ECB-cuof/ofcu Software Reference Manual

Ethernet Can Bus Board Description

L. Castellani

I.N.F.N. sez. PADOVA

25 July 2014

Firmware Version 2.1

Introduction

The board is a bridge from Ethernet to two CAN bus controllers. The connection to the CAN controllers can be done also through USB.
The CAN controllers are configured to transfer all messages received to the TCP connections and/or to the USB connections, and all message received from TCP and/or USB to the CAN controllers.
The firmware implements two TCP servers for Ethernet connections and two USB Communications Device Class (CDC) that are seen on the PC as two serial port, the two channels are independent of each other.
The firmware for the configuration and status implement also a HTTP server, protected by password, by default user is admin and password is admin.
Throw USB connection also implement a Mass Storage Device (MSD), where are stored the files for the HTTP server, the INF file to install CDC on WindowsXP and other configurations files (On Linux the CDC are seen as ttyACM0 and ttyACM1). To see the MSD on WindowsXP at the first time the USB cable is connected you must CANCEL the new hardware installation, than when you see the MSD you must update driver of the new unknown hardware. The driver inf file can be found also on http://www.pd.infn.it/~caste/pub/Pic32_inf/ .
The frame accepted through the various connections have the following formats:

[SOF_L][SOF_H][size_L][size_H][data(0)]…..[data(size-1)]

The SOF=0x5AA5, the size is the number of CAN message multiplied by 16 , and the data content is constitute by a multiple of 16 byte that representing the CAN controller message registers in **little-endian** order:

```
struct CANRxMessageBuffer{
      // SID of the Received CAN Message.
      unsigned SID:11;
    // Filter which accepted this message.
      unsigned FILHIT:5;
    // Time stamp of the received message. This is
    // valid only if the Timestamping is enabled.
      unsigned CMSGTS:16;
    // Data Length Control. Specifies the size of the
    // data payload section of the CAN packet. Valid
    // values are 0x0 - 0x8.
      unsigned DLC:4;
    // Reserved bit. Should be always 0.
      unsigned RB0:1;
      unsigned :3;
    // Reserved bit. Should be always 0.
      unsigned RB1:1;
    // Remote Transmit Request bit. Should be set for
    // RTR messages, clear otherwise.
      unsigned RTR:1;
    // CAN TX and RX Extended ID field. Valid values
    // are in range 0x0 - 0x3FFFF.
      unsigned EID:18;
    // Identifier bit. If 0 means that message is SID.
    // If 1 means that message is EID type.
      unsigned IDE:1;
    // Susbtitute Remote request bit. This is
    // always set for an EID message. It don't care
    // for an SID message.
      unsigned SRR:1;
      unsigned :2;
    // This is the data portion of the CAN TX message.
      BYTE data[8];
};
```

```
struct CANTxMessageBuffer{
    // CAN TX Message Standard ID. This value should
    // be between 0x0 - 0x7FF.
      unsigned SID:11;
      unsigned :21;
   // Data Length Control. Specifies the size of the
    // data payload section of the CAN packet. Valid
    // values are 0x0 - 0x8.
      unsigned DLC:4;
    // Reserved bit. Should be always 0.
      unsigned RB0:1;
      unsigned :3;
    // Reserved bit. Should be always 0.
      unsigned RB1:1;
    // Remote Transmit Request bit. Should be set for
    // RTR messages, clear otherwise.
      unsigned RTR:1;
    // CAN TX and RX Extended ID field. Valid values
    // are in range 0x0 - 0x3FFFF.
      unsigned EID:18;
    // Identifier bit. If 0 means that message is SID.
    // If 1 means that message is EID type.
      unsigned IDE:1;
    // Susbtitute Remote request bit. This bit should
    // always be set for an EID message. It is ignored
    // for an SID message.
      unsigned SRR:1;
      unsigned :2;
    // This is the data portion of the CAN TX message.
      BYTE data[8];
};
```

For a compete description see http://ww1.microchip.com/downloads/en/DeviceDoc/61154C.pdf
page 56 (**34.7 TRANSMITTING A CAN MESSAGE**)
and page 75 (**34.9 RECEIVING A CAN MESSAGE**).

Ethernet configuration

The Ethernet configuration can be modified editing the ETH_CFG2.BIN file with and binary editor, or if the file is deleted when the board restart is generated a file with the following default settings :
IP=192.168.8.XXX where XXX is the value of the low byte of the BoardID
GATEWAY=192.168.8.254
DHCP=ENABLED
NETBIONAME=ECBx where x is the value of low byte of the BoardID

ETH_CFG2.BIN File format:

```
struct IP_CONFIG {
      DWORD ip_addr;
      DWORD ip_mask;
      DWORD ip_gateway;
      BYTE mac_addr[6]; // by default must be zero.
      BYTE en_dhcp;
      char NetBIOSName[16];
};
```

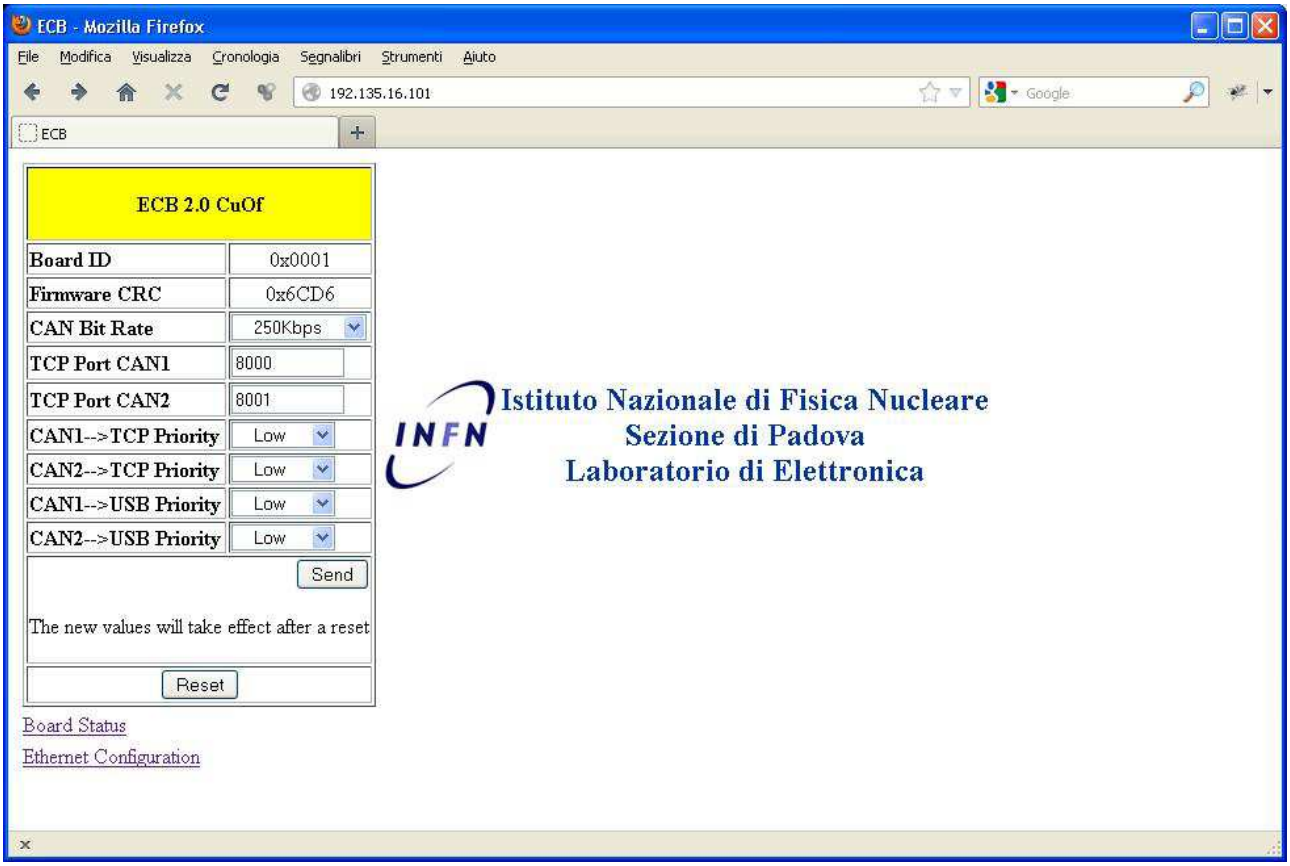
Board Start Up

At start-up the board execute for 30 seconds a boot program for firmware update and if the USB cable in connected during this period the countdown is stopped and remaining in the boot program until the MSD is safety removed from the PC.
The firmware can be update throw USB or Ethernet connection.
When is running boot program the yellow led blink at 1Hz if the board found ETH_CFG2.BIN file or blink 2Hz if not found the file and program generate the file with default parameters.
When is running ECB program the green led blink at 1Hz if the firmware found ETH_CFG2.BIN file or blink 2Hz if not found.

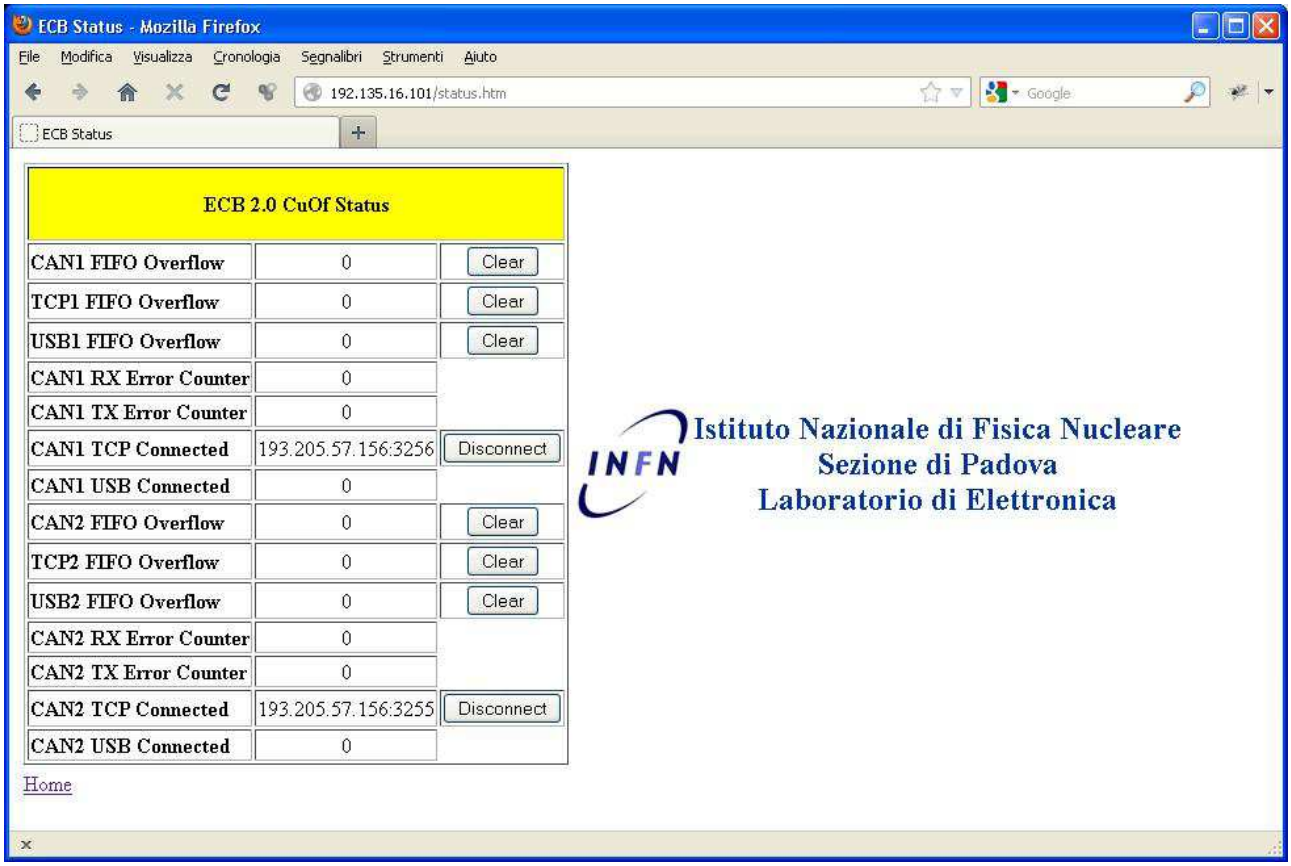
Functional Description

The firmware take all CAN messages from TCP and/or USB connection and transfer it to the CAN controller TX FIFO. The messages transmission is always guaranteed because firmware not accept more data from TCP and/or USB if the TX FIFO is full, the are two independent FIFO, 32 messages deep , one for TCP connection with a 4096 byte buffer and one for USB connection with 4096 byte buffer.
The received messages on the RX FIFO of the CAN controller are transferred to the TCP and/or USB, but the CAN RX FIFO can overflow if TCP and/or USB connection can't receive more data or if the firmware process is too slow to process the message flux.
In the actual firmware version the CAN RX FIFO is 32 message deep (32*16=512 byte), the TCP buffer is 4096 byte, the USB buffer is 4096 byte.
The received messages on the CAN RX FIFO are transferred to the TCP and/or USB in different mode, it depend by the priority settings.
If is connected only TCP or USB the priority setting have no effect.
If together TCP and USB are connected and priority is TCP=LOW, USB=LOW the messages transferred from the CAN RX FIFO to the connection buffers is the minimum number accepted by TCP or USB buffer, so only on the CAN RX FIFO the overflow flag can be set and TCP and USB receive the same messages.
If priority is TCP=HIGH, USB=LOW or TCP=LOW, USB=HIGH the messages transferred from the CAN RX FIFO to the high priority buffer is the maximum messages number accepted by the buffer, instead if the low priority buffer can't accept the same number of messages the buffer overflow flag is set, so the overflow flag can be set on the CAN RX FIFO and in the low priority buffer .
If priority is TCP=HIGH, USB=HIGH the messages transferred from the CAN RX FIFO to the buffers are the maximum messages number accepted by one of the two buffers, and the corresponding overflow buffer flag is set if one of the two buffers can't accept that number of messages.

General Settings and Board info



Board Status



Ethernet Configuration