Automotive Technology
VE/TI Lab

Experiment 4

Name	ABDULLAH AZAM BIN FAKHARULDIN
Group Number	1
Date	30-08-2020

4. CAN-LIN Gateway (Part 2)

In the previous part, we have activated the keypad via LIN bus and we have visualized the pushing of the hazard lights button in CANoe. In the following part, we will activate the instrument cluster and show the hazard lights function both visually and acoustically on this component.

Sending and Receiving LIN messages with a development ECU

4.1. Introduction

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

a) How is a CAN message set up?

➢ The CAN Bus protocol can be summarized in the following manner: The physical layer uses differential transmission on a twisted pair wire. A non-destructive bit-wise arbitration is used to control access to the bus. The messages are small (at most eight data bytes) and are protected by a checksum.

b) What does stuffing bit mean and what is its polarity?

➢ When a CAN node detects an error in a transmitted message, it transmits an error flag which consists of six bits of the same polarity. The bit stuffing mechanism prevents six consecutive bits from having the same polarity by inserting a bit of opposite polarity after the fifth bit.

c) How is the messages' possible collision on the bus avoided?

➢ Logical collisions can be avoided completely by making source node ID a part of arbitration field and enforcing node ID uniqueness.

d) What are the properties of the CAN bus concerning interference immunity?

➢ The signals on the two CAN lines will both be subject to the same electromagnetic influences, and so the difference in voltages between the two lines will not vary. Because of this, the bus is also immune to electromagnetic interference. Electrical noise immunity achieved by transmitting simultaneously one bit on two lines CAN High and CAN Low with a potential change in opposite directions.

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

➢ This bus uses differential wired-AND signals. Two signals, CAN high (CANH) and CAN low (CANL) are either driven to a "dominant" state with CANH > CANL, or not driven and pulled by passive resistors to a "recessive" state with CANH ≤ CANL. A 0 data bit encodes a dominant state, while a 1 data bit encodes a recessive state, supporting a wired-AND convention, which gives nodes with lower ID numbers priority on the bus.

4.2. Vector driver configuration (only with real hardware)

If ever you want to set up the experiment with real hardware, you will have to open the driver configuration of the Vector hardware by clicking on hardware -> bus hardware -> drivers… *for checking the channel attachment we saw during CAN LIN gateway experiment Part 1.*

4.3. Representing the CAN signal

- Copy the data used in the previous part on your desktop.

- *If ever you wish to simulate with real hardware, you will have to do the following steps:*

    1. *Connect the vector hardware VN8950 to the USB port on your workplace.*
    2. *Connect the instrument cluster and the LIN keypad with the operating voltage needed (see previous lab sheet).*
    3. *Connect the instrument cluster and the VN8950 (Channel 1) with a serial cable.*
    4. *Connect the LIN keypad with the VN8950 (Channel 3).*

- Start the program CANoe with the configuration file you worked on in the last part.

- Open the simulation setup window by clicking on: *simulation* -> *simulation setup* and add a CAN node named *Kombiinstrument* to the CAN network

- What would be another form of sending CAN messages within the CAPL environment, different from the one you used so far? Write down a small example.

In the following list, you can find two messages with the corresponding signals that are necessary for the activation of the hazard lights function. Thereby, we will simulate the state of the ignition lock via the signal *Klemmeninformation* in the message *Klemmenstatus*. The message *TIM* contains all the signals necessary for the activation of the hazard lights. Unlike the keypad of the Mercedes CL, the instrument cluster needs no additional timer for the generation of the flashing frequency. You rather have to write the desired duty cycle t_{ein} into the signal *TurnLampONDur* of the message TIM. Therefore, please consider the corresponding transfer formula, as well as the number system (see Wertedarstellung). The two other signals (*TurnInd_LT_ON*, *TurnInd_RT_ON*) are used for controlling the indicator lights (on the left and on the right).

Software-Klemmen

Botschaftsname: Klemmenstatus	
Identifier:	0x1
Byteanzahl:	8
Zyklus:	100 ms
Signalname: Klemmeninformation	
Byte:	0
Bit:	0 bis 3
Wertetabelle:	
IGN_OFF	0x1
IGN_ON	0x4

Warnblinker

Botschaftsname: TIM	
Identifier:	0x29
Byteanzahl:	3
Zyklus:	680 ms

Signalname: Turnlnd_LT_ON	
Byte:	0
Bit:	6
Wertetabelle:	
OFF	0x0
ON	0x1

Signalname: Turnlnd_RT_ON	
Byte:	0
Bit:	7
Wertetabelle:	
OFF	0x0
ON	0x1

Signalname: TurnLampONDur	
Byte:	1
Bit:	0-7
Wertedarstellung:	

$t_{ein}[hex] = t_{ein}[dez]/10$

Attention: all the messages are sent in Intel format. Please consider that the transmitting behaviour has to be programmed in CAPL code.

- Open the database *Kombiinstrument* you downloaded from the online platform and add the messages and signals listed above.

- Moreover, add a network node named *Kombiinstrument* to the database.

- Integrate the database into the CANoe simulation setup and assign the node to a database as well.

4.4. Connecting the CAN/LIN network

In order to use the state of the hazard light (on the LIN keypad) in both networks, it is necessary to save the value into a global variable (system variable). System variables can be administrated in CANoe via *environment* -> *system variables*. The system variable Status_Warnblinker used here has already been created in your configuration.

- Extend your CAPL code already created for the LIN keypad by adding the system variable Status_Warnblinker. When the hazard light is activated, the variable should have the value 1, otherwise, the value 0.

- In the next step, connect the CAPL file Kombiinstrument.can with the network node and open the source code afterwards.

- Create the following function that you will visualize in a flowchart at first.

Please consider the following constraints:

- The state of the ignition will be activated durably from the start of the measurement on. Use an appropriate event handler for realizing this.

- The request for the lights' flashing (*message "TIM"*) should take place every 680 ms, the duration (*TurnLampONdur*) should be 340 ms.

- Please take into consideration that the flashing of the two control units should occur simultaneously.