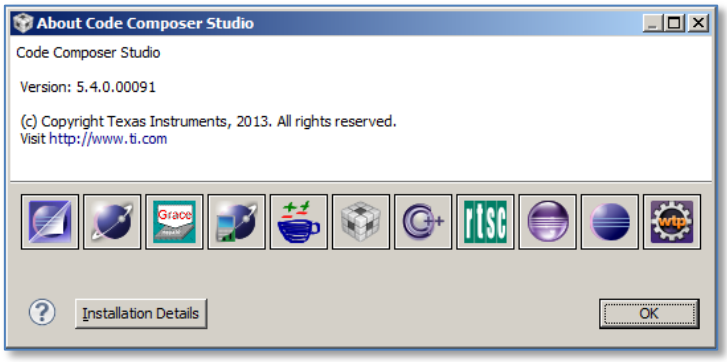
RM48L952 HDK Lab3_2

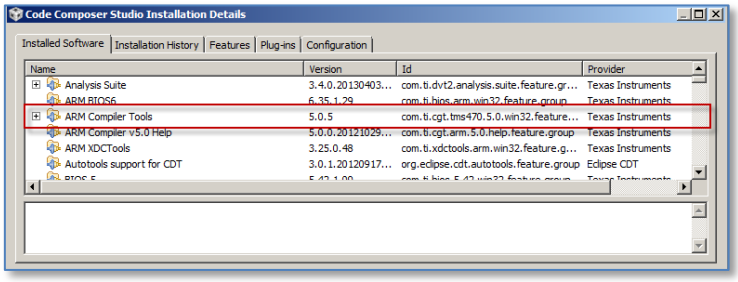
DCAN1 Transmit and Receive; no interrupts

1. Project Dependencies

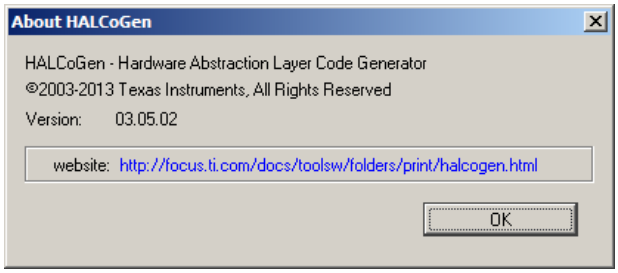
- Code Composer Studio Version 5.4.0:



- ARM Code Generation Tools Version 5.0.5:



- HalCoGen – Version 3.05.02:



2. Project Objective

- CAN Transmit at CAN data rate: 100 kBit/s.
  - Identifier 0x12000000, DCAN1 Message-box 1, Byte Message
  - 1 Byte message; incremental pattern every 100 milliseconds
- CAN Receive: Identifier 0x10000000 DCAN1 Message-box 2;
  - if a meesage has been received, bits 0-2 of the 1st message byte are displayed at LED's at NHET1-25, 1-18 and 1-29.

3. Hardware Setup

No special setup required. Connect the RM48HDK with a USB cable to your computer and plug in the external +5V or +12V power supply unit. The four blue LEDs DS2...DS5 "Power" should be "ON". Connect DCAN1 (Jumper J2) with "CAN1H", "CAN1L" and "GND" to a CAN system. Use a CAN – Analyzer to monitor the CAN – Traffic.

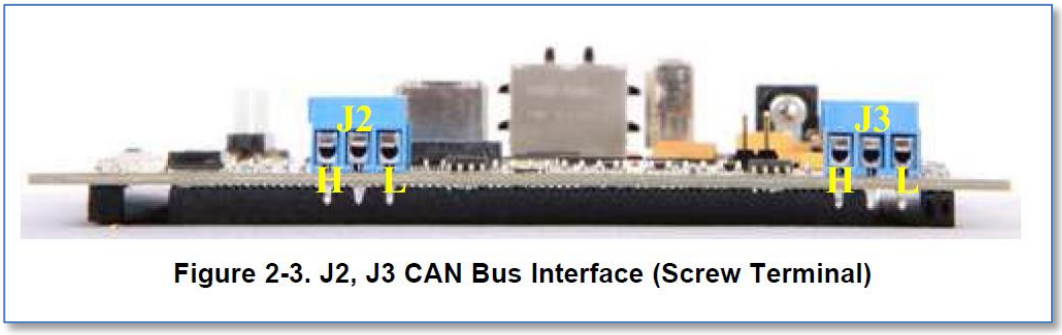
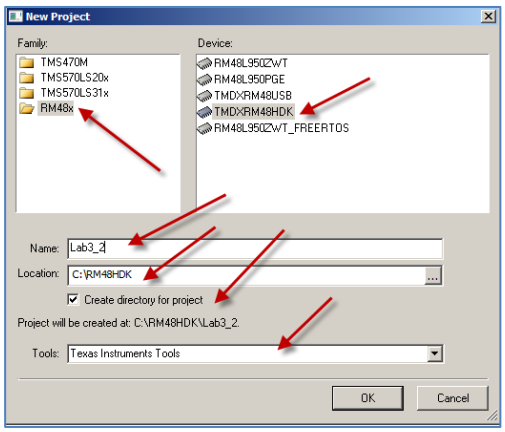


Figure 2-3. J2, J3 CAN Bus Interface (Screw Terminal)

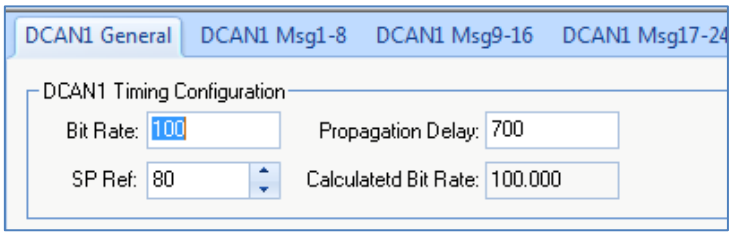
4. HalCoGen Project Design

4.1. Create a new project

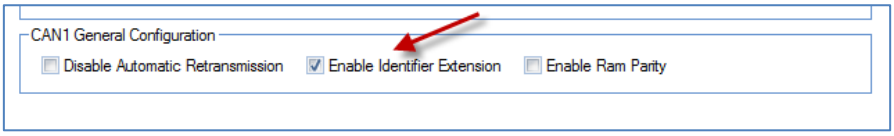
➔ File ➔ New ➔ Project:



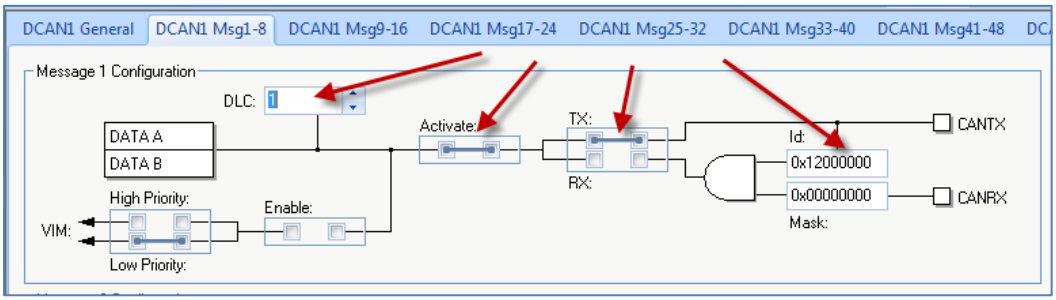
- In the “Driver Enable” Menu, enable just the GIO and CAN1 driver.
- In the CAN1 – Module set Bit Rate to 100 kBit/s and the Sample Point Reference to 80%:



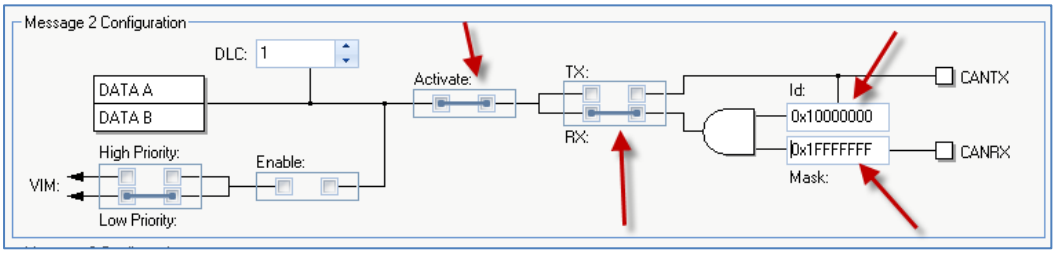
- Enable also the “Extended Identifier Extension” at the end of this window:



- Prepare the DCAN1 Message Box 1 as transmit mailbox with identifier 0x1200000 and DLC = 1 Byte:



- Prepare the DCAN1 Message Box 2 as receive mailbox with identifier 0x10000000, DLC = 1 and Mask = 0x1FFFFFFF:



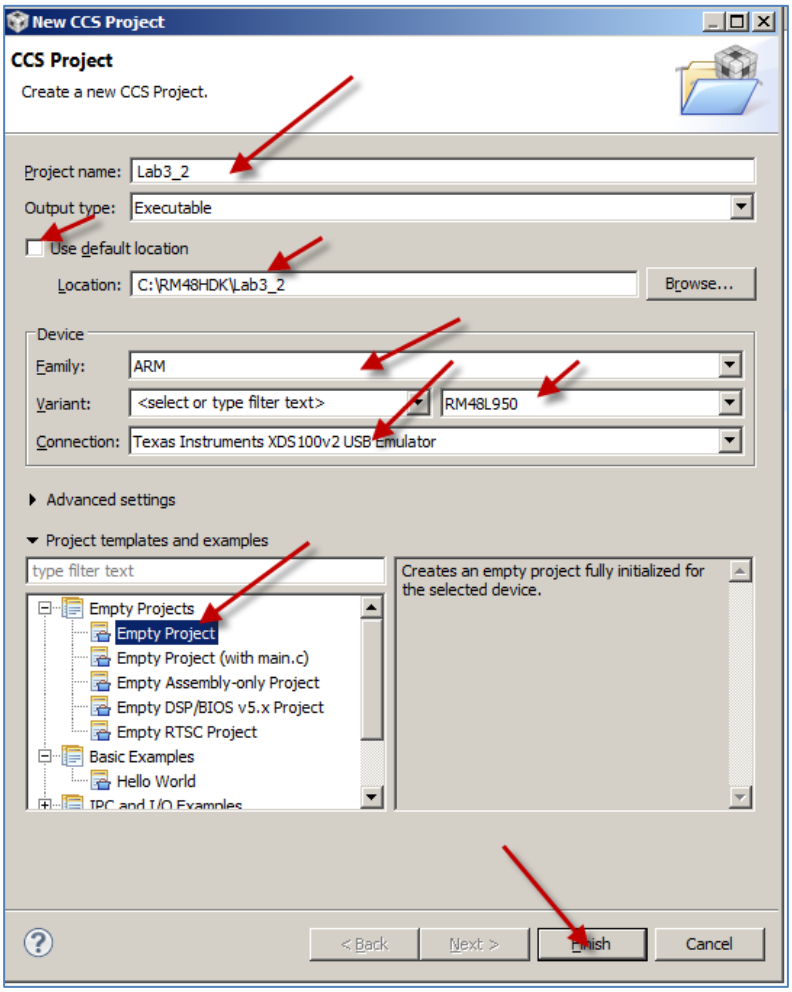
➔ File ➔ Generate Code

5. CCS Project Design

5.1. Create a CCS Project

For the CCS project we have to use the identical path as we used in HalCoGen:

➔ **File ➔ New ➔ CCS Project:**



5.2. Edit the project

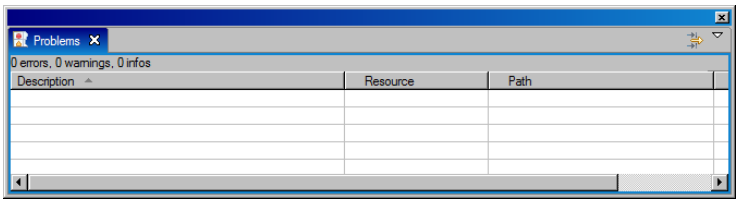
- Right Click at the project “Lab3_2” and select “Properties”:
  - In the “Build”, “ARM Compiler”, “Include Options” add a new #include search path:

    ${PROJECT_ROOT}/include

5.3. Build the Project

➔Project ➔ Rebuild Active Project (Alt + Shift + P)

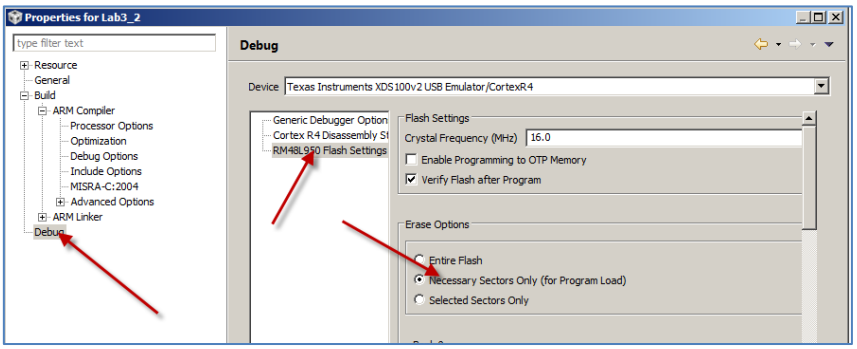
Monitor the build procedure of the tools in the output window. The final message should give you "0 Errors, 0 Warnings, 0 Remarks".



5.4. Load the code into the target

The code will be programmed into the RM48 internal FLASH memory. To avoid unnecessary long programming times, we should set the configuration to erase and re-program FLASH sections only, if required:

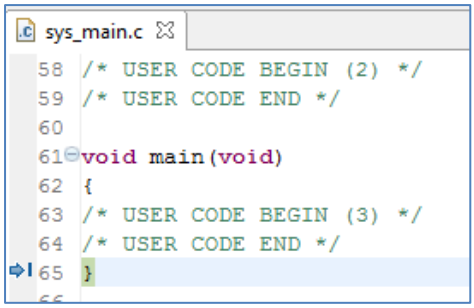
➔ Project ➔ Properties ➔ CCS Debug ➔ Target ➔ RM48L950 FLASH Settings ➔ Necessary Sectors Only (For Program Load)



Now load the machine code into the device. Click:

➔ Run ➔ Debug (F11)

A blue arrow should now point to the end of function "main()" in file "sys_main.c". This is an indication that the machine code has been downloaded properly into the RM48L950.



6. Code Implementation

6.1. Modify file “sys_main.c”

Now that we have a working project framework, we can start to develop our own code for Lab3_2.

⇨ switch back to the “CCS-Edit” -Perspective and edit the file “sys_main.c”.

Between the two comment lines shown below, add the following include lines:

```
/* USER CODE BEGIN (1) */
#include "gio.h"
#include "het.h"
#include "can.h"
/* USER CODE END */
```

After the label “/* USER CODE BEGIN(2) */” add the following code:

```
unsigned char rx_message[8];
void delay(void)
{
	volatile unsigned int delayval = 1000000; // appr. 100 ms
	while(delayval--)
	{
		if(canIsRxMessageArrived(canREG1,canMESSAGE_BOX2))
		{
		canGetData(canREG1,canMESSAGE_BOX2,rx_message);
		if(rx_message[0] & 1) gioSetBit(hetPORT1, 25,1);
			else gioSetBit(hetPORT1, 25,0);
		if(rx_message[0] & 2 )gioSetBit(hetPORT1, 18,1);
			else gioSetBit(hetPORT1, 18,0);
		if(rx_message[0] & 4) gioSetBit(hetPORT1, 29,1);
			else gioSetBit(hetPORT1, 29   ,0);
		}
	}

}
```

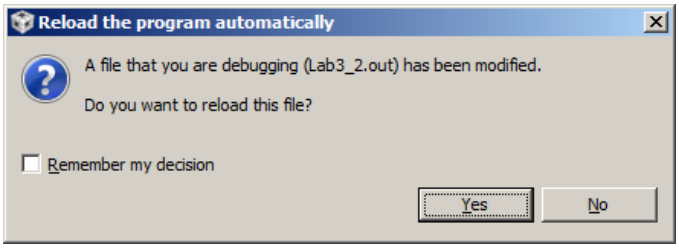
After the label “/* USER CODE BEGIN(3) */” add the following code:

```
/* USER CODE BEGIN (3) */
unsigned char tx_data = 0;
gioSetDirection(hetPORT1, 0xFFFFFFFF);
canInit();
while(1)
{
	if(canIsTxMessagePending(canREG1,canMESSAGE_BOX1)==0)
	// msgbox empty?
	{
		canTransmit(canREG1,canMESSAGE_BOX1,&tx_data);
		tx_data++;
	}
	delay();
}
/* USER CODE END */
```

6.2. Rebuild Project

➔Project ➔ Rebuild All (Ctrl + B)

If the new build is successful a message window will pop-up:



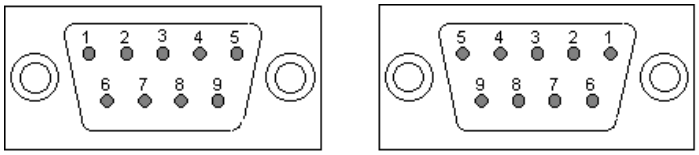
7. Run the Code

- Now perform a Run (F8)!
- A test option would be to use a low cost CAN – Analyser, e.g. the USB-CANmodul from Systec Electronic (www.systec-electronic.com).

A common technique for physical CAN cables is based on DB9 connectors, according to CiA DS 102 (www.can-cia.org):

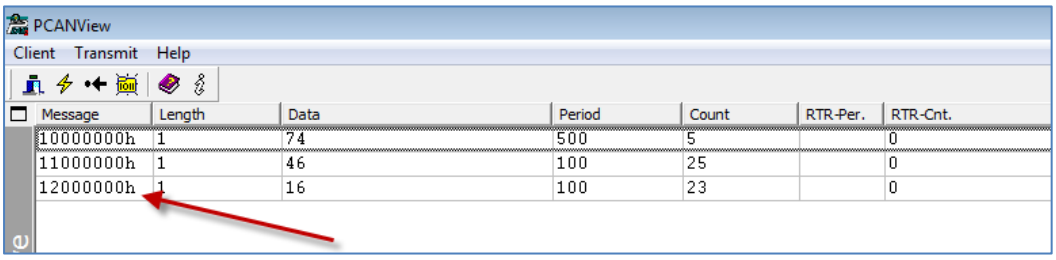
Pin Nr.	Signal	Description
1	-	Reserved
2	CAN_L	CAN Bus Signal (dominant low)
3	CAN_GND	CAN ground
4	-	Reserved
5	CAN_SHLD	Optional shield
6	GND	Optional CAN ground
7	CAN_H	CAN Bus Signal (dominant high)
8	-	Reserved
9	CAN_V+	Optional external voltage supply Vcc

At minimum we need CANL (pin 2), CANH (pin 7) and CAN_GND (pin3).



Pins Socket

The picture below is a screenshot of a CAN - Anaylzer:



8. Test Result

You will need a 2nd RM48HDK board.

This 2nd board must periodically transmit a 1 byte message with identifier 0x10000000 at a CAN data – rate of 100 kBit/s. Use Lab3_1 for this part and change the ID to 0x10000000.

The device under test (the board used for Lab3_2) should receive this message and display the last 3 bit of byte 1 at the 3 LEDs HET1:25,1:18 and 1:29.

If the message is repeadedy received with different payloads, e.g. an incremental value, the 3 LEDs should reflect this.