

8-bit

Microcontroller

AP08071

Hardware and Software Description
DriveMonitor

Application Note

V2.0 2009-03-20

Microcontrollers

Edition 2009-03-20

**Published by
Infineon Technologies AG
81726 Munich, Germany
© 2009 Infineon Technologies AG
All Rights Reserved.**

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

DriveMonitor

Revision History: V2.0, 2009-03-20

Previous Version(s):

V1.0

Page	Subjects (major changes since last revision)
------	--

Update from Drive Monitor Software V5.1 to V6.0

3	Figure 3
---	----------

4 - 5	Section 2.2
-------	-------------

Additional Chapter

13 - 19	Chapter 3: Target Firmware
---------	----------------------------

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing?
Your feedback will help us to continuously improve the quality of this document.

Please send your proposal (including a reference to this document) to:

ipdoc@infineon.com



Table of Contents

1	Overview	2
2	PC Host Software (GUI)	4
2.1	JTAG Toolbar	4
2.2	CAN Toolbar	4
2.3	CAN Control Window	5
2.4	Customizing the CAN Control Window	7
2.4.1	Group Entries	7
2.4.2	Display Field	8
2.4.3	Buttons	9
2.4.4	Status Flags	9
2.4.5	Progress Bars	10
2.4.6	Oscilloscope	11
2.5	Troubleshooting	12
2.5.1	Problems with the Target	12
2.5.2	Problems with the DriveMonitor USB Stick	12
2.5.3	Problems with the Host Computer	14
3	Target Firmware	15
3.1	Program Flow	15
3.2	Command Structure	16
3.3	Receiving a CAN Message	16
3.3.1	Command Execution	16
3.3.2	Command SET	17
3.3.3	Command GET	17
3.3.4	Start, Stop and Ramp Down Buttons	18
3.3.5	Button Scope	18
3.4	Transmitting a CAN Message	19
3.4.1	Oscilloscope and Progress Bar	20
3.4.2	Status Flags and Display Fields	20
3.4.3	Command GET	21
3.4.4	Example for Get and Set Commands in Grouped Entries	22
4	Hardware Description	23
4.1	USB Interface	23
4.2	CAN Microcontroller XC886CM	25
4.3	CAN Transceiver and Target Connector	26
4.4	PCB Layout	27

1 Overview

The DriveMonitor is a USB stick (see [Figure 1](#)) providing JTAG, Virtual COM (VCOM) and CAN interfaces in one device. The DriveMonitor USB stick is designed to be used in combination with Infineon DriveCards. DriveCards are small microcontroller evaluation boards with a standard connector to power inverter boards, and a digitally isolated debug interface. A JTAG interface is used for software download and OCDS debugging, while a CAN connection is also provided for real time monitoring and parameter setup when the application is running.



Figure 1 DriveMonitor USB Stick

[Figure 2](#) shows the block diagram of the DriveMonitor.

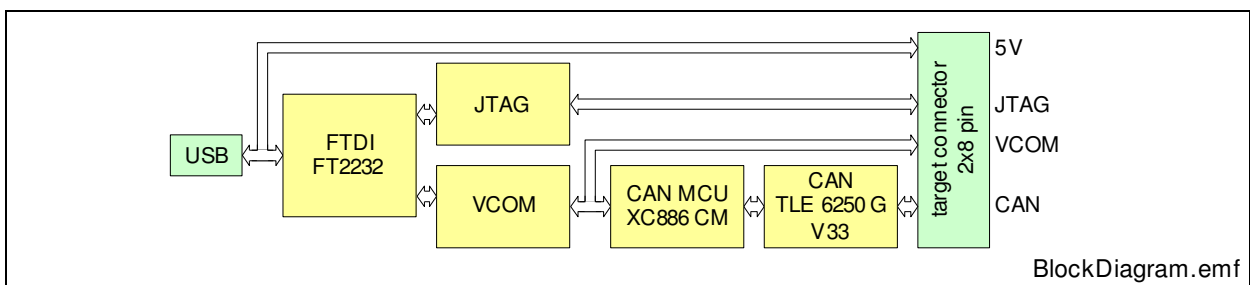


Figure 2 Block Diagram of DriveMonitor

The USB 2.0 compliant interface device (FT2232) provides a JTAG and a VCOM port. Both the JTAG and the VCOM port are directly connected to the 2 x 8-pin target connector. The VCOM port is additionally connected to the CAN microcontroller XC886CM. A driver application can convert CAN message objects in UART protocol, and vice-versa. As a result, PC software is able to monitor and generate CAN messages via USB and UART.

The DriveMonitor can be used for the CAN message monitoring/generating task. After clicking the connect button, a small driver software package is downloaded to the

DriveMonitor USB stick. CAN messages can then be sent to and received from the target device.

Figure 3 shows a screen shot of the DriveMonitor software V6.0.

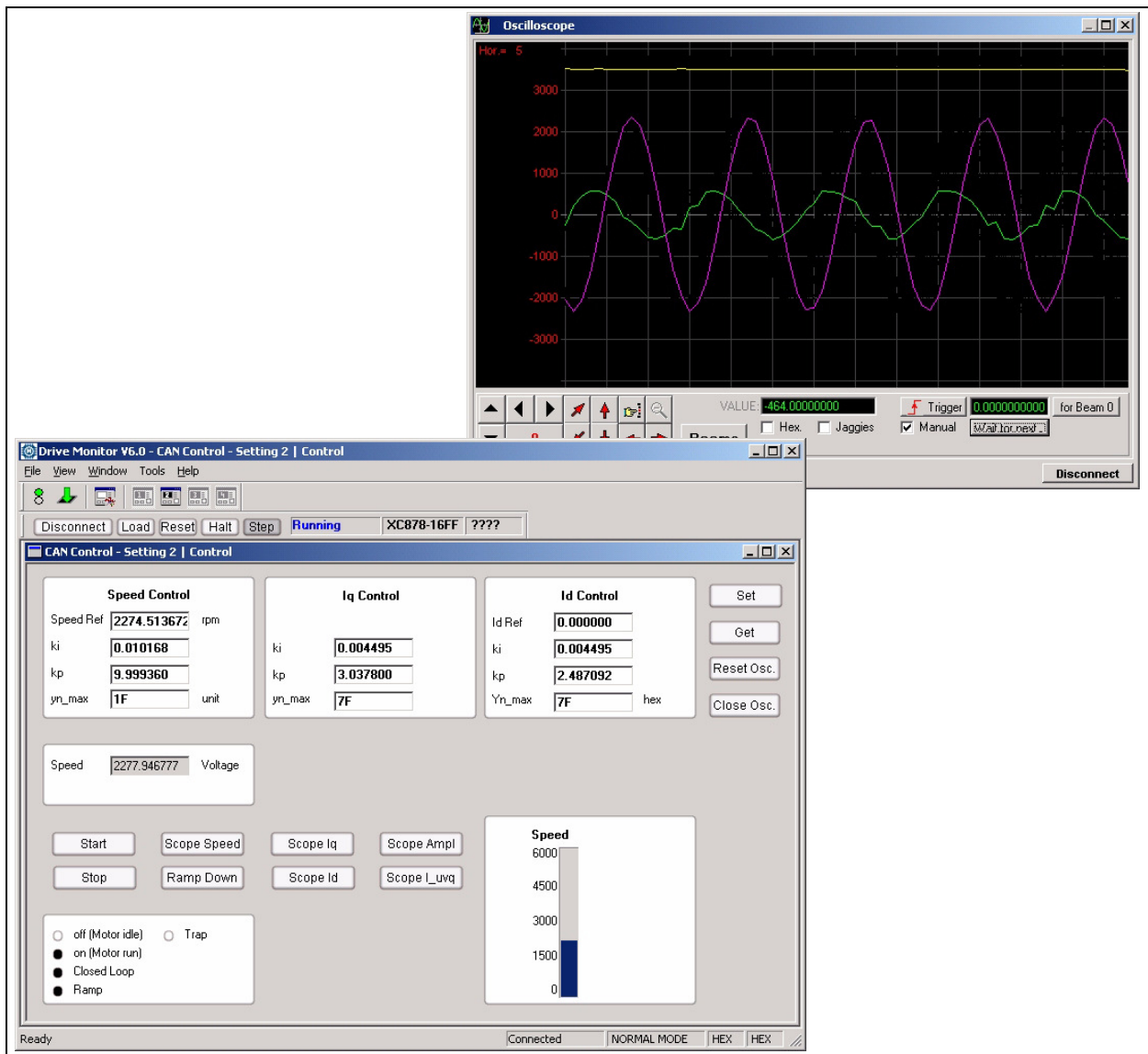


Figure 3 Screenshots of the DriveMonitor V6.0 Software

The main window of the DriveMonitor Software provides access to the target via the CAN control window and the JTAG control toolbar. The CAN control window is fully configurable and can be customized according to the target software. An oscilloscope window is also available to monitor up to three signals in real-time.

The DriveMonitor USB stick provides data rates up to 500 kBaud and a data throughput of about 300 kBaud. Because of the fast data streaming capability, the DriveMonitor is an ideal tool for motor control debugging.

2 PC Host Software (GUI)

The DriveMonitor Software V6.0 provides a versatile GUI based on CAN messages, as well as a low-level JTAG tool. The GUI and JTAG tool are described in detail in this section.

2.1 JTAG Toolbar



Figure 4 JTAG Tool Control

The JTAG Tool Control provides the following features:




- **Connect:** The DriveMonitor software connects as client to the Device Access Server (DAS).
- **Load:** After pressing the Load button, a hex-file can be selected for download to the target system. After download, the code is verified automatically.
- **Reset:** A hardware reset is performed on the target system and the MCU is halted.
- **Halt:** The target system is halted and the program counter address is shown.
- **Run:** The target system continues operation after a halt or reset.
- **Step:** The target system moves one step further and the program counter address is shown.



2.2 CAN Toolbar



Figure 5 CAN Tool Control

The CAN Tool Control provides the following features:

- **Connect** 
 - After selecting the connect button, the driver firmware is downloaded to the CAN controller XC886CM and placed on the DriveMonitor USB stick. After successful download, the button changes to a green light.
- **Download via CAN** 
 - A CAN Boot Strap Loader (BSL) is provided on the Infineon microcontroller devices. After clicking the CAN download button, a hex-file can be selected for download to the target system. After download, the code is verified automatically.
- **Open CAN Control Window** 
 - The CAN Control Window is described in [Section 2.3](#).

- **Open CAN Control Editor** 
 - The CAN Control Editor is used to customize the CAN Control Window, and is described in [Section 2.4](#).
- **CAN Control Settings** 
 - There are up to four different settings for the CAN Control window available. The settings can be configured individually and can be switched by choosing one of the four buttons in the CAN toolbar. The currently visible setup is indicated by the bold button. This allows very flexible use of the oscilloscope and the display fields for different states of the application; for example, at startup, at runtime, and at off-state.

2.3 CAN Control Window

The CAN Control Window provides six elements to display and setup data for a target application by using the CAN bus protocol:

- **Group Entries**
 - 3 x 4 hexadecimal or floating point values can be sent to (set) or read from (get) the target.
- **Display Field**
 - Six hexadecimal or floating point values can be displayed.
- **Buttons**
 - Eight buttons can execute state machine commands to the target.
- **Status Flags**
 - Sixteen status flags can show bitwise information.
- **Progress Bar**
 - Two progress bars can show hexadecimal or floating point values
- **Oscilloscope**
 - Three hexadecimal or floating point values can be displayed in a software oscilloscope.

All values can be 8 to 32 bits wide and are scaled by a given factor.

In [Section 2.4](#), the details of the protocol are described and the extent of the flexibility becomes apparent.

Figure 6 shows an example of a CAN control window, which uses some of the available elements.

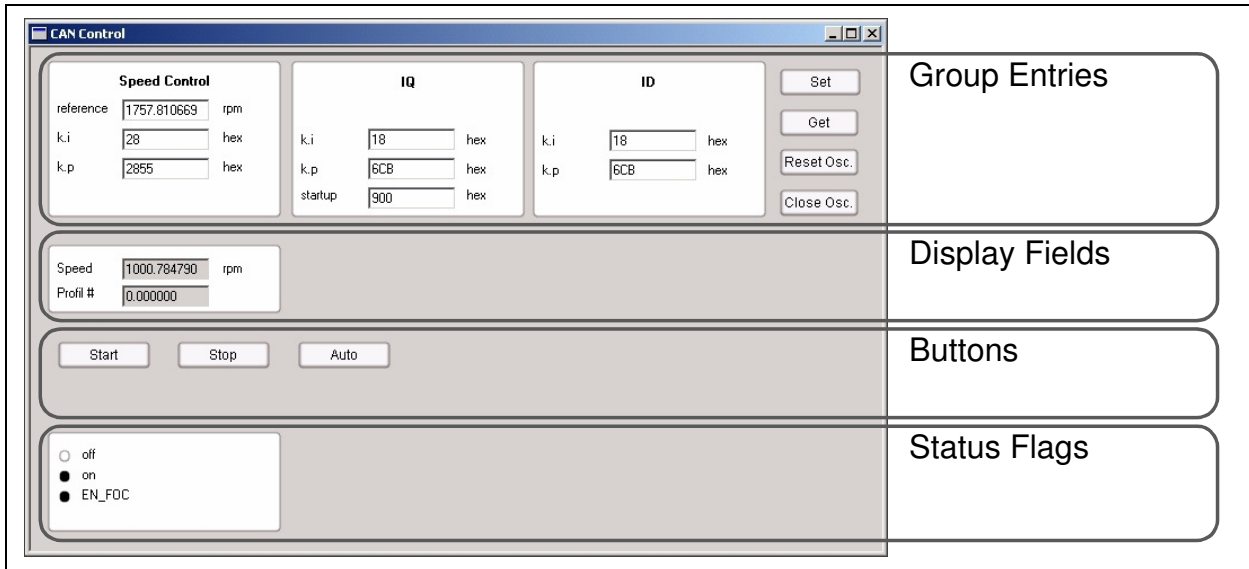


Figure 6 Example of a CAN Control Window

The CAN control window is designed to be used as a real-time communication channel to a real-time system such as motor control.

The example in **Figure 6** shows the control used for an FOC algorithm implemented on XC886CM:

- The group entries give access to the PI controllers of the algorithm
- The display field shows the actual speed
- The buttons are used to start and stop the motor
- The status flags show control specific information

2.4 Customizing the CAN Control Window

The CAN Control Window can be customized by defining details for the available elements. The CAN communication is based on a CAN frame of eight bytes, with the following specification:

Command	D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
Commands from DriveMonitor to target									
Set integer	adrL	adrH	valL	valH	0	0	size	SET	5
Get integer	adrL	adrH	valL	valH	0	0	size	GET	5
Button	0	0	0	0	0	0	0	BUT	5
Set Beams	green	0	pink	0	yellow	0	0	SB	5
Commands from target to DriveMonitor									
Match	adrL	adrH	xx	xx	xx	xx	xx	xx	57
Display	status flags		D2	D3	D4	D5	D6	D7	7
Oscilloscope	D0	D1	D2	D3	D4	D5	D6	D7	77

2.4.1 Group Entries

There are 12 group entries available in three groups, with four entries per group. A title can be defined for each group, and a label, unit and radix can be defined for each entry. The value from the group entry field is divided with a Scale Factor before being sent via a CAN message.

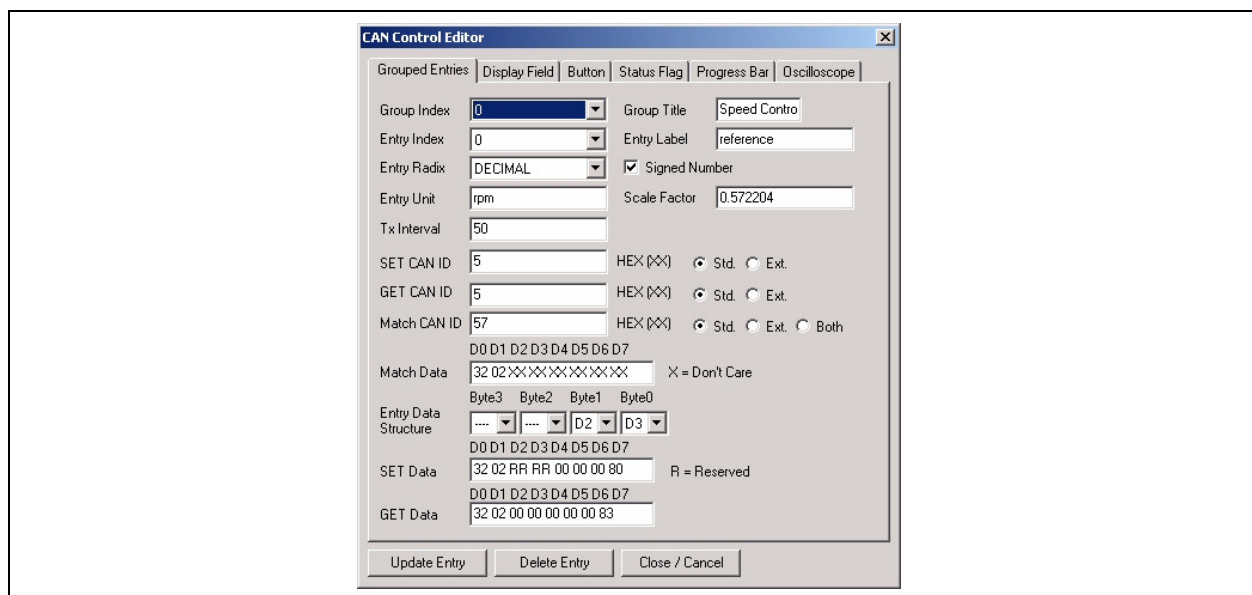


Figure 7 CAN Control Window: Group Entries

All group entries are transmitted to the target by pressing the <set> button.

In order to prevent data overflow at the target, a transmit delay (TX interval) can be specified.

Selecting the <get> button sends “get” commands for all group entry fields to the target. The target responds to each get command with a match response which contains the data.

2.4.2 Display Field

There are six display fields available. A label, unit and radix can be defined for each field. The received value from the CAN message is multiplied with a Scale Factor and is shown in the display field.

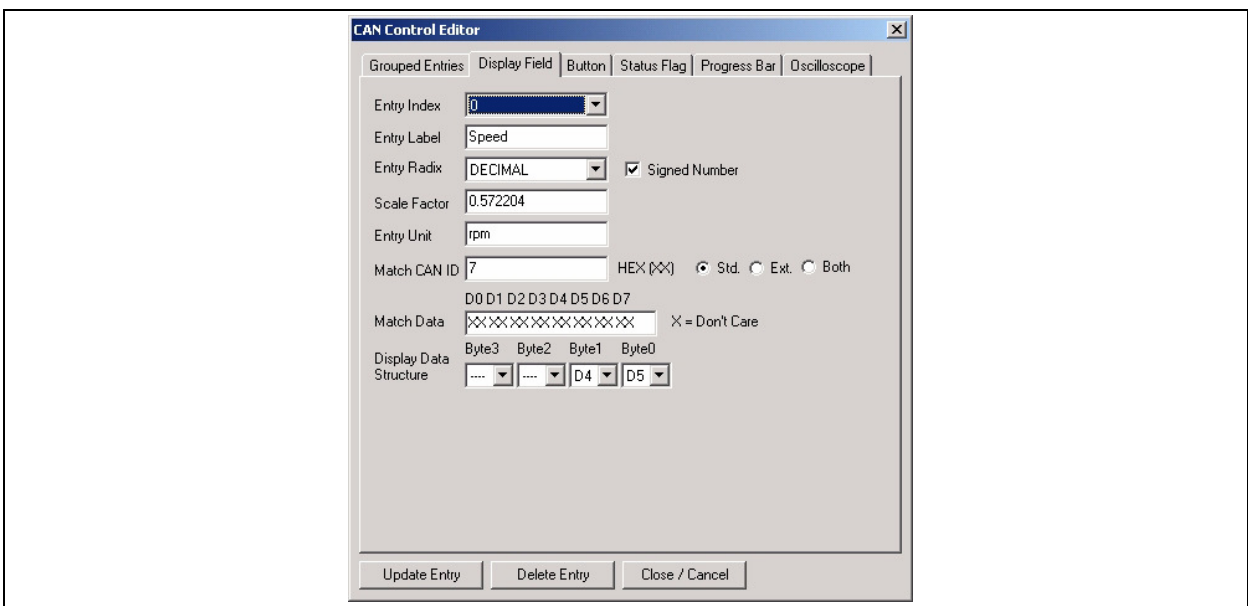


Figure 8 CAN Control Window: Display Field

2.4.3 Buttons

Eight buttons are available for free configuration.

For each button a label, the transmit CAN ID, and the data frame can be configured.

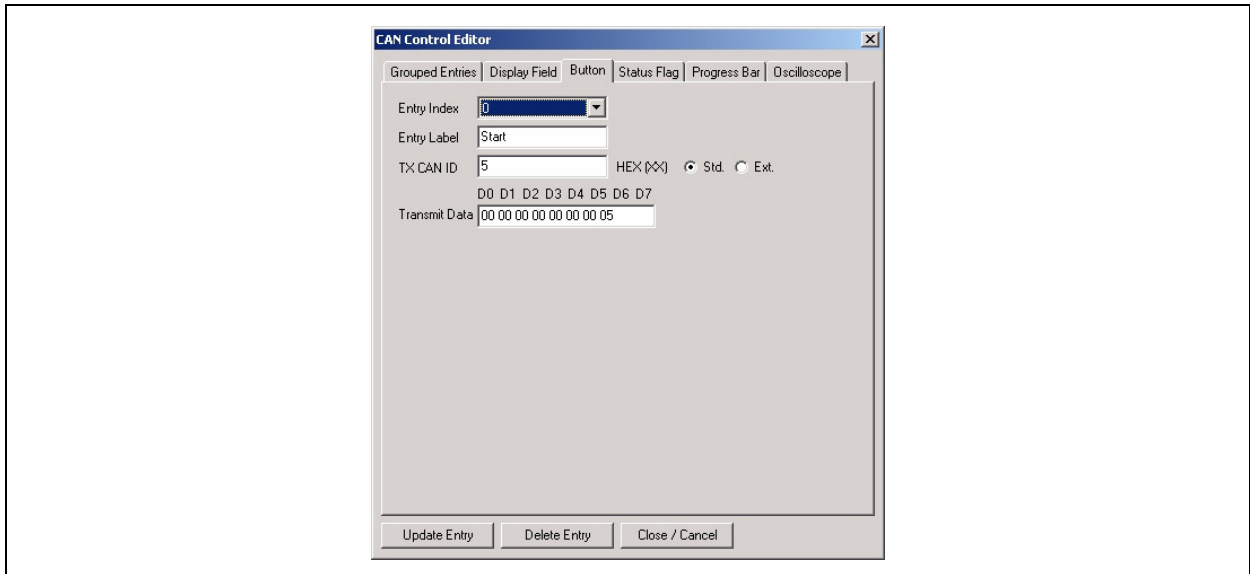


Figure 9 CAN Control Window: Buttons

2.4.4 Status Flags

16 status flags are available. They can be received in the same frame together with display data or oscilloscope data.

A label, CAN ID and Match data can be specified on byte level. The bitwise masking is taken from the masking data information. The flag is set when the logical AND combination of received data and masking data is not 0.

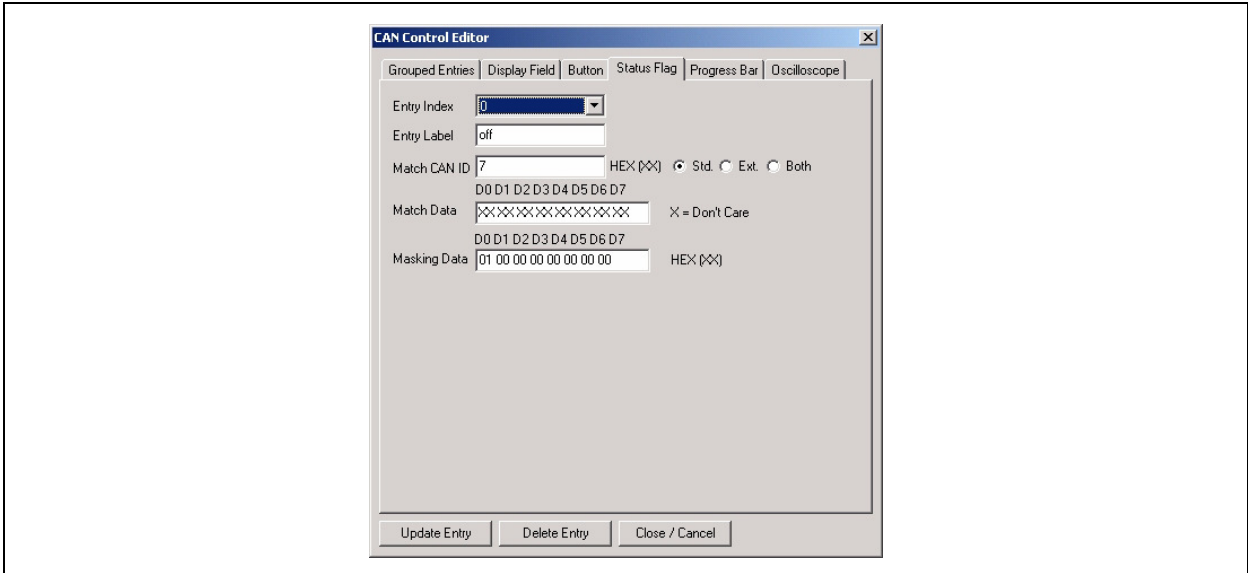


Figure 10 CAN Control Window: Status Flags

2.4.5 Progress Bars

Two progress bars are available. They are handled like display fields, but are displayed in a progress bar. The minimum and maximum level (lower range and upper range) can be specified.

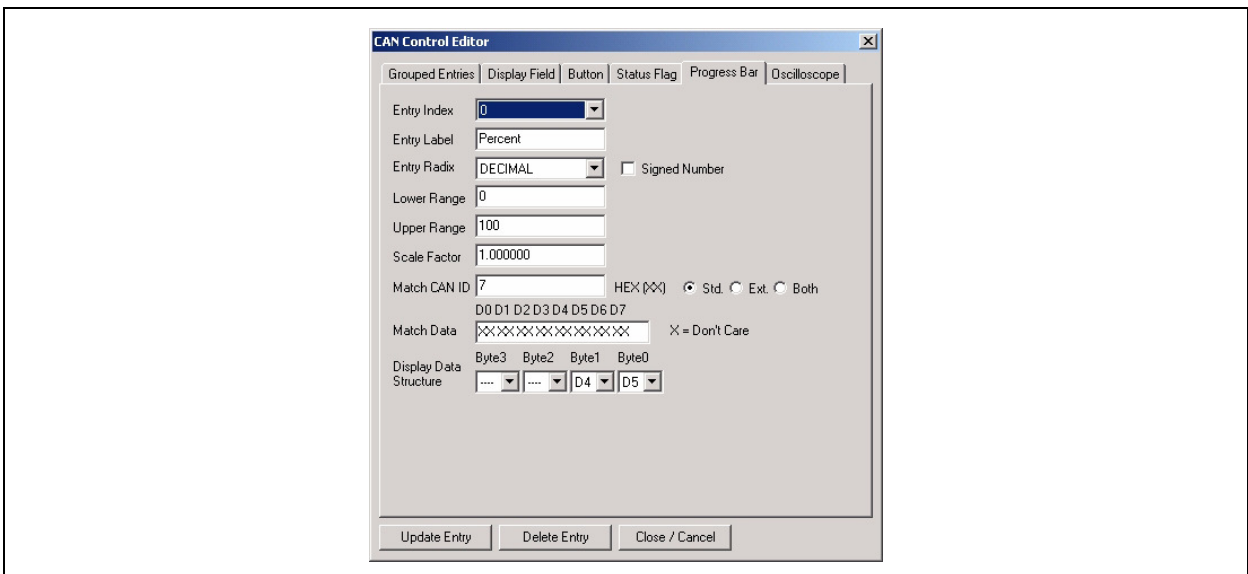


Figure 11 CAN Control Window: Progress Bar

2.4.6 Oscilloscope

A virtual oscilloscope with three beams is available.

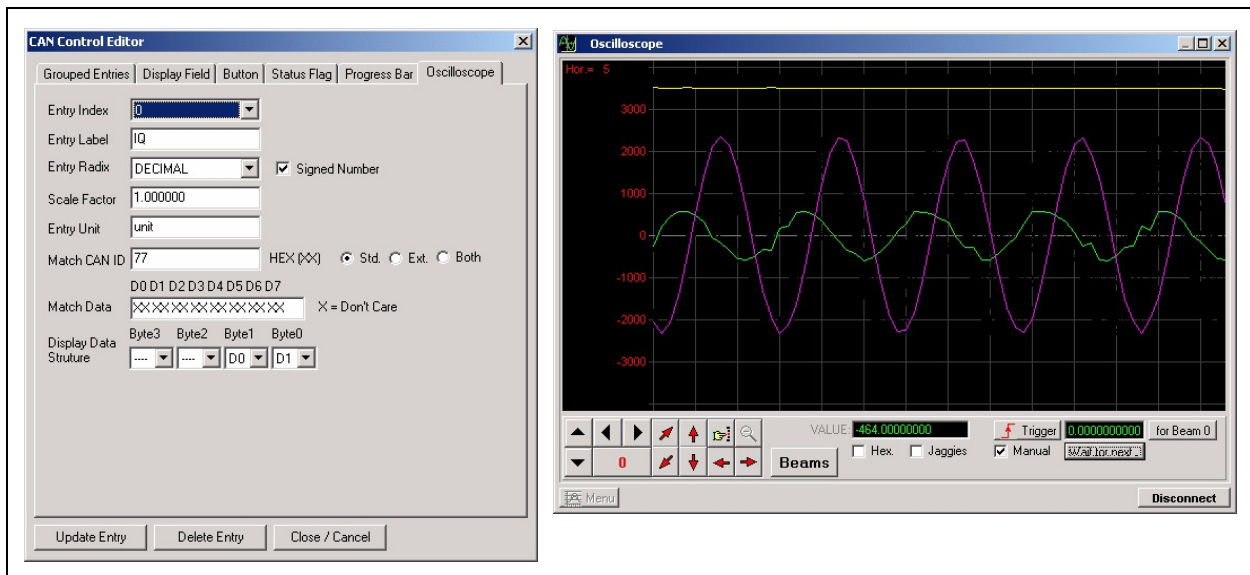


Figure 12 CAN Control Window: Oscilloscope

Oscilloscope GUI DLL

The virtual oscilloscope is realized by a freeware DLL.

<http://www.oscilloscope-lib.com/>

The author of this DLL calls it a “*Universal Real-Time Software Oscilloscope GUI DLL Library for data acquisition and logging, computer simulation and debugging programs*”

The library makes it possible to show on the beams of the oscilloscope over a million quantization steps of signal per second (on the Pentium III machine) – less than one micro-second is sufficient for one signal sample. This software can be used for linking to real-time controlling programs as there is no delay in relaying data to the oscilloscope. The data is instantly displayed in beams, the process of their relay and display is deterministic (and may be infinite). The relayed data is stored in the oscilloscope memory and can be displayed graphically as beams at any time, without the need for any “solution” such as decimation or excerption, strobe effect, and so on. It works regardless of the relay speed.

2.5 Troubleshooting

Sometimes the communication between Host and Target does not work as expected, or is cut-off completely. In particular, the virtual oscilloscope may show long response times or even pauses. If the connections and baud rates are set correctly, then there are essentially three different root causes of these problems:

- [Problems with the Target](#)
- [Problems with the DriveMonitor USB Stick](#)
- [Problems with the Host Computer](#)

2.5.1 Problems with the Target

It may happen that the target is dumping data too fast to the DriveMonitor USB stick. If the target starts CAN transmission before the DriveMonitor hardware is ready, CAN error messages may block the bus. In this instance, resetting the target while COM port connection is established may help. Sometimes the COM Port connection has to be re-established by clicking on the “Connect to COM Port” button (see [Section 2.2](#) for details).

It should be noted that CAN communication requires very accurate matching of the baud rate - the baud rate must be correctly configured. When using customized target hardware it is recommended to check the timing of the CAN_RXD and CAN_TXD signals on the target pins.

2.5.2 Problems with the DriveMonitor USB Stick

The CAN to USB bridge is implemented using the 8 bit microcontroller XC886CM. This device has to shuffle the data received from the USB side (FTDI, see [Chapter 4.3](#)) to the CAN side, and vice versa.

Once the DriveMonitor is connected to the host USB and the “Connect to COM Port” button is pressed, a small firmware package is downloaded to the bridge device. There are a few settings which can influence the communication behaviour, but the main one is the CAN baud rate. This can be adjusted in the U2CAN configuration menu. This menu is usually hidden but can be opened by selecting: **View >> Toolbars >> Advance**. A baud rate of 500 kbps at node 1 should be selected on both the target and the bridge device.

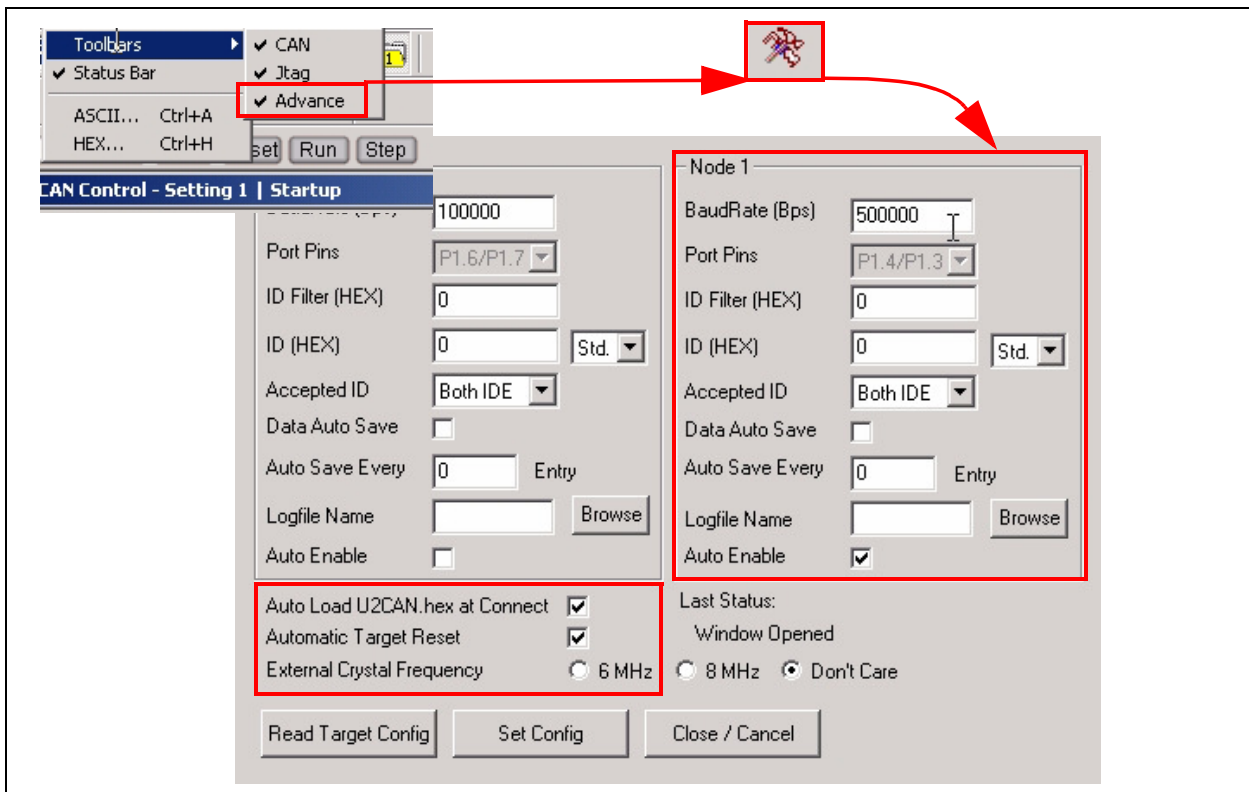


Figure 13 U2CAN Configuration Window in Advanced Toolbar

The pinout of the DriveMonitor Header (2 x 8 pins) has the same JTAG signals at the same position as the standard Infineon JTAG header. In addition the DriveMonitor offers a few more signals to support CAN (CANH and CANL), a UART (RXT and TXD) and to supply the DriveCard (5 V). These additional signals may conflict with other target hardware, especially the supply pins.

Note: Please note that Infineon Easy Kits and Starter Kits are usually supplied by a power plug or via USB. These kits offer the pin compatible JTAG header, so a hardware conflict would be caused if the DriveMonitor were connected to this header.

2.5.3 Problems with the Host Computer

The DriveMonitor needs some drivers on the PC side. Usually these drivers come with the DriveMonitor installation and need to be installed first.

The Infineon DAS (Device Access Server) is required. After installation has been completed the DriveMonitor enumerates with "DAS JTAG over USB plus CAN" and the Windows Device Manager (select **Start >> Control Panel >> System**) shows the following new entries:

- Ports (COM & USB): Infineon USB COM Port (COMxx)
- Universal Serial Bus Controllers: Infineon USB COM Port,
Infineon USB Debug Port

If the entries listed above are not visible the installation should be repeated as follows:

- If the DriveMonitor is plugged into the USB port, uninstall the "Unknown device" in the device manager window of your system control panel
- Remove the DriveMonitor from the USB port
- Install the DAS Software
- Plug in the DriveMonitor

USB hubs can show different behavior, such as long response times or even pauses on the USB transfer, which is visible on the virtual oscilloscope. In most cases this can be optimized by adjusting the properties of the Infineon USB COM Port in the advanced settings menu. Here the packet size (USB Transfer Sizes) for receive and transmit can be reduced to 128 bytes (default: 4096 bytes). Furthermore the Latency Timer (BM options) can be reduced to 1 msec (default 16 msec). This usually helps to get better performance when monitoring data. However some USB hubs cannot be adjusted properly. In this instance other PC hardware has to be used instead.

3 Target Firmware

This chapter describes the implementation of the target firmware for the XC800 family (XC886, XC888, XC878). The implementation for the XE166 family is very similar.

3.1 Program Flow

The program flow of the target microcontroller is an embedded real-time code which runs in several interrupt service routines. Timer T21 provides a system tick of 500us, which calls a scheduler. The scheduler state machine controls the application and reacts to host commands.

The CAN controller is configured to three interrupt vectors for receive, transmit and error handling. The CAN Message Objects have the following IDs:

- ID5 - receive object - MO 0: SET/GET command and Buttons
- ID55 - receive object - MO 2; unused
- ID7 - transmit object - MO1; slow data for status flags and display field
- ID77 - transmit object - MO 3; fast data for oscilloscope and progress bar
- ID57 - transmit object - MO 4; reply to GET command

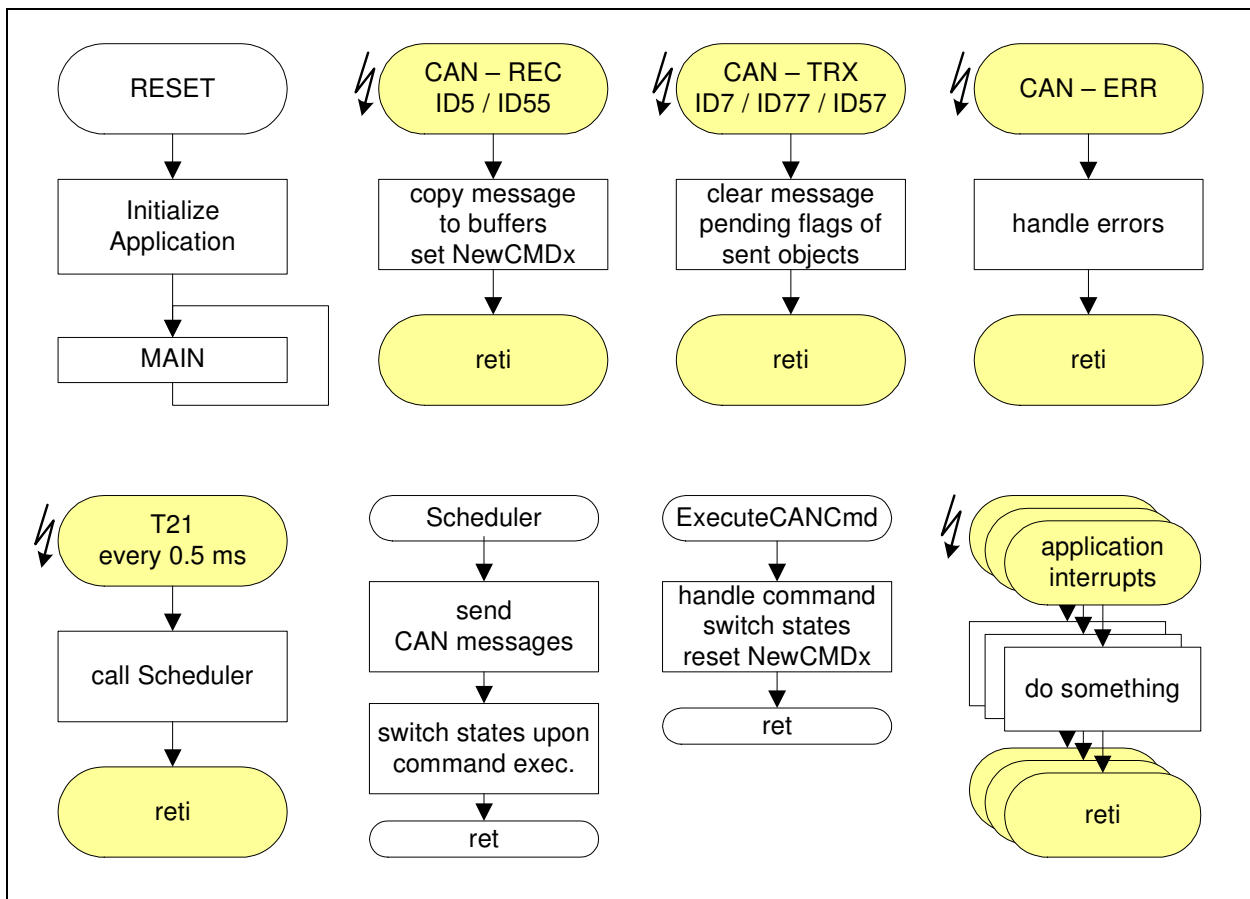


Figure 14 Program Flow of the Client Firmware

3.2 Command Structure

The command structure is described in [Chapter 2.4](#):

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
Commands from DriveMonitor to target								
adrL	adrH	valL	valH	0	0	sze	CMD	5

3.3 Receiving a CAN Message

Upon receipt of a CAN message object (MO 0 / MO 2 with ID5 / ID55) the receive interrupt is vectorized. The corresponding message object is then copied in to the eight byte wide receive buffer in case there is no command pending. The global command variable is updated and a bit indicating a new command is set.

The following shows a code extract of the CAN Receive Interrupt Service Routine (ISR):

```

if (gb_NewCMD0==0) // write to buffer if ready for new data
{
    CAN_vWriteCANAddress(CAN_MODATAL0); // access MO_0 low part
    CAN_vReadEN(); // and read
    CANRxdBuf0[0] = CAN_DATA0;
    CANRxdBuf0[1] = CAN_DATA1;
    CANRxdBuf0[2] = CAN_DATA2;
    CANRxdBuf0[3] = CAN_DATA3;

    CAN_vWriteCANAddress(CAN_MODATAH0); // access MO_0 high part
    CAN_vReadEN(); // and read
    CANRxdBuf0[4] = CAN_DATA0;
    CANRxdBuf0[5] = CAN_DATA1;
    CANRxdBuf0[6] = CAN_DATA2;
    CANRxdBuf0[7] = CAN_DATA3;
    guc_NewCMD0 = CANRxdBuf0[7]; // update command variable
    gb_NewCMD0 = 1;
}

```

3.3.1 Command Execution

At every system timer tick the scheduler state machine is executed and the bit indicating a new command is polled. If set, the corresponding command is executed and the bit is cleared afterwards in order to accept new commands.

This handshaking mechanism with the receive interrupt ensures that an incoming command has to be served first before a new command is accepted.

The following shows a code extract of the CAN Receive Interrupt Service Routine:

```

if (gb_NewCMD0) // gb_NewCMD0 is set in CANreceive interrupt XINTR6INT
{
    ExecuteCANCmd();
    gb_NewCMD0=0;
}

```

3.3.2 Command SET

With the SET command, the variable with the SET address is updated. It is therefore necessary to have the address information of the respective variables. Address and data size information are read from the receive buffer and the contents of the address is written with the new value.

The following shows a code extract of the `ExecuteCANCmd()` function:

```
case CMDSET: // adrL / adrH / valL / valH / 0x00 / 0x00/ sze / CMDSET
    adr = (unsigned char*) CANRxdBuf0[0];
    sze = CANRxdBuf0[6];
    i = CANRxdBuf0[2];
    *adr++ = i;
    if( sze == 2 )
    {
        i = CANRxdBuf0[3];
        *adr = i;
    }
    break;
```

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
Commands from DriveMonitor to target								
adrL	0	valL	valH	0	0	sze	SET	5

3.3.3 Command GET

With the GET command the host can request to read the contents of a variable. The host has to send the address and size information first before the target responds with the data on transmit message object MO 4.

The following shows a code extract of the `ExecuteCANCmd()` function:

```
case CMDGET: // adrL / adrH / valL / valH / 0x00 / 0x00/ sze / CMDGET
    adr = (unsigned char*) CANRxdBuf0[0];
    sze = CANRxdBuf0[6];

    // Transmit MO 4 on request
    ...
    break;
```

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
From DriveMonitor to target								
adrL	0	valL	valH	0	0	sze	GET	5

3.3.4 Start, Stop and Ramp Down Buttons

When selecting the Start, Stop, and Ramp Down buttons, the state variable of the scheduler is updated and will be entered with the next timer tick.

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
From DriveMonitor to target								
0	0	0	0	0	0	0	Button	5

3.3.5 Button Scope

Real-time monitoring of fast changing data is one of the key features of the DriveMonitor. Therefore the CAN message object MO 3 - ID 77 is sent with every system tick to the host. With this message, three 16 bit values can be monitored on the three beams of the virtual oscilloscope. With the button scope the addresses of the three monitoring variables can be changed.

The following shows a code extract of the `ExecuteCANCmd()` function:

```

case CMDSETBEAMS:
    // change monitoring variables for scope beams
    // adr0 | 0x00 | adr1 | 0x00 | adr2 | 0x00 | xx | CMDSETBEAMS
    //   green   |   pink   |   yellow   |
    CANTrxBuf1[3] = CANRxdBuf0[0]+1;
    CANTrxBuf1[2] = CANRxdBuf0[0];
    CANTrxBuf1[5] = CANRxdBuf0[2]+1;
    CANTrxBuf1[4] = CANRxdBuf0[2];
    CANTrxBuf1[7] = CANRxdBuf0[4]+1;
    CANTrxBuf1[6] = CANRxdBuf0[4];
    break;

```

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
From DriveMonitor to target								
green	0	pink	0	yellow	0	0	SB	5

3.4 Transmitting a CAN Message

There are three transmit message objects configured for sending data from the target to the host. The transmit Interrupt Service Routine is vectorized once the CAN message is completed, and the corresponding transmit pending flag is cleared. Each transmit message object has a different trigger event.

The transmit messages have to be initialized. The transmit buffers `CANTrxBuf0[8]` and `CANTrxBuf1[8]` are written with the address of the data to be transmitted. This is done in function `Scope_vInit()`.

```
void Scope_vInit( void )
{
// slow CAN transfer for display fields sent at CAN-ID 7
CANTrxBuf0[0] = (unsigned char) &Status_word+1;
CANTrxBuf0[1] = (unsigned char) &Status_word;
CANTrxBuf0[2] = (unsigned char) &gi_delta_angle+1;
CANTrxBuf0[3] = (unsigned char) &gi_delta_angle;
CANTrxBuf0[4] = (unsigned char) &gi_Speed+1;
CANTrxBuf0[5] = (unsigned char) &gi_Speed;
CANTrxBuf0[6] = (unsigned char) &t21+1;
CANTrxBuf0[7] = (unsigned char) &t21;

// fast CAN transfer for oscilloscope sent at CAN-ID 77
//green (beam 0)
CANTrxBuf1[3] = (unsigned char) &gi_Speed+1;
CANTrxBuf1[2] = (unsigned char) &gi_Speed;
//pink (beam 1)
CANTrxBuf1[5] = (unsigned char) &gi_Speed_reference+1;
CANTrxBuf1[4] = (unsigned char) &gi_Speed_reference;
//yellow (beam 2)
CANTrxBuf1[7] = (unsigned char) &gi_V_q+1;
CANTrxBuf1[6] = (unsigned char) &gi_V_q;
//not used in scope but possible in progress bar
CANTrxBuf1[1] = (unsigned char) &gi_Amplitude+1;
CANTrxBuf1[0] = (unsigned char) &gi_Amplitude;
} // End of function Scope_vInit();
```

The transmitting function reads the contents of the address of the transmit buffer `CANTrxBuf0[8]` and `CANTrxBuf1[8]`, and copies it to the `CAN_DATAx` access mediator register. The CAN message is then transmitted by the function `CAN_vTransmit(#MO)`. The following shows this code snippet:

```
void CAN_Transmit_MO1()
{
// MO1 is shown as Value
CAN_DATA0 = *((unsigned char data *) CANTrxBuf0[0]);
CAN_DATA1 = *((unsigned char data *) CANTrxBuf0[1]);
CAN_DATA2 = *((unsigned char data *) CANTrxBuf0[2]);
CAN_DATA3 = *((unsigned char data *) CANTrxBuf0[3]);
CAN_vWriteCANAddress(CAN_MODAL1);
CAN_vWriteEN(ALL_DATA_VALID);

CAN_DATA0 = *((unsigned char data *) CANTrxBuf0[4]);
CAN_DATA1 = *((unsigned char data *) CANTrxBuf0[5]);
CAN_DATA2 = *((unsigned char data *) CANTrxBuf0[6]);
```

```

CAN_DATA3 = *((unsigned char data *) CANTrxBuf0[7]);
CAN_vWriteCANAddress(CAN_MODATAH1);
CAN_vWriteEN(ALL_DATA_VALID);

CAN_vTransmit(1); // Send MO1 via CAN bus
}

```

3.4.1 Oscilloscope and Progress Bar

With every system timer tick the data for the soft oscilloscope and the progress bar are sent to the host; i.e. it is synchronous to the scheduler tick. This data has the highest transmit rate. The user has to ensure that the real-time conditions are met. This can be tuned with the T21 overflow rate. The fast data uses ID77 with MO 3.

Command	D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
From target to DriveMonitor									
oscilloscope	D0	D1	beam 0: green	beam 1: pink	beam 2: yellow	77			

3.4.2 Status Flags and Display Fields

For data that changes at a slower rate, such as the status flags and display fields, the data transmit rate does not have to be so frequent. The transmit rate is defined by a variable which counts a defined number of system timer ticks. This 'slow data' uses ID7 with MO 1.

The following shows a code extract of the synchronous transmit trigger event at the beginning of the Scheduler() function:

```

if( guc_countMO1-- == 0 )
{
    guc_countMO1 = CAN_MO1_RATE;
    CAN_Transmit_MO1();
}

if( gb_Off == 0) // Send continuous fast data only, when motor is started
{
    CAN_Transmit_MO3();
}

```

Command	D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
From target to DriveMonitor									
display	status flags		D2	D3	D4	D5	D6	D7	7

3.4.3 Command GET

The GET command is user-initiated; i.e. asynchronous. Once the GET command is received the scheduler state machine responds to the requested data with transmit message object MO 4 / ID57.

The following shows a code extract of the ExecuteCANCmd() function:

```

case CMDGET: // adrL / adrH / valL / valH / 0x00 / 0x00/ sze / CMDGET
    adr = (unsigned char*) CANRxdBuf0[0];
    sze = CANRxdBuf0[6];

    // Transmit MO 4 on request
    CAN_DATA0 = adr;
    CAN_DATA1 = 0x00;
    CAN_DATA2 = * (unsigned char data *) adr;
    CAN_DATA3 = * (unsigned char data *) (adr+1);
    CAN_vWriteCANAddress(CAN_MODATAL4);
    CAN_vWriteEN(ALL_DATA_VALID);
    CAN_DATA0 = 0x00;
    CAN_DATA1 = 0x00;
    CAN_DATA2 = sze;
    CAN_DATA3 = CMDGET;
    CAN_vWriteCANAddress(CAN_MODATAH4);
    CAN_vWriteEN(ALL_DATA_VALID);
    CAN_vTransmit(4); // Send MO4 via CAN bus
    break;

```

D0	D1	D2	D3	D4	D5	D6	D7	CAN ID
From target to DriveMonitor								
adrL	0	valL	valH	0	0	sze	GET	57

3.4.4 Example for Get and Set Commands in Grouped Entries

The following figure shows an example of grouped entries for SET and GET commands on the host and target.

The screenshot shows the 'Grouped Entries' configuration window with the following parameters:

- Group Index: 0
- Group Title: Speed Control
- Entry Index: 0
- Entry Label: Speed Ref
- Entry Radix: DECIMAL
- Entry Unit: rpm
- Scale Factor: 1.716614
- Tx Interval: 75
- SET CAN ID: 5
- GET CAN ID: 5
- Match CAN ID: 57
- Match Data: 54 00 XX XX XX XX XX XX XX
- Entry Data Structure: D2 D3
- SET Data: 54 00 RR RR 00 00 02 80
- GET Data: 54 00 00 00 00 00 02 83

Below the window, a table maps these parameters to host and target data structures:

Command	ID	TRX	ADL	ADH	RR	RR	00	00	SIZE	Value
CMDSET	ID5	TRX	ADL	ADH	RR	RR	00	00	80	80
CMDGET	ID5	TRX	ADL	ADH	00	00	00	00	83	83
	ID57	REC	ADL	ADH	data	data	00	00	83	83

Figure 15 Grouped Entries for SET and GET Commands on Host and Target

4 Hardware Description

The DriveMonitor USB Stick consists of three main blocks:

- The USB interface provides the JTAG and VCOM port
- The XC886CM bridges the UART to a CAN protocol
- A connector provides all interfaces to the target system

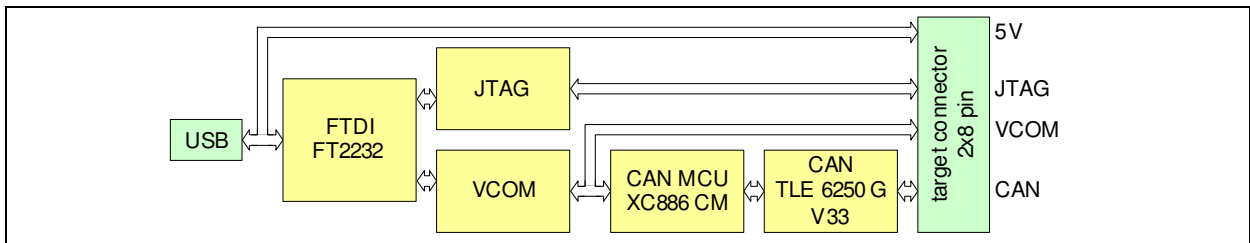


Figure 16 Block Diagram of DriveMonitor

4.1 USB Interface

The USB interface is provided by a dual Multi-Purpose UART / FIFO controller (FT2232C) which can be configured individually in several different modes. For the DriveMonitor it is configured as follows:

- Channel A makes use of the Multi-Protocol Synchronous Serial Engine interface which provides the synchronous serial protocol for JTAG.
- Channel B is configured as a fast UART with FIFO.
- The EEPROM configures the FT2232 in order to appear as an USB 2.0 full speed device (12 Mbit/s).

Hardware Description

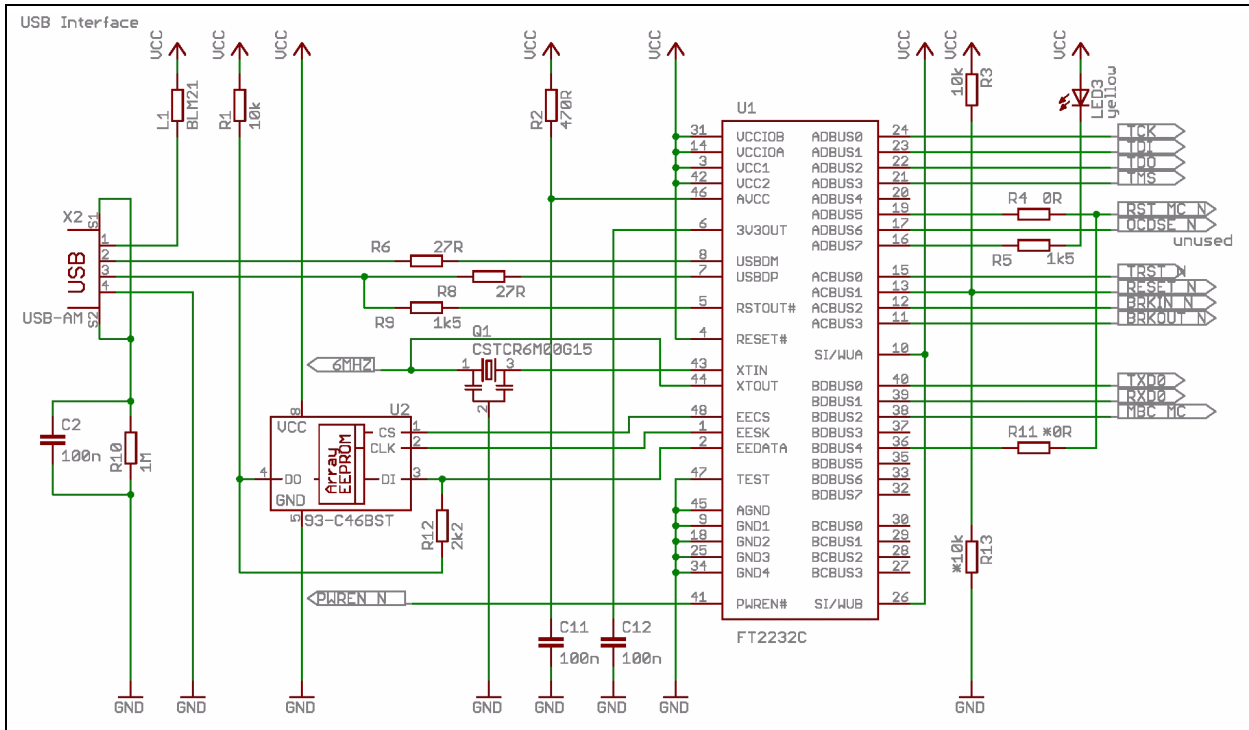


Figure 17 USB Interface

The driver software is included in the DAS (Device Access Server) architecture which provides one single interface for all types of debug tools fulfilling all the performance and reliability requirements.

The tool interface is on the software level (DAS API) and implemented in a generic DLL. It provides the abstraction of the physical device connection, which becomes a parameter value in the connection setup phase. During operation the physical connection (e.g. JTAG) is fully transparent to the tool. The DriveMonitor can therefore be used for JTAG debugging with all third-party debugger tools that support the DAS interface.

The UART interface is provided as a standard virtual COM port and is visible for all applications. It can be configured in the Windows Control Panel.

For software updates and further details, please refer to:

<http://www.infineon.com/DAS>

4.2 CAN Microcontroller XC886CM

The XC886CM provides advanced networking capabilities by integrating a CAN controller (V2.0B active) and up to 32 KByte of embedded Flash memory on a single chip. The on-chip CAN module reduces the CPU load by performing most of the functions required by the networking protocol (masking, filtering and buffering the CAN frames).

The XC886/888CLM offers an optimized fit to a wide range of CAN networking applications including automotive body applications, control for industrial and agricultural equipment, building control for lifts/escalators, intelligent sensors, distributed I/O modules, and industrial automation. For further details, please refer to:

<http://www.infineon.com/XC886>.

In the DriveMonitor USB stick the XC886 is used as a UART-CAN bridge. Most of the on-chip peripherals are unused, although a user LED is connected to port 3.1.

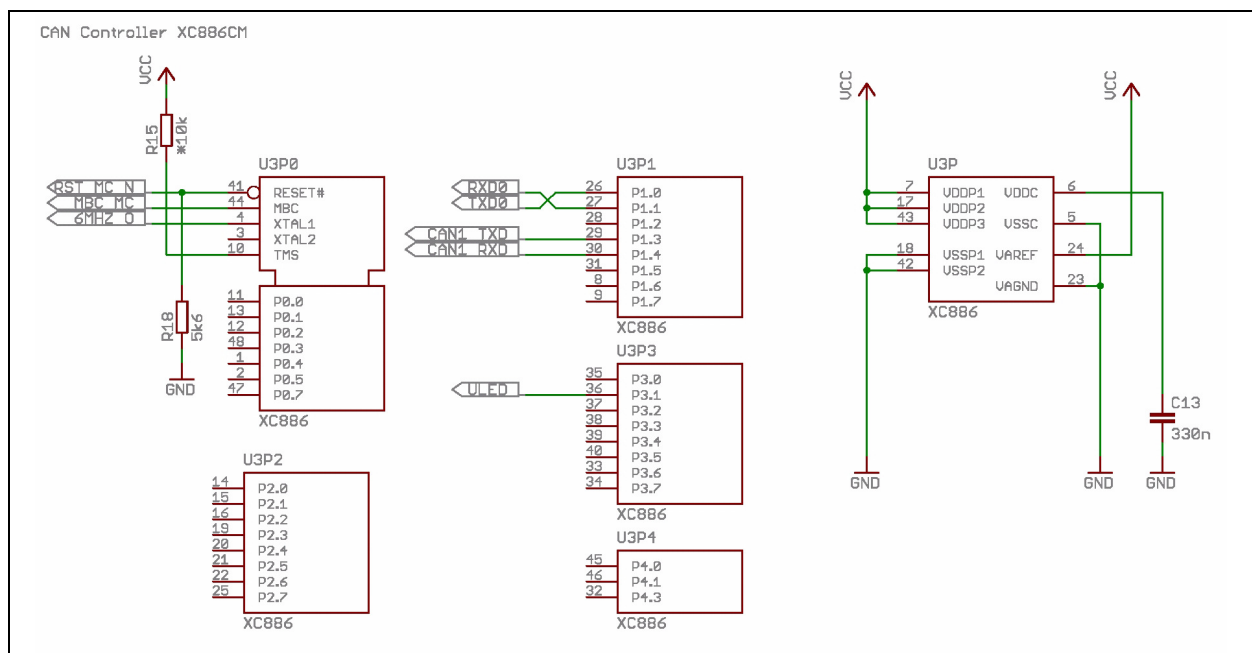


Figure 18 CAN Microcontroller XC886CM

The clock for the DriveMonitor USB stick is a common 6 MHz clock for FT2232 and XC886CM. A ceramic resonator is connected to the FT2232 device. The output (signal 6 MHz) of the integrated oscillator circuit is taken and amplified by a single buffer device whose output is connected to the clock input of p3.0 of XC886CM (signal 6 MHz_O).

4.3 CAN Transceiver and Target Connector

The CAN interface of the XC886CM is connected to the CAN transceiver TLE 6250GV33. The CAN-Bus is terminated by a 120 Ω resistor.

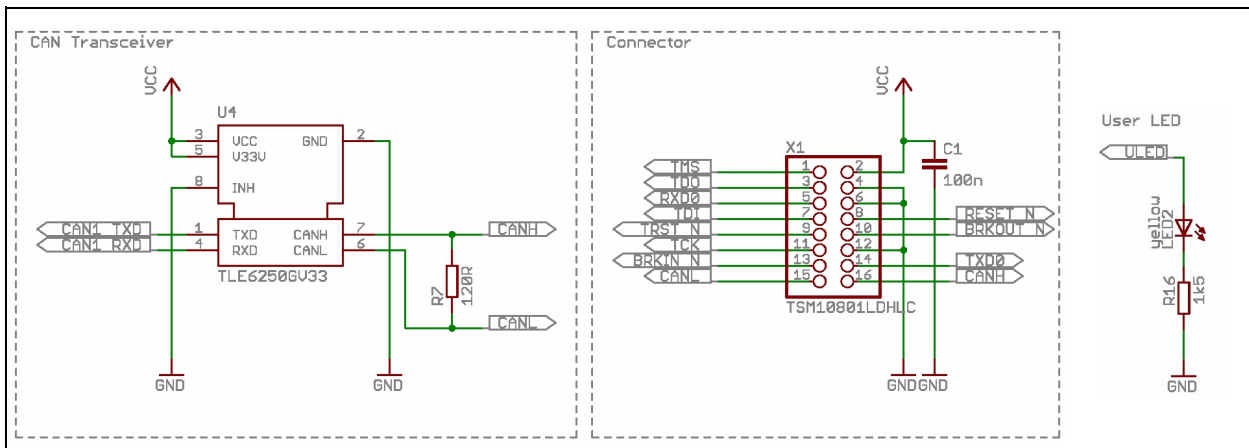


Figure 19 CAN Transceiver and Target Connector

The target connector provides all signals at a 2 x 8 pin header in an extended OCDS pin configuration. The CAN and VCOM signals are placed to ensure that the DriveMonitor can be used with a very wide variety of evaluation kits.

When used with Infineon DriveCards, the target connector can be used directly with a 16-pin cable. These DriveCards provide a digital isolation for CAN and JTAG and need a 5 V supply for the isolation devices.

Note: When using with Infineon Easy Kits or Starter Kits, ensure that the 5 V VCC supply (pin 2 of target connector) is NOT connected to the OCDS connector, because the standard OCDS connector provides the supply voltage of the target to a level shifter. If there is a power supply of 5 V at the target, the DriveMonitor can be used directly. At a 3.3 V target, level shifters must be used.

4.4 PCB Layout

The following figures show the PCB layout of the DriveMonitor.

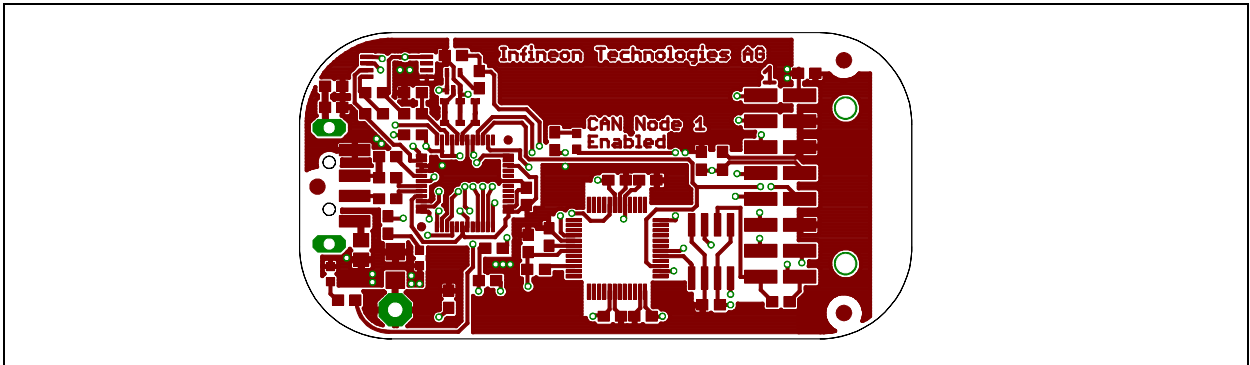


Figure 20 Top Layer

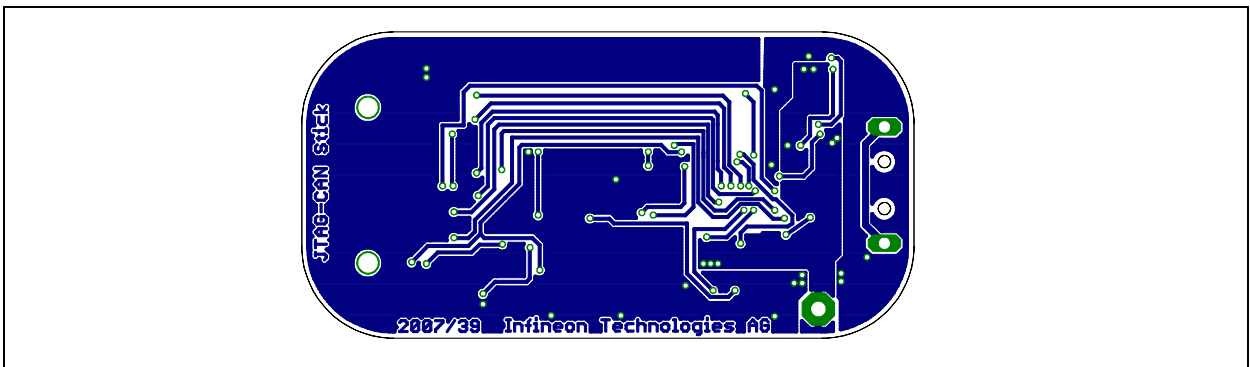


Figure 21 Bottom Layer

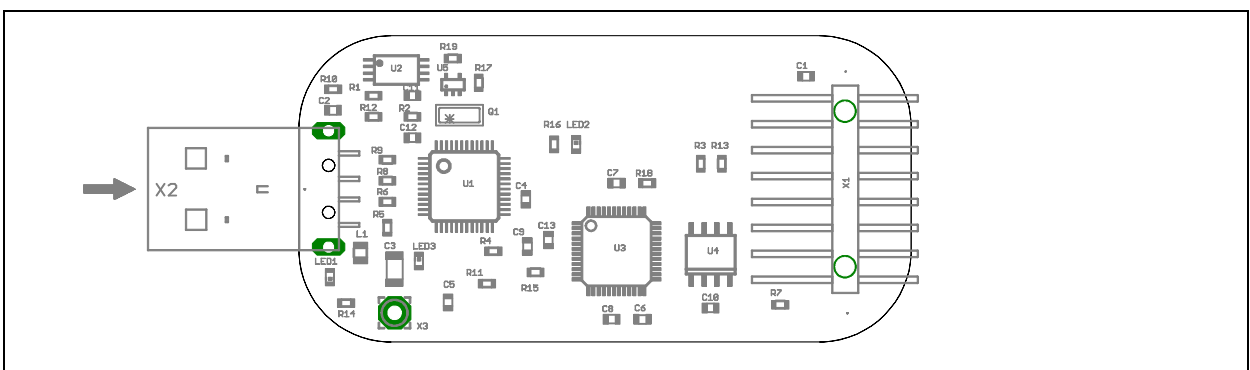


Figure 22 Components

www.infineon.com

Published by Infineon Technologies AG