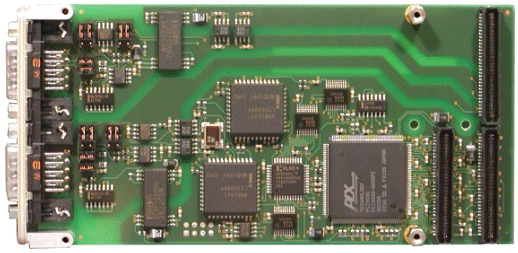
IRAM-COMP-057

Revision: 0
2009-03-26

Contact Author

Institut de RadioAstronomie Millimétrique

Can IP



Owner Sebastien Blanchet

Keywords: CAN

Approved by:	Date:	Signature:
A.Perrigouard	March 2009	

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS

Content

1 Summary

CanIP is a C++/Qt library to manage the CAN bus. Historically, this library has been written in 2008 for the IRAM needs but the library has been carefully designed to be reusable by any CAN applications.

2 Features

- IP-encapsulated CAN packets.
- SQL database backend to retrieve configuration information.
- CAN message monitoring tools
- Powerful graphical CAN message reader/writer.
- CanOpen protocol support
- Many code examples to show how to use it.

3 Requirements

3.1 Software requirements

The software is written in C++ with the Qt 4 Framework, therefore the software requirements are:

- Linux Operating system
- C++ compiler
- Qt 4.4.3 or more recent
- Utils library (a IRAM C++ library)

The software has been tested successfully on the following platforms:

- Fedora Core 4 and 6 i386
- Debian 4.0 (Etch) and 5.0 (Lenny) i386

3.2 Hardware requirements

The software runs on any PC, but a CAN controller is obviously needed to control a real device. For the moment, the only supported CAN controller is TPMC816 from Tews Technologies. This card uses the PMC format, so if you do not have PMC slot in your computer, you need to use a PCI carrier to plug in the TPMC816 card.

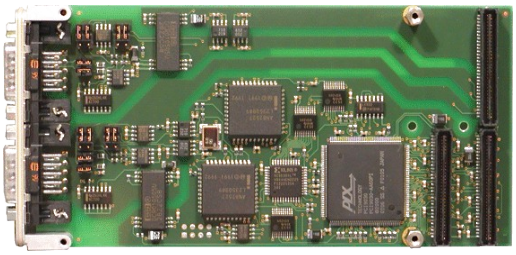


Figure 1: TPMC816 from TEWS Technologies

The procedure to support CAN controllers from other manufacturers is available later in this document.

4 Installation

4.1 Building

Get the sources:

```
$ mkdir develSVN
$ cd develSVN
$ svn co svn://svn.iram.fr/general/drivers/can/tpmc816/trunk tpmc816
$ svn co svn://svn.iram.fr/general/cplusplus/Utils/trunk Utils
$ svn co svn://svn.iram.fr/general/cplusplus/CanIp/trunk CanIp
```

Edit the file *Utils/conf/conf.pri* so set the paths for Utils dependencies.

Build the TPMC816 driver (not required if you use only the simulation)

```
$ cd ~/develSVN/tpmc816
$ su -c "make install"
```

To create devices (and to load the driver)

```
$ su -c "./create_devices.sh"
```

Note for the Debian users:
The *create_devices.sh* script creates nodes in */etc/udev/devices*, but by default there will not be recreated at startup.

Therefore, create a startup script to recreate devices on boot:

```
# echo "rsync -a --devices /etc/udev/devices/ /dev/" \
     > /create_devices.sh
# chmod +x /create_devices.sh
```

Modify */etc/rc.local* to load driver and to call */create_devices.sh*

```
# echo "/sbin/modprobe tpmc816drv" >> /etc/rc.local
# echo "/create_devices.sh" >> /etc/rc.local
```

Now build the CanIp library:

```
$ cd CanIp
$ qmake -r
$ make all
```

The output is stored in the bin subdirectory. There are both debug and release version.

4.2 API documentation

The API documentation can be automatically extracted with doxygen.

```
$ make doc
```

The output is stored in the *doxydoc* subdirectory.

5 CAN Infrastructure

5.1 Overview

The Linux device node that represents the CAN controller cannot be opened simultaneously by several processes. Therefore a little server program called **CanManager**, has been written to share the CAN with all the applications that want read/write CAN messages. Each CAN message is encapsulated into an UDP packet to be exchanged between server and clients

The main advantages of this architecture are:

- Transparency: the clients nodes read/write CAN messages as if they have direct access to the CAN controller..
- Flexibility: virtual devices, like simulators, can be easily plugged to the CAN/IP bus
- Simplicity

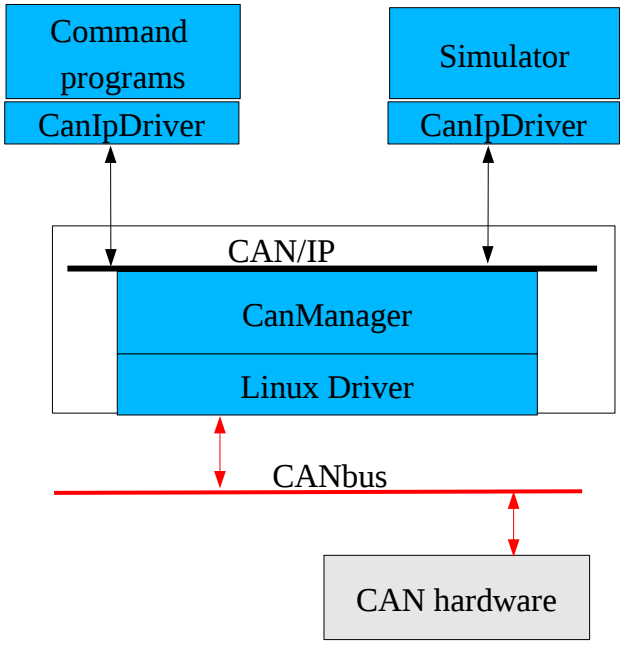


Figure 2: CanIp Infrastucture

5.2 CAN/IP protocol

This section explains the CAN/IP protocol.

5.2.1 Packet definition

A CAN/IP packet is basically a fixed-length UDP packet that contains a C data structure,

```
#define CAN_MAX_DATA_LENGTH 8

/* define a can message */
typedef struct {
   uint16_t msgType;     /* message type    */
   uint8_t  extended;    /* =1 for 29 bits identifiers, else =0 */
   uint8_t  msgLen;      /* data length (CAN)  */
   uint32_t identifier;  /* CAN Identifier     */
   uint8_t  data[CAN_MAX_DATA_LENGTH];   /* CAN data     */
} CanIpMsg_NetStruct_t;
```

```
typedef enum  {
   INVALID,
   RECORD,
   RECORD_ACK,
   DELETE,
   DELETE_ACK,
   CAN_MSG,
   CAN_MSG_ACK
} MsgType;
```

5.2.2 Protocol description

Initialisation
When *CanIpDriver* starts, it tries to initialize a connection with the *CanManager*, it sends a *CanIpMsg_NetStruct_t* with *msgType = RECORD*. Then it waits for a RECORD_ACK message.
Note: This record sequence is not mandatory since a normal CAN_MSG is enough to be registered in the CanManager. Though it is very useful for a CAN node to know if the server is alive or dead.

Keep connection alive
To avoid accumulation of dead connections, *CanManager* drops all connections that have not send a message for 6 seconds. Therefore *CanIpDriver* send automatically a RECORD message every 2 seconds, only to keep alive the connection.

Can traffic
All messages with msgType = CAN_MSG are sent to the other CAN/IP nodes and to the CANbus.
All messages from the CANbus are converted to CAN/IP and sent to the CAN/IP bus.

The CAN_MSG_ACK messages are defined but are not used yet.

Closing
To close the connection, *CanIpDrv* has to send a DELETE *CanIpMsg*. The CanManager replies with a DELETE_ACK message.

6 Database

A SQL database is used to store requires information such as:
- CAN Identifier from the symbolic name
- CAN bus to use (an application may use simultaneously several CAN bus)

For convenience, CanIP use SQLite as SQL database.
SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain. For more information see http://www.sqlite.org

Therefore, the environment variable CAN_DB_FILE must be set to the SQLite database file.

The minimal SQL table definition is stored in *sql/canip.sql* file.

7 CAN libraries

7.1 CanIp

This library provides the base classes to build a CAN application
The source code is in directory CanIp/libs/CanIp

Main classes:
- *CanDevice*: Mother class for all Can devices.
- *CanIpDriver*: To dialog with *CanManager* with the UDP protocol. Useful to write CAN monitor tools, but for normal applications it is better to inherit from the *CanDevice* class.
- *CanMsg*: class that represents a CAN message

7.2 CanDevice

The source code is in directory CanIp/libs/CanDevice

This library provides a set of CAN devices:
- CanAdc
- CanDac
- CanIo
- CanMotor
- CanOpen (Mother class for CanOpen devices) For more detail on the CanOpen protocol see http://en.wikipedia.org/wiki/CANopen

8 CAN Applications

8.1 CanManager

CanManager is the gateway between the CANbus and the CAN/IP bus.

It is basically a 3-thread application:
- *processIPtoCan*: listen for CAN/IP packet and according to the message type, register/unregister a node, transmit a copy of the packet to the other CAN nodes and write the corresponding CAN message to the CANbus
- *processCanToIP:* listen for CAN message on the CANbus, then build the corresponding CAN/IP packet and send a copy to all registered CAN/IP nodes.
- *cleanInactiveClients:* remove the inactive connections (i.e. the connection with no traffic for 6 seconds), and monitor the CAN bus status. If there is a problem the CAN bus is reset and the status register is printed.

Note:
- Unlike the real CAN protocol, the sender does not receive a copy of its transmission.
- CanManager handles only CAN device controller, each CAN controller requires its CanManager. For example the TPMC816 has two CAN port, so we need two CanManager.
- CanManager can also be run without any CAN device controller, for example to use with a CAN device simulator.

8.1.1 Syntax

```
$ CanManager -h
CanManager is a bridge between the CAN bus and the CAN/IP protocol
Usage: CanManager [options]
Options:
  -d=name        CAN controller device name to use. If missing,
                      the application runs in simulation mode
  -p=N           Listen to UDP port N
  -l=N           Limit write speed base CAN messages.
                      'N' is in message/sec. Default value = 0 (no limit)

  -v             Display version information
  -h, -?         Display help
```

```
Example:
        CanManager -d=/dev/tpmc816_0 -p=2500 -l=20
```

8.2 CanLogger

It is a CAN monitor tool, that can optionally decode the CanID into symbolic names.

8.2.1 Syntax

```
$ CanLogger -h
CanLogger - CAN logger
Usage: CanLogger [options]
Options:
  -p=N                UDP port to contact (mandatory)

  -v                  Display version information
  -h, -?              Display help


Example:
        CanLogger -p=2500

Note: If the environment variable CAN_DB_FILE is set,
the program loads the database to decode CanID into symbolic names.
```

8.2.2 Example

```
$ export CAN_DB_FILE=/home/introot/emir/data/emir.db
$ CanLogger -p=2501
Settings:
Port= 2501
DatabaseFile= /home/introot/emir/data/emir.db

Load data from /home/introot/emir/data/emir.db
CAN_MANAGER_SERVER='localhost'
Connect to CanManager localhost:2501
2009-03-09T16:27:11.471: msg 1 : io_B1_LoSwitches_getStatus : X 0x01000100 []
2009-03-09T16:27:11.471: msg 2 : adc_B1_V1_ifLevel : X 0x13040100 []
2009-03-09T16:27:11.471: msg 3 : adc_B1_V2_ifLevel : X 0x13040101 []
2009-03-09T16:27:11.472: msg 4 : adc_B1_H1_ifLevel : X 0x13040102 []
2009-03-09T16:27:11.472: msg 5 : adc_B1_H2_ifLevel : X 0x13040103 []
```

8.3 UtilCan

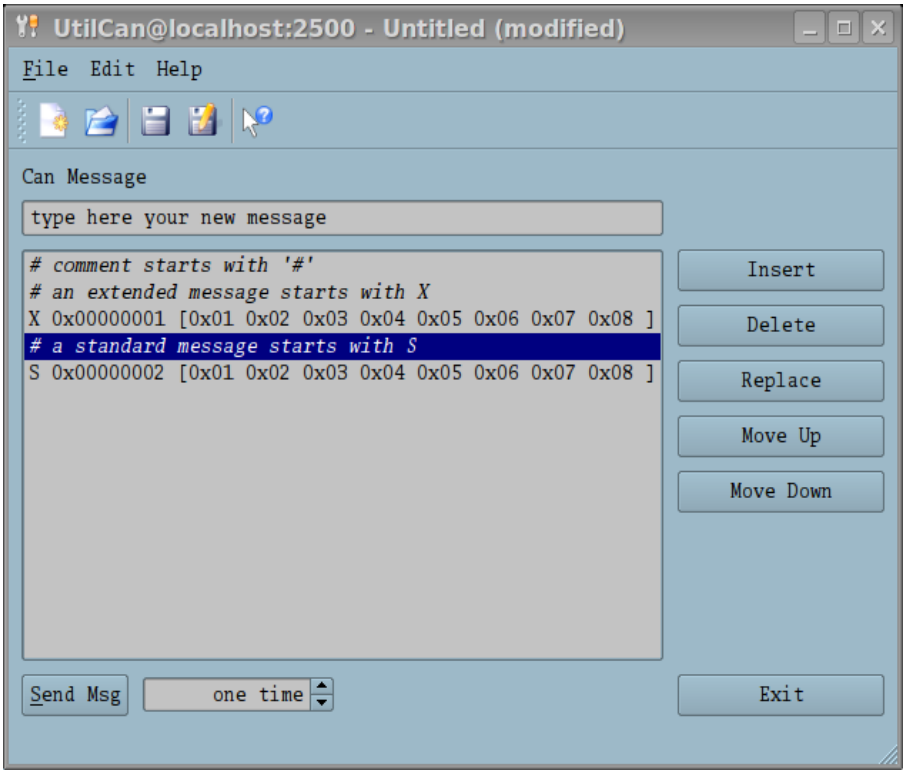
8.3.1 Syntax

```
$ UtilCan -h
UtilCan - CAN Utility to read/write CAN messages
Usage: UtilCan [options]
Options:
  -p=N                UDP port to contact (mandatory)
  -f=filename         File to load (optional)
```

```
  -v                  Display version information
  -h, -?              Display help


Example:
        UtilCan -p=2500 -f=myfile.txt
```

8.3.2 Usage



To add a new message:

- Enter the message in the top line edit, and press enter.

To move up or down messages:

- Select messages in the list.
- Click MoveUp or MoveDown button.

To modify an existing message:

- Double-click on the message in the list and enter the new value.