Automotive Technology

VE/TI Lab

Experiment 3

Name	Hassan Al-Faiz Bin Mohamed Ahmed (HB17031)
Group Number	
Date	21/8/2020

3. CAN-LIN Gateway

In this experiment, we want to analyze the activation of two connected bus systems by a control unit used for development purposes, once more issued by Vector itself.

In the experiment, the hazard light function in the instrument cluster, which is already known from the previous experiment, should be realized via the CAN bus. This function should be controlled by a LIN keypad.

The experiment is divided into two parts. In the first part, the LIN keypad will be put into operation, in the second part, the same will be done for the instrument cluster.

Sending and Receiving LIN messages with a development ECU

3.1. Introduction

- Revise the chapter dealing with the LIN bus in your lecture documents.

  a) How is a LIN message set up?
  All data is sent in a Frame Slot which contains a Header, a Response and some Response Space so the Slave will have time to answer. Every frame is sent in a frame slot determined by the LIN Description File (LDF). Messages are created when the Master node broadcasts a frame containing a header. The Slave node(s) then fills the frame with data depending on the header sent from the Master. The data that are exchanged in the frames are referred to as Signals.

  b) What does start bit and stop bit mean and what is their polarity?
  Two bytes are sent, each consisting of a start bit, followed by eight data bits (bits 0-7), and one stop bit, for a 10-bit character frame. The last data bit is sometimes used as a parity bit.

  The **start bit (LOW)** is used to signal the beginning of a frame. The **stop bit (HIGH)** is used to signal the end of a frame. The data is contained in the data bits and the parity bit is an extra bit that is often used to detect transmission errors.

  Stop bit is required so that there is always a transition at the leading edge of the start bit. Otherwise, if the last data bit ov the previous character happened to be of the same polarity as a start bit, the receiver wouldn't be able to see the start bit if a new character was sent immediately.

c) What are the roles of the master and the slave in the LIN network?
   LIN is a broadcast serial network comprising 16 nodes (one master and typically up to 15 slaves). All messages are initiated by the master with at most one slave replying to a given message identifier. The master node can also act as a slave by replying to its own messages.

   The slave waits for synch break and then the synchronization between master and slave begins on synch byte. Depending on the identifier sent from the master the slave will either receive or transmit or do nothing at all. A slave that should transmit sends the number of bytes which the master has requested and then ends the transmission with a checksum field

d) What ae the signals' levels that are defined in the LIN bus?

e) What are the electrical connections needed for the LIN bus?

- Low-cost silicon implementation based on common UART/SCI interface hardware, an equivalent in software or as a pure state machine.
- Self-synchronization without a quartz or ceramics resonator in the slave nodes
- Low cost, single-wire to connect automotive electronic control units (ECUs)

3.2. Introduction to CAPL

The behaviour of simulated network nodes (LIN, CAN) in CANoe on the bus can be programmed directly in CANoe by the programming language CAPL (Communication Access Programming Language) that is similar to C. Follow the Video "Experiment 3 – Introduction to CAPL" by composing your own short examples of the functions explained. Use the help function (index search) of CANoe for the following functions:

- on start

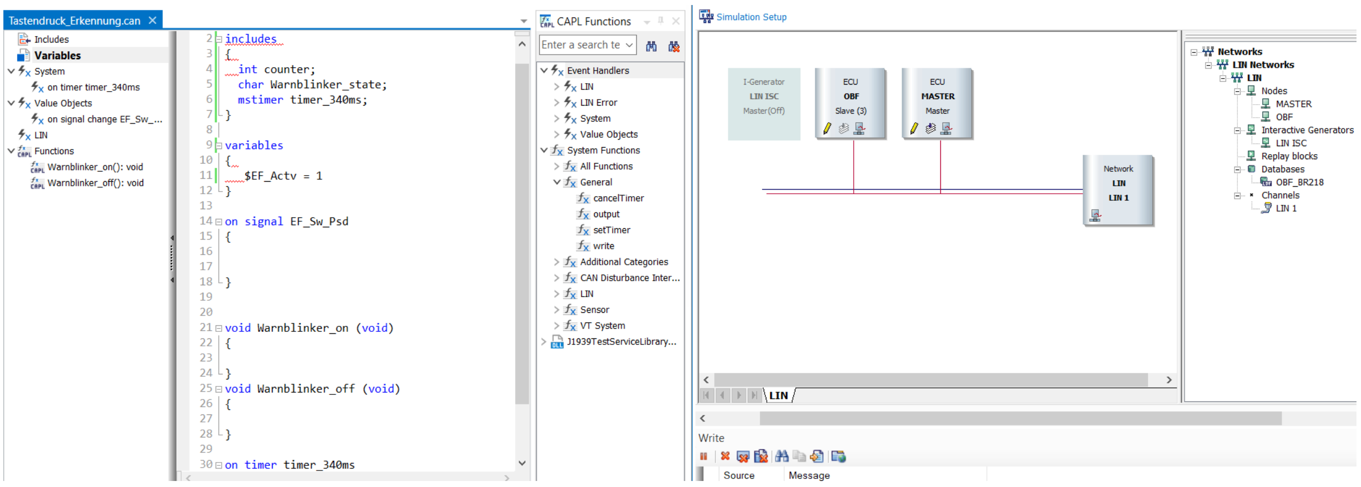
- on signal

- on timer
  (Creating a timer, initializing a timer, starting a timer, stopping a timer)

- What are the different timers that can be used?

- How can you get access to signals contained in databases by using CAPL?

- How can you get access to system variables by using CAPL?



```
includes
{
  int counter;
  char Warnblinker_state;
  mstimer timer_340ms;
}

variables
{
   $EF_Actv = 1
}

on signal EF_Sw_Psd
{


}


void Warnblinker_on (void)
{

}
void Warnblinker_off (void)
{

}

on timer timer_340ms
```

3.3. Vector driver configuration (channel settings Hardware/software)

For the hardware setup, watch the video “Experiment 3 – Hardware setup of LIN keypad”. As far as configuration in CANoe is concerned, you can skip it because you do not have the hardware necessary. If ever you want to set it up, you can do it by opening the driver configuration of the Vector hardware in CANoe clicking on *Hardware* -> *Bus hardware* -> *drivers*…

Now you can define the channels by right-hand clicking according to the following figure 3.1.

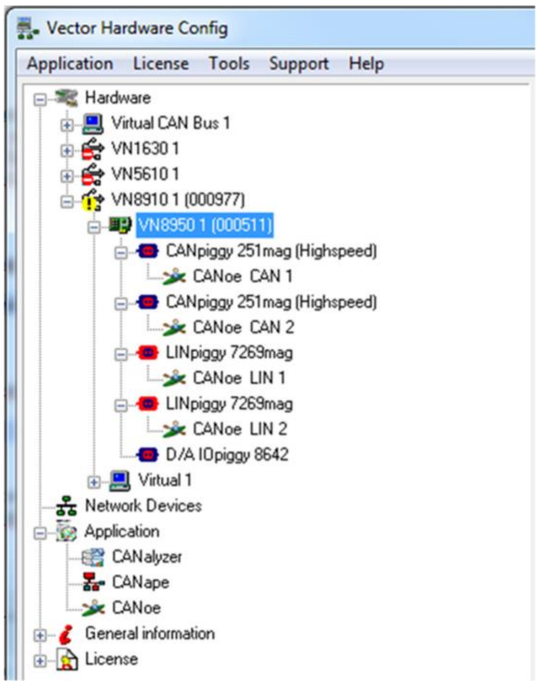


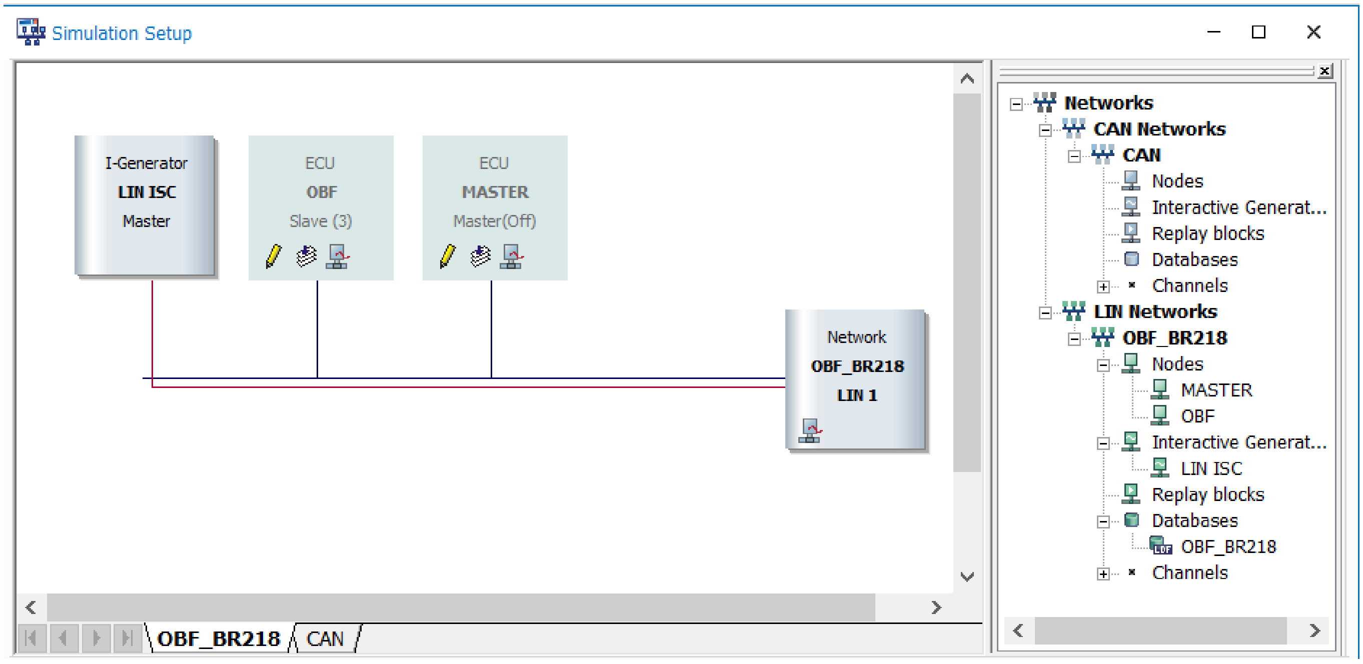
Figure 1: Vector hardware configuration

3.4. The LIN node

The LIN keypad is a component taken out of a Mercedes Benz CLS, it is manufactured by the company Marquardt. An extract of the documentation is available in the file *Kurzbeschreibung OBF.pdf*.

3.5. Visualizing LIN signals

- Copy the lab files necessary on your desktop.
- For hardware configuration, please refer to the video.
- Open the configuration file *Gateway_LIN_CAN.cfg* in CANoe.
- Open the simulation setup window by clicking on *View* -> *simulation setup* and add the database *OBF_BR218.ldf* to the LIN network. After this, deactivate the nodes *LIN master* and *OBF* by right-hand clicking on the nodes.
- Open the panel designer by clicking on *Tools* -> *Panel Designer* and create a panel that visualizes the pushing of the button on the real hazard light button.
  Find the signal *EF_SW_Psd* (edge recognition of the hazard lights button) in the database and connect it to a corresponding element.
- Save the panel as *"Tasterleiste.xvp"* in your repository.



3.6. Connecting the hazard lights function/hazard lights button

To make sure that the hazard lights button will shine when pushing it, the signal *EF_Actv* must have the value 1.

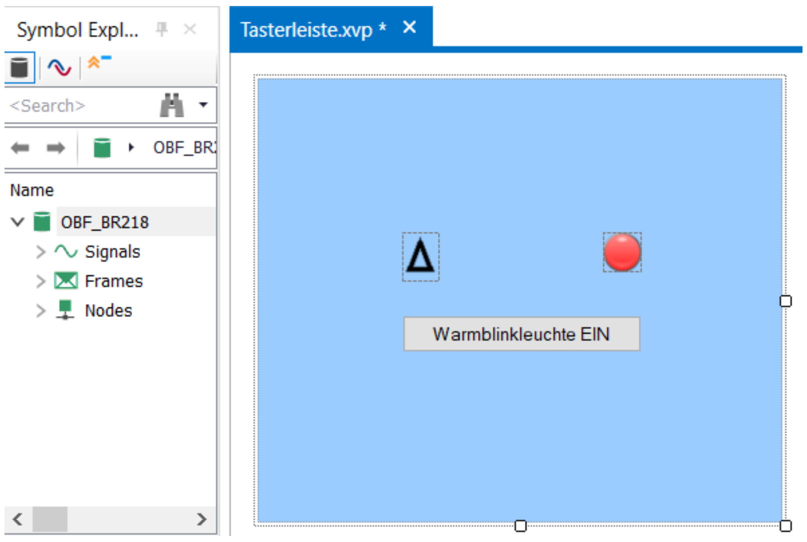
- Reopen the panel designer and add a button to the panel. Label it “*Warnblinkleuchte EIN*” in the panel.
- Assign the signal *EF_Actv* (Activation of the hazard light button’ LED) of the database to this button.

Now, when the button in the panel is pushed, the real hazard light button should flash. The software of the LIN keypad will reset the light individually after some milliseconds. As you do not have the real hardware, just put your files in the online repository. We will test them for you later.

3.6.1. Programming the hazard light button

In the following step, the buttons’ blinking behaviour will be realized when pushing the button on the real keypad.

- By means of a flowchart, show how you would realize the connection between the button and its lighting. Take into consideration, when creating the logic, that the signal *EF_SW_Psd* jumps up on value 1 (rising and falling edge) for a short moment whenever you push or release the button!
- In the simulation setup window, add the CAPL code to the master node by right-hand clicking on the *node* and then -> *configuration,* in the tab *common* -> *node specification* -> *file* and then adding the *Tastendruck.Erkennung.can* that you can find online.
- Realize the flashing of the hazard light button when pushing it, by the means of the function framework and your flowchart.
- Extend your already existing panel by a hazard light symbol that flashes when pushing the button. Therefore, use the *switch/indicator* element you can find in the library. By clicking on *Choose image* in the element’s properties, you can add the corresponding symbol. You can download the graphic file online.
- Extend your flowchart by the real blinking function of the hazard light. It should be activated when pushing the button and the hazard light button should blink by a frequency of 2.94 Hz. When pushing the button one more time, the blinking should stop.
- Transfer your extended flowchart into CAPL code and realize the function. Always be careful on using the predefined function framework.



3.7. Finishing your work

The lab folder will be needed one more time in the next experiment so do not delete it!