CANalyzer-Getting Started Tutorial

Introduction

This tutorial will focus on the software development of CAN tool. Including the database creation, control panel configuration. Note that this tutorial is based on you have certain knowledge of CAN bus, structure layer, and protocols.

What is CAN?

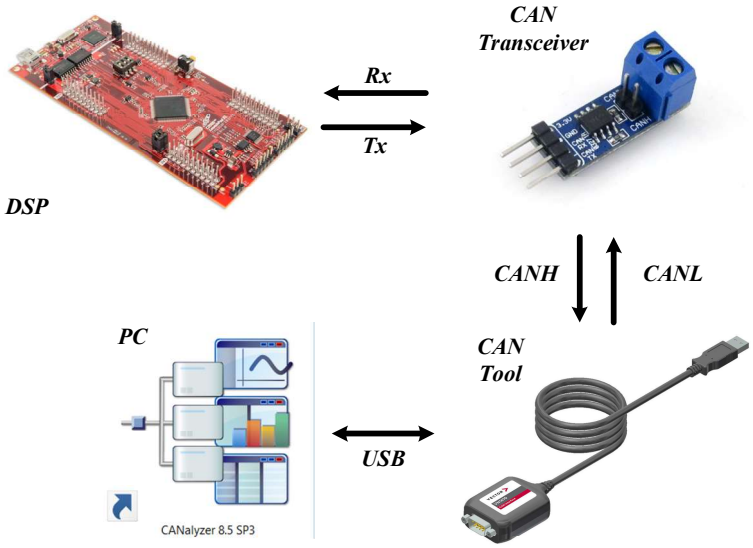
A Controller Area Network (CAN bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but is also used in many other contexts.

What is CANalyzer?

CANalyzer is the comprehensive software tool with intuitive operation for analysis and stimulation of network communication. Use CANalyzer to check whether and what type of communication is occurring on the network. In addition to sending or recording data, interactive ECU diagnosis is also possible. For every application it offers powerful basic functions for beginners as well as extensive detailed functions for experienced users.

Before started

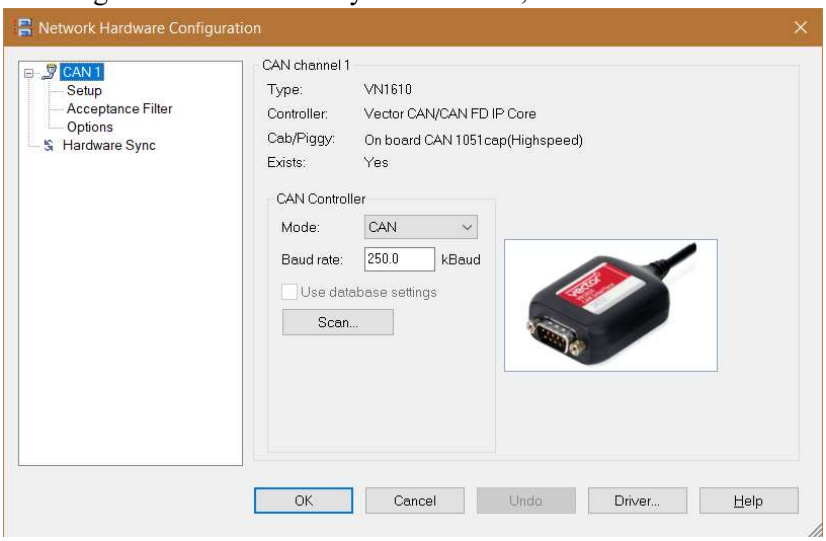
To realize the CAN control, certain hardware is needed. The typical system connection diagram is as shown below.



CANalyzer installation

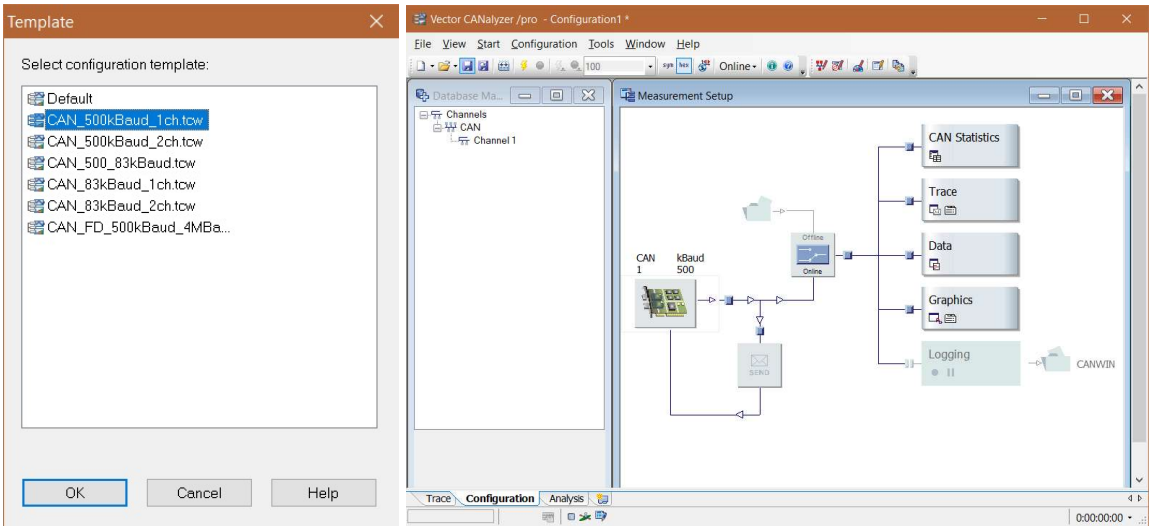
The installation for CANalyzer is pretty easy, just pay attention that the driver of the CAN tool is needed. In this tutorial, the CAN tool is VN1610 from Vector. The CANalyzer software version is 8.5 SP3. After the installation of the driver and software, follow the steps below to check the device connection status:

1. connect the USB of the CAN tool to the PC;
2. Open CANalyzer;
3. On the top of the software, click Configuration->Network Hardware;
4. If you could see the figure of the CAN tool you connected, it's done.



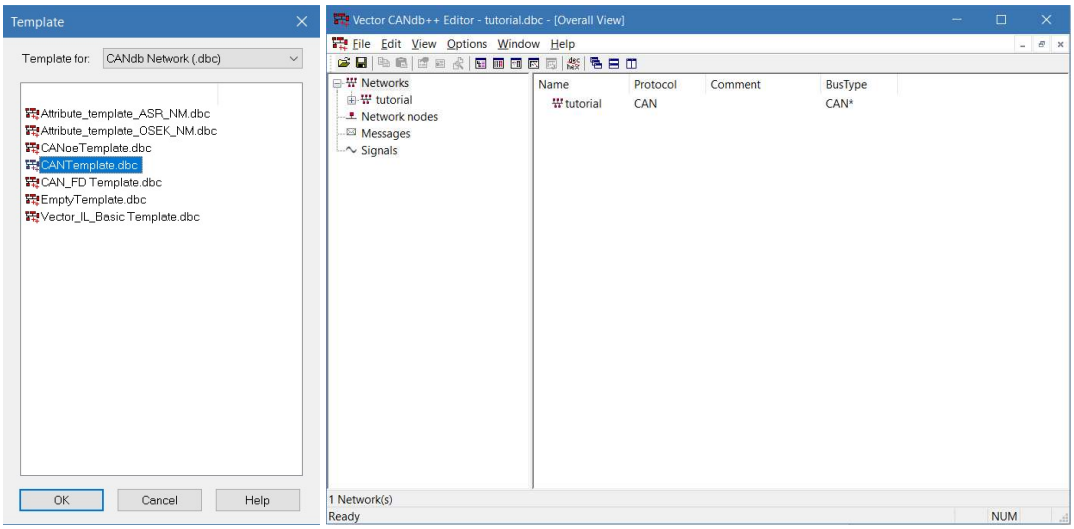
Create a new project

Click File->New Configuration. You can select the template of the new project. Here in this tutorial, let's select CAN_500kBaud_1ch.tcw. Therefore, a CAN project with 500kBaud is created. You can save the project by click File->Save Configuration. The initial interface is as shown below.



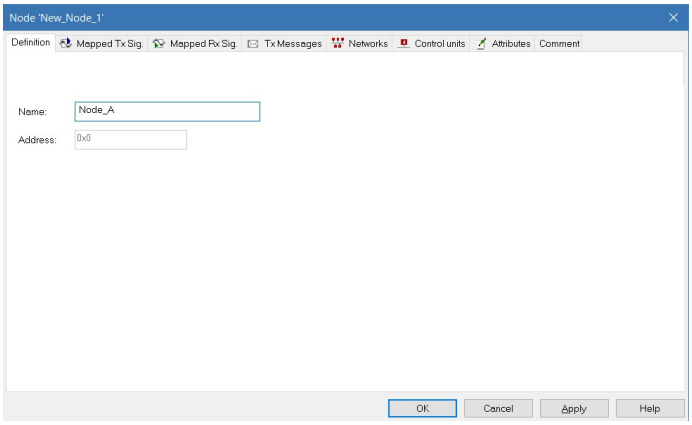
Using CANdb++ Editor to create the database

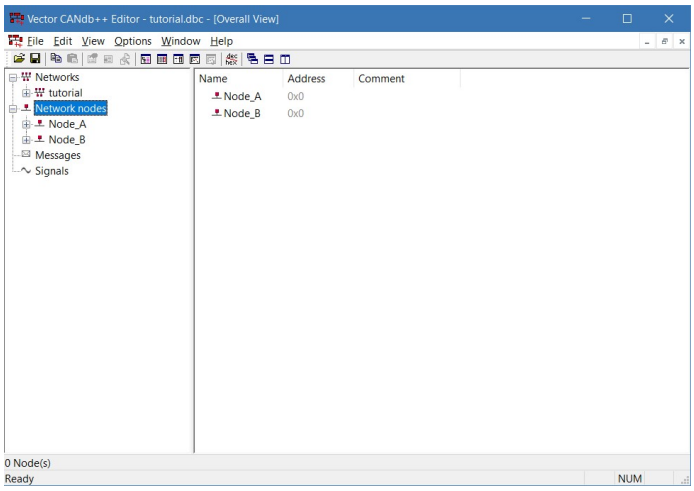
Click the CANdb++ Editor button on the top bar to start the CANdb++ Editor. Then, click File->Create Database. Select CANTemplate.dbc. Name the dbc file and select the desired path to save the database.



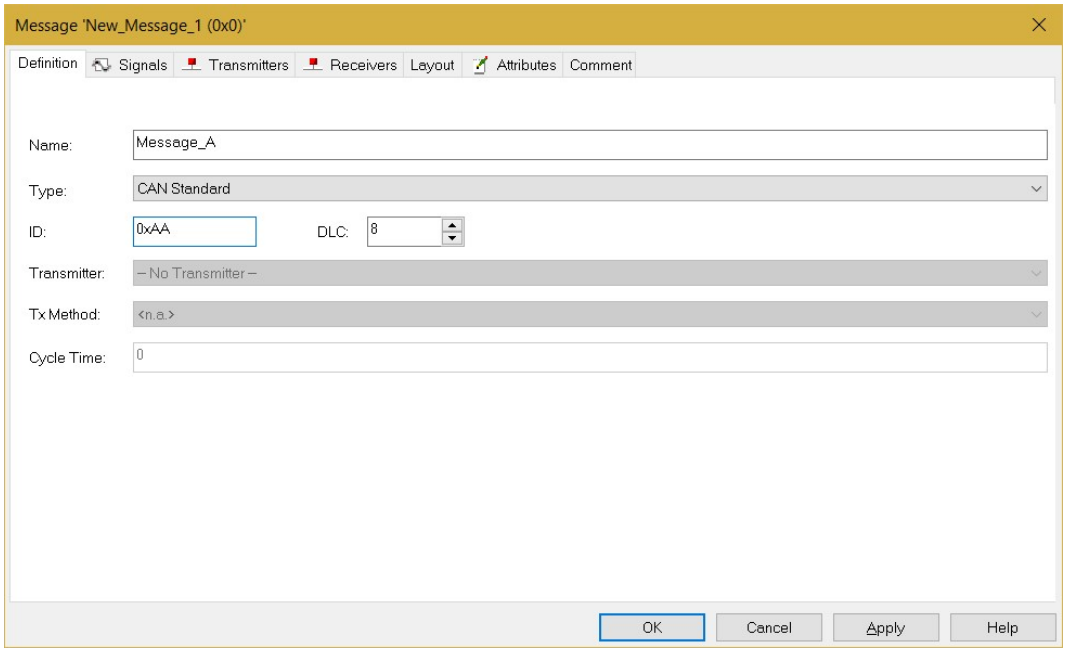
Click the CANdb++ Editor button  on the top bar to start the CANdb++ Editor. Then, click File->Create Database. Select CANTemplate.dbc. Name the dbc file and select the desired path to save the database.

Right click Network nodes->New to define the network nodes. In this step, the only thing you need to do is to name the nodes. Here we will create 2 nodes. Simply named as Node_A and Node_B. After the nodes are created, you will see that two new nodes are listed under the Network nodes content.

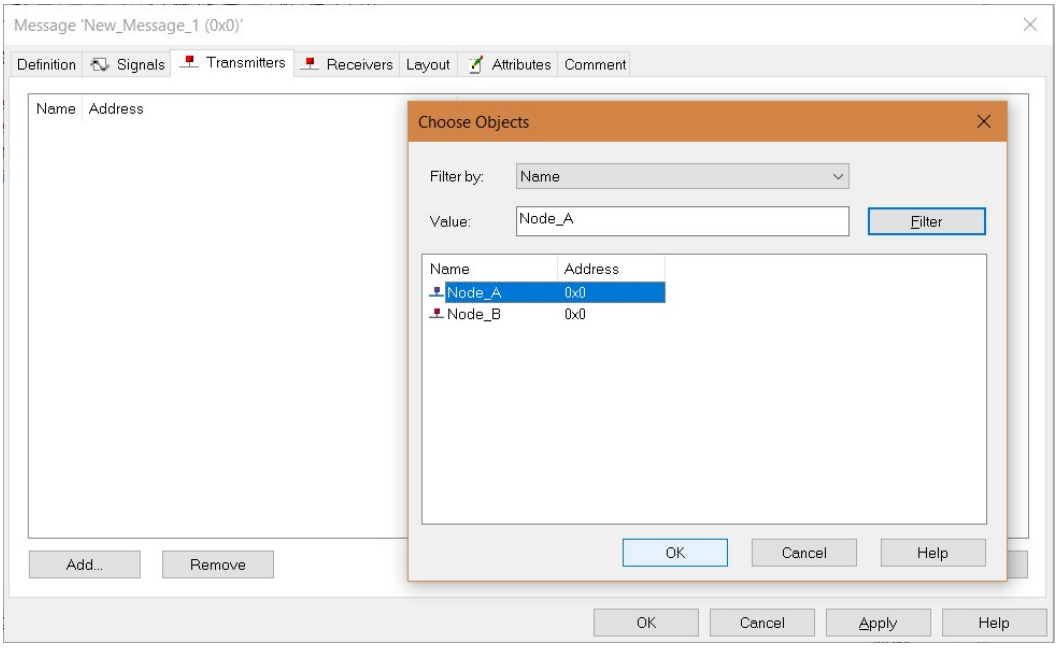




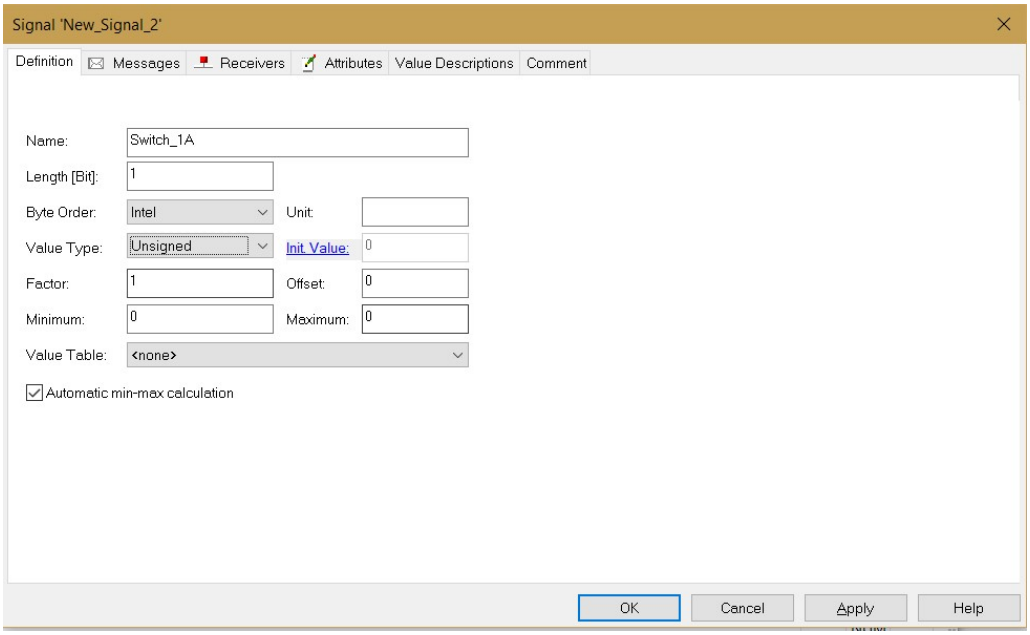
After the nodes are added, the next step is to define the CAN message. Right click Message->New. In the pop-up window, you need to define the message name, ID(mailbox ID), DLC(data length code), etc.



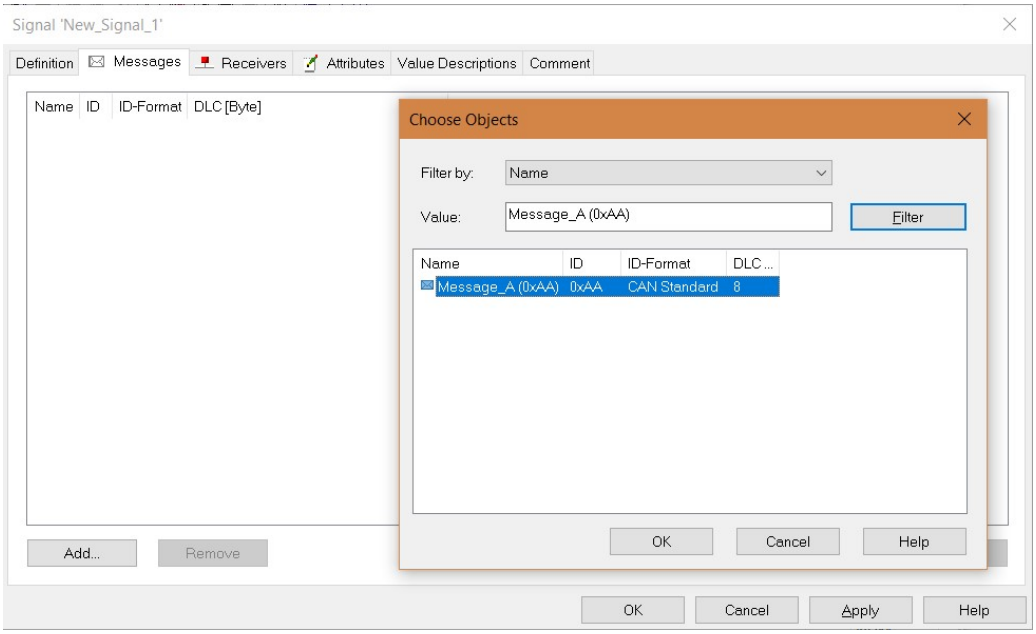
Under the “Transmitters” tab, click “Add” button to add the Node_A as the transmission node. Which means, this message is send out form Node_A. Another way to do this is simply drag the message to the desired node to add this node as a transmission node.



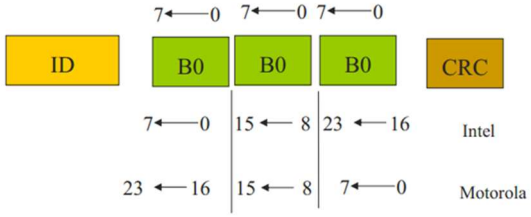
As the message is defined, it’s time to include some signals in the message. To create a signal. Right click Signal->New, fill out the pop-up window as shown below. Here we will define a signal that represents the switching stauts, which only has two values, 0(OFF) or 1(ON). Thus, 1 bit data length is enough.



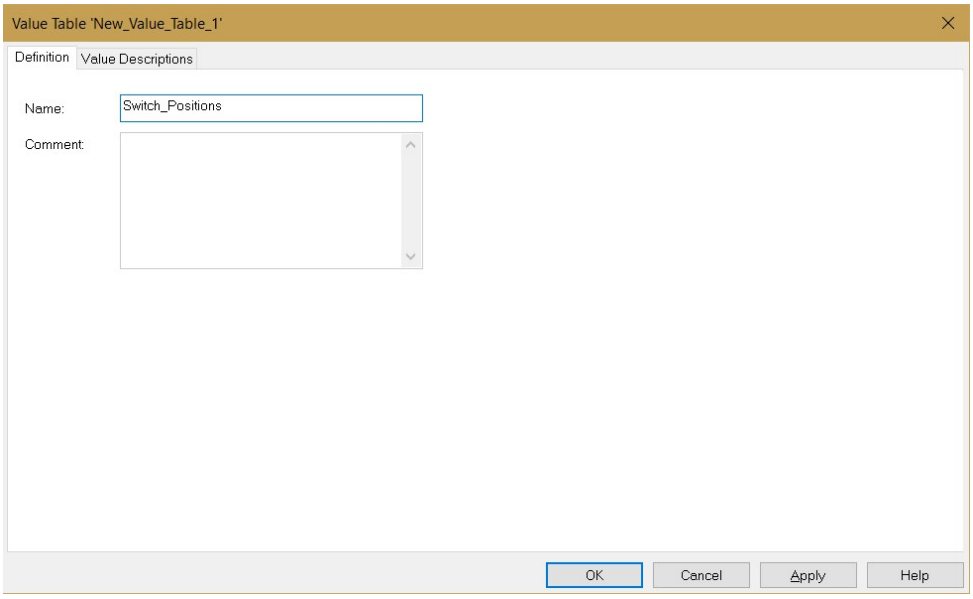
To put this signal to the message we just created, switch to the “Message” tab, add Message_A. Of course, drag the signal to Message_A also works.

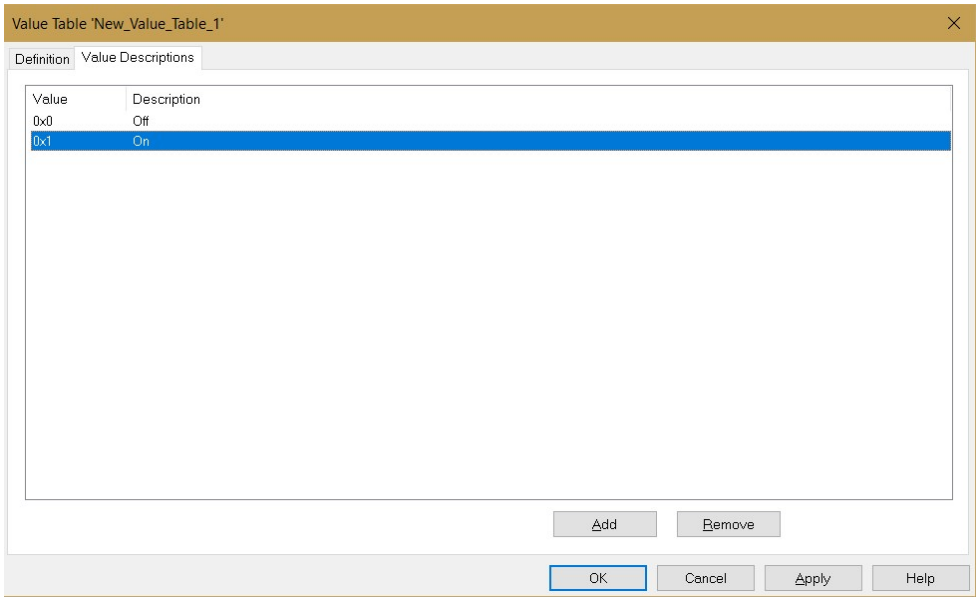


Note that in the signal definition window, the “Byte order” has two options, “Intel” and “Motorola”, the difference between these two fommat can be identified as below:

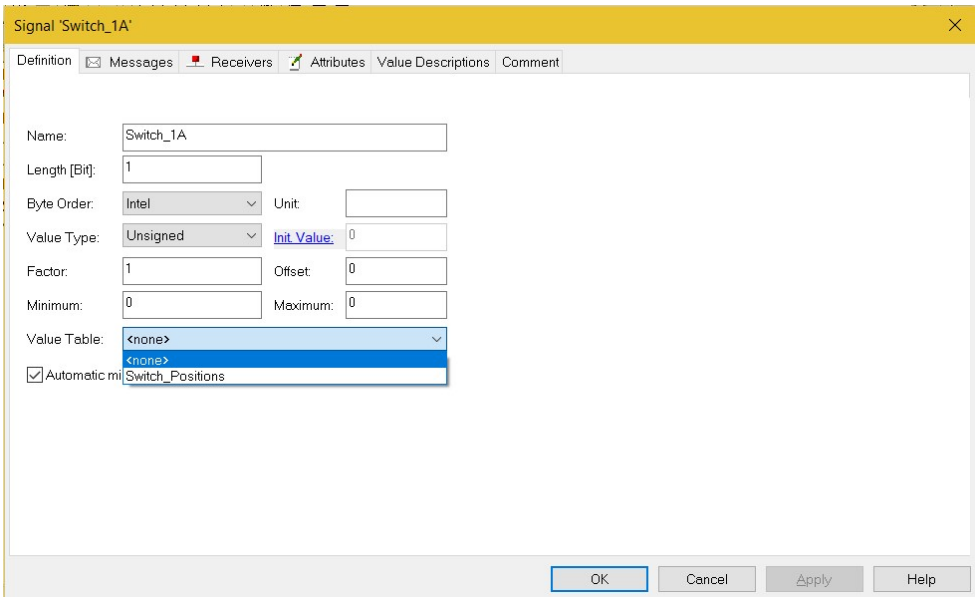


You might notice that using 0 and 1 to represent OFF and ON is not very intuitive. Why not directly use “On” and “Off” to define the signal value? You can! On the top bar of CANdb++ Editor, click View->Value Tables, right click the blank area->New. Type the information as shown below, be sure to save the work.



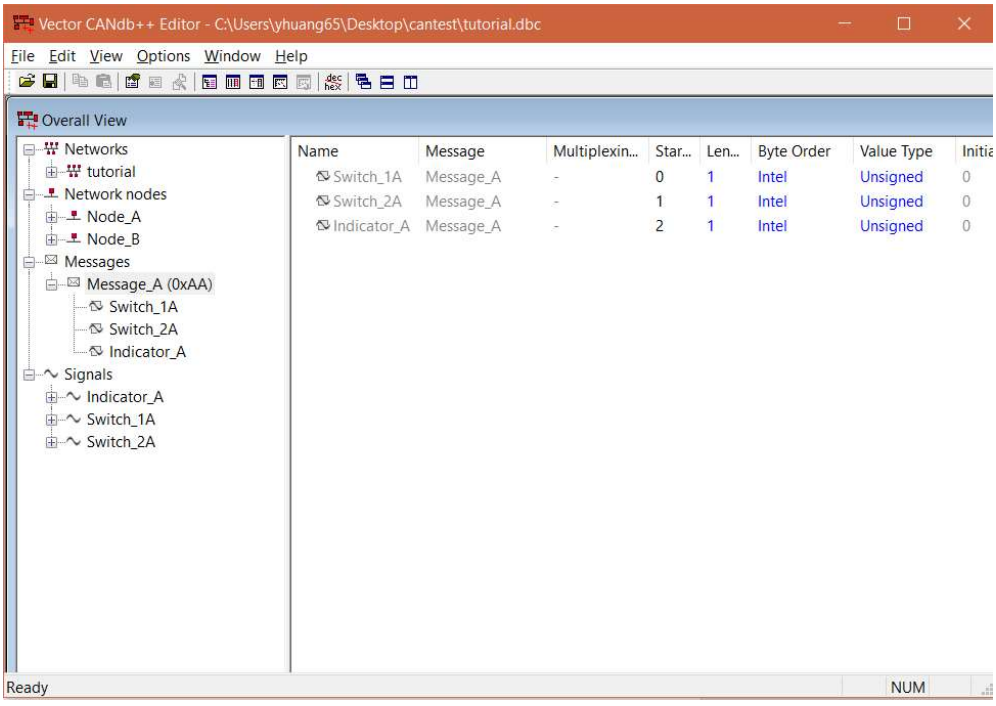


After the value table is defined. It has to be associate to the signal. Right click the signal that need to be associated with(Switch_1A)->Edit Signal. Under the defination tab, unfold drop-downlist of Value Table. You will see the value table you just created, select that table. Then your signal defination is done.



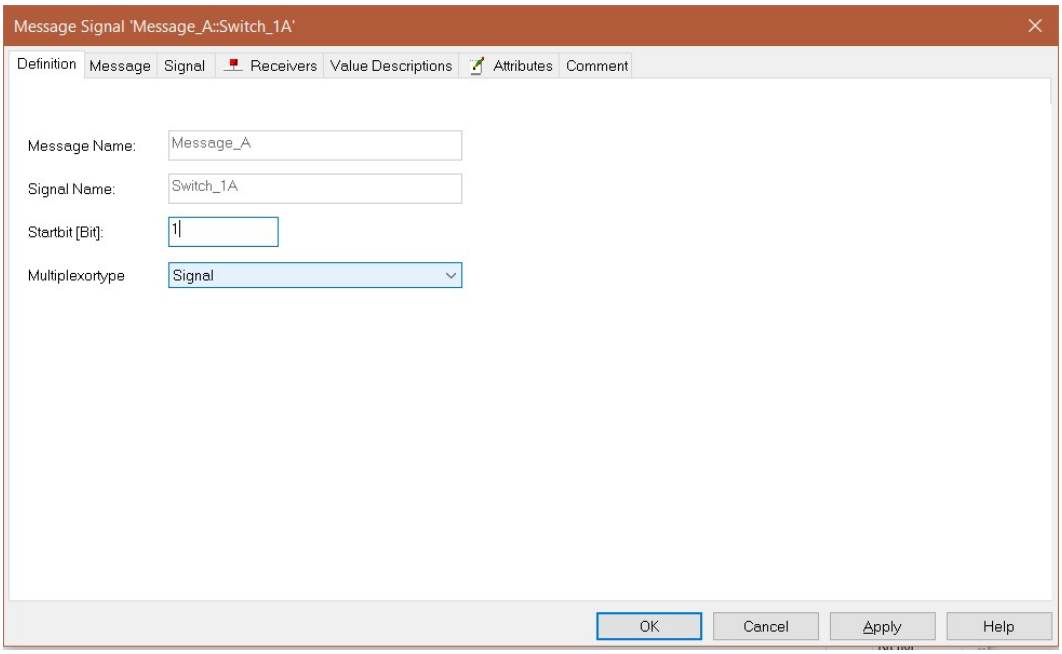
Follow the steps we've done to create another signal. You can right click the signal that already crated under the root directory of "Signals", select "copy", and then paste it. Double click the pasted signal, rename it as "Switch_2A". Use the same method to associate this signal to Message_A as we did before, which is very efficient.

Let's continue to create the third signal, "Indicator_A", same as what we did before.

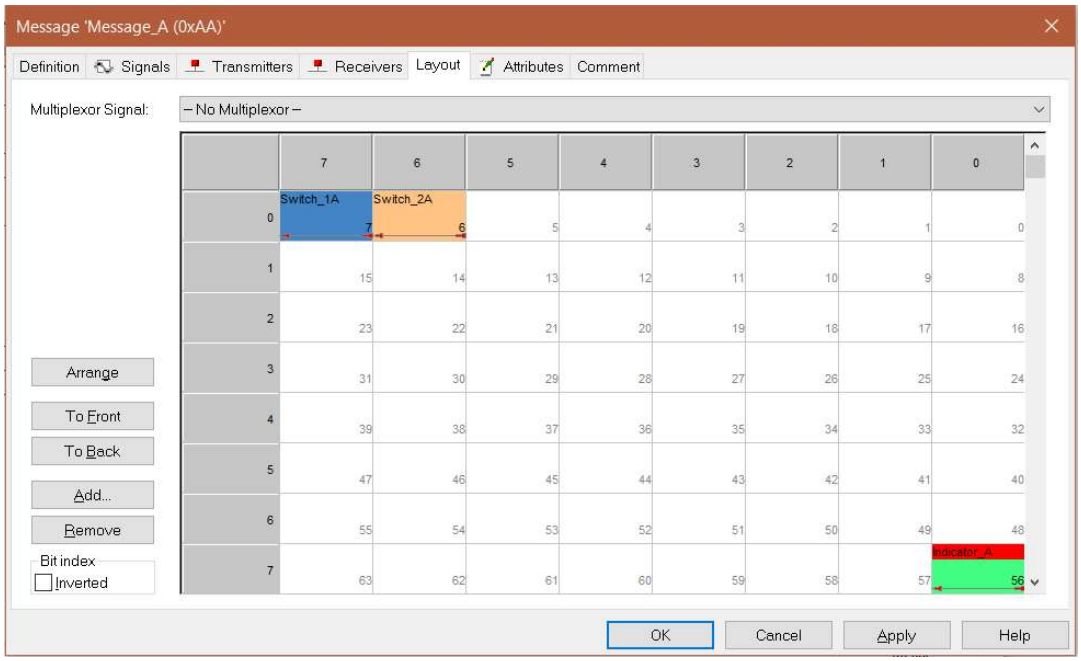


Since we have multiple signals in Message_A now, how to arrange these signals in the message remains to be done. Message_A is defined as a 8-byte data. Assume we want the first bit of Message_A represents the signal "Switch_1A", and the second bit represents "Switch_2A", make the last bit represents "Indicator_A". There are two approach to make it.

First, right click the signals that has already been associated to Message_A, under the root directory of messages. Click edit mapped signal, in the pop-up window, fill the “Startbit” with the position you want the signal begins with. For example, for “Switch_1A”, fill in 1. For “Switch_2A”, fill in 2. For “Indicator_A”, fill in 56.



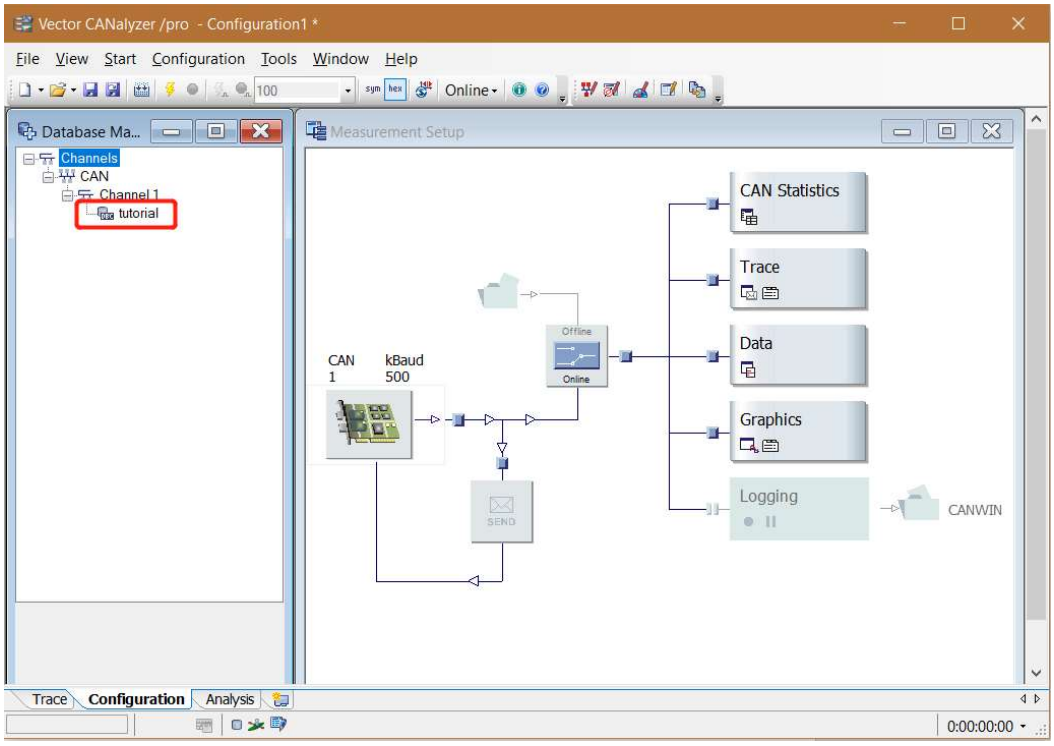
Second method is, right click Message_A-> Edit Message. Go to the “Layout” tab. You can drag the colored blocks, which represents different signals in this message, to relocate the position.



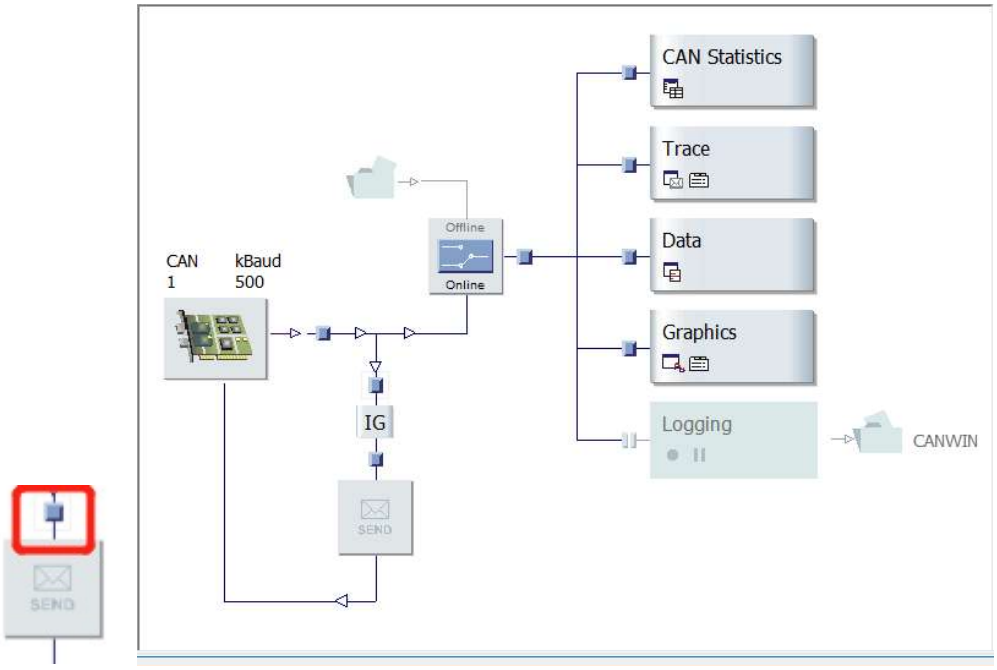
By now, you’ve succesfully created a complete message: Message_A. Don’t forget to save the database.

Configure the CAN UI

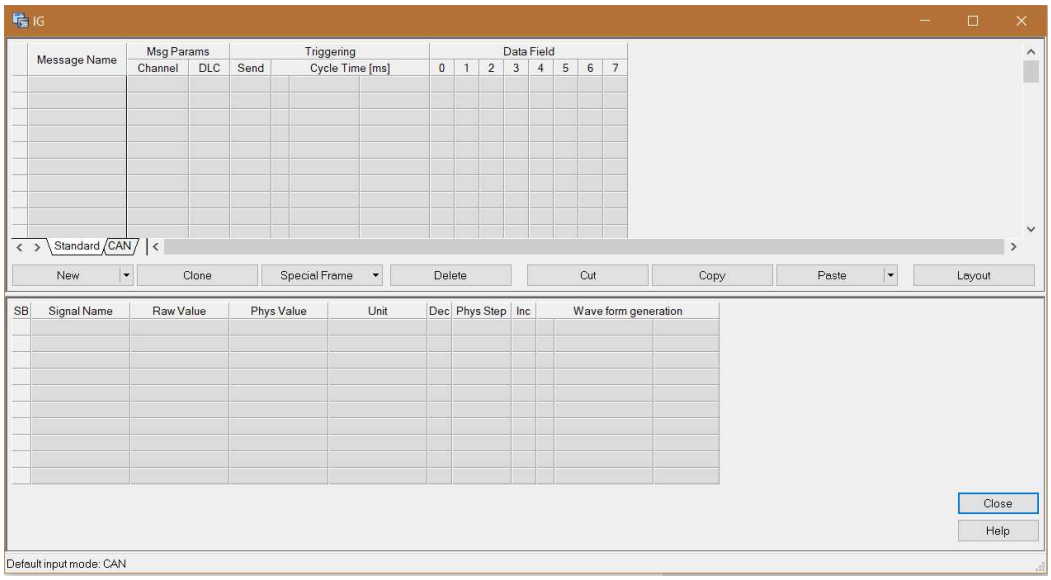
With the database successfully created, let's go back to the CANalyzer main interface. On the left side, the database management window, right click Channel 1->Add Database, select the database you just saved.



On the right side, the Measurement Setup window, in the SEDN loop, right click the square node in the loop->Insert Interactive Generator Block. This node is help you to send command to the controller though CAN bus.

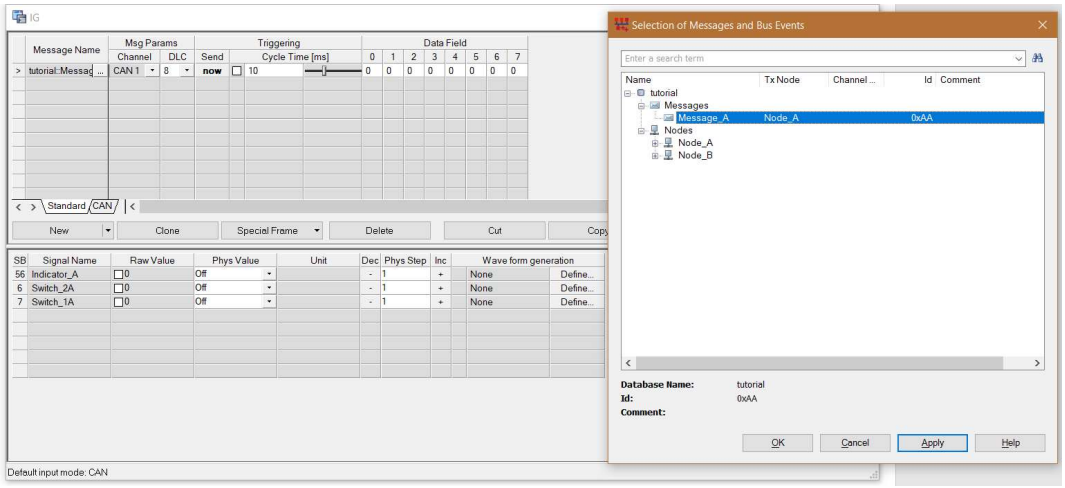


Double click the IG block, you will see interface below:



Double click the first row, select the Message_A we created in database. The included message and signals will appear on the IG interface. On the right side of Message_A, it gives the related setting. When sending this message, you can either click “now” to send it once, or check the box right after “now” button to make this message been send periodically. The cycle time could be adjusted also.

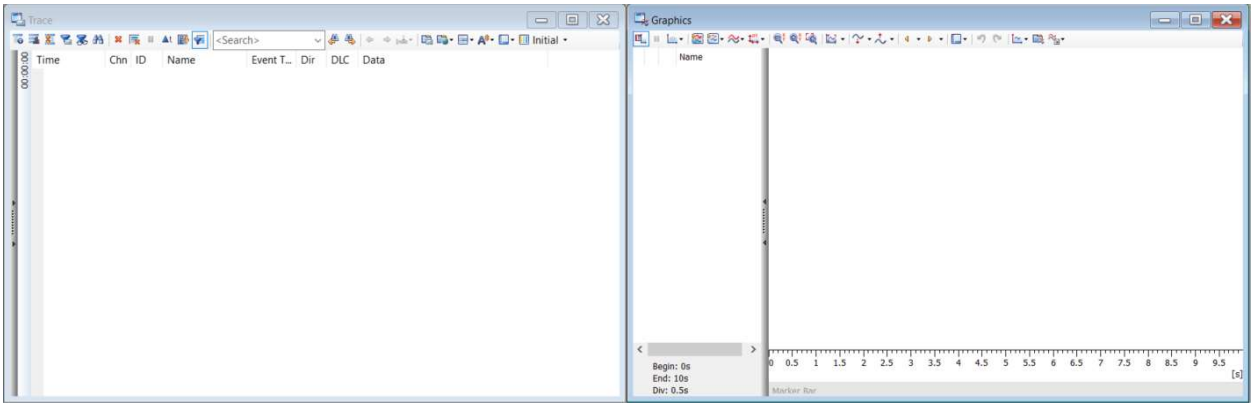
On the bottom part. Three signals that be involved in Message_A are also listed. You can choose the value or status of each signals, these settings will been send out simultaneously as the message.



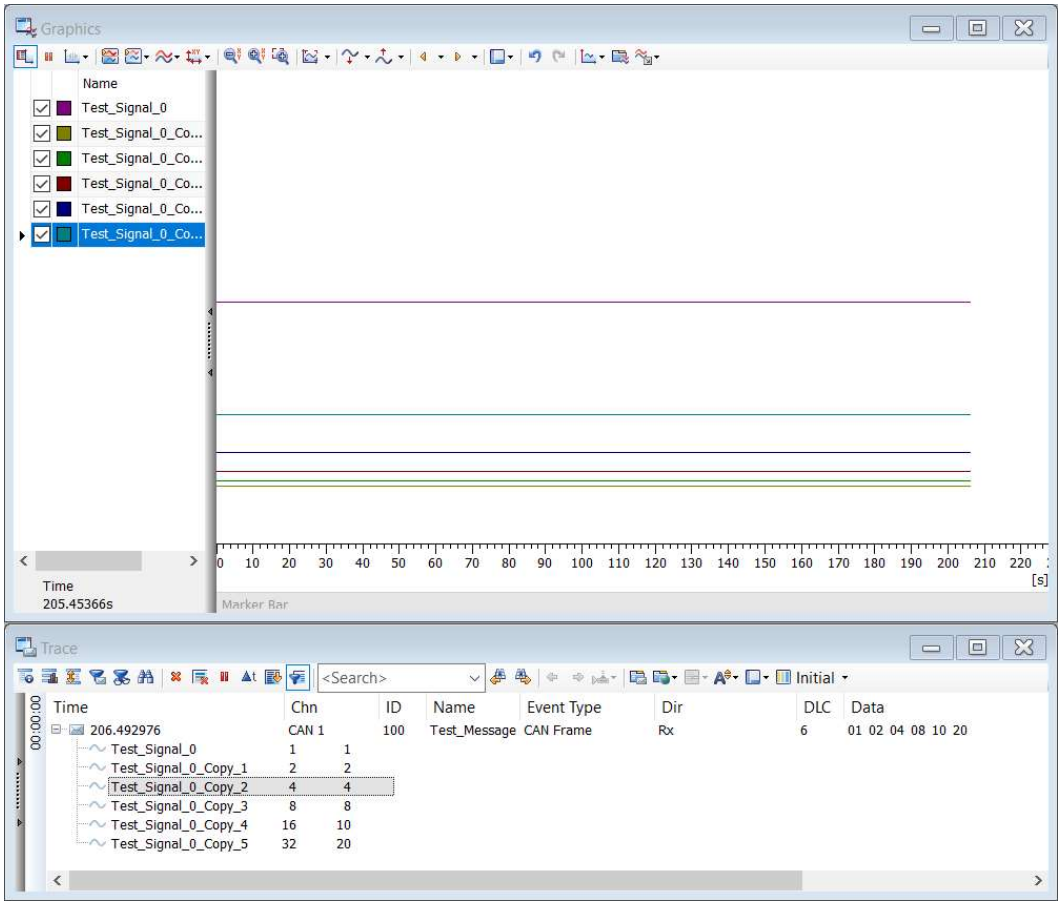
Signal Monitoring

We definitely want to see the signals that coming back from the controller. To monitor these data, two tools are often used.

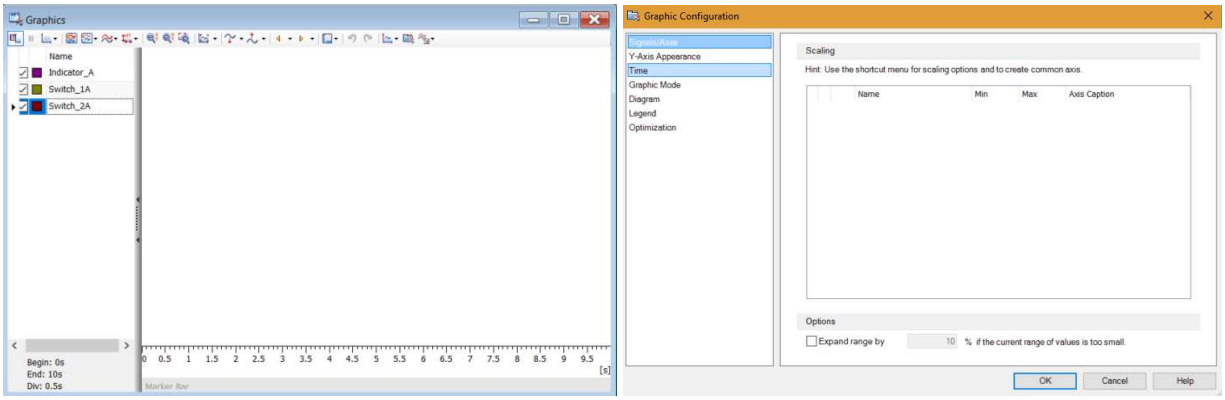
On the Measurement Setup window, double click “Tace” and “Graphics”, you will find that two more windows will appear on your interface as below:



On the Trace window, you don't have to do anything, because once the connection is built, during the data communication between controller and PC, the defined messages and signals will automatically appear on this window and refresh with every communication cycle. An example during the communication is as shown below:



On the Graphic window, in order to see the variation trend of specific signal, right click the left side blank area-> Add signals to add the items you want to monitor. Then during the communication, you can observe the real-time plot here on the right. More plot settings, such as axis settings, colors could be found if you double click the right side blank area.



Till now, the configuration part for the CANalyzer side is finished.

Other Related Work

DSP coding:

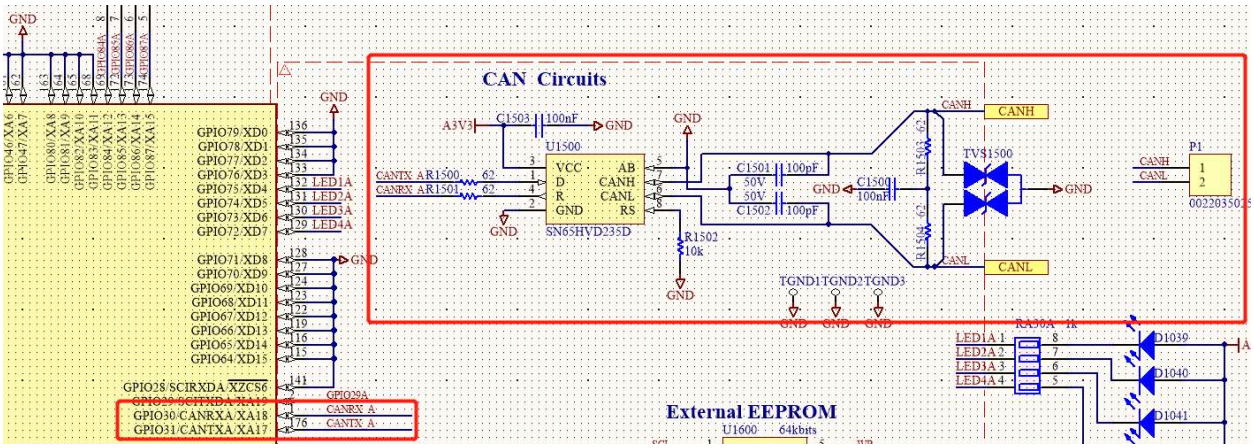
Follow the document: “Enhanced Controller Area Network (eCAN) Reference Guide”. The eCAN module should be configured, all the mailboxes that will be used should have unique message ID.

```
void CAN_Config(void) {
    ECanaRegs.CANME.all = 0x00000000;       //Ensure all mailboxes disabled; required before writing the MSGIDs
    ECanaRegs.CANMD.all = 0xE0008000;       //We will use mailboxes 0-14 & 16-31 as general TRANSMIT mailboxes
    ECanaMboxes.MBOX0.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX1.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX2.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX3.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX4.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX5.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX6.MSGID.all             = 0x80000000;       //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX7.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX8.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX9.MSGID.all             = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX10.MSGID.all            = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX11.MSGID.all            = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX12.MSGID.all            = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX13.MSGID.all            = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX14.MSGID.all            = 0;        //Use 11-bit ID, No acc Mask, No auto-answer
    ECanaMboxes.MBOX0.MSGID.bit.STDMSGID    = 0x00A0;   //Set Std Msg ID value; Product Name
    ECanaMboxes.MBOX1.MSGID.bit.STDMSGID    = 0x00A1;   //Set Std Msg ID value; SW Revision
    ECanaMboxes.MBOX2.MSGID.bit.STDMSGID    = 0x00A2;   //Set Std Msg ID value; VIN# Higher byte
    ECanaMboxes.MBOX3.MSGID.bit.STDMSGID    = 0x00A3;   //Set Std Msg ID value; VIN# Lower Byte
    ECanaMboxes.MBOX4.MSGID.bit.STDMSGID    = 0x00A4;   //Set Std Msg ID value; Security Response
    ECanaMboxes.MBOX5.MSGID.bit.STDMSGID    = 0x00A5;   //Set Std Msg ID value; Vehicle Status
    ECanaMboxes.MBOX6.MSGID.bit.EXTMSGID_L  = 0x00A6;   //Set Std Msg ID value; Speed/Power
    ECanaMboxes.MBOX7.MSGID.bit.STDMSGID    = 0x00A7;   //Set Std Msg ID value; Vehicle Energy trip
    ECanaMboxes.MBOX8.MSGID.bit.STDMSGID    = 0x00A8;   //Set Std Msg ID value; Vehicle Energy Life
    ECanaMboxes.MBOX9.MSGID.bit.STDMSGID    = 0x00A9;   //Set Std Msg ID value; Motor Temps
    ECanaMboxes.MBOX10.MSGID.bit.STDMSGID   = 0x00AA;   //Set Std Msg ID value; Motor Current
    ECanaMboxes.MBOX11.MSGID.bit.STDMSGID   = 0x00AB;   //Set Std Msg ID value; INV_internal status
    ECanaMboxes.MBOX12.MSGID.bit.STDMSGID   = 0x00AC;   //Set Std Msg ID value; Performance Setting
    ECanaMboxes.MBOX13.MSGID.bit.STDMSGID   = 0x0623;   //Set Std Msg ID value; Battery Voltages
    ECanaMboxes.MBOX14.MSGID.bit.STDMSGID   = 0x0624;   //Set Std Msg ID value; Battery Currents
```

```
void SendDataCAN(void)
{     if (CpuTimer1.InterruptCount % 1 ==0)
      {   ECanaMboxes.MBOX3.MDL.word.HI_WORD  = (int16)(FAULT_PFC);//                    //FaultNo
          ECanaMboxes.MBOX3.MDL.word.LOW_WORD = (int16)(IGA_FLT.Out);//vga            //Ia
          ECanaMboxes.MBOX3.MDH.word.HI_WORD  = (int16)(IGB_FLT.Out);//picd           //Ib
          ECanaMboxes.MBOX3.MDH.word.LOW_WORD = (int16)(IGC_FLT.Out);//picq           //Ic
          ECanaRegs.CANTRS.all = 0x0008;

          //ECanaMboxes.MBOX4.MDL.word.HI_WORD  = (int16)(sys.RunStatus);//pidc.err*10    //Ua
          ECanaMboxes.MBOX4.MDL.word.HI_WORD  = (int16)(UGA_FLT.Out);
          ECanaMboxes.MBOX4.MDL.word.LOW_WORD = (int16)(UGB_FLT.Out);//pllangle         //Ub
          ECanaMboxes.MBOX4.MDH.word.HI_WORD  = (int16)(UGC_FLT.Out);//dqugD           //Uc
          ECanaMboxes.MBOX4.MDH.word.LOW_WORD = (int16)(VDC_FLT.Out);//dqugQ           //PLL_Angle
          ECanaRegs.CANTRS.all = 0x0010;
```

Circuit layout:

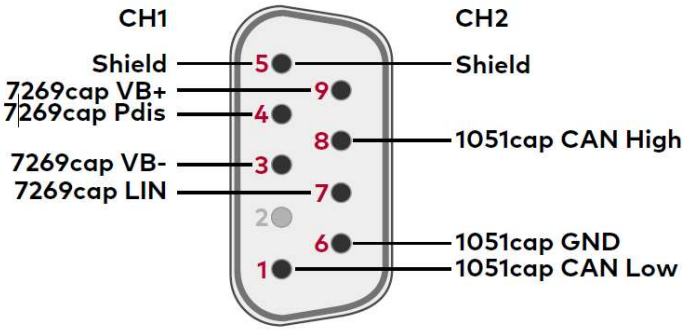


CAN interface:

D-SUB9 connector

The pin assignment of the D-SUB9 connector (CH1 and CH2) is as follows:

CH1/CH2



Pdis: power disable