from the viewpoint of the CVCP board transferring files from the Host PC.

```
## CVCP Board using tftp
$ cd /home/econolite
$ tftp -g <ipAddressOfTftpServer> -l <filename>
$ chmod a+x <filename>
```

You can either use tftp or ssh to transfer files to the CVCP. This is an scp example.

```
## CVCP Board
$ cd /home/econolite
$ scp <user>@<ipAddress>:~/<filename> .  // Trailing dot means "current directory"
$ chmod a+x <filename>                   // Make it executable
```

ssh, scp to the CVCP

From an external host, one can ssh or scp to the CVCP with the user: "econolite" and the password "ecpi2ecpi"

```
## Host PC
Host PC
$ ssh econolite@<boardIpAddress>
CVCP Board# //execute command on CVCP board now
```

Create a new microSDcard

It may become necessary to create a new sdcard sometime. If that occurs, use the rootfs.ext2.gz file from Econolite and both the zcat and dd utilities to reflash a microSDcard.

Take the microSDcard, connect it to a microSD-USB adapter and connect to a Host PC. Then execute the command below to write it to the microSDcard.

Use the "dmesg" utility to see that the microSDcard has mounted as mmcblk0p1. On some systems it may mount as a different device such as sdc. Be careful with this command and only an advanced user should perform this action.

The CVCP board is accompanied with a zip file that contains the rootfs.ext2.gz file used to create the original microSDcard.

```
$ zcat output/images/rootfs.ext2.gz | sudo dd of=/dev/mmcblk0p1 bs=1M
```

External microSD

Plugging in an sdcard into the external microSD connector on the front panel of the CVCP board will mount the card on /media/mmcblk1p1. Logging or other files may be written to or read from the mounted microSD card.

```
## CVCP Board
# ls /media/mmcblk1p1 -l
```

Programming Example Using Test net-snmp

The programs snmpwalk and snmpget are pre-installed and below is a description of how to configure and compile from source as references for application programs.

Given an snmpwalk program, a controller may be queried and will display its standard MIBS from a command prompt on the CVCP board. The arguments to snmpwalk are the community, "public" in this case, and the ipaddress:port of the controller. The arguments to snmpget are the community, the ipaddress:port and the variable to be read.

```
## CVCP Board
# snmpwalk -c public 10.1.15.12:501
SNMPv2-MIB::sysUpTime.0 = Timeticks: (4022798) 11:10:27.98
SNMPv2-MIB::sysContact.0 = STRING: Econolite Control Products
SNMPv2-MIB::sysName.0 = STRING: ASC3 Actuated Controller
## Use snmpget to get some standard MIBS
# snmpget -c public 10.1.15.12:501 system.sysUpTime.0
SNMPv2-MIB::sysUpTime.0 = Timeticks: (4103588) 11:23:55.88
## This is NTCIP maxPhases MIB
# snmpget -v 1 -c public 10.1.15.12:501 1.3.6.1.4.1.1206.4.2.1.1.1.0
SNMPv2-SMI::enterprises.1206.4.2.1.1.1.0 = INTEGER: 16
```

The programs in net-snmp-5.7.3 may be compiled standalone and the headers along with libraries and Makefiles can be used as an example to create other snmp programs. In order to run any of these programs, they must be copied via scp or tftp to the CVCP board from the ~/temp2 directory. This ~/temp2 directory is created in the configure step and is necessary to describe a directory to install the binaries to in order to avoid writing over the default binaries on the Host PC.

Go to http://www.net-snmp.org/download.html and get the source files for netsnmp-5.7.3. Expand these out in a directory on a host PC. In the directory apps/ are the source files for snmpwalk.c and snmpget.c. Use these source files as examples to help understand how to create an application program to do snmp communication with a controller.

First untar the source and setup the environment

```
## Host computer compilation environment
$ cd ~
$ tar xvf ~/Downloads/net-snmp-5.7.3.tar.gz
$ export PATH=$PATH:~/arm-2014.05/bin
$ export CROSS_COMPILE=arm-none-linux-gnueabi-
```

Then configure and compile the net-snmp source. This configure step is important to copy paste as is along with the line continuation characters "\". Most source packages require a custom crafted configure step and this is the one for the net-snmp-5.7.3 package. Google "cross compile configure" or refer to https://www.gnu.org/software/autoconf/manual/autoconf-2.69/html_node/Hosts-and-Cross_002dCompilation.html for additional information on a configure step.

```
## Host computer configure and compile
$ cd net-snmp-5.7.3
net-snmp-5.7.3$ mkdir ~/temp2
net-snmp-5.7.3$ ./configure --target=arm-none-linux-gnueabi \
 --host=arm-none-linux-gnueabi --build=x86_64-unknown-linux-gnu \
 --with-persistent-directory=/var/lib/snmp --with-defaults \
 --enable-mini-agent --without-rpm --with-logfile=none \
 --without-kmem-usage --enable-as-needed --without-perl-modules \
 --disable-embedded-perl --disable-perl-cc-checks --disable-scripts \
 --with-default-snmp-version="1" --enable-silent-libtool \
 --enable-mfd-rewrites --with-sys-contact="root@localhost" \
 --with-sys-location="Unknown"\
 --with-out-transports="Unix" --disable-manuals \
 --with-install-prefix=~/temp2 --with-cc=arm-none-linux-gnueabi-gcc \
 --with-ld=arm-none-linux-gnueabi-ld --with-endianness=big \
net-snmp-5.7.3$ make clean && make
net-snmp-5.7.3$ make install
```

At this point, you should have snmpset, snmpget and other binaries in the ~/temp2 directory. Use tftp or scp to copy to the CVCP board and test them.

Use the Makefile for snmpset & snmpget as an example to add a new application program to develop for your purpose. The reason we are building this source file completely is to gain access to the header files and shared object libraries files necessary for the cross compilation and to avoid bringing in header or librariy files from the Host PC.

Accessing the traffic controller data

The primary method of accessing data on the traffic controller is via NTCIP. Because this protocol is based on SNMP as a transport mechanism, the installed net-snmp library can be used to provide most of the required communications support. In this section we will use examples from the net-snmp source code to illustrate how to work with these functions to execute NTCIP data exchanges.

Source files for the library and some command line SNMP tools are in the net-snmp-5.7.3 directory and its sub-directories. As an example of how to access data from an application program, below we explain how the snmpget application (source code is in net-snmp-5.7.3/apps /snmpget.c) can be used to access data on the controller by setting up a session to read an SNMP MIB.

The first requirement is to open communications with the remote device; in this example we use the Cobalt traffic controller. As shipped from Econolite, the CVCP is configured with a network IP address that is compatible with the default traffic controller configuration, but our application will need to provide some basic information to the library functions to establish the connection. In the SNMP world, this connection is called a session. Before we can call the appropriate function, we need to add the required data to a structure that describes the required connection. There are many elements to this structure and more detailed descriptions can be found at http://www.net-snmp.org/wiki/index.php/TUT:Simple_Application which describes an application using snmp_session and the setting of variables in this structure.

For this example, we will set the minimum values in snmp_session to version, peername, community and community_len.

Session.version is set to 0 meaning version 1.

Session.peername is set to the string containing the IP Address of the traffic controller, a colon ":", and the receive port of the traffic controller.

Community is a string used to identify access to variables across the network.

Session.community is set to the string "public" & the length of this string is set to session.community_len.

After the structure is populated, a session is opened with snmp_open() function.

```
netsnmp_session session, *ss;
session.version = 0;
session.peername= = "10.1.15.12:501"
session.community = "public"
session.community_len = strlen(session.community);
ss = snmp_open(&session);
```

After a session is open, the application can then make its request for data. The library does this using data formatted into a pdu which is a netsnmp_pdu structure defined along with netsnmp_session in net-snmp/types.h.

We first create a formatted pdu with the snmp_pdu_create() function. The parameter passed to the function defines the type of pdu we need. In this case, we use SNMP_MSG_GET, which is the simplest. Other available message types such as SNMP_MSG_SET allow a program to set an OID using the same subroutines if desired. Refer to https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol.

```
netsnmp_pdu    *pdu;
pdu = snmp_pdu_create(SNMP_MSG_GET);
```

The created pdu is, in effect, a template. In order to explicitly identify the data we are looking for, we need more information; this data is the OID of the required data. You can find the standard MIBs and OIDs defined by snmp in the install directory (~/temp2/snmp/mibs/*.txt).

Add the OID to the pdu using snmp_add_null_var()

This is the key part of the program. It takes the pdu created in the previous step and adds the oid argument string to it. That is, the string that looks like "1.3.6.1.4.1.1206.4.2.1.1.1.0 whose length is 28.

```
// Add the oid to the pdu
snmp_add_null_var(pdu, "1.3.6.1.4.1.1206.4.2.1.1.1.0", 28);
```

We are now ready to make the request for data. This example uses a function that sends the request message and waits for a response.

Perform the request with snmp_sync_response(). Two netsnmp_pdu structures are used. One is "pdu" and the other is "response". "Pdu" sends the variable requested and "response" contains the response from the peer. The variables and their values are in the response->variables member of the response structure.

```
int             status;
netsnmp_pdu    *response;
 status = snmp_synch_response(ss, pdu, &response);
    if (status == STAT_SUCCESS) {
        if (response->errstat == SNMP_ERR_NOERROR) {
            for (vars = response->variables; vars;
                 vars = vars->next_variable)
                print_variable(vars->name, vars->name_length, vars);
        }
```

This process can be repeated as required from the PDU create step until all required data has been retrieved. For retrieving additional variables, it is best to free the pdu with snmp_free_pdu() and create a new one with snmp_pdu_create().

After they are complete, we need to free the resources we allocated and close the connection.

```
snmp_free_pdu(response);
snmp_close(ss);
```

For more information on the NTCIP MIBS and OIDS for connected vehicles, refer to the companion pdf file Econolite-ConnectedVehicle-Cabinet_SpaT-Guide-16Mar15.pdf pages 8-17.

Explanation of snmpget Code

The source file for snmpget is snmpget.c and it is in the net-snmp-5.7.3/apps directory. This program is small, about 250 lines and parses a set of command line arguments containing version, community, address:port and the OID to be read. The binaries are installed in the ~/temp2 directory which is set by the configure step with the –with-install-prefix command. This set of programs and libraries may be compiled on an x86 as well as cross compiled for the CVCP board.

This program and its supporting libraries are compiled by default with debug symbols with the "-g" compiler option. This means single stepping through the program and its associated library calls with gdb, and either ddd or eclipse is reasonable.

The main start of the snmpget.c program is at line 106 and parses the command line with snmp_parse_args() in line 126 to get the version, community, ipaddress:port of the peer, and OID to be read. These are filled out into the structure "session" by snmp_parse_args() and available later in the program. A session with the netsnmp library is opened at line 162 with snmp_open(&session) and a protocol data unit (pdu) is created at line 176 with snmp_pdu_create().

The request & response are processed between lines 198 and 237. The key subroutine here is snmp_sync_response() which is passed the session (ss), the protocol data unit (pdu) and a structure to receive the response as arguments. The response is printed on line 204 with the routine print_variable().

Explanation of snmpset Code

Snmp can be used to write or "set" a variable by defining a PDU with SNMP_MSG_SET instead of SNMP_MSG_GET. Snmpset.c shows a similar parsing of arguments for version, community, ipaddress:port and OID along with the variable type and value.

The source file for snmpget is snmpget.c and it is in the net-snmp-5.7.3/apps directory along with snmpget.c. This program is very similar to snmpget.c, but does its snmp_pdu_create call with SNMp_MSG_SET.

This program and its supporting libraries are compiled by default with debug symbols with the "-g" compiler option. This means single stepping through the program and its associated library calls with gdb, and either ddd or eclipse is reasonable.

The main start of the snmpset.c program is at line 113 and parses the command line with snmp_parse_args() in line 137 through 194 to get the version, community, ipaddress:port of the peer, OID to be written, its variable type and value to be written. These are filled out into the structure "session" and available later in the program. A session with the netsnmp library is opened at line 212 with snmp_open(&session) and a protocol data unit (pdu) is created at line 225 with snmp_pdu_create(SNMP_MSG_SET).

The next step is to add the OID, its type and value to the pdu before sending it with snmp_sync_response().

```
// Add the oid to the pdu
//snmp_add_var(pdu, name, name_length, type, value)
snmp_add_var(pdu, "1.3.6.1.4.1.1206.4.2.1.4.14.0", 28, "i:1", value);
```

The request & response are processed at line 248. The key subroutine here is snmp_sync_response(), which is passed the session (ss), the protocol data unit (pdu) and a structure to receive the response as arguments. The response is printed on line 204 with the routine print_variable() and generally contains the value written as well as an indication of success.

The routine snmp_sync_response() performs the request to write and receives the response. Two netsnmp_pdu structures are used, one is "pdu" and the other is "response". Pdu sends the variable requested and response contains the response from the peer. The variables and their values are in the response->variables member of the response structure.

```
int             status;
netsnmp_pdu    *response;
 status = snmp_synch_response(ss, pdu, &response);
    if (status == STAT_SUCCESS) {
        if (response->errstat == SNMP_ERR_NOERROR) {
            for (vars = response->variables; vars;
                 vars = vars->next_variable)
                print_variable(vars->name, vars->name_length, vars);

        }
```

Here is an example using snmpset with a traffic controller:

```
## CVCP Board
## This sets the systemPatternControl to flash (255) and then back to standby (0)
# snmpget -v 1 -c public 10.1.15.12:55502 1.3.6.1.4.1.1206.4.2.1.4.14.0
iso.3.6.1.4.1.1206.4.2.1.4.14.0 = INTEGER: 0
# snmpset -v 1 -c public 10.1.15.12:55502 1.3.6.1.4.1.1206.4.2.1.4.14.0 i:1 255
iso.3.6.1.4.1.1206.4.2.1.4.14.0 = INTEGER: 255
# snmpget -v 1 -c public 10.1.15.12:55502 1.3.6.1.4.1.1206.4.2.1.4.14.0
iso.3.6.1.4.1.1206.4.2.1.4.14.0 = INTEGER: 255
# snmpset -v 1 -c public 10.1.15.12:55502 1.3.6.1.4.1.1206.4.2.1.4.14.0 i:1 0
iso.3.6.1.4.1.1206.4.2.1.4.14.0 = INTEGER: 0
# snmpget -v 1 -c public 10.1.15.12:55502 1.3.6.1.4.1.1206.4.2.1.4.14.0
iso.3.6.1.4.1.1206.4.2.1.4.14.0 = INTEGER: 0
```

Additional Tools Installed on the CVCP

Programming Front-Panel LEDs

The CVCP module has two front-panel LEDs, shown on Page 6, that are user-programmable from Linux via console commands. From the left side of the front panel, they are designated USER_LED1 and USER_LED2. Both LEDs are pre-configured to turn ON at power up.

To turn ON the USER_LED1, enter:

```
echo 1 > /sys/class/gpio/gpio3/value
```

To turn OFF the USER_LED1 enter:

```
echo 0 > /sys/class/gpio/gpio3/value
```

To turn ON the USER_LED2 enter:

```
echo 1 > /sys/class/gpio/gpio0/value
```

To turn OFF the USER_LED2 enter:

```
echo 0 > /sys/class/gpio/gpio0/value
```

Test gdb & gdbserver

In embedded development, the gdb debugger is split into two parts. One part runs on the host PC where the program is compiled from and the other part runs on the CVCP board. The two parts connect via ethernet to each other in order to step through source code on the host PC with a binary on the CVCP board.

The CVCP board is known as a "target". The host Linux PC that compiled the program to be debugged is known as a "host". Remote gdb is used on this board. That means gdbserver runs the program on the target CVCP board and waits for a connection from gdp on the host PC where the program was compiled. In other words, the host PC that compiled the program runs gdb and connects to the remote CVCP board target.

```
// First, copy the binary to the target and execute it under gdbserver
// using the hostIP address and a port such as 2001
CVCP board$ scp user@host:/<binfile>
CVCP board$ gdbserver <binfile> hostIp:port
// Then startup the arm gdb server on the host with the original binfile
// in the source tree as an argument
host PC$ cd /where/binfile/was/compiled/from
host PC$ arm-none-linux-gnueabi-gdb <binfile>
 (gdb) target remote targetIP:port
 (gdb) break main
 (gdb) continue
```

Python

Python is installed and this is an example of using the Python interpreter from a terminal connected to the CVCP board. Python may either be used with its interpreter from a command line or a Python script will execute and support import of libraries. Below shows the commands to verify the Python installation.

```
## CVCP Board
# python
Python 2.7.9 (default, Dec  1 2015, 10:03:05)
[GCC 4.8.3 20140320 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World"
Hello World
>>> quit()
#
```

Test sqlite3

An sql compatible datebase program called sqlite3 is installed on the CVCP board. This database program supports simultaneous reads & writes of multiple entries at high speed. This is an example of starting sqlite3, creating a database from its command shell and performing a query.

A program may contain embedded sql commands to open, read & write a database and support dynamically-changing data in a traffic control environment. Here is how to generate a table and query it:

```
## CVCP Board
# sqlite3
SQLite version 3.8.8.2 2015-01-30 14:30:45
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> create table tbl1(one varchar(10), two smallint);
sqlite> insert into tbl1 values('hello!',10);
sqlite> insert into tbl1 values('goodbye', 20);
sqlite> select * from tbl1;
hello!|10
goodbye|20
sqlite> .help
.backup ?DB? FILE      Backup DB (default "main") to FILE
.bail on|off           Stop after hitting an error.  Default OFF
.clone NEWDB           Clone data into NEWDB from the existing database
.databases             List names and files of attached databases
.dump ?TABLE? ...      Dump the database in an SQL text format
                         If TABLE specified, only dump tables matching
                         LIKE pattern TABLE.
.echo on|off           Turn command echo on or off
.eqp on|off            Enable or disable automatic EXPLAIN QUERY PLAN
.exit                  Exit this program
.explain ?on|off?      Turn output mode suitable for EXPLAIN on or off.
                         With no args, it turns EXPLAIN on.
... only the first few dot commands are shownUsing an external sdcard
```