

## CAN Simulation with $\mu$ Vision2

## Implementation Suggestion for Temic CANray

July 26, 2000, Munich, Germany

by Hans Schneebauer, Keil Elektronik GmbH hs@keil.com ++49 89 456040-14

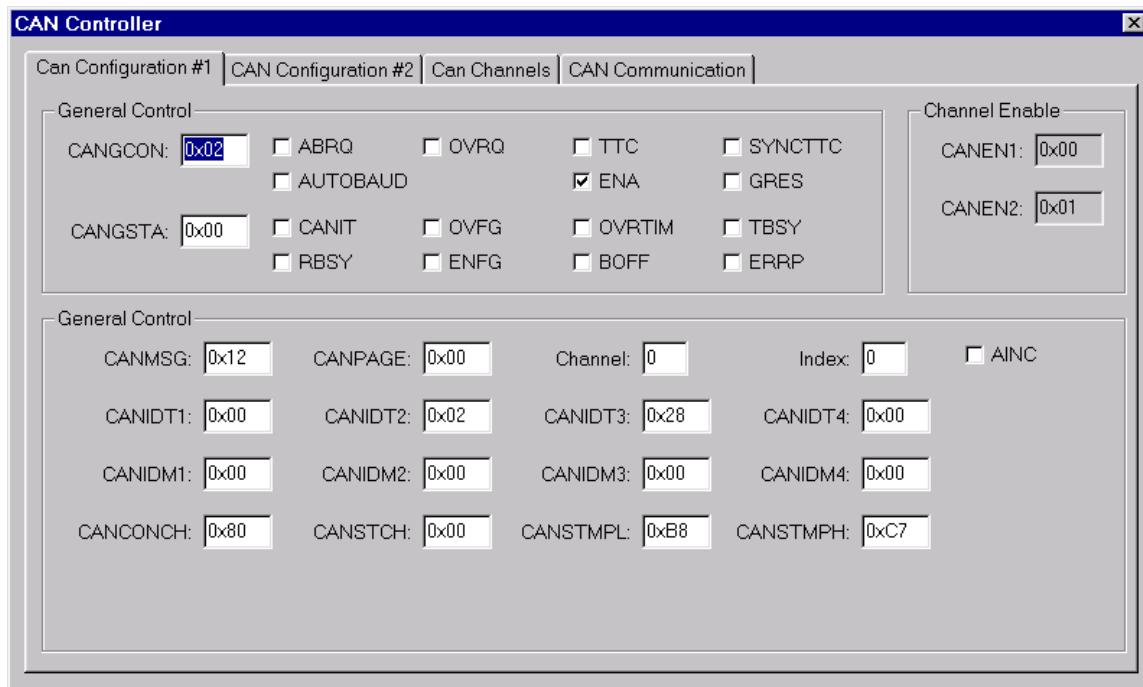
This Implementation suggestion describes the CAN simulation with  $\mu$ Vision2. With  $\mu$ Vision2 it is possible to:

- Simulate incoming messages and outgoing messages
- View the CAN communication and the parameters of the CAN controller
- write signal functions that supply messages automatically.

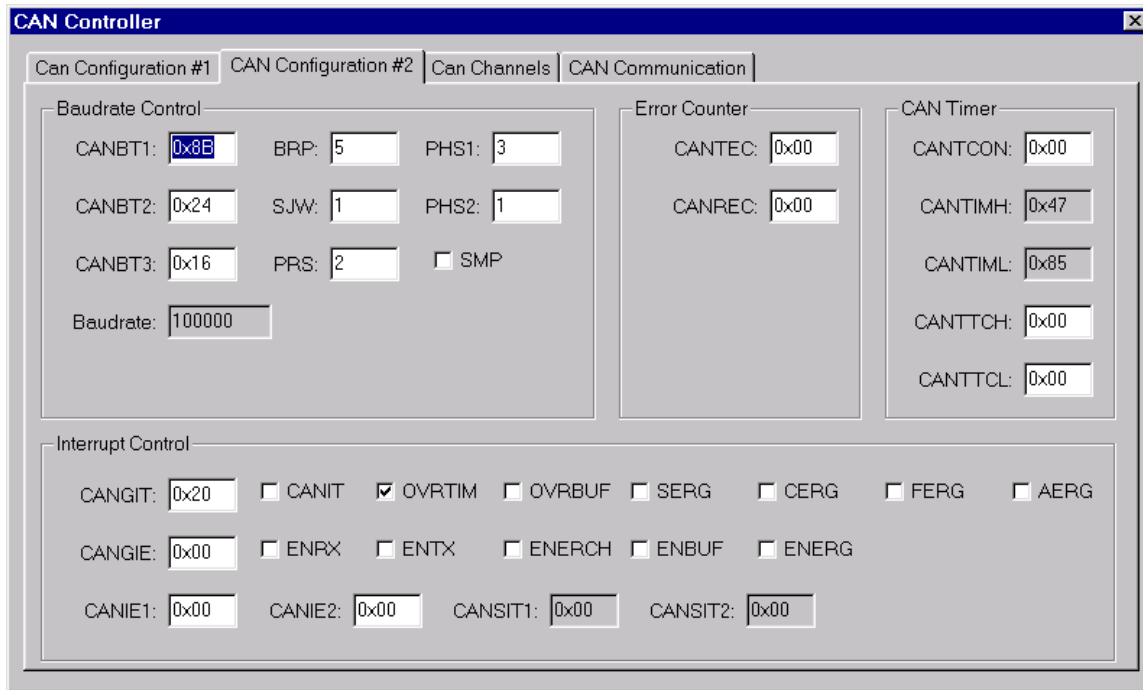
### Peripheral Dialog for CAN

The  $\mu$ Vision2 Debugger offers a CAN dialog that allows you to review the actual settings of the CAN interface. This dialog opens via the main menu under **Peripherals – CAN**.

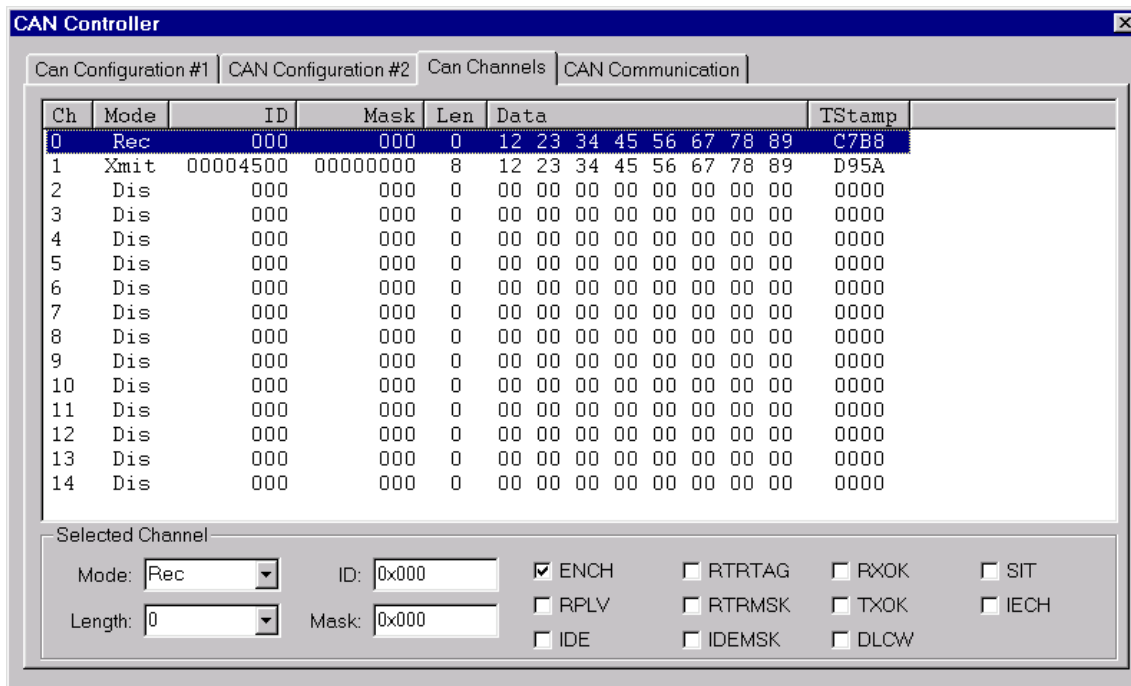
This dialog offers several tabs that are described in the following.



**CAN Configuration #1:** review and modify the general CAN controller registers. (Message ID Mask registers, ect.)



**Configuration #2:** review and modify the CAN Baudrate and Interrupt registers.



**CAN Channels:** review the CAN message object registers including status information

CAN Controller													
Can Configuration #1		CAN Configuration #2		Can Channels		CAN Communication							
No	States	ID	dir	Ch	Len	Data							
0	3186	100	Xmit	1	8	55	AA	BB	11	22	33	44	66
1	4303774	00004500	Rec	0	8	12	23	34	45	56	67	78	89
2	4308290	00004500	Xmit	1	8	12	23	34	45	56	67	78	89
3	5099297	064	Rec	0	8	98	87	76	65	54	43	32	21
4	5102596	064	Xmit	1	8	98	87	76	65	54	43	32	21
5	5597091	00004500	Rec	0	3	12	23	34					
6	5600250	00004500	Xmit	1	3	12	23	34					
7	6392631	064	Rec	0	3	98	87	76					
8	6394575	064	Xmit	1	3	98	87	76					
9	7317659	075	Rec	0	0	Remote Frame							
10	7318834	075	Xmit	1	0	Remote Frame							
11	8476288	00004500	Rec	0	3	12	23	34					
12	8479449	00004500	Xmit	1	3	12	23	34					
13	9947999	00004500	Rec	0	8	12	23	34	45	56	67	78	89
14	9952513	00004500	Xmit	1	8	12	23	34	45	56	67	78	89

**CAN Communication:** review incoming and outgoing messages. This window works similar to a CAN analyzer and shows all messages on the simulated CAN BUS.

---

## Virtual Simulation Registers (VTREG)

The  $\mu$ Vision2 Debugger implements virtual simulation registers (VTREG) that can be used to review the CAN communication on the Debugger command line level or within Debug and Signal functions. The following registers are implemented:

VTREG	Description
<b>CAN0ID</b>	Is an 11-bit or 29-bit identifier of the message currently transferred. The ID size is specified with the values in the CAN0IN or CAN0OUT VTREG. For short 11-bit identifiers only the 11 LSB bits are used of this VTREG.
<b>CAN0L</b>	The data length of the CAN message. Valid values for CAN0L are 0...8.
<b>CAN0B0 ... CAN0B7</b>	The data bytes of the CAN message. 8 data bytes are implemented to access the individual message bytes of the current object.
<b>CAN0IN</b>	Is set by the user or within debug functions to simulate incoming messages. The following values are possible: CAN0IN = 1 receive the current message using a 11-bit identifier. CAN0IN = 2 receive the current message using a 29-bit identifier. CAN0IN = 3 request a remote frame from the application program with matching 11-bit identifier. CAN0IN = 4 request a remote frame from the application program with matching 29-bit identifier. CAN0IN = 0xFF simulate BUSOFF mode of the CAN controller.  CAN0IN is set to 0 by the simulator when the message has been processed by the $\mu$ Vision2 simulator.
<b>CAN0OUT</b>	Is set by the simulator when transmitting a message by the application program. The following values are possible: CAN0OUT = 1 transmit the current message using a 11-bit identifier. CAN0OUT = 2 transmit the current message using a 29-bit identifier. CAN0OUT = 3 request a remote frame with matching 11-bit identifier from the user or debug function. CAN0OUT = 4 request a remote frame with matching 29-bit identifier from the user or debug function.
<b>CANTIMING</b>	Allows you to set a performance factor that controls the simulated communication timing within $\mu$ Vision2. This performance factor simulates a busy CAN network. With a performance factor of 0 an CAN network with infinite baudrate is simulated. With a factor of 1 the CAN messages are transferred in real-time taking care of the current selected communication baudrate. A factor of 2 simulates the performance that is identical to 50% of the communication baudrate. CANTIMING is a floating point value that can be between 0 .. 1000.

The VTREG naming conventions **CAN0xx** is used to allow also the simulation of devices with dual CAN interfaces. On such devices the VTREG names of the 2<sup>nd</sup> CAN controller are named **CAN1xx**

## Simulate Incoming Messages

The following example shows the simulation of an incoming message object with 2 message bytes.

```
>CAN0ID = 0x4500 // message ID = 0x4500
>CAN0L = 2 // message length 2 bytes
>CAN0B0 = 0x12 // message data byte 0
>CAN0B1 = 0x34 // message data byte 1
>CAN0IN = 2 // set CAN message with 29-bit identifier to the application program
```

---

### NOTE

*You may also enter incoming messages directly in the CAN communication dialog page.*

---

## Process Outgoing Messages

By using a breakpoint on the CAN0OUT register it is possible to invoke debug functions. The debug function might check the CAN message object and may send a reply to the CAN message or invoke a signal function that sends a delayed reply. The following example shows the usage:

```
FUNC void CANmessage (void) {
    switch (CAN0OUT) {
        case 1: printf ("\nSend Message (11-bit ID=%04X)", CAN0ID); break;
        case 2: printf ("\nSend Message (29-bit ID=%08X)", CAN0ID); break;
        case 3: printf ("\nRequest Message (11-bit ID=%04X)", CAN0ID); return;
        case 4: printf ("\nRequest Message (29-bit ID=%08X)", CAN0ID); return;
    }
}
```

```
printf ("\nMessage Length %d, Data: ", CAN0L);
printf ("%02X %02X %02X %02X ", CAN0B0, CAN0B1, CAN0B2, CAN0B3);
printf ("%02X %02X %02X %02X \n", CAN0B4, CAN0B5, CAN0B6, CAN0B7);
}

>BS WRITE CAN0OUT, 1, "CANmessage ()" // call CANmessage debug function with outgoing messages
```

---

### NOTE

All CAN messages can be review also in the CAN communication dialog page.

---

## Using Signal Functions

With signal functions it is possible to send periodically messages to the user applications. Signal functions are created within the  $\mu$ Vision2 debugger. Refer to  $\mu$ Vision2 Debug Functions for more information on this topic.

The following signal function sends a message with two bytes 10 times per second. The signal function receives an info value that is transmitted to the application program. This info value is constantly incremented:

```
SIGNAL void sendCAN (unsigned int info) {
    while (1) {
        CAN0ID = 0x4510;           // CAN message ID
        CAN0L = 2;                // message length 2 bytes
        CAN0B0 = (info & 0xFF);   // message data byte 0
        CAN0B1 = (info >> 8);    // message data byte 1
        CAN0IN = 2;              // send CAN message with 29-bit ID
        twatch (CLOCK / 10);     // send message 10 times per second
        info++;                  // increment info value
    }
}

>sendCAN (500); // invoke sendCAN function
```

## Simulating Communication Errors

The  $\mu$ Vision2 Debugger is not able to simulate directly CAN communication errors. These errors are typically handled within the CAN controller hardware. The user application does not care of communication error conditions since they are automatically handled by the CAN controller. If it is required to simulate error conditions the user might directly set the error status information in the **Configuration #2** dialog page.