
	<p style="text-align: center;">Professorship Computer Engineering</p> <p style="text-align: center;"><b>Automotive Software Engineering Practical</b></p> <p style="text-align: center;">Prof. Dr. Wolfram Hardt, Dipl.-Inf. Norbert Englisch</p>	 <p style="text-align: center;">TECHNISCHE UNIVERSITÄT CHEMNITZ</p>
<p>WiSe 2014/15</p>	<p style="text-align: center;"><b>Experiment 1</b></p> <p style="text-align: center;">Basics of automotive software engineering and an introduction to the demo boards</p>	<p style="text-align: center;">20.10.2014</p>

## Content

1. Basics .....	1
1.1. ECUs in modern automobiles .....	3
1.2. Sensors – ECU – Actuators .....	4
1.3. Evaluation Board .....	5
1.4. Practical Examples .....	7
2. Experimental Procedure .....	8
2.1. Experimental setup and –procedure .....	8
2.2. Task 1 .....	8
2.3. Task 2 .....	10
2.4. Task 3 .....	10
2.5. Additional Task 4 .....	11
3. Appendix .....	12
List of References .....	14

## 1. Basics

For more than a hundred years, motor vehicles are produced and improved. The use of micro-electronics provided new possibilities for engineering and implementation of complex functionalities for automobiles. Today most car drivers use their automobile for more than one hour per day, this results in increasing demands of the driver to the vehicle. To meet all the demands the usage of hardware and software in automobiles increases rapidly. The fast and reliable reaction of automotive functions is assured by software, electronic control units and their communication among each other. Furthermore former mechanical functions were replaced by software, electronic control units, sensors and actuators. Today luxury class automobiles contain up to 80 connected electronic control units i.e. engine management, airbag control, power locks and electric windows. In the case of the electric windows the software automatically reverses/stops the window motor if it senses an obstruction while closing.

Nowadays 90% of the innovations in car development concern the electronics and the software. That includes the conversion of classic mechanic functions as well as new features for example the automatic parking system.

For the practical automotive specific terms are necessary which are explained in the following table:

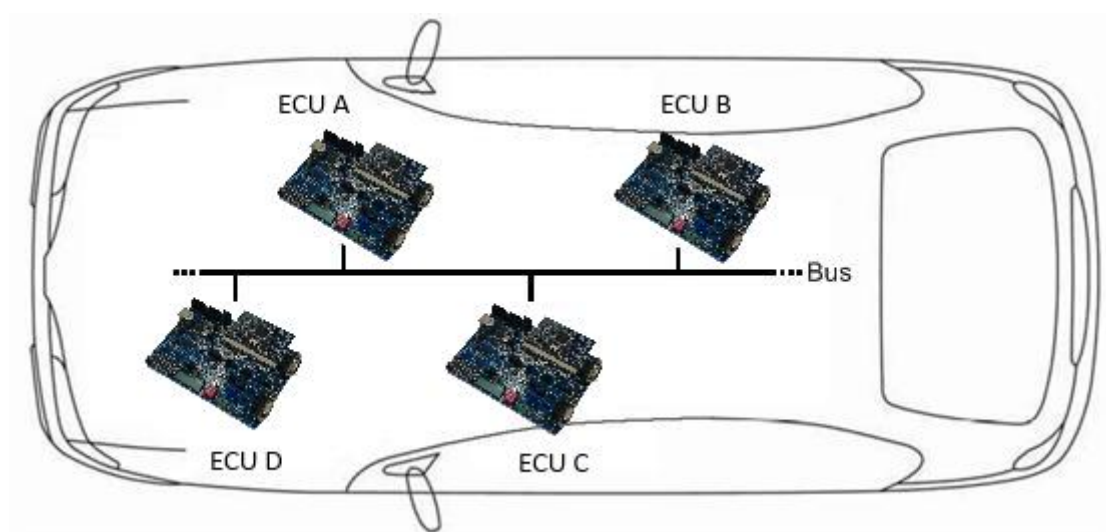
<b>Term</b>	<b>Definition</b>
<b>Embedded system</b>	Computer with special purpose. Serves to control and monitor a system.
<b>Distributed systems</b>	Computer network which communicates by message exchange.
<b>Interactive systems</b>	Hardware and software components which process user input and return results.
<b>Reactive systems</b>	Hardware- and software components which interact permanently/cyclic with the environment. Reactive systems normally do not terminate.
<b>Real-time system</b>	Computer system, which guarantees the calculating of a job within a predefined deadline (time limit). It is differentiated between hard, firm and soft real time.
<b>Communication</b>	The communication in the automotive domain describes the message exchange of distributed and embedded systems with the help of protocols.
<b>Reliability</b>	Ability of a system to consistently perform its required function within a specified time limit without degradation or failure.
<b>Availability</b>	Proportion of time a system is in a functioning condition.
<b>Security</b>	Extent to which a system is protected against data corruption, destruction, interception, loss or unauthorized access.
<b>Monitoring</b>	Observing of the current system status to detect failures and to initiate counteractions.
<b>Electronic control unit</b>	An electronic control unit (ECU) is an embedded system which controls one or more of the electrical subsystems in a vehicle. Sensors, actuators and ECUs are often one unit in mechatronic systems.
<b>Sensors</b>	Electronic device used to measure a physical quantity such as temperature, pressure or loudness and convert it into an electronic signal
<b>Actuators</b>	Transformation of electronic signals from an ECU into mechanical respectively physical quantities.

## 1.1. ECUs in modern automobiles

Increasing customer demands, such as lower fuel consumption, improving of driving safety and comfort are closely related to the use of more and more electronics and software in modern automobiles. The requirements of the ECUs in today's automobiles are the following:

- Reliable under rough and changing ambient conditions like temperature, humidity and vibration
- High demands to the dependability and availability
- High demands to the security

ECUs work as a reactive system. The driver has no influence on the functionality except from activating sensors (e.g. switches).



This easy, general example demonstrates the interconnection of the controllers and their communication over a bus. It is typical for a bus that the data transfer is realized via messages. Therefore the developer has the ability to send information resp. messages (for example the outdoor temperature) from one ECU to 1 or several others over the bus. The installed ECUs are divided into the following subsystems:

- Engine
- Chassis
- Car body
- Multimedia
- Security

To stress the importance of the interconnection and the categorization of ECUs, the following example shall be used. The ACC (Adaptive Cruise Control) is an add-on for the cruise control. The system maintains a steady speed as set by the driver and automatically keeps the necessary security-distance. When approaching a slower vehicle the speed is adjusted to the speed of the preceding vehicle and thus the security distance is kept. If the distance increases the systems accelerates up to the defined speed. To realize this function several ECUs are necessary in the automobile. The ACC-

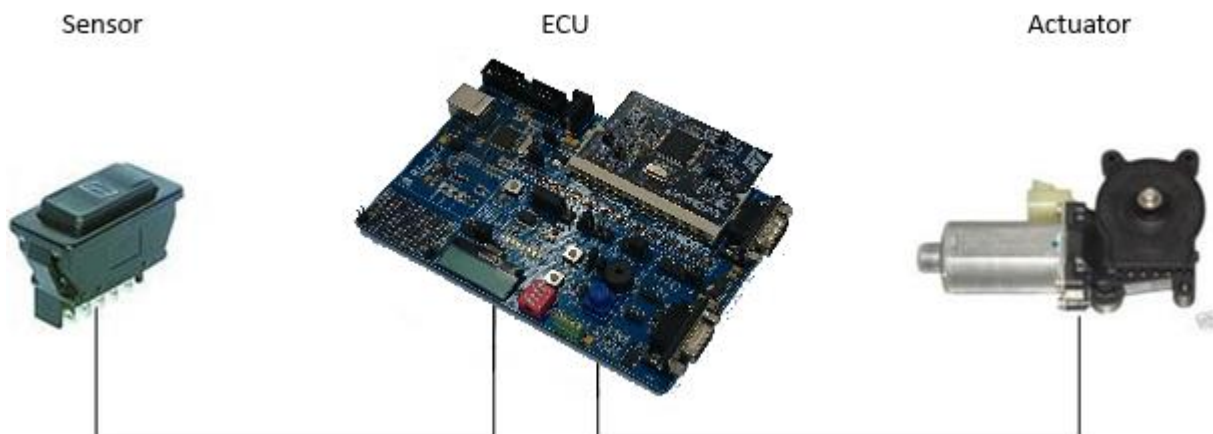
ECU communicates with the engine-, the ESP- and the gearbox ECUs to adjust the driving speed. ESP is necessary to selectively decelerate individual wheels to avoid the vehicle breaks away.

This simple example shows the importance of the communication between the affected ECUs. Only by interaction of the different ECUs the functionality of the ACC can be displayed. It also shows that simple features need complex implementations.

## 1.2. Sensors – ECU – Actuators

Sensors deliver the data which is then analyzed by the ECU. They measure physical quantities and convert them into electric quantities, so they can be processed. There are various types of sensors that react to different physical quantities, such as air pressure, humidity, light intensity, distance, temperature or density. The ECU periodically reads the sensor data and processes the implemented software. At this point it is decided whether an actuator is activated or not. If the sensor data differs between the periods in most cases the actuators are accessed by the ECU to react on the environmental changes. Furthermore the actuators can be accessed if sensor data have arrived at certain threshold. A simple example is the headlight which is automatically switched on at nightfall. A photodiode measures the light intensity and forwards it to the ECU. If the intensity of the light falls below a threshold the headlights are turned on.

The following graphic shows the relation between sensor, ECU and actuator.



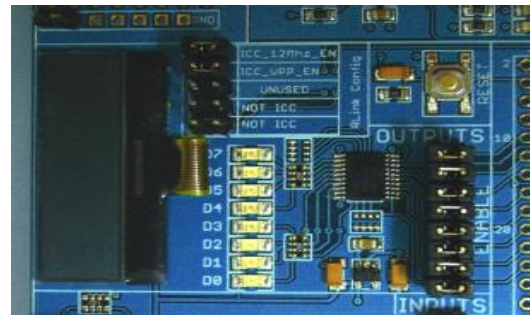
The graphic shows the function of an electric window. A simple switch serves as the sensor. Once the switch is pressed the ECU activates the actuator, in this case the electric window motor. The ECU is responsible for automatic reversing the electric windows if it senses an obstruction while closing.

### 1.3. Evaluation Board

The evaluation board from Raisonance serves as development and testing platform for the SPC560P controller from STMicroelectronics. It can be used as a standalone application or in a network. Furthermore it is possible to flash developed software onto the board and to run it. If software is transferred to the board, it can also be operated with a power supply and without a computer. The Board provides different interfaces to access external devices, like sensors and actuators. The following graphic shows the provided sensors, actuators und further interfaces on the board:

#### *Digital Outputs Area*

- Eight red transistor-driven LEDs (D0 – D7) that light up when the command is low. Current is supplied by the power supply (9V or USB).
- Eight jumpers enabling the independent use of each LED. The use of an LED is enabled when the corresponding jumper is plugged in.
- LCD



#### *Digital Inputs Area*

- Two-position jumper to choose the polarity of the BT6 push button.
- Two push buttons and four switches. When the switches are on or the buttons pressed, the corresponding signal is tied to the ground (except for BT6 which depends on the chosen polarity).



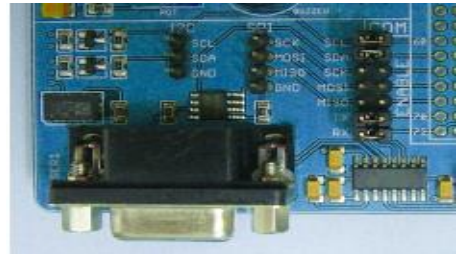
#### *Analog Area*

- Four-position analog connector which includes:
- Two analog inputs that can be connected directly to the ADC input pins of the microcontroller
- Analog output resulting from the integration of a PWM output. The integration is done with a simple RC filter with a characteristic time of 100ms
- Ground connection
- Potentiometer, that can be connected directly to the ADC input pins of the microcontroller
- Temperature sensor, that can be connected directly to the ADC input pins of the microcontroller. The output voltage of this sensor is given by the formula  $V=0.01(T+1)$ .
- Buzzer that can be connected to the PWM microcontroller output through the PWM/BUZ jumper



### *Communication Area*

- Three-position connector for I<sup>2</sup>C communication
- RS-232 connector,
- Four-position connector for SPI communication



### *Secondary Serial / Can Area*

- This area allows the use of other serial communication protocols (CAN for example).



There are several sensors and actuators already on the board, no more components are needed for simple experiments. Please study the tasks under point 2 and the source code files in the appendix before the experiment.

## 1.4. Practical Examples

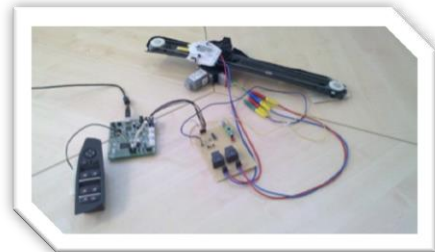
### *Development of functionality for the Yellow Car*

The Yellow Car provides many sensors and actuators which can be used for the implementation of different functions. Amongst others the flashing indicator control and engine management are implemented.



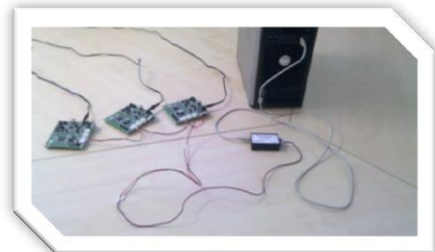
### *Interaction of sensors and actuators via an ECU*

The window regulator is a practical example which shows the interaction of a sensor, an ECU and an actuator. The switch panel serves as the sensor and the window motor as the actuator. The entrapment protection and the up and down moving functions are a software implementation on the ECU.



### *Communication between ECUs*

To enable the communication between ECUs, messages are defined and transmitted over the bus. One ECU serves as the user interface (e.g. the steering wheel) which sends messages to the indicator ECU when an indicator button is pressed. There the message is then analyzed and afterwards the respective indicator light is turned on/off.



### *Armature of a BMW 7Series*

The armature serves to demonstrate a cockpit. The infotainment-system works with the help of an evaluation-board





## 2. Experimental Procedure

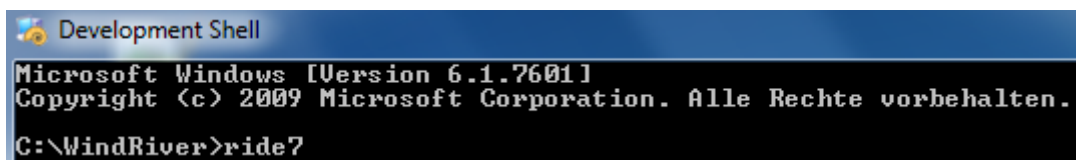
### 2.1. Experimental setup and -procedure

To implement functions on the ECU, the **“Raisonance Ride7”** IDE will be used. To flash code to the board **“Rflasher7”** is used.

The compiler must be started by opening **“WindRiver -> Development Shell”** in the start menu or on the desktop.



Then it is necessary to enter the command **“ride7”** in the development shell to start the IDE.



It is mandatory to do this first. Otherwise the code will not be compiled. The flashing program can be found under **“Raisonance tools -> Ride7 -> RFlasher7”** in the start menu. The project **“Unit1\Unit1”** (Source code in the appendix) has to be opened in the ride7 IDE. Analyze and study the source code on the file **“main.c”** and the user interface of Ride7 and Rflasher7. To flash the program, connect the board with the computer, load the output file **“Unit1\output\unit.s19”** and click the **“Go”** button. Please note that the output file has to be created by the compiler, hence pressing the **“Build”** button from the IDE will create (if there are no errors in the code) an output file in the specified location.

### 2.2. Task 1

The I/O pins of the main controller situated on the daughter board are multiplexed (they can be used for multiple functions). The aim of the first subtask is to configure these pins as general purpose input and output for the LEDs. Therefore it is necessary to refer to the additional document (**“additional.pdf”**) which is provided. In this document the corresponding **“PCR”** register associated with each LED can be found. The first page of the additional document shows the mapping of peripherals onto different pins of the controller. The correct pins/ports for the LEDs (D0-D7) with the corresponding **“PCR”** register number can be found in the additional document.

The table below (table 2.2.1) shows IO port pin mapping. As an example port pin PA9 corresponds to LED D0, and the PCR register number for this LED from the additional document is 9. To use an LED the corresponding pin has to be configured as output. Furthermore, since the pins are multiplexed, the correct function of the pin has to be chosen. For the LEDs this can be done by writing **“0x0200”** to the corresponding **“PCR”** register.

The function **“configure\_io()”** shall be used for this task.



In order to light an LED, a low signal should be applied on the I/O pin. This is possible by writing “0” in the corresponding register. For turning off a high signal by writing “1” to the corresponding register.

Example for lighting an LED: “*SIU.GPDO[X].R = 0;*” where X stands for the corresponding PCR register.

Feature	CPU pin name	CPU pin number	Uses	Comments
D7	PA11	120	1	Light when command is low
D6	PD11	78	1	Light when command is low
D5	PD13	95	1	Light when command is low
D4	PD14	105	1	Light when command is low
D3	PA12	128	1	Light when command is low
D2	PA13	136	1	Light when command is low
D1	PC10	3	1	Light when command is low
D0	PA9	125	1	Light when command is low
BT6	PA0	73	1	No pull-up/down, off=float, on=low/high jumper
BT5	PA1	74	1	No pull-up/down, off=float, on=low
SW4	PA2	84	1	No pull-up/down, off=float, on=low
SW3	PA3	92	1	No pull-up/down, off=float, on=low
SW2	PA4	108	1	No pull-up/down, off=float, on=low
SW1	PC12	82	1	No pull-up/down, off=float, on=low
ANA IN1	PC1	41	1	External analog input
ANA IN2	PC2	45	1	External analog input
ANA OUT	PC9	123	1	PWM connectable to the buzzer or to ANA_OUT
POT	PE1	39	1	Potentiometer analog input, min=0V, max=VDD
TEMP	PE2	49	1	Temperature sensor analog input
I2C SCL	PE5	44	1	I <sup>2</sup> C clock, pull-up by jumper
I2C SDA	PE6	46	1	I <sup>2</sup> C data, pull-up by jumper
SPI SCK	PC5	13	1	SPI clock, no pull-up/down
SPI MOSI	PC6	142	1	SPI master out slave in, no pull-up/down
SPI MISO	PC7	15	1	SPI master in slave out, no pull-up/down
UART TX	PB2	114	1	Serial transmit
UART RX	PB3	115	1	Serial receive
CAN TX	PB0	109	1	CAN transmit
CAN RX	PB1	110	1	CAN receive

Table 2.2.1

## 2.3. Task 2

The light sensor (photoconductive cell) is used in automobiles to automatically turn on/off the head lights. The implementation of this exercise shows the automatic activation of actuators regarding sensor data.

### **Subtask 1:**

The value of the sensor, which is converted by the ADC module of the controller, is stored in the register ***“ADC\_0.CDR[2].B.CDATA”***.

This subtask concerns with displaying the value of the light sensor data through LEDs.

The controller pin, which is mapped to the ADC module, has to be configured for analog input which can be done by writing ***“0x2500”*** to the corresponding PCR register. Please refer to table 2.2.1 for the corresponding port/pin. The light sensor is connected to the ***“ANA IN1”*** input pin. The code for the analog pin configuration should be entered in the ***“configure\_io()”*** function.

The function ***“showdata( value )”*** contains the code for the different value levels. The task is to glow LEDs corresponding to each level, i.e. for the highest value of the sensor all LEDs should be switched on. Please check and modify the function ***“showdata( value )”*** in the provided code.

### **Subtask 2:**

This subtask concerns with the usage of the potentiometer provided on the board. The converted value from the potentiometer is being stored in the register ***“ADC\_0.CDR[4].B.CDATA”***. Use this data register, instead of light sensor data register, with the same function (***“showdata( value )”***).

Please note: To read converted values from the ADC data register, the corresponding controller pin has to be configured in the same manner as the light sensor. The corresponding PCR register is not the same in this case!

### **Subtask 3:**

The digitally converted values of the ADC unit lie between 0 – 1024 (10bits).

Use the data from the light sensor and invert the behavior of the function. Change the function ***“showdata( value )”*** so that 5 levels of light intensity are indicated, utilizing LED D4. For minimum light incidence LEDs D0 to D4 should glow.

## 2.4. Task 3

The PIT (Periodic Interrupt Timer) module on the board provides 4 different timer channels. In the following tasks digital inputs and timer-functionality will be used.

### **Subtask 1:**

Similar to the LEDs, the buttons and switches ***“BT5”***, ***“BT6”***, ***“SW1”*** etc., are also mapped to each pin of the controller. The correct ***“PCR”*** register can be found by following the same procedure as LED configuration. A button can be configured as input by writing ***“0x0100”*** to the corresponding PCR register.

A button press or a switch on action results in a low signal (“0”) on the pin. This signal can be checked by the input register of the corresponding pin (“*SIU.GPDI[X].R*”, where X stands for the corresponding PCR register number).

The button/switch action should be indicated by switching on an LED.

#### **Subtask 2:**

The PIT timer channels 0 and 1 have already been configured. The timers can be started by calling the function “*conf\_timer0( int mode, int value )*” respectively “*conf\_timer1()*”. The description of the parameters can be found in the table below.

Parameter	Description
Mode	0 – Turn off the timer channel 1 – Turn on the timer channel
Value	Timer period in milliseconds. (1 sec = 1000ms)

The timer should be configured with a valid mode and a valid time. The configured period causes an interrupt at the end of each time-period. When the interrupt occurs, the function “*timer0\_intr*” is being called. To indicate a working timer, LED D0 should blink at every timer interrupt. “*SIU.GPDO[X].R = ~SIU.GPDO[X].R*” (where X stands for the PCR register number) can be used to let the LED blink. Consider the “~”-Operator, which negates the following value, in this case the current value of the register for the corresponding LED.

This implementation is an endless loop. The timer channel should be switched on/off using SW1, if the switch state is changed. Implement this functionality.

This task shows a part of the indicator control of an automobile. In a later practical the indicator control will be expanded of the possibilities of left, right and hazard flashing.

## **2.5. Additional Task 4**

Read the document PIT.pdf and write the code for the configuration of timer-channel 1 in the function “*conf\_timer1()*”. Implement the same functionality as task 3 - subtask 2 is using timer channel 1. The periodic interrupt, by this timer, causes the function call “*timer1\_intr()*”.

Please note: The system clock frequency is 16MHz. Think about how you have to use this information to get a timer which can be configured in millisecond-steps.

### 3. Appendix

```
#include "api.h"

void showdata(int value);
#pragma inline showdata

void configure_io(void)
{
    //Task1
    //Please enter the PCR register numbers and initialitation code here
    // LEDs

    // SIU.PCR[].R = ;      // D0
    // SIU.PCR[].R = ;      // D1
    // SIU.PCR[].R = ;      // D2
    // SIU.PCR[].R = ;      // D3
    // SIU.PCR[].R = ;      // D4
    // SIU.PCR[].R = ;      // D5
    // SIU.PCR[].R = ;      // D6
    // SIU.PCR[].R = ;      // D7

//task 3
// Buttons & switches
    //SIU.PCR[].R = ; //BT5
    //SIU.PCR[].R = ; //SW1

//task 2
// Analog inputs configuration
    //SIU.PCR[].R = ;
    //SIU.PCR[].R = ;
}

void conf_timer1(void)
{
    // Task 4
    // Timer channel Channel 1

    //PIT.CH[1].LDVAL.B.TSV = ; //Time Start Value Bits
    //PIT.CH[1].TCTRL.B.TIE = ; // Timer interrupt Enable
    //PIT.CH[1].TCTRL.B.TEN = ; // Timer enable
    //PIT.CH[1].TFLG.B.TIF = ; // Clear Timer Flag
}

void timer0_intr(void)
{
    // task 3 subtask 2 - LED blink code goes here

}

void timer1_intr(void)
{
    // task 4 - timer 1 interrupt function
}
}
```

```

void main(void)
{
    int value;

    init(); // board initialization

    for(;;) // infinite loop
    {
        //task 1 - please enter your code for checking LEDs here

        // task 3/ task 4 - code for checking button press and timer enable/disable goes here

        //task2
        //value = ;
        //showdata(value);

    } // End for loop
} // End main

```

```

void showdata(int value) // task2
{
    if(value <= 256)
    {
        // task 2
        // LED status to be shown
        // D0 - on, D1 - off, D2 - off D3 - off
    }
    else if(value > 256 && value <= 512)
    {
        // task 2
        // LED status to be shown
        // D0 - on, D1 - on, D2 - off D3 - off
    }
    else if(value > 512 && value <= 768)
    {
        // task 2
        // LED status to be shown
        // D0 - on, D1 - on, D2 - on D3 - off
    }
    else
    {
        // task 2
        // LED status to be shown
        // D0 - on, D1 - on, D2 - on, D3 - on
    }
}
}

```

## List of References

- (1) JÖRG SCHÄUFFELE, THOMAS ZURAWKA: Automotive Software Engineering, 2010
- (2) ST Electronics, SPC560P Reference Manual
- (3) Raisonance REva motherboard V3 User guide