

DevCon 2008 Training

## CAN Development & Demo Kit Lab

**Description:** Using the Renesas M16C CAN API

### Objectives

After completing this lab you will be able to use the CAN API to

1. Initialize the on chip CAN module
2. Transmit from a CAN mailbox
3. Receive data in a CAN mailbox
4. Use the SYSTEC bus monitor to view and send CAN data

### Lab Materials

You will use the following lab components:

- HEW + NC30WA V.5.43
- M16C E8 Emulator V.2.02.00 or above
- Systec PCAN View software

#### CAN-D-Kit components:

- R8C/23 RSK
- M16C/29 RSK
- E8 Debugger with USB cable
- Systec CAN Monitor with white USB cable
- Red/White and 9 pin D-sub CAN sniffer cable

**Skill Level:** Familiar with Renesas tools and MCUs

**Total Time to Complete Lab:** 100 minutes

## Lab Sections

### Part I

#### 1 Connecting E8, Systec CAN Monitor, and RSK boards

Time to complete task: 10 minutes

#### 2 Launch HEW and firmware project

Time to complete task: 10 minutes

#### 3 Enabling CAN View CAN data with monitor

Time to complete task: 10 minutes

### Part II

#### 4 Transmit from a CAN mailbox

Time to complete task: 10 minutes

#### 5 Receive to a CAN mailbox by interrupt

Time to complete task: 10 minutes

#### 6 Transmit CAN Data with monitor The HEW Watch window

Time to complete task: 10 minutes

## Lab Procedure

### PART I

#### 1 Connecting E8, Systec CAN Monitor, and RSK boards

10 minutes

**Overview:** Connect the E8, the Systec CAN Monitor, and the RSK boards.

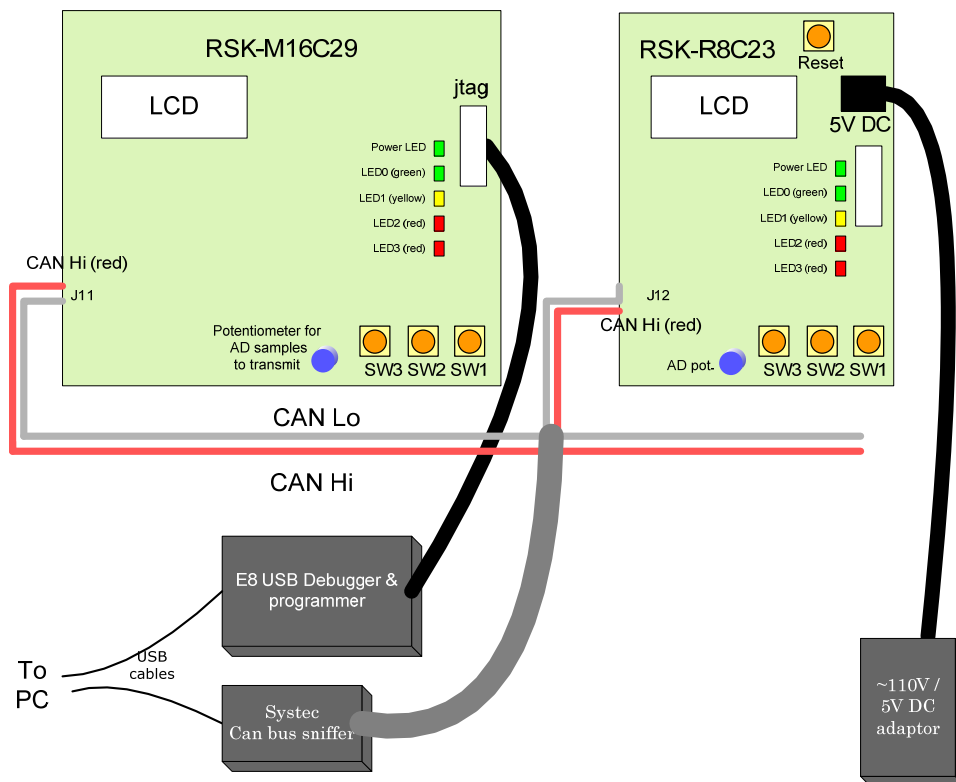
#### Procedural Steps:

Skip to next section if both RSKs and the sniffer are already connected!

#### E8

- 1.1) **Connect the E8** to the **RSK-M16C29** board with the colored ribbon cable
- 1.2) Plug the **USB cable** into the E8 and also into a PC USB port. The E8 requires a high power USB connection so it will not work if plugged into an un-powered hub.

### CAN Lab Set Up



### Systemc CAN Monitor

- 1.3) Plug the **Systemc USB connector cable** (maxi type) into the Systemc monitor and into the PC.
- 1.4) Connect the **9 pin D-sub** (serial port type) **CAN sniffer cable** to the other side of the CAN sniffer.
- 1.5) Connect other side of the sniffer cable, the **red & white wires**, to the **J11** port of the RSK29, and **J10** of the RSK23. The connectors are marked with an arrow for **CAN Hi** , which is the thick red wire of the sniffer cable..

### Supply voltage

- 1.6) Connect the **5V** power jack to the **RSK23** board.



## **Launch HEW and firmware project**

Time to complete task: 10 minutes

**Overview:** Start HEW and the lab's firmware project.



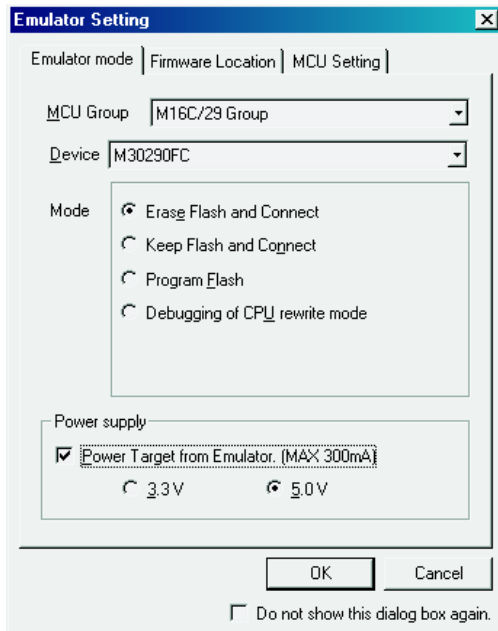
The RSK23 board is already programmed. You are just going to work with the RSK29 board.

### **Procedural Steps:**

#### HEW

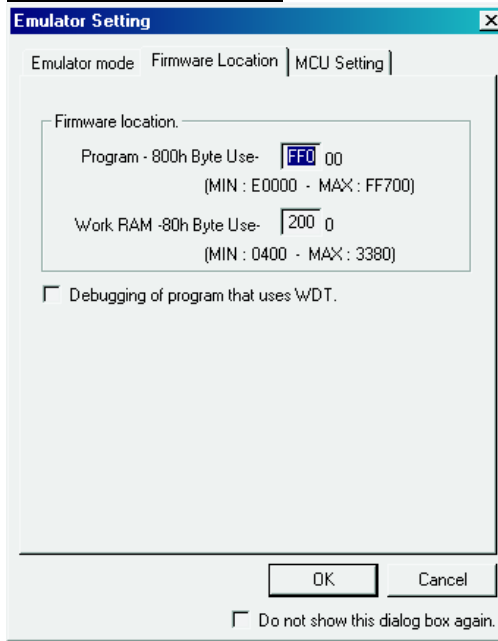
- 2.1) Launch HEW from the shortcut on the desktop or from the **Start Menu >> Programs>>Renesas> High Performance Embedded Workshop>> High Performance Embedded Workshop**.
- 2.2) When the Welcome screen comes up select **Browse to another project workspace** and click OK.
- 2.3) When the browse window comes up go to the **C:\WorkSpace\CAN\_API\_Lab\_005\CanDkit29\_DevCon\_lab** and in that folder you will find a **CANDKit.hws**. Select that file and Open.
- 2.4) You will now get a series of E8 connect screens. If you don't have the Emulator Settings window, Switch to the E8\_Debug session by selecting Debug->Debug-Sessions->Current session.

### Emulator Mode tab



- 2.5) MCU group: **M16C/29**
- 2.6) Select device: **M30290FC**
- 2.7) Select **“Erase flash and connect”**.
- 2.8) Select **“Power target from emulator”**, and **5V** if choose to let the E8 to power the board.
- 2.9) Do **not** click OK yet. (If you did, disconnect and reconnect again). Select the Firmware location tab.


### Firmware location tab

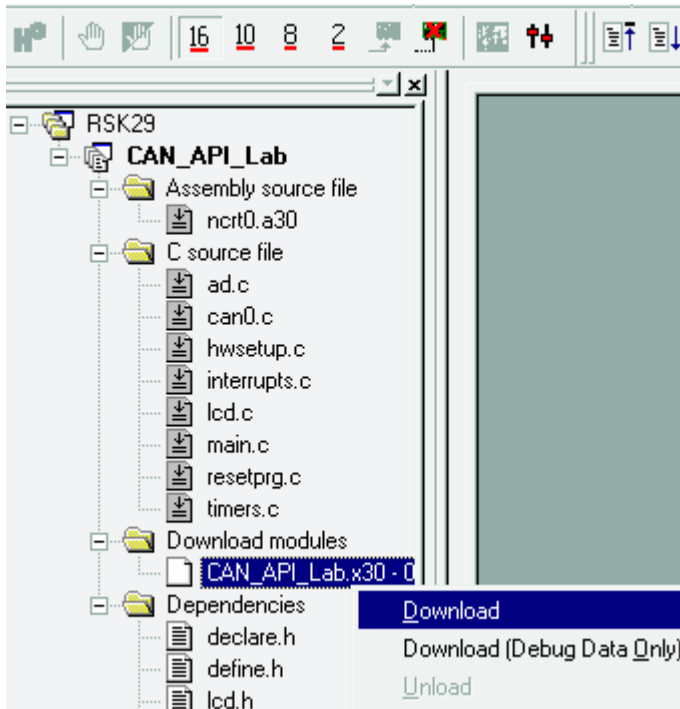




- 2.10) Enter for example **FF000** for the debug program kernel location on the target and **2000** for the work RAM location.
- 2.11) The checkbox “Debugging of a program that uses wdt” should not be checked since the code to be debugged is not running a watchdog.

- 2.12) Click OK, and the **E8 monitor** should be **downloaded** into the microcontroller
- 2.13) Accept any dialogue warning that a different version of the E8 firmware is necessary and let it reprogram the E8.

**Build and download**

- 2.14) **Build** this project by pressing F7, or click the Build icon .
- 2.15) If a window comes up and asks if you want to download the module select **Yes**.
- 2.16) If the window did not appear, right-click on the **CAN\_API\_Lab.x30** file and select “**Download module**”.



- 2.17) **Run**. You can now run the code by selecting **Debug→Reset→GO**,  or Shift+F5. Make sure it is running by checking that the stop sign  is red in HEW, and that the orange LED is blinking.
- 2.18) The RSK29 board's LCD should display “**Renesas CanDkit**”.

**3 Enabling CAN**  
**View CAN data with monitor**  
 10 minutes

**Overview:** Study CAN data using the CAN Monitor. We will set up the CAN peripheral to receive a message.

**Procedural Steps:**

**Enable the MCU's CAN module**



To enable the CAN peripheral so we can use the API, just call `can_initial()` after the MCU has been initialized. This will initialize the can peripheral to a chosen baud rate (can be changed in `can01.h`).

- 3.1) Press Stop (Escape-button).
- 3.2) Uncomment the call to `can_initial()` at the beginning of the function `main()` in file "**main.c**" (line 87). Right after also uncomment the `mailbox_init()` function that we will later change.
- 3.3) **Compile, download, and run** as above.
- 3.4) On the other board, the RSK-R8C23, **press** the **RESET** switch along the edge of the board under the LCD to make sure it is running properly.
- 3.5) On the RSK-R8C23 board the LCD will show "**Renesas CanDkit**" briefly then;  
**"CAN: "** on line 1, and  
**"A-D Rx "** on line 2.

**Errors** If you see the message "bus0: 0x", where x=1-5 you have problems with the CAN bus:

**"bus0: 03"** probably means you have a bad physical connection and the CAN peripheral cannot transmit. Are the wires reversed? See step 1.

**"bus0: 05"** is more serious as the device is going bus-off and not able to recover due to disturbances. Check the CAN network.

- 3.6) The code in your RSK29 project is already set up to **transmit** an **A-D value** over CAN once per second with CAN-id 1. The RSK23 is set to **receive** CAN frames with that same CAN-id.  
**Turn** the **potentiometer** on the RSK29 and **check** that the **A-D field** is **updated** on the RSK23.

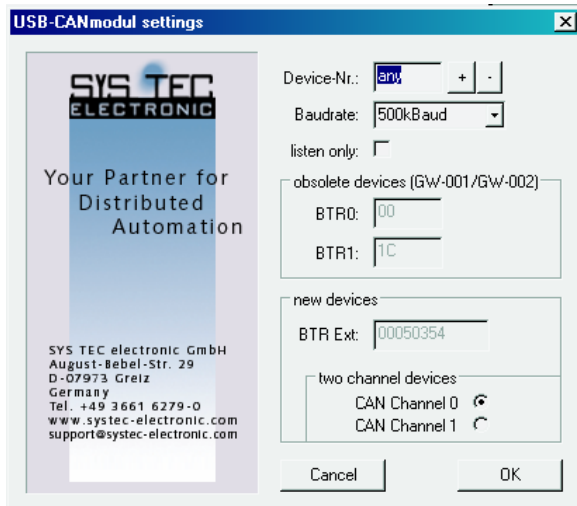
### Question

1. On the RSK23, does the green LED blink once per second, and the A-D value should show up on the LCD?

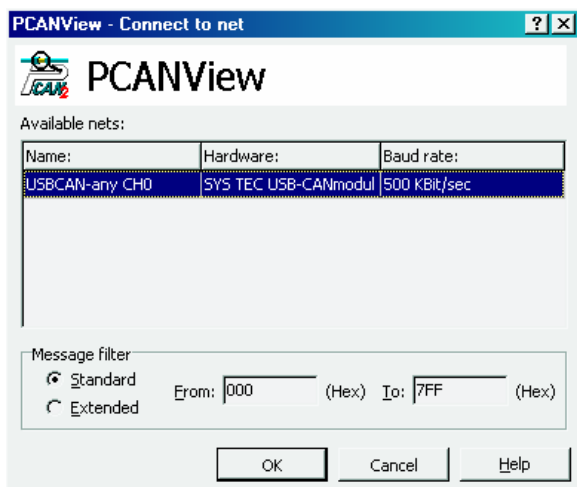
If the answer is yes, continue.

### Run the monitor

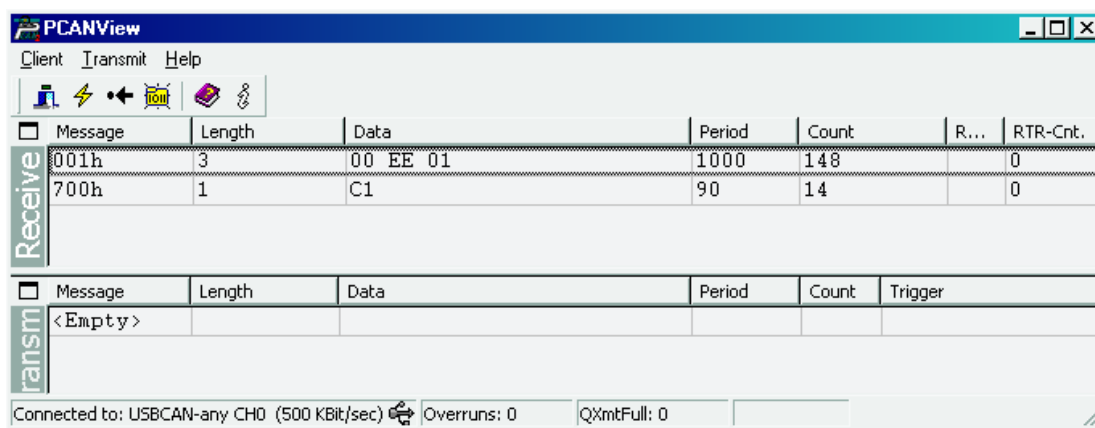
- 3.7) Minimize (don't close!) HEW so we can work with the Systec CAN Monitor.
- 3.8) Launch the PCANView from the desktop or the Start Menu >> Programs>> USB\_CANmodul Utilities >> PCANView (USBCAN)
- 3.9) On the first dialog screen **check** that  
Device-Nr = **any**  
Baud rate = **500** kbaud  
CAN Channel is **0**.  
Click **OK**



- 3.10) On the next screen **verify** the **Message Filter** is set for “**Standard ID**” and “**From 000-7FF**”. Click **OK**.



- 3.11) Turn the R9 potentiometer of the RSK23 to have it send a CAN message with **ID 700h**.



Each line shows the latest frame for a particular message id (CAN-id).

**The columns of the PCAN screen:**

**Message:** CAN-id.  
**Length:** Number of bytes in data field.  
**Data:** Data field content.  
**Period:** Time since last message with this id, in ms.  
**Count:** Number of frames with this id since reset.  
 (Reset, or clear the window with the Escape button).

- 3.12) **Look** at the data that is being **received**. You should see that there is data on the bus with an **id** of **1**, which has the **AD value** and the status of the RSK29's **SW1 in data byte 3**. Press and hold switch 1 on the RSK29 . Watch the PCAN monitor.
- 3.13) The RSK23 will also transmit A-D values, but with another id, as soon as the value changes. **Turn the potentiometer** on the **RSK23** to change the value. Check that it is received by the sniffer.

**Question**

What CAN-id does the RSK23 use for transmitting A-D values? \_\_\_\_\_

## PART II

### 4 Transmit from a CAN Mailbox

10 minutes

**Overview:** Configure and enable the CAN peripheral, and then use it to transmit from a CAN Mailbox.

#### Procedural Steps:

##### Transmit from a CAN Mailbox



We will create a CAN transmit message using the API with the following characteristics:

Message ID = 30h  
 Data length = 2 bytes  
 CAN mailbox = 13

We will use the PCAN monitor to verify that the message is actually being sent.

- 4.1) Back in HEW, press Stop (hit Escape).
- 4.2) **Let us create** a CAN dataframe **data structure**. To do this we will copy one of the existing structures. In the `mailbox_init()` function in "**main.c**", line 240, **uncomment** the code

```
/* Add for lab TX dataframe */
//lab_tx_dataframe.id           = 0x30;
//lab_tx_dataframe.dlc         = 2;
//lab_tx_dataframe.data.data[0] = 8;
//lab_tx_dataframe.data.data[1] = 0;
//mailbox_lab_tx               = 6;
```

- 4.3) The structure and the mailbox variable need to be defined. Open the **declare.h** file. Near the top of the file on line 44-45 you will find the definition of CAN variables as below.

```
/*Added for lab */
```

```
//EXTERN can_std_data_def lab_tx_dataframe;
//EXTERN int mailbox_lab_tx;
```

Uncomment these lines. (For the curious: 'EXTERN' is replaced with 'extern' for all files except the file in which the data is defined. In that file 'EXTERN' is replaced with nothing (blank space).)

4.4) We have now completely setup a CAN mailbox and data structure. Now we just need to have the program transmit the data.

4.5) In the "main.c" file, in function **main()** line 127, uncomment these lines:

```
/* Added for lab */
//set_trm_std_dataframe_can0(mailbox_lab_tx, &lab_tx_dataframe);
//lab_tx_dataframe.data.data[1]++;
```



**The first line** calls a standard function in the API that will transmit a frame on channel CAN0, using the mailbox given by the first parameter `mailbox_lab_tx`, and the data from the structure we created, `&lab_tx_dataframe` (the '&' is for the address of that structure to be passed). **The second line** just increments one of the data values so that we can see that the data is actually being transmitted continuously.

### Question

What should happen when the code is run? \_\_\_\_\_  
\_\_\_\_\_

- 4.6) **Build** the program. You should not have any errors, if you do; double click on the error and it will take you to the line with the problem.
- 4.7) **Run** the program Reset-GO (Shift+F5).
- 4.8) **Look** at the **data** with the **sniffer**. You should see that there is now also this new message that we just created. It should have an **ID of 30hex** and the first byte should be fixed at **'08'** while the second byte is **rapidly changing** since the transmit frame command was put in the main loop.

5

## Receive to a CAN mailbox by interrupt

10 minutes

**Overview:** We will look at how simple it is to receive a message using the API and the sample project. We will set up the CAN peripheral to receive a message with the following parameters:

Message id = 44h;  
Message length = 4 bytes  
Mailbox nr = 4

Then, we will use the PCAN monitor to send a message and then check that it is actually received.

### Procedural Steps:

#### Set up to receive in a CAN mailbox

Let's **define** a CAN data **receive structure**, so we don't mix the send and receive data in our application. To do this, we'll copy and edit one of the existing structures.

5.1) Press Stop (Escape-button).

5.2) In the mailbox\_init() function in main.c, **uncomment** the code:

```
/* Add for lab RX dataframe */
//lab_rx_dataframe.id           = 14; //= 0x0E;
//for (i=0; i<8; i++)
//    lab_rx_dataframe.data.data[i] = 0;
//mailbox_lab_rx                = 7;
//set_rec_std_dataframe_can0(mailbox_lab_rx, lab_rx_dataframe.id );
```

5.1) The structure and the mailbox variables need to be defined. **Open** the “**declare.h**” file and uncomment them (as you did in 0 above.)

We now have a CAN mailbox and data structure set up to receive data.

### Question

What dataframe needs to be sent onto the CAN bus for this slot to pick up a message?

### The CAN interrupt

When the mailbox receives data it will generate an interrupt . The interrupt routine `Can0_Rx_ISR()` will be called when a CAN frame arrives because in the function `can_initial()` the CAN receive interrupt is set (`c0recic = CAN0_LVL`). Polled mode is available as an alternative.

5.2) **Open** the “**interrupts.c**” file.

At the bottom of this file is the **CAN receive interrupt** function `Can0_Rx_ISR()`. A section of code at the bottom of this interrupt service uses the `check_rec_success_can0()` function to see if the interrupt was caused by a reception into the mailbox `mailbox_lab_rx` that we just set up. **Uncomment** the code.

```
/* Added for lab */
/* uncomment the lines below to set a flag which gets set to 1 whenever
   an interrupt is received and the mailbox we setup has valid data */
//    if (check_rec_success_can0(mailbox_lab_rx, &lab_rx_dataframe) == 1)
//    {
//        /* Flag to update display */
//        lab_msg_flag = 1;
//    }
//    /* lab-add end */
```

Now we need to add code in the main routine to do something with the received frame whenever `lab_msg_flag` has been set by the interrupt routine!

- 5.3) In function **main**, line ~130, **uncomment** the following code.

```
//if (lab_msg_flag)
//{
//    RED_LED = LED_ON;
//    DisplayString(LINE1_1STCH, "Received");
//    DisplayString(LINE2_1STCH, "CAN msg");
//    lab_msg_flag = 0;
//}
```

The if “conditional” checks whether the interrupt flag was set. If so, the code turns the red led on and clears the flag.

### Question

CAN message slot \_\_\_\_\_ is now ready to receive CAN data frames with the ID \_\_\_\_\_.

If a message is received, what will happen? \_\_\_\_\_

- 5.4) In the next section we will use the sniffer to send a message and hopefully receive it with our newly set-up mailbox.

6


## Transmit CAN data with monitor Debug with HEW Watch window

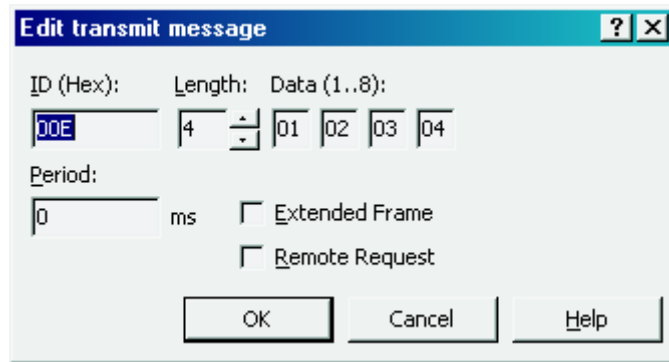
10 minutes

**Overview:** We will transmit data with SysTec CAN Monitor. We will then debug received can data by

- setting a breakpoint
- adding a variable to the HEW Watch window
- looking at the received data.

### Procedural Steps:

- 6.1) Press Stop  (Escape-button)
- 6.2) **Set a breakpoint** at the line `lab_msg_flag = 0`, the line you added in section 5. Do this by double-clicking in the left margin, in the “event gutter” creating a **blue dot**, or a ‘hardware breakpoint’ (a red dot is a software breakpoint and is slower to use).
- 6.3) Hit Reset->Go, or **Shift+F5**.
- 6.4) Switch back to viewing the **PCAN monitor** and right click in the **Transmit** section of the window.
- 6.5) Select **New** in the dialog box that comes up.
- 6.6) In the New Transmit Message dialog you can see that you can modify the ID, the number and value of the bytes in the frame and the period. **Set**
  - **ID** to **00E** (hex; this is 14 decimal).
  - **Length** to **4**.
  - In the **dataframe** value fields, enter values. For example **01, 02, 03, 04**.
  - Leave **Period** at **0** as we will manually transmit the messages rather than have them be periodically sent by the monitor.



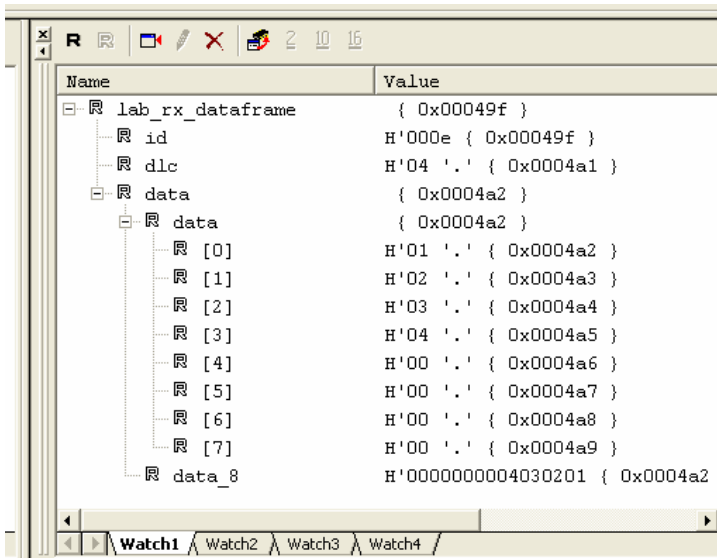
- Click **OK**.

- 6.7) **Transmit** the frame by double-clicking on the message number in the transmit window – you should see the Count column increment.
- 6.8) The red led should now be lit.

### Question

What is shown on the RSK LCD? \_\_\_\_\_

- 6.9) Open a watch window in HEW with **Ctrl+W**. If you cannot see anything, you may need to move a window frame to see it.
- 6.10) Add `lab_rx_dataframe` that you added in section 5. **Right click** inside the window, select **Add Watch** and type in the variable name. You could also highlight `lab_rx_dataframe` in the `mailbox_init()` function and then drag it into the watch window.



- 6.11) **Check** that the **received data** is the same as the **sent data**.

/End of lab.