
	<p>Professorship Computer Engineering</p> <p><b>Automotive Software Engineering Practical</b></p> <p>Prof. Dr. Wolfram Hardt, Dipl.-Inf. Norbert Englisch</p>	 <p>TECHNISCHE UNIVERSITÄT CHEMNITZ</p>
<p>WS 2014/2015</p>	<p><b>Experiment 3</b></p> <p>CAN Networks</p>	<p>26.11.2014</p>

## Content

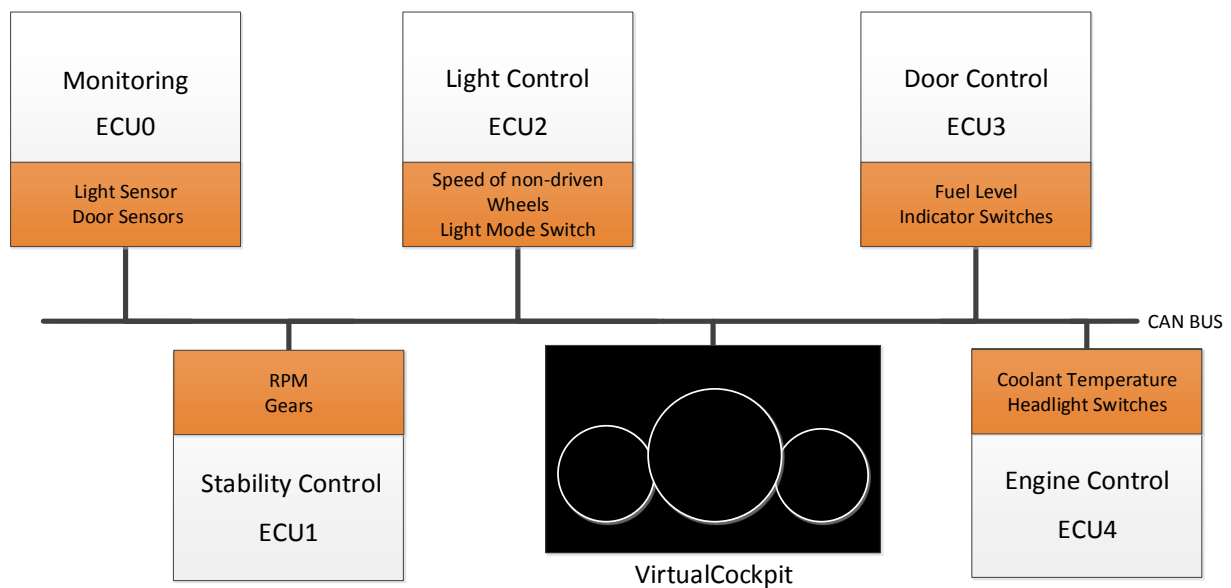
1. Overview.....	2
2. Tasks .....	3
2.1. ECU0 – Monitoring .....	3
2.2. ECU1 – Stability Control .....	3
2.3. ECU2 – Light Control.....	4
2.4. ECU3 – Door Control .....	4
2.5. ECU4 – Engine Control.....	4
3. Hints and Notes .....	5

## 1. Overview

The purpose of unit 3 is to learn how ECUs and applications interact. The goal is to control a virtual dashboard with a set of ECUs. Therefore it is necessary that the ECUs exchange information over the CAN bus. Each board realizes sensor functionality and additional logical functionality which controls the dashboard. The main task of this unit is to implement these ECUs.

A message catalog can be found as .xls file in OPAL. It describes all messages sent by the ECUs. Moreover it gives an overview of the message timings. Each message is periodically sent. Furthermore information about the data sent is included. Please refer to the message catalog if there are questions regarding the messages. Attention: All timings must be fulfilled! Moreover the message format must comply with the message format specified in the message catalog! Otherwise problems occur.

Figure 1 shows an overview of the system. The orange boxes represent the sensors which should be used on the boards. Every board has its own task and is connected to bus. Like in the last units a CAN bus is used to communicate. The virtual dashboard application which is called VirtualCockpit is added as an additional bus participant. This application will visualize the bus messages as in a real car dashboard and will be used to evaluate the functionality of your solution. VirtualCockpit is provided on each development PC and can be found in the path "C:\Program Files (x86)\VirtualCockpit". Figure 2 shows the program after start-up.



**Figure 1** System overview with bus and bus participants



Figure 2 The VirtualCockpit application

## 2. Tasks

Five groups are needed to implement the whole functionality. Every group in the team must implement one of the following ECUs.

### 2.1. ECU0 – Monitoring

- Sensor functionality:
  - Light sensor: The light value received by the photo resistor connected to this board should be sent to the bus.
  - Door sensors: Two switches should represent sensors included in the doors. One switch for the left door and another switch for the right door. If the doors are opened, a logical '0' should be written to the bus. A logical '1' should signalize that the door is closed.
- ECU functionality:
  - This ECU receives the fuel level and the engine coolant temperature. If the fuel level reaches  $1/10^{\text{th}}$  of the maximum a signal should be written to the bus. If the coolant temperature reaches  $9/10^{\text{th}}$  of the maximum another warning signal written to the bus.
  - Moreover it must check if ECU1 – Stability Control is present. If ECU1 is not present LED7 should periodically blink. Additionally an error message must be sent.

### 2.2. ECU1 – Stability Control

- Sensor functionality:
  - RPM: The potentiometer of this ECU should be used to read the RPM value. This value must be sent to the bus. Note: The value received by the potentiometer is  $1/10^{\text{th}}$  of the actual RPM. So a value of 410 would be 4100 RPM.
  - Gear sensor: Two buttons on this ECU should be used as gear sensors. One button should indicate that the user shifted up. The other button is for down shifting. In every case a signal including the gear should be sent to the bus. Note: Let's assume a 6-gear sequential manual transmission is used. At start up the gearbox is always in neutral (N). The 6-gear transmission should have the following gears: R – reverse, N – neutral, 1, 2, 3, 4, 5, 6. One byte should be used to send the data. Every bit in this byte represents one gear. So the reverse gear should be represented by '0b00000001'. The neutral gear is represented by '0b00000010' and so on.
- ECU functionality:
  - The Stability Control receives the speed of the drive wheels and the speed of the non-driven wheels. If the speed difference is bigger than 15 km/h a signal should be sent to the bus.

- Moreover it must check if ECU2 – Light Control is present. If ECU2 is not present LED7 should periodically blink. Additionally an error message must be sent.

### 2.3. ECU2 – Light Control

- Sensor functionality:
  - Speed of non-driven wheels: The potentiometer should be used to determine the speed of the non-driven wheels. This value must be sent to the bus. Additionally the potentiometer value must be mapped to a value between 0 and 300.
  - Light mode switch: A switch on this ECU should be used as mode switch. The mode switch toggles between the manual and the automatic light mode on this ECU.
- ECU functionality:
  - This ECU implements two modes. The first mode is the manual mode. In manual mode the ECU receives 6 signals which represent different light states. Each state switches a light on or off. A logical '0' received means the light is switched on. A logical '1' should switch off a light.
  - In automatic mode the low beam and the parking light should be switched on automatically if the light value falls below 512. The message with ID 0x101 holds the light value.
  - Moreover it must check if ECU3 – Door Control is present. If ECU3 is not present LED7 should periodically blink. Additionally an error message must be sent.

### 2.4. ECU3 – Door Control

- Sensor functionality:
  - Fuel level: The potentiometer is used to represent the fuel level of the car. It should be sent to the bus.
  - Indicator switches: 3 switches should be used to control the state of the indicators. There should be a switch for the left and right indicator, and the last switch should be used for the hazard light (left and right indicator at the same time). Note: To signalize that an indicator is switched on a logical '0' should be sent. If it is switched off a logical '1' must be sent.
- ECU functionality:
  - This ECU receives the speed of the car and the state of the doors. The state of the doors must be sent to the bus. If the car moves faster than 5 km/h and a door is opened a warning signal should be sent additionally.
  - Moreover it must check if ECU4 – Engine Control is present. If ECU4 is not present LED7 should periodically blink. Additionally an error message must be sent.

### 2.5. ECU4 – Engine Control

- Sensor functionality:
  - Engine coolant temperature: The potentiometer should be used to determine the engine coolant temperature and must be sent to the bus.
  - Headlight switches: 3 switches should be used as headlight switches. One switch is used as high beam switch, another one as low beam switch, and the third one as switch for the parking light. A logical '0' sent means the corresponding light is switched on and a logical '1' should be sent if the light is switched off.
- ECU functionality:
  - This ECU should calculate the speed depending on gear and RPM. The speed value must be sent to the bus. The speed can be calculated by the following formula:

$$speed = \frac{gear\ ratio * rpm}{100000}$$

The gear ratios can be found in the following table:

Gear	R	N	1	2	3	4	5	6
Value	7842	0	7842	13112	19861	27038	33149	40035

The speed should have a resolution of 10 bits and must be sent to the bus.

- Moreover it must check if ECU0 – Monitoring is present. If ECU0 is not present LED7 should periodically blink. Additionally an error message must be sent.

### 3. Hints and Notes

- A new template is provided. Please get familiar with it. Registers can be accessed like in the old template. You can use the macro '*LEDX*' to access the LEDs 0-7. The macro '*POT*' returns the potentiometer value. '*IN1*' returns the value of the device connected to the IN1 Pin (light sensor). Open the "spc560p\_io.h" file to see all predefined macros.
- It is not necessary to configure any pin. Everything is preconfigured.
- Analog values are 10 bits wide. If nothing special is mentioned in the tasks the 8 least significant bits should be in the 0<sup>th</sup> data field of the CAN message. The remaining 2 most significant bits should be in the 1<sup>st</sup> data field.
- Refer to the message catalog if something is not clear regarding the messages. It could help to solve your problem.
- Try to use different buffers for different messages.
- Try to use the individual mask registers for every buffer. So the interrupt controller needn't handle all messages received. This can solve some specific problems.
- Try to set only flags in the interrupt routines and handle the logic within the for-loop.
- Use LEDs to debug your code
- Because of technical issues it is not possible to use switch 2.
- Message buffers should be written to the '*CAN\_MessageBufferInit()*' function.
- Try to test your implementation using the VirtualCockpit application as soon as possible
- You can also test your ECUs using the TinyCAN Viewer application as before
- You can only use either VirtualCockpit or TinyCAN at a time (due to the exclusivity of bus access) so close the one you are not using
- There is some new application interface you should use. The new functions are explained below.

*void ADC\_StartConversion(void)*

- This function starts an ADC conversion. It must be called before analog values are needed. Otherwise the values are not updated.

*void PIT\_ConfigureTimer(char timerChannel, unsigned int loadValue)*

- This function configures the timer '*timerChannel*' and initializes it with '*loadValue*'. '*loadValue*' is the time the interrupt is called in milliseconds. Timer channels 0 and 1 are preconfigured and can be used. The corresponding interrupt functions can be found in the interrupt section at the end of the template.

*void PIT\_StartTimer(char timerChannel)*

- This function starts the timer '*timerChannel*'.

*void PIT\_StopTimer(char timerChannel)*

- This function stops the timer '*timerChannel*'.