PSIM

TUTORIAL

Implementing CAN Bus Communication in PSIM

June 2020

PSIM's SimCoder Module together with Hardware Targets provide the necessary functionality to implement CAN bus communication easily.

Three function blocks are provided in the F2833x, F2803x, and F2806x, F2837x, and F28004x Targets for CAN bus communication:

- CAN Configuration:	It defines the configuration for communication
- CAN Input:	It receives CAN messages from a CAN bus
- CAN Output:	It transmits CAN messages to a CAN bus

Examples are given to illustrate how these blocks are used.

1. Defining CAN Bus Function Blocks

The definitions of the three CAN bus function blocks are first described below.

For the *CAN Configuration* block:

CAN Source: The CAN source can be one of the two groups: *CAN A* and *CAN B*. Depending on the type of DSP, CAN A and CAN B use combinations of different GPIO pins.

For F2833x:

- CAN A: GPIO 19, 31 to transmit, and GPIO 18 and 30 to receive.
- CAN B: GPIO 8, 12, 16, 20 to transmit, and GPIO 10, 13, 17, 21 to receive.

For F2803x/F2806x (CAN A only):

- CAN A: GPIO 31 to transmit, and GPIO 30.

For F2837x:

- CAN A: GPIO 5, 18, 30, 36, 62, 70 to receive, and GPIO 4, 19, 31, 37, 63, 71 to transmit.
- CAN B: GPIO 7, 10, 13, 17, 21, 39, 73 to receive, and GPIO 6, 8, 12, 16, 20, 38, 72 to transmit.

For F28004x:

- CAN A: GPIO 5, 18, 30, 33, 35 to receive, and GPIO 4, 31, 32, 37 to transmit.
- CAN B: GPIO 7, 10, 13, 17, 39, 59 to receive, and GPIO 6, 8, 12, 16, 58 to transmit.

CAN Speed: Communication speed, in Hz. It can be set to one of the following preset values: 125kHz, 250kHz, 500kHz, and 1MHz. Or you can set the speed manually by typing the text in the parameter field. Note that the speed should not exceed 1MHz.

Data Byte Order: The order of the data bytes. It can be one of the following:

- *Least significant byte 1st*: The least significant byte is placed first

- *Most significant byte 1st*: The most significant byte is placed first

Number of Input Mailboxes: Number of input mailboxes. The total number of mailboxes is 32. The rest of the mailboxes are assigned as output mailboxes.

Checking Receive Mail Lost: If it is set to *Enable*, the error report function "_ProcCanAErrReport(nErr)" (for CAN A) will be called if the received mail is lost. The value nErr in the function parameter list is copied from the CAN register CANGIF0.

If it is set to *Disable*, the error report function will not be called when this error occurs.

Checking Bus Off: If it is set to *Enable*, the error report function "_ProcCanAErrReport(nErr)" (for CAN A) will be called if the bus is in the off state. The value nErr in the function parameter list is copied from the CAN register CANGIF0.

If it is set to *Disable*, the error report function will not be called when this error occurs.

Error Check Mode: The error check mode can be either *Passive* or *Active*. If it is in the error-passive mode, an interrupt will be generated when the error count reaches 128. If it is in the error-active mode, an interrupt will be generated every time.

For the *CAN Input* block:

Number of Inputs: Number of CAN inputs. It can have up to 8 inputs.

CAN Source: The CAN source can be either *CAN A* or *CAN B*.

Use Extension ID: If this is set to *Yes*, the ID of a message is a 29-bit integer. If this is set to *No*, the ID of a message is a 11-bit integer.

Message ID: ID of a message, for example 0x23. It is a 29-bit integer for extended ID, and an 11-bit integer for non-extended ID.

Local Mask: Mask for the message. It is a 29-bit integer for extended ID and an 11-bit integer for non-extended ID. If the current message ID and the received message ID are identical at the bits that local mask are 1, the message will be received. Otherwise, it will be ignored.

For example, if the mask is 0x380 and the message ID is 0x389, then the received message with ID 0x387 will be accepted, because 0x380 & 0x389 equals to 0x380 & 0x387. But if the received message ID is 0x480, the message will be ignored, because 0x380 & 0x389 does not equal to 0x380 & 0x480.

Overwrite Flag: It can be set to *Allow overwrite* or *Do not allow overwrite*. Assume a coming message is receivable by the mailboxes with the same local mask, and all these mailboxes keep old messages and these messages are waiting for process. If the overwrite flag is set to *Allow overwrite*, the new message will be accepted by one of these mailboxes, the old message in the mailbox will be overwritten. If the

	flag is set to *Do not allow overwrite*, the new message will not be accepted, and the old one will be kept.
Receive Message Rate:	The number of messages received by the input block in a specific period of time. For example, there are two input blocks, with the first input block receiving 20 messages per second, and the second input block receiving 30 messages per second. The parameter "Receive Message Rate" will be set to 20 for the first block, and 30 for the second block.
Input$_i$ Gain:	Gain to the i_{th} input where *i* can be 1 to 8. The input is multiplied by the gain.
Input$_i$ Data Start Position:	A message can have up to 8 data points. A data point can have 1 bit up to 32 bits. This defines the start position of the current data point in the message.
Input$_i$ Data End Position:	This defines the end position of the current data point in the message.
Input$_i$ Data Type:	The data type can be either *Float*, *Integer*, or *IQ1* to *IQ30*.
Input$_i$ Default Data:	Default data value of the input block.

For the *CAN Output* block:

Number of Outputs:	Number of CAN outputs. It can have up to 8 inputs.
CAN Source:	The CAN source can be either *CAN A* or *CAN B*.
Use Extension ID:	If this is set to *Yes*, the ID of a message is a 29-bit integer. If this is set to *No*, the ID of a message is a 11-bit integer.
Message ID:	ID of a message. It is an integer, for example, 0x23.
Trigger Type:	The trigger type can be one of the following: - *No trigger*: No triggering. - *Rising edge*: Triggering occurs at the rising edge of the trigger source. - *Falling edge*: Triggering occurs at the falling edge of the trigger source. - Rising/*falling edge*: Triggering occurs at both the rising edge and the falling edge of the trigger source. A rising/falling edge is considered to have occured if the difference between the current value of the trigger source and the value at the last triggering instant is equal to or greater than 1.
Trigger Source:	A trigger source can be either a constant of a global variable depending on the Trigger Type. If the Trigger Type is set to *No trigger*, the trigger source defines the counter limit. For example, if the trigger source is a constant or a global value, and the value is 5, it means that triggering will occur

once out of every 5 times. In another word, the data will be sent out once per every 5 cycles.

If the Trigger Type is set to edge trigger, the trigger source can only be a global variable. Triggering will occur when the global variable has the rising or falling edge or both depending on the Trigger Type.

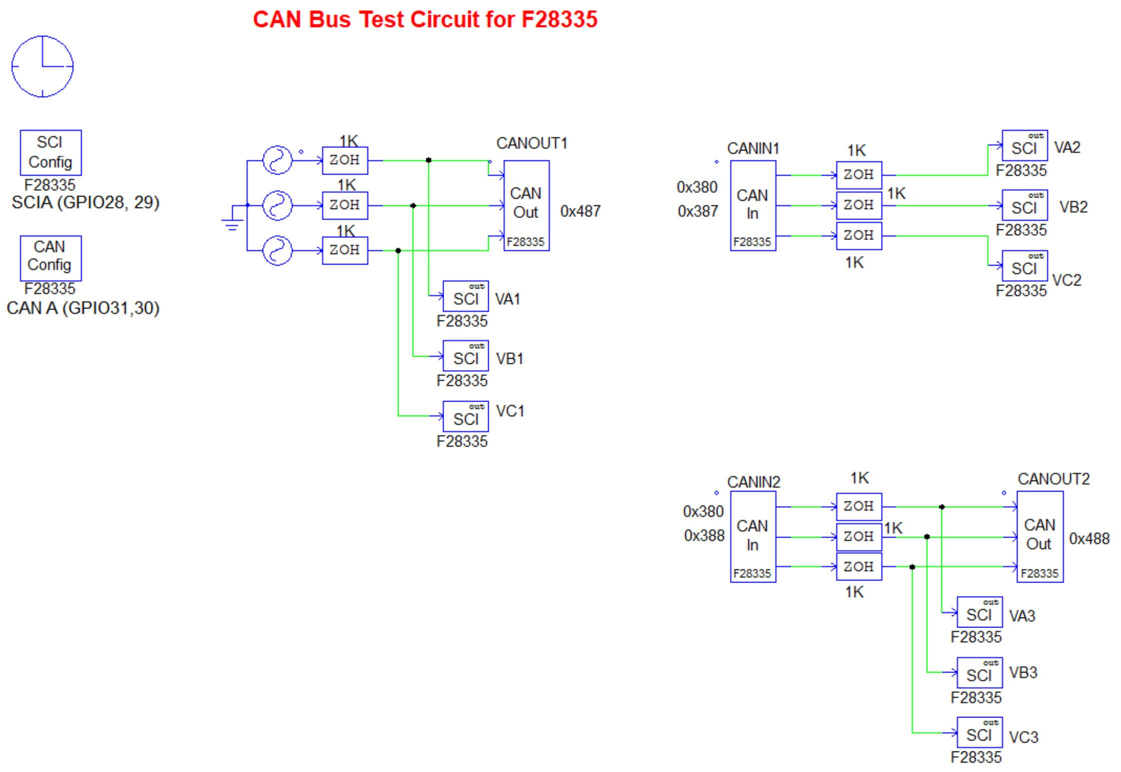
Output$_i$ Gain:	Gain to the i_{th} output where *i* can be 1 to 8. The output is multiplied by the gain.
Output$_i$ Data Start Position:	A message can have up to 8 data points. A data point can have 1 bit up to 32 bits. This defines the start position of the current data point in the message.
Output$_i$ Data End Position:	This defines the end position of the current data point in the message.
Output$_i$ Data Type:	The data type can be either *Float*, *Integer*, or *IQ1* to *IQ30*.
Output$_i$ Default Data:	Default data value of the output block.

For the CAN Configuration block, the variable nErr in the callback function "_ProcCanAErrReport(nErr)" (for CAN A) is a 32-bit integer. It obtains its value from the Global Interrupt Flag Register CANGIF0. After returning from the function "_ProcCanAErrReport(nErr)", the register CANGIF0 will be cleared.

Also, if you wish to take actions on a specific error, you can add you own code within the "_ProcCanAErrReport(nErr)" function.

2. CAN Bus Example

To illustrate how to use these CAN bus function blocks, the example “test_CAN_bus_1 (F2833x).psimsch” is shown below.

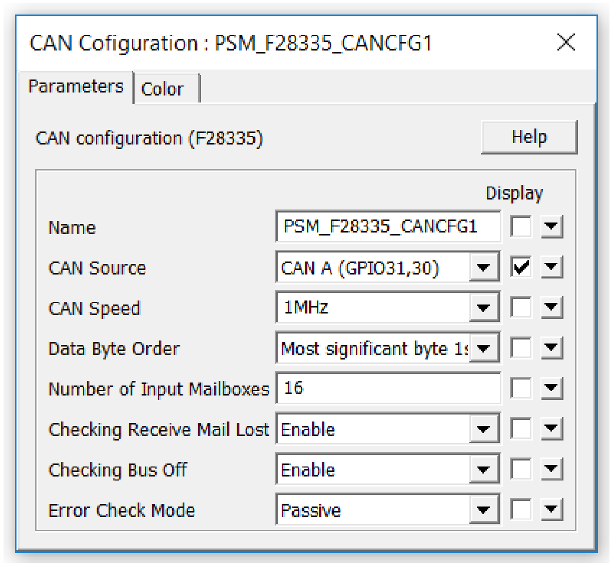


This is the example Test 1. The circuit does the following:

- It generates 3-phase sine waves VA1, VB1, and VC1 and sends them the CAN bus through the CAN Output block CANOUT1 with the message ID 0x487. These three signals are also sent to PSIM's DSP Oscilloscope for display.
- It receives 3 signals through the CAN Input block CANIN1 with the local mask of 0x380 and the message ID 0x387, and displays these signals as VA2, VB2, and VC2 through the DSP Oscilloscope.
- It receives 3 signals through the CAN Input block CANIN2 with the local mask of 0x380 and the message ID 0x388, and displays these signals as VA3, VB3, and VC3 through the DSP Oscilloscope. It then sends these signals to the CAN bus through the CAN Output block CANOUT2.

The SCI Configuration block and SCI Out blocks are used to set up the serial communication and monitor the DSP waveforms in real time using PSIM's DSP Oscilloscope. For more information on how to use the serial communication for waveform monitoring, refer to the document "Tutorial – Using SCI for waveform monitoring.pdf".

The input dialog of the CAN Configuration block is shown below.


CAN Cofiguration : PSM_F28335_CANCFG1

Parameters | Color

CAN configuration (F28335) Help

		Display
Name	PSM_F28335_CANCFG1	☐
CAN Source	CAN A (GPIO31,30)	✓
CAN Speed	1MHz	☐
Data Byte Order	Most significant byte 1s	☐
Number of Input Mailboxes	16	☐
Checking Receive Mail Lost	Enable	☐
Checking Bus Off	Enable	☐
Error Check Mode	Passive	☐

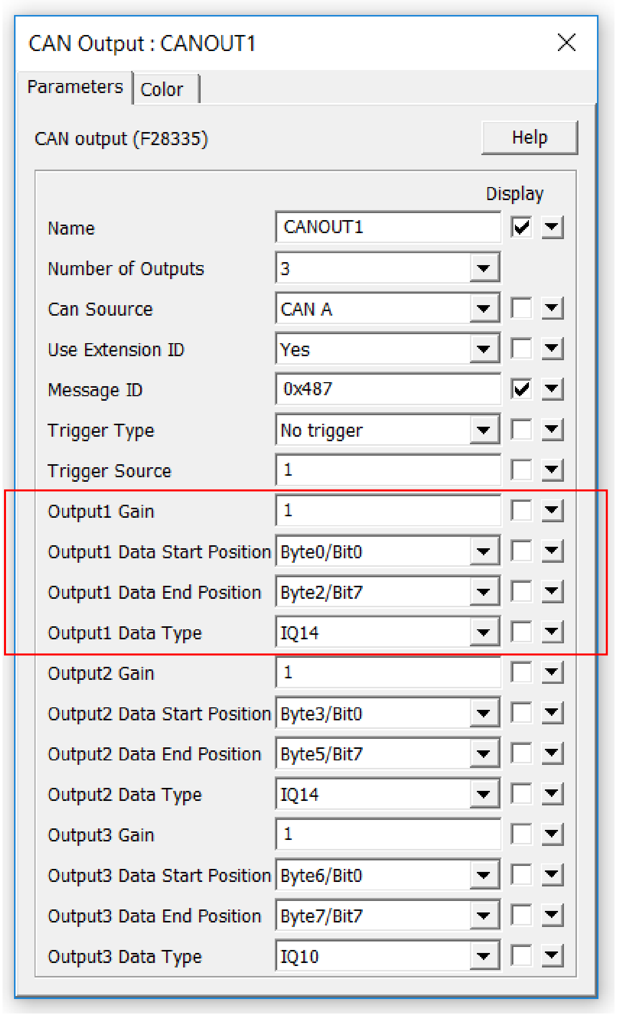
It defines that CAN A with GPIO31 and 30 is used. The CAN speed is 1MHz. The number of input mailboxes is 16.

The input dialog of the CAN Output block CANOUT1 is shown below.

It defines that the block has 3 outputs. It uses CAN A, and uses extended CAN. The message ID is 0x487.

There are four input fields correspond to each data: Gain, start position, end position, and data type. Note that when defining the data start and end positions, one must make sure that enough bits are allocated depending on the data type. A float data type has 32 bits, and an integer or IQ data type can have 1 bit up to 32 bits. For example, if a data is float, there must be 32 bits between the start and end positions.

Also, to make sure that data values are consistent, data types at the transmit end and the receive end must be the same. For example, if a data is sent as IQ14, it must be received as IQ14.



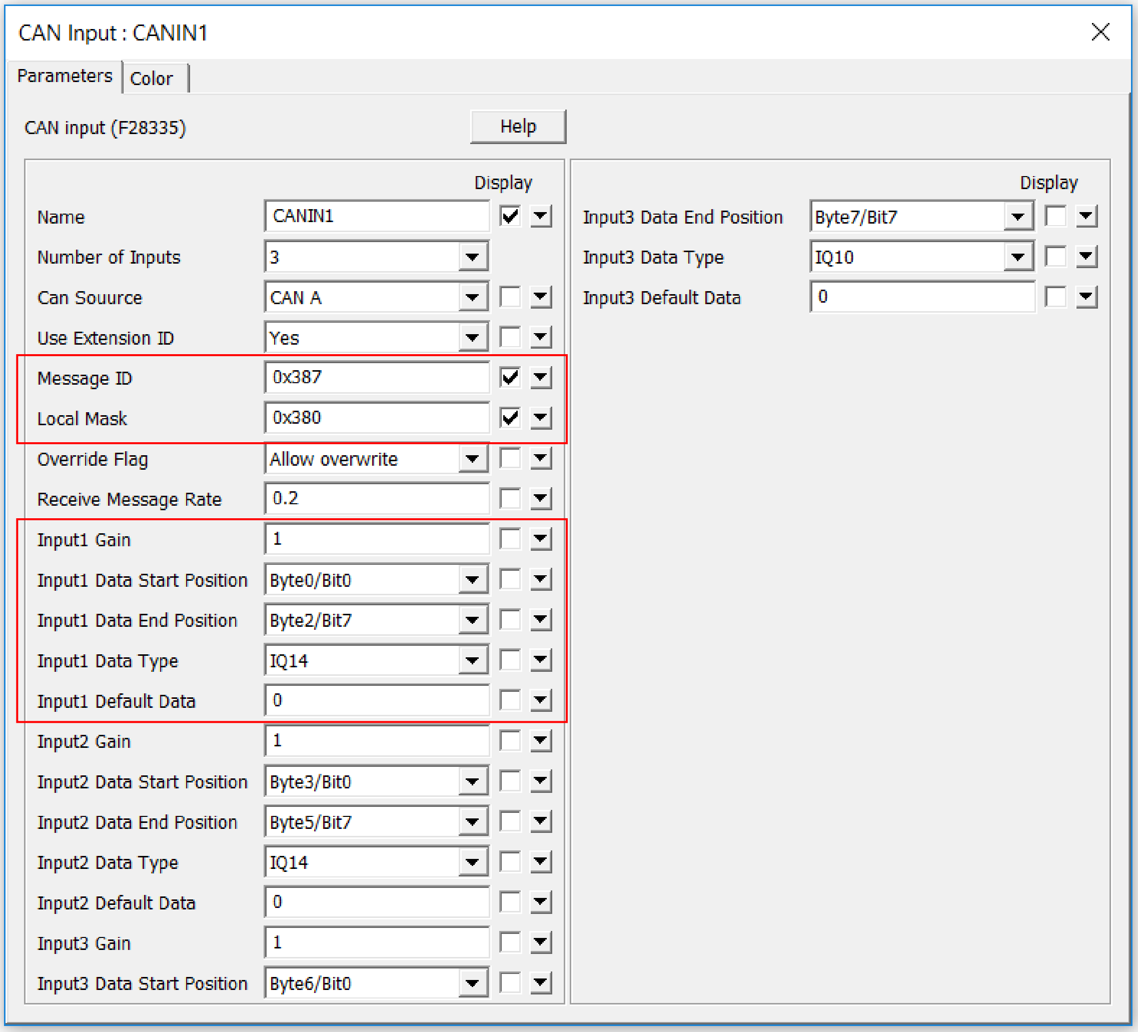
CAN Output : CANOUT1

Parameters | Color

CAN output (F28335) Help

		Display
Name	CANOUT1	✓
Number of Outputs	3	
Can Souurce	CAN A	
Use Extension ID	Yes	
Message ID	0x487	✓
Trigger Type	No trigger	
Trigger Source	1	
Output1 Gain	1	
Output1 Data Start Position	Byte0/Bit0	
Output1 Data End Position	Byte2/Bit7	
Output1 Data Type	IQ14	
Output2 Gain	1	
Output2 Data Start Position	Byte3/Bit0	
Output2 Data End Position	Byte5/Bit7	
Output2 Data Type	IQ14	
Output3 Gain	1	
Output3 Data Start Position	Byte6/Bit0	
Output3 Data End Position	Byte7/Bit7	
Output3 Data Type	IQ10	

The input dialog of the CAN Input block CANIN1 is shown below.



CAN Input : CANIN1

Parameters | Color

CAN input (F28335) Help

		Display
Name	CANIN1	✓
Number of Inputs	3	
Can Souurce	CAN A	
Use Extension ID	Yes	
Message ID	0x387	✓
Local Mask	0x380	✓
Override Flag	Allow overwrite	
Receive Message Rate	0.2	
Input1 Gain	1	
Input1 Data Start Position	Byte0/Bit0	
Input1 Data End Position	Byte2/Bit7	
Input1 Data Type	IQ14	
Input1 Default Data	0	
Input2 Gain	1	
Input2 Data Start Position	Byte3/Bit0	
Input2 Data End Position	Byte5/Bit7	
Input2 Data Type	IQ14	
Input2 Default Data	0	
Input3 Gain	1	
Input3 Data Start Position	Byte6/Bit0	
Input3 Data End Position	Byte7/Bit7	
Input3 Data Type	IQ10	
Input3 Default Data	0	

It defines that the block has 3 inputs. It uses CAN A, and uses extended CAN. The local mask is 0x380, and the message ID is 0x387.

As explained before, the local mask is used to determine if the CAN Input block should receive a particular message or not. Therefore, one needs to set the local mask and the message ID properly. Only if the message ID has the same bits as the local mask bits where the local mask bits are 1, the message will be received.

Similar to the CAN Output block, when defining the data start and end positions, one must make sure that enough bits are allocated depending on the data type. Also, the data type must match the data type at the transmit end.

3. Testing CAN Bus Example with CANUSB

To test the Test 1 example in Section 2 above, one can use the CANUSB device (available from www.can232.com/?page_id=16).

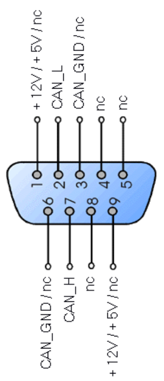
We will use the TI F28069M LaunchPad hardware and run the Test 1 example for F28069M (available in "examples\F2806x Target\CAN bus"). It is assumed that the driver for CANUSB from the manufacturer has been installed on the computer already. Please install the 32-bit driver as the CAN test program is 32-bit.

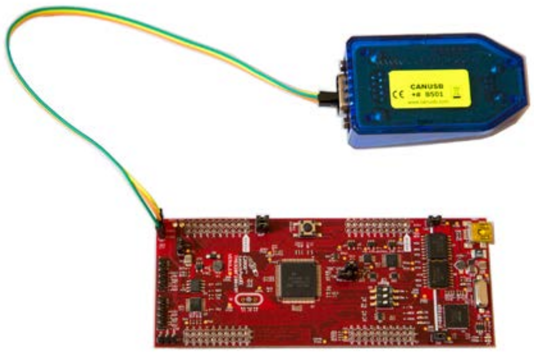
Below is the test procedure:

- Make sure that all 3 positions of the DIP switch S1 on the F28069M LaunchPad are set to ON.
- Use a jumping cable to connect the DB-9 connector (male) of CANUSB to J12 of the F28069M LaunchPad in the following way:

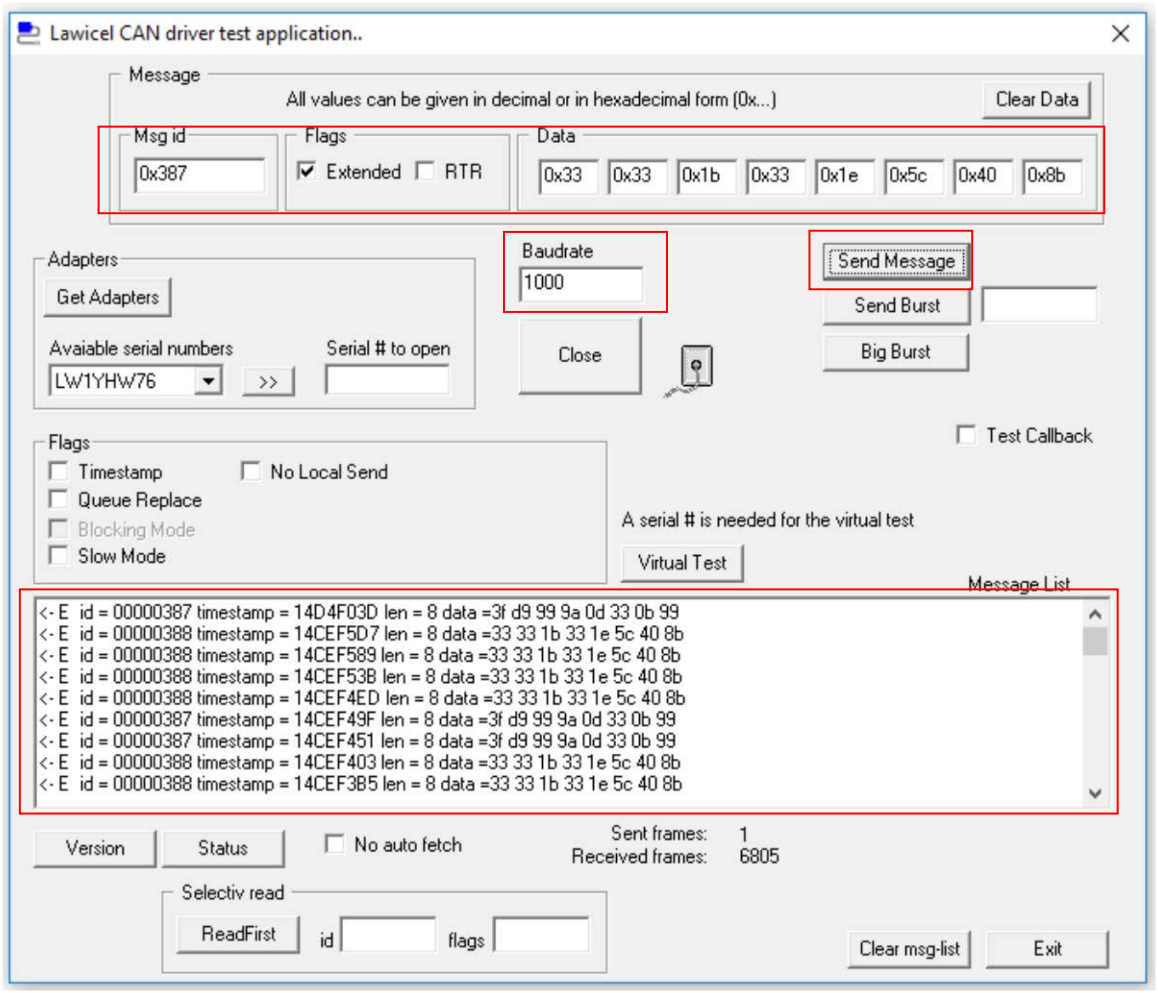
CANUSB		LaunchPad
Pin 7	-------	CAN-H
Pin 2	-------	CAN-L
Pin 3	-------	Pin Ground

The pin assignment of the DB-9 connector of CANUSB is shown on the left below:
The figure on the right below shows the connection of the LaunchPad and CANUSB.





- Connect the LaunchPad and CANUSB to the computer.
- Generate the code from the example "test_CAN_bus_1 (F2806x).psimsch" in the folder "examples\F2806x Target\CAN bus". Compile the code in the Code Composer Studio, and upload it to the LaunchPad.
- Run the testDriver.exe program from CANUSB. The interface is as below.



Msg id: 0x387

Flags: [✓ Extended] RTR

Data: 0x33 0x33 0x1b 0x33 0x1e 0x5c 0x40 0x8b

Baudrate: 1000

Avaiable serial numbers: LW1YHW76

```
<- E  id = 00000387 timestamp = 14D4F03D len = 8 data =3f d9 99 9a 0d 33 0b 99
<- E  id = 00000388 timestamp = 14CEF5D7 len = 8 data =33 33 1b 33 1e 5c 40 8b
<- E  id = 00000388 timestamp = 14CEF589 len = 8 data =33 33 1b 33 1e 5c 40 8b
<- E  id = 00000388 timestamp = 14CEF53B len = 8 data =33 33 1b 33 1e 5c 40 8b
<- E  id = 00000388 timestamp = 14CEF4ED len = 8 data =33 33 1b 33 1e 5c 40 8b
<- E  id = 00000387 timestamp = 14CEF49F len = 8 data =3f d9 99 9a 0d 33 0b 99
<- E  id = 00000387 timestamp = 14CEF451 len = 8 data =3f d9 99 9a 0d 33 0b 99
<- E  id = 00000388 timestamp = 14CEF403 len = 8 data =33 33 1b 33 1e 5c 40 8b
<- E  id = 00000388 timestamp = 14CEF3B5 len = 8 data =33 33 1b 33 1e 5c 40 8b
```

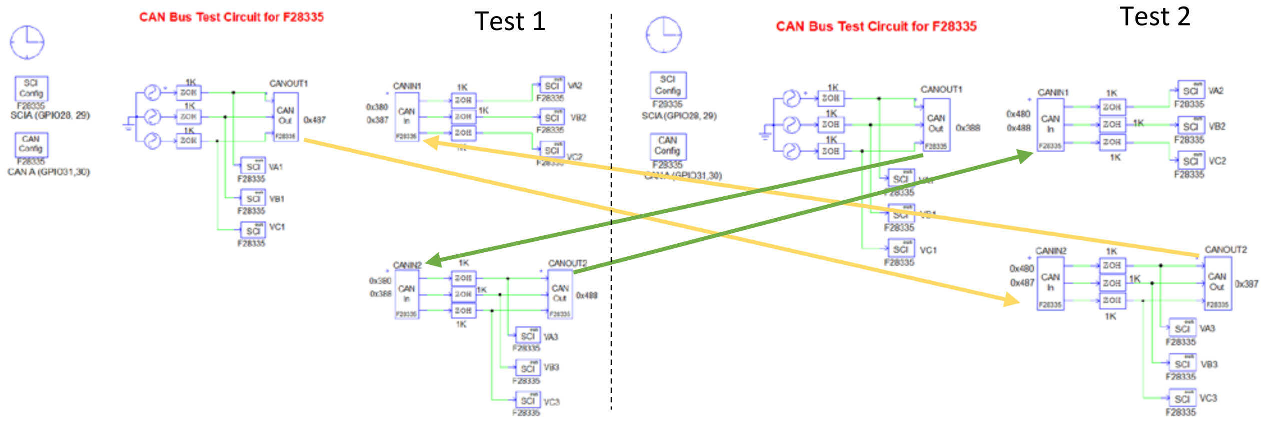
Sent frames: 1
Received frames: 6805

- Set the **Baud rate** to 1000. This represents 1000kHz, and this must match the 1MHz setting in the CAN Configuration block in PSIM. Click on the **Open** button below the baud rate box. The **Open** button will change to the **Close** button.
- Run the code on DSP. Messages should appear in the **Message List** in the testDrive program. These messages are sent by the DSP and received by CANUSB.
- To test sending messages from CANUSB to DSP, set the message ID to **0x387**. Check the box **Extended** under **Flags**. Fill in the data field as shown above, and click on **Send Message**. If a breakpoint is set in CCS in Line 239 (below “case 0x387”), the execution will stop at the breakpoint.

4. Testing CAN Bus Examples Between Two Hardware Boards

Beside example Test 1, another example Test 2 is provided in PSIM so that one can test CAN bus communication between two hardware boards.

The figure below shows the schematic of both Test 1 and Test 2.



Test 2 has the same schematic as Test 1, except that the local mask and message IDs for the CAN Input and CAN Output blocks are different.

Let's assume that Hardware Board 1 runs Test 1, and Hardware Board 2 runs Test 2. These two examples are set up in the following ways:

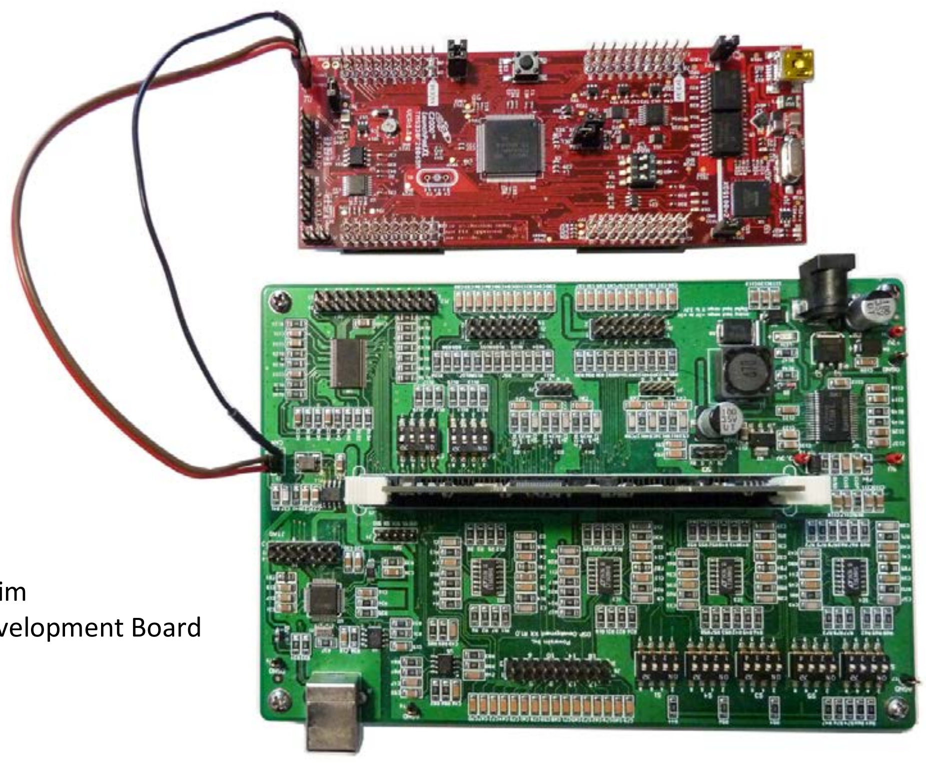
- Hardware 1 will send out VA1, VB1, and VC1 through CANOUT1 to the CAN bus. Hardware 2 will receive these signals through CANIN2 and display them as VA3, VB3, and VC3. These signals will be sent back to the CAN bus through CANOUT2. Hardware 1 will receive these signals through CANIN1, and will display them as VA2, VB2, and VC2.

  This process is illustrated through the yellow arrows.

- Similarly, Hardware 2 will send out VA1, VB1, and VC1 through CANOUT1 to the CAN bus. Hardware 1 will receive these signals through CANIN2 and display them as VA3, VB3, and VC3. These signals will be sent back to the CAN bus through CANOUT2. Hardware 2 will receive these signals through CANIN1, and will display them as VA2, VB2, and VC2.

  This process is illustrated through the green arrows.

The hardware-to-hardware test has been done using the F28069M LaunchPad and Powersim's DSP Development Board. The arrangement is shown below.



Use a jumping cable to connect J3 of the Powersim DSP board to J12 of the F28069M LaunchPad in the following way:

Powersim Board		LaunchPad
Pin 1	-------	CAN-H
Pin 2	-------	CAN-L
Pin 3	-------	Pin Ground

One can run Test 1 of any of the F2833x/F2803x/F2806x Target on one hardware, and run Test 2 of any of the F2833x/F2803x/F2806x Target on another hardware.

5. Conclusions

PSIM's SimCoder, together with CAN bus function blocks, provides a quick and convenient way of implementing CAN bus communications.