

OVERVIEW

This application note demonstrates programming and simulation of the on-chip CAN interface of the C167 / ST10-F168. These devices integrate a full CAN controller. The Infineon C167CS includes a dual CAN controller that can be programmed and tested using the Keil 166/ST10 development tools.

This application note provides sample source code that can be compiled using the Keil C166 compiler. To test the program you may use the μ Vision2 debugger/simulator that is part of the Keil PK166 package. Even the Keil PK166 Evaluation Version that is available at www.keil.com can be used, since the program of this sample CAN application fits into the limitations of the evaluation version. You may adapt the sample CAN application and re-use it for your own programs.

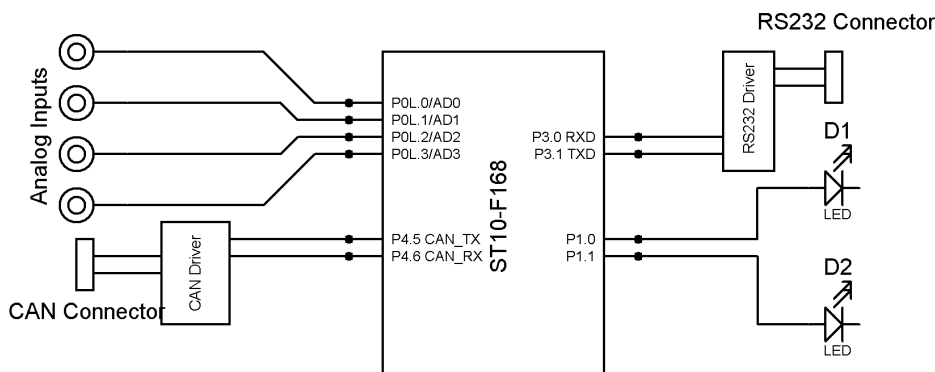
This application note gives you a detailed insight into the usage of the on-chip CAN controller. The sample CAN application uses several different methods to transmit and receive CAN objects:

- Simple send and receive routines that use status polling.
- Interrupt driven send and receive routines.
- Request remote frames and automatically reply to remote frames.

The CAN (Controller Area Network) is a serial bus originally developed for use in automobiles. It is finding additional applications in other areas such as factory automation. The physical layer is usually a differential twisted wire pair. This application note assumes some familiarity with CAN and its associated terminology. Introduction documents on CAN can be accessed through links on the Keil web site (www.keil.com/can).

HARDWARE

The sample CAN application can be used with a single chip hardware, if the controller owns an on-chip flash ROM. Below is the schematic for a ST10 F168 single chip hardware.



The sample CAN application uses the hardware components as follows:

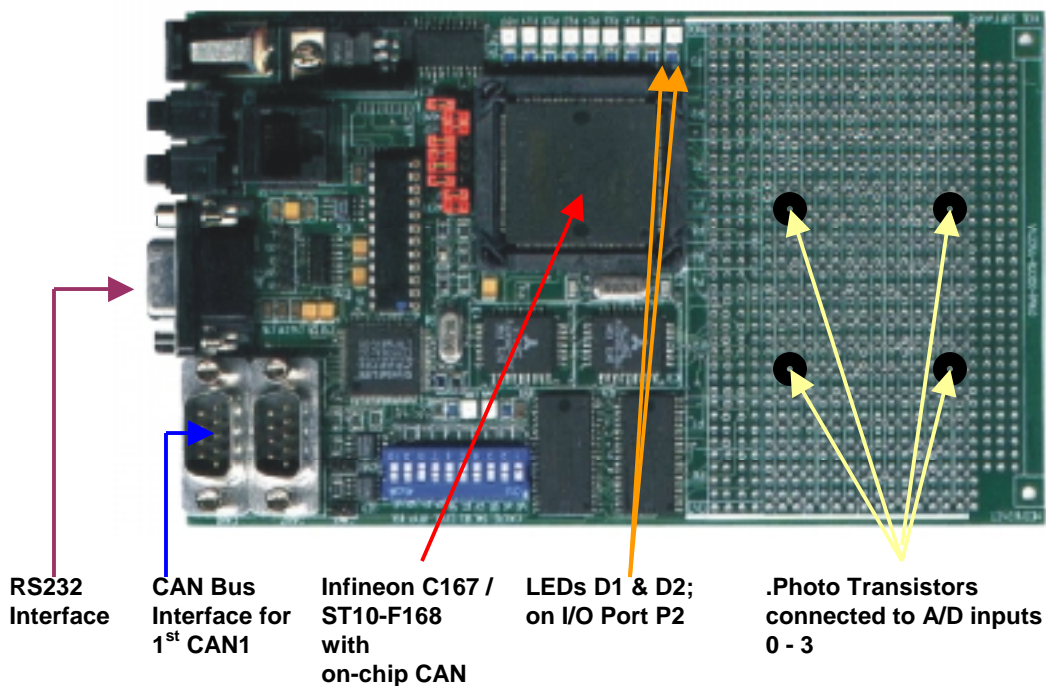
- The CAN connector is used to connect the board to a CAN bus.
- The two LEDs signal the status of one of the incoming CAN messages.
- The analog inputs provide input for the remote frames.
- The RS232 connector is used to output text messages and to receive commands.

The functions are described in detail under “Re-use of the sample CAN application” on page 8.

You may use the μ Vision2 simulator to test the sample CAN application without real hardware.

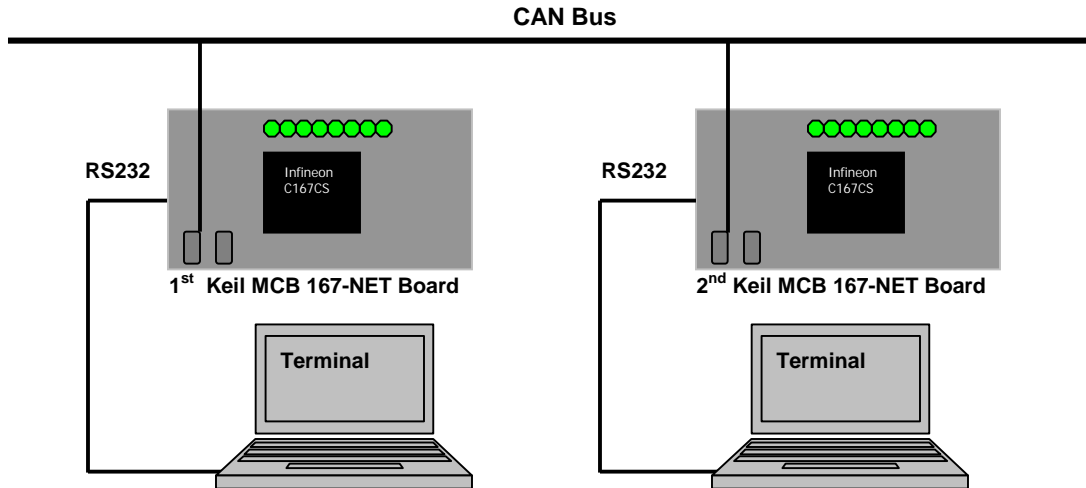
Another alternative is to use evaluation boards, like the Keil MCB167-NET. This standard hardware provides all components needed for running the sample CAN application. The sample CAN application that we are supplying, is configured for the Keil MCB167-NET

The picture below shows the Keil MCB167-NET board. To provide input for the A/D converter we have connected four photo transistors to the A/D inputs.



To run the sample CAN application you may use two identical hardware boards that are connected via CAN Bus. For text output and command input, each hardware board might be connected to a PC with terminal emulation or a real terminal.

The complete hardware is shown below:



USING THE SOFTWARE

Characters which are sent through the RS232 trigger outgoing CAN messages. Any status change of these commands is transmitted via CAN object 1 to the opposite CAN board. The commands are used to generate additional messages that are listed in the table below:

Terminal Command	Description
0x10	The application will send a long text string over channel 3 in interrupt mode.
0x20	The application sends a remote frame request over channel 5.

Channel 1 sends the status of RS232 input in polling mode.

The two LEDs signal the RS232 commands received from the opposite CAN board.

The A/D input values AIN0-AIN3 are captured and stored to the automatic reply message object for the remote frame mode. If the character 0x20 on the opposite controller is sent, this data is requested.

The RS232 connector is also used to output text messages. Text messages are the text strings, that are sent when 0x10 or 0x20 are received on the serial interface.

SOURCE FILES

The following table gives you an overview of the files that are included in this application note:

Filename	Description
CANT1.UV2 CANT1.OPT	µVision2 project file that allows you to built and test the sample CAN application.
MAIN.C	Includes the main routine and calls the CAN interface routines.
CAN1.C	Includes the CAN interface routines and configures the message objects including the CAN ID's for the user application.
CAN1.H	Extern definitions for the CAN interface routines.
DEBUGGER.INI	Debug-Script that defines functions for I/O Simulation.

You can build the project within µVision2 with the following commands:

- Open the project file **CANT.UV2** with **Project – Open Project**.
- Translate and link the project with **Project – Build Target**. This generates object files, listing files, and the executable program in absolute and Intel HEX format.

The screenshot shows the Keil µVision2 IDE. The top window displays a C code snippet:

```

unsigned int  sv;

for (i = 0; i < 4; i++) {
    sv = ((*(v) >> 8) | (*(v) << 8));
    printf ("AIN%04.2f ", i, (f1
    v++;
}

```

The bottom window shows the build log for 'Target 1':

```

Build target 'Target 1'
compiling MAIN.C...
compiling CAN1.C...
assembling Start167.a66...
linking...
creating hex file from "CanT1"...
"CanT1" - 0 Error(s), 0 Warning(s).

```

DOWNLOAD THE APPLICATION VIA MONITOR

The executable file can be loaded into memory of the MCB167-NET with the µVision2. Setting up the MCB167-NET board and µVision for monitor debugging is described in the manual for the MCB167-NET Evaluation Board. To enable monitor download, make sure the Bootstrap mode is active by setting the dipswitch BSL to on.

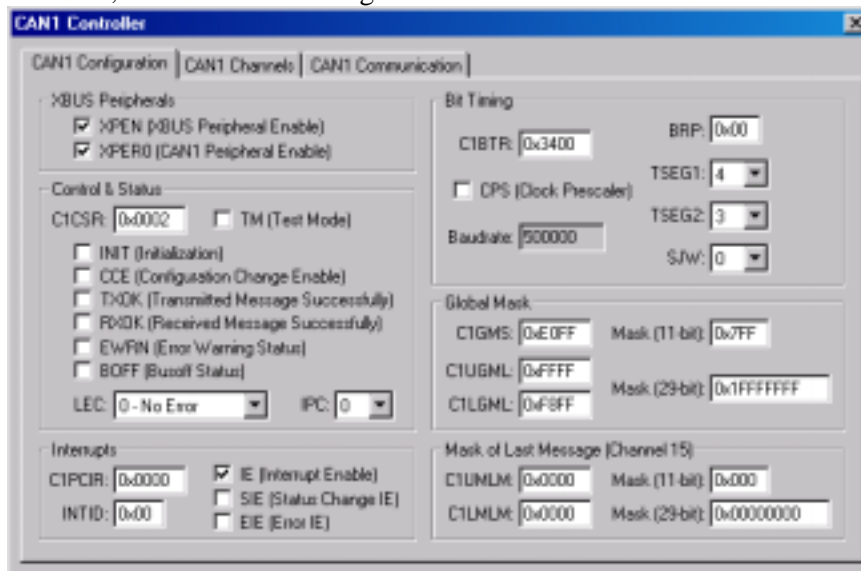
PROGRAMMING THE APPLICATION TO FLASH


The executable HEX file, which is generated by the compiler can be programmed into the flash memory of a of the ST10-F168 device. ST provides a tool, “ST10-Flasher”, that can do this. Setting up the MCB167-NET board for flash programming is described in Application Note 157 “Using the ST10 FLASHER Tool with the MCB167-NET Board”.

SIMULATION WITH THE μ VISION2 DEBUGGER

 Start the μ Vision2 simulator with **Debug – Start/Stop Debug Session**.


- μ Vision2 simulates the behavior of all on-chip peripherals. In case of the Infineon C167 or the ST10-F168, also the CAN controller is fully simulated. You can review the status by using the **Peripherals** menu items.
- **Peripherals – CAN** opens the CAN dialog, that shows the CAN configuration, the CAN channels, and the CAN messages that are transmitted via the CAN network.



 The project in this application note includes a **Debugger.INI** file. This file defines μ Vision2 debug functions that provide simulation for CAN message I/O. The **Debugger.INI** file is specified under **Options for Target – Debug – Initialization File** and therefore automatically loaded at startup of the μ Vision2 simulator. This file defines also Toolbox buttons that allow quick access to the debug functions. The Toolbox opens with **View – Toolbox** and provides the following buttons:

Toolbox Button	Description
Send S1 Command	Simulates a pressed button S1 of the PHYTEC board
Send S2 Command	Simulates a pressed button S2 of the PHYTEC board
Send S1+S2 Command	Simulates two pressed buttons S1 and S2 of the PHYTEC board.
Receive S1+S2 Command	Simulate an incoming "Switches on" status message, which is send with CAN ID 101h;
Receive Button Reset	Simulate an incoming "Switches off" status message, which is send with CAN ID 101h;
Send Remote Frame	Generates the reply frame for the remote frame, requested by ID 104h
Simulate Reception Traffic	Send a 5*8 Bytes long string, which is received by buffered channel 3/4 in interrupt mode.

Each toolbox button invokes a debug function that is defined in the **Debugger.INI** file. Details of the CAN simulation with μ Vision2 can be found in the Application Note 147, CAN Simulation in μ Vision2.

 When you execute the program with **Debug – Go** you can review the CAN traffic in the CAN communication dialog page. By using the Toolbox buttons described above you can generate CAN messages.

- You may use all features of the μ Vision2 debugger. You may single-step through the code or set breakpoints. You can use the memory windows, the performance analyzer, the code coverage feature, or just view some variables.

The content of the file **DEBUGGER.INI** is listed below. The functions of this file are available during debugging. This **.INI** file is automatically loaded at start of a debug session since it is specified under **Options for Target – Debug**.

```

/* =====DEBUGGER.INI=====
* Contains debug functions for the sample CAN project (App_Nt 166).
*
* - PressSwitch enables simulation of buttons on memory mapped I/O
* - CANMessage prints message information of send messages
* - SendInfo sends 8Byte messages on a arbitrary ID
* - SimTraffic simulates sending of messages in a short interval
*
* =====
*/

/*
* Simulate a 0.1 Second Click on Switch I/O Port (for S1 & S2)
*/
SIGNAL void PressSwitch (unsigned char v) {
    unsigned char cx;

    S0OUT = v;                // set I/O Port value for switch v
    swatch (0.1);            // press switch for 0.1 seconds
}

/*
* Define Buttons for the toolbox with presets for PressSwitch */
define button "Send S1 Command",    "PressSwitch (1)"
define button "Send S2 Command",    "PressSwitch (2)"
define button "Send S1+S2 Command", "PressSwitch (3)"

/*
* Print the last message sent by the controller

```

```

*/
FUNC void CANmessage (void) {
    switch (CAN1OUT) {
        case 1: printf("\nSend Message (11-bit ID=%04X)", CAN1ID);    break;
        case 2: printf("\nSend Message (29-bit ID=%08X)", CAN1ID);    break;
        case 3: printf("\nRequest Message (11-bit ID=%04X)", CAN1ID); return;
        case 4: printf("\nRequest Message (29-bit ID=%08X)", CAN1ID); return;
    }
    printf("\nMessage Length %d, Data: ", CAN1L);
    printf("%02X %02X %02X %02X ", CAN1B0, CAN1B1, CAN1B2, CAN1B3);
    printf("%02X %02X %02X %02X\n", CAN1B4, CAN1B5, CAN1B6, CAN1B7);
}

/*
 * Set Breakpoint on CAN Output VTREG
 */
BS WRITE CAN1OUT, 1, "CANmessage()"

/*
 * Send Information on any ID
 */
FUNC void SendInfo (unsigned long id,    // message ID
                   unsigned char len,   // message length
                   unsigned char val) { // 1. value byte
    CAN1ID = id;        // Set VTREG that keeps next ID
    CAN1L = len;        // Set VTREG with message length
    CAN1B0 = val;       // Set the data registers
    CAN1B1 = val+1;
    CAN1B2 = val+2;
    CAN1B3 = val+3;
    CAN1B4 = val+4;
    CAN1B5 = val+5;
    CAN1B6 = val+6;
    CAN1B7 = val+7;
    CAN1IN = 2;        // Send message to simulated controller
}

/*
 * Send 5 messages in short intervals on ID 103h
 */
SIGNAL void SimTraffic (void) {
    int i;
    for (i=0; i<=5; i++) {
        SendInfo(0x103,8,'0'+(i*8));    // Send CAN message on ID 103h
        swtch(0.00025);                // wait 0.00025sec before next message is send
    }
}

/*
 * Define Buttons for the toolbox for easy access to SendInfo and SimTraffic
 */
define button "Receive S1+S2 Command", "SendInfo(0x101,1,0x30)"
define button "Receive Button Reset", "SendInfo(0x101,1,0x00)"
define button "Send Remote Frame", "SendInfo(0x104,8,0x00)"
define button "Simulate Reception Traffic", "SimTraffic()"

```

RE-USE OF THE SAMPLE CAN APPLICATION

The files **candrv.h** and **candrv.c** of this application note can be re-used in your own CAN projects. The can driver provides the following functions:

Functions	Description
CAN1Init	Initialization of the CAN Controller.
CAN1Send	Sends a message over a channel. The message length must not exceed channel definition
CAN1Read	Reads a message received by a channel. The message length is specified in the channel definition.
CAN1SendIsr	Interrupt driven Send Routine.
CAN1ReadIsr	Interrupt driven Receive Routine.
CAN1SetRemote	Set the data for remote requested frame
CAN1ReadRemote	Read received data after a remotely requested frame
CAN1ReqRemote	Request data via remote frame
CAN1Interrupt	The interrupt service routine for the CAN Controller.

The file **candrv.c** needs to be adapted for your application as described below:

- The section CAN ID definitions specifies the CAN message objects.
- The table **id_typ** needs entries for all CAN message objects.
- Under CAN baud rate calculation you can configure the baud rate of the CAN bus.
- In the function **CAN1Init** all CAN message objects need to be initialized.
- Adapt the interrupt driven CAN I/O routines to the requirements of your application.
- To add or remove interrupt driven CAN message objects, adapt the function **CAN1Interrupt**.

The include file **CAN1.h** enables you to access the functions of the file **candrv.c** in own source files.

CONCLUSION

This sample CAN application shows that the implementation of a CAN interface is straightforward. You may adapt this sample CAN application for your own software projects. You may also convert this application software for other on-chip CAN controllers.

The Keil C166/ST10 development tools allow you fast and reliable development of complete applications. With the μ Vision2 simulator you can debug the application without real hardware. μ Vision2 simulates the behavior of all on-chip peripherals, including complex peripherals like the CAN controller and the A/D converter. In fact, this sample CAN application was completely tested with the simulator before we checked it in real hardware.

This application note uses the μ Vision2 debug functions to simulate external hardware components as well as CAN communication. Specific information on CAN simulation can be found in **Application Note 147 “CAN Simulation in μ Vision2”**.

It is even possible to extend the build-in peripherals of μ Vision2 with custom define peripherals. This is described in the **Application Note 154 “Implementing μ Vision2 DLLs for Simulating User Defined Hardware”**.

Copyright © 2001 Keil Software, Inc. All rights reserved.

In the USA:
Keil Software, Inc.
1501 10th Street, Suite 110
Plano, TC 75074
USA

Sales: 800-348-8051
Phone: 972-312-1107
FAX: 972-312-1159

E-mail: sales.us@keil.com
support.us@keil.com

Internet: <http://www.keil.com/>

In Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn b. Munchen
Germany

Phone: (49) (089) 45 60 40 - 0
FAX: (49) (089) 46 81 62

E-mail: sales.intl@keil.com
support.intl@keil.com