Exercises and assignments for the course DIT632 Development of Embedded systems _ Part 2 of 3

Exercise points: For each of the handed in and approved exercises you get a number of exercise points from zero up to a maximum specified point. You can get a total sum of 54 exercise points for all handed in exercises. Conditions to pass the sub course "Assignments" and to get bonus points to final written exam is described in part 1 of Exercises and assignments.

Terms of assignments and bonus

All exercises that are submitted (possible for all in WP2 - WP6) must meet the following requirements to give bonus points at final exam.

Includes a program header as below:

/ * ==========

File name: exerc_x_y.c (or cpp)

Date: 2019-mm-dd

Group Number:

Members That contributed:

xxxxxxx xxxxxxxx

yyyyyy YYYYYYYYYY

zzzzz ZZZZZZZ

Members not present at the demonstration :

Xxxxxxx

Demonstration code: [<Ass code xxxxx>] **Important , No code no exercise points !**

========== * /

All submissions must be handed in by one of the group members via course homepage. Before handed in all programs should be demonstrated for a teaching assistant (TA) and if the TA approves the program you will get a specific unique demonstration code <xxxxx> to type in to the program information header.

You should only hand in one file per WP nr. All programs should be included in one xxx.zip file. If there are any solutions containing more than two files you had to put them in a separate directory.

Work package nr 4/ Low level C-programming (Max 10p)

Introduction information for this course week

For this week tasks we will, for some exercises, use an IDE intended for Motorola HC12 microcontroller-based system XCC12. It gives you an opportunity to develop programs in C and/or in Motorola assembly language. We will use it to study low level C-programming and later on real-time system. We use this environment because it includes a simulator which gives a good opportunity to study what happens when we control individual bits or bytes in I/O-ports and later study some aspects on real-time systems.

The XCC IDE (XCC12) is developed for Windows environment but it can be executed in Wine environment on a Mac or a Linux computer. All solution has been tested with XCC12 running on a Mac laptop in the Wine environment. It works best in a Windows environment, but it still works well in Wine environment. The difference is that the simulation is a bit slower. If you don´t have the possibility to install XCC12 or Wine/XCC12 you can do alternative exercises using a web-based environment Tinkercad in which it is possible to design systems including an Arduino computer board and then simulate the system.

For those who can install the XCC program it is very easy to download and install. You can download an installer file from the course homepage in Files/ XC12 documents…. There you also can find a short quick guide for start using XCC12, ***Intro XCC12 Quick Guide_17.pdf*** and some demo-files.

Exerc_4_ 1 **(**Filename code.c) (1p)

Pack and unpack variables into a byte. You need to store 4 different values in a byte. The values are:

Name	**Range**	**Bits**	**Info**
engine_on	0..1	1	Is engine on or off . This is bit no 7 (MSB)
gear_pos	0..4	3	What gear position do we have
key_pos	0..2	2	What position is the key in
brake1	0..1	1	Are we breaking (told by sensor 1)
brake2	0..1	1	Are we breaking (told by sensor 2) = bit no 0 (LSB)

We should store them in a byte like this:

[engine_on]	[gear_pos]	[key_pos]	[brake1]	[brake2]	
1 bit	3 bits	2 bits	1 bit	1 bit	(LSB, Least significant bit)

(8 bits in total)

Write a program ***code.c*** that takes 5 arguments (less or more should be treated as an error). The arguments should correspond to the values/variables above.

Example for a start of the program from command line:

code 1 2 2 1 1 (In the console window for Windows, ./code 1.. in Linux)

The above should be treated as:

Name	Value	
engine_on	1	Bit no 7
gear_pos	2	
key_pos	2	
brake1	1	
brake2	1	Bit no 0

The program should pack these values together in a byte (unsigned char) and print it out to the console in hexadecimal form. For this example it should be 'AB' corresponding to bits '10101011. After this your program should exit. If your program finds anything wrong (too many/few arguments, faulty input values..) your program should print out an error message and exit.

Exerc_4_ 2 (decode.c) (2p)

Write a program ***decode.c*** that takes 1 argument , in the example 'AB' (less or more argument should be treated as an error). The arguments should correspond to the byte as printed by your earlier code program. If your program finds anything wrong (too many/few arguments, faulty input values..) your program should print out an error and exit.

The program should unpack the bytes according to the specification above. Print out the result as below:

Start program in the console window : ***decode AB*** *(If* Compiled to code.exe)
Should print out.

Name	Value
engine_on	1
gear_pos	2
key_pos	2
brake1	1
brake2	1

The following exercises 4.3 – 4.5 are aimed to be developed and tested in the XCC12 IDE and its´ including simulator.

If you **not have the possibility to install XCC12 on a Windows computer or a Mac-computer in Wine environment** or not willing to spend some extra time for a possible, better understanding of IO management by this IDE you can **chose to solve the alternative exercises 4.3 AT – 4.5 AT** that you can solve independent of your own computer environment. You find information for this further on in this document.
For information regarding the use of Wine on you Mac se the Week #4 information page on the course homepage.

XCC12 is a Programming IDE for a hardware based on a Motorola MC12 CPU. The system includes a CPU board ML12 and a couple of IO unit (ML 5 , ML 13 , ..) that can be connected to the CPU board. The IDE includes an editor, cross compiler for generate executable code for the system and a very nice and easy to use simulation environment. The simulator makes it possible to debug program in combination with a simulated part of the IO unit without having access to the real hardware.
The IDE gives you very good opportunities for illustrate program developments dealing with reading/ writing to IO units. It will give you the opportunity to solve and test a couple of exercises (4.3 – 4.5) with help of this environment. There will later also be several exercises in WP 6 that can be tested in this IDE.
To be able to solve the exercise 4.3 – 4.5 read the information of XCC12 IDE including compiler and simulation possibilities in the document: ***Intro XCC12 Quick Guide_12.pdf*** . You find it at the course homepage in Documents / Materials for exercises and assignments.
You need also to install the IDE on your computer which is very easy to do. You find the installer file at the same place as the Quick Guide.
You can find two demoprograms, *progfil.c* (general C-program) and *xcc_demo_prog.c* (for demo of use of I/O unit) at the same place. You can download the file and try to compile and run them as described in the Quick guide.

Exercises for the XCC12 environment.
(If you not will use XCC : se alternative exercises further on in this document**)**

Exerc_4.3_XCC (Filename exerc_4_3_xcc.c) (2p)
Write a program with the function described below that uses the 8 bit In and Out ports (ML4) connected to XCC12. You can read about how to connect ML4 to the simulator in the Quick Guide
You can also find information about all IO devices in the IDE under Help / Help on XCC / IO simulator.

The program should work as follows:
1. Print 3 (binary 0000 0011) on the OUT port.
2. In an infinitive loop :
   Read bit no 4 on the INPORT
   - If bit 4 is set (1) rotate the content on OUT-port one step left. If bit 7 is set before the left shift bit 0 should be set after the shift.
   - If bit 4 is clear (0). Do not change the value on OUT-port
   Run some delay function made by a simple for-loop.

The frequency of this bit shift is controlled by the delay function and can be chosen arbitrary and adapted till simulator.
Write the program and test it in the XCC simulator. Connect the ML4 IO units to address 0X400 for the INPORT (dipswitch) and address 0x402 for the OUT-port LED.

Exerc_4.4_XCC (Filename exerc_4_4_xcc.c) **Keyboard reading** (2p)
Write a program with the function described below that uses the ML 2 Keyboard (Address 0x09C0) and Interface **ML15 (**menu Interface on Keyboard(default)) and the console connected in XCC.

In ML15 mode the keyboard reading is supported by some hardware on the keyboard and to read a pressed key number the program just read he value in a register with a specific address.

To read a pressed key number the program can either check if a key is pressed for the moment by checking if the DAV-bit (bit 7) is zero and then read the value in the register or wait for an interrupt signal (Interrupt turned on for the keyboard) and read the register content by the interrupt routine. An IRQ-request occur for every time a key is pressed and if IRQ mode is turned on for the keyboard. For both methods you read the value from the specific register in the device.

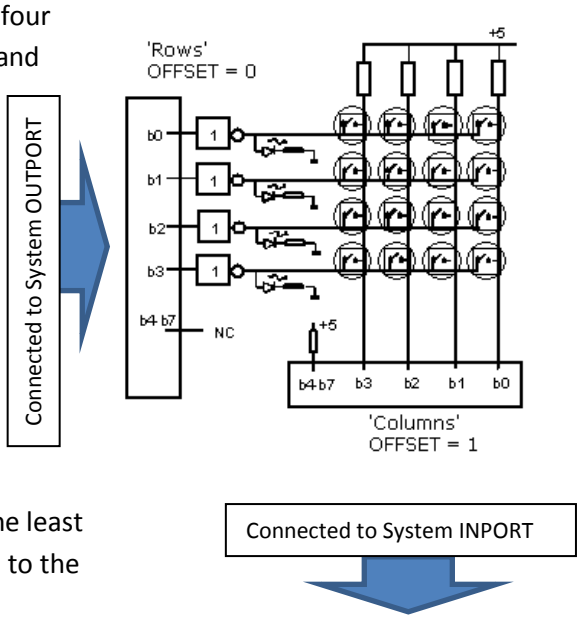
In this exercise you should develop program that periodically check the keyboard and if a key is pressed, read the value in the register and convert it to corresponding Hexadecimal integer 0 – 9, A-F and print it out in the console window. If no key is pressed when checking no new value is printed out.
Note: The program should consist of appropriate number of functions with well specified tasks.

Exerc_4_ 5 XCC (Filename exerc_4_5_xcc.c) / **Keyboard scanning** (3p)
Some cheap keyboards are not connected to a supporting hardware device and is a bit harder to read. The ML2 Keyboard can act in this simple mode if choose ML5 mode.

The principle for this mode is described below.

The keyboard is built up by four row connectors and four column connectors. At the 16 crosses between raw and column there is normally no electrical connection. At each cross there is a key and if key is pressed there will be an electrical connection between the column and raw.

All columns are on one side connected to a voltage source (5V) through a resistor and the other side of the columns are connected to the four least significant bits (b3 - b0) on a 8 bit out port connections from the keyboard.



All four rows are through an inverter connected to the least significant bits (b3 – b0) on a 8 bit in port connection to the keyboard.

The keyboard is connected to the computer system by two 8 bit I/O ports. The system INPORT (Address 0x09C1) reads the column value through bit 0 to bit 3 and the system OUTPORT (0x09C0) controls the row values by bit 0 to 3. All as the figure above shows.

If no key is pressed the value on INPORT[3..1] (columns) always will be high on all four bits. If all four bits on OUTPORT[0…3] are zero the value on all rows are high due to the inverters. If any key now is pressed it will not give any change on any BITS on INPORT (column bits). They will all still be high.

The only way to detect a pressed key is to set a row to low (0) by setting the corresponding row bit in OUTPORT to high (1). If any key in that row now is pressed it will give the corresponding column a zero value that can be detected by reading the INPORT.

To read the keyboard you must in a loop clear one rows value (set the corresponding OUPORT bit) and check if INPORT is not equal to four set bits. If still four set bits you clear next row and check the columns value. As soon as you find a column value with not four set bits you have detect a pressed key. If so you know which row and column and you can from that calculate which key number is pressed. The pattern for these two nibbles (nibble = four bits) is unique for each key. If you check all four rows without finding any INPORT value not equal to four set bits there is no key pressed for the moment.

We can assume that Key is numbered as from top row from left 0 , 1 , 2 , 3 , 4 and next row 5 , 6 , 7 , 8 etc.

Your task is to write a program with the function described below that uses the ML 2 Keyboard in **ML5** mode reading keyboard as above description. Connect the ML2 unit to the base address 0x09C0 which gives row register the address (OUTPORT) 0x09C0 and the column register 0x09C1. The program also uses the console in XCC so it must also be connected for testing the program.

The program should periodically check the keyboard. If any key is pressed search for the column and row number for the pressed key and then calculate the key nr (0-15), convert it to corresponding hexadecimal integer. 0 – 9, A-F. Finally, the program shall print out the number in the console window. If no key is pushed when checking no printing is done.

Alternative exercises 4.3 AT – 4.5 AT for those not able to use the XCC12 environment.
The following exercises number 4.3 AT – 4.5 AT is for those who choose not to installed XCC12 which is necessary for the previous exercises 4.3 XCC – 4.5 XCC. The alternative exercises will use a web-based environment in which it is possible to set up a system and corresponding software for a Arduino supported electronic system.

Introduction to Arduino, ATmega microcontroller and TinkerCad

Arduino is a small computer board designed to make it easy and quickly to develop small embedded systems. The various Arduino boards are designed with different ATmega microcontrollers. Arduino Uno, which we also will use later in this course, is equipped with an ATmega 328 microcontroller.

The Arduino system also includes its own development environment (IDE) which, with the help of extensive associated library functions, facilitates software development and makes it easy for a beginner to develop the software for a system in the program language C supplemented by all Arduino's own library functions and without knowing much about lots of low-level details.

In our course we will later build a simple hardware system, a thermometer, to give a little insight into working with real hardware. (See WP # 5).

Tinkercad

In two of the exercises in WP # 4 we will, as an alternative to some other possible exercises needed XCC12, use a development environment, **Tinkercad**, which makes it possible to virtually build an electronics system around an Arduino card computer and then write the software to the system. The entire design can then be tested by simulating the systems function in an associated simulator. You can easily test your own software solutions for a specific hardware design.

Tinkercad is a web-based development environment which makes it independent of the user's own computer environment. To use Tinkercad you only need to create an account on their website. If you want to use this option for solving the exercises 4.3 and 4.5, you probably create one common account for your project group.

Link to Tinkercad: https://www.tinkercad.com/

Under Circuits you will find the development environment that we will use in the exercises. When choosing components, you should choose all to find the needed components.

To set up the circuits you will need to use a **breadboard** that you also find in the component list.

If you never used a breadboard se the first 5 to 6 minutes of this video :

https://www.youtube.com/watch?v=oiqNaSPTI7w

Briefly about the ATmega 328 microcontroller

To test the solution of the alternative exercises exerc_ard_4.3 and 4.5 we will use Tikercad and a hardware build around an Arduino Uno card. Before you start solving the exercises you need some information about how we should design the hard and software.

First some general about micro controllers. A Micro controller is a bit different designed in comparison to a more general processor. A general CPU circuits use to have connections for address

bus, data bus and a lot of control bits like R/W etc. A controller doesn't have these buses but instead they have ready-made IO ports for both digital and analog signals. They also often have built-in AD/DA converters, timer circuits, etc. The different IO ports have fixed addresses. IO-ports and the internal modules are easily configured by writing bit patterns in special registers. ATmega 328 has two digital 8 bits IO ports (In Arduino used as one 8 and one 6 bits digital IO) as well as a number of analog inputs and outputs with associated AD and DA converters, respectively.

The digital IO ports are configured by writing 1 or 0 in the corresponding bit in a so-called data direction register DDRD and DDRB respectively. Writing and reading can then easily be done if you know the address of the ports. In Arduino's IDE you can via simple function calls or use of so-called Macron configure the port to the desired function and then easily read and write on the IO ports.

You can find a pin-layout picture fort he use of ATmega328 on the Arduino Uno computer board at:

https://i2.wp.com/marcusjenkins.com/wp-content/uploads/2014/06/ARDUINO_V2.png

Exerc_4_3_AT **(** File name exerc_4_3_at.c**)** **(2p)**

In Thinkercad you should drew a new circuit as the figure below shows. In the circuit PD0-PD7 is used as out bits and connected to the LED: s anode side via a resistor. Bit PB0 is used as inbit reading the status from one side of the DIP switch. The value to PB0 will be either 1 or 0 depending on if the DIP switch is on or not.

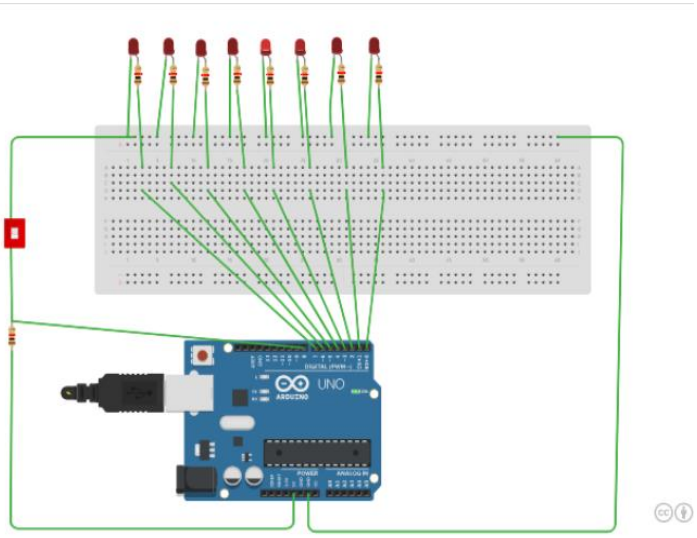
Write then a program with the function described below that uses out- and in-port as described above.

The program should work as follows:

1. Configure the PORTB to inport and PORTD as out port.
2. Write 3 (binary 0000 0011) on the OUT_PORTD .
3. In an infinitive loop :
   Read bit PB0 on the IN_PORTB
   - If bit is set (1) rotate the content on OUT-port one step left. If bit 7 is set before the left shift bit 0 should be set after the shift.
   - If bit is clear (0). Do not change the value on OUT-port.

For the delay use the delay function as shown in the program skeleton below.

You are **not allowed to use the Arduino-style** of the program and must use the way of read and write as viewed in the program skeleton below.



Run the program in the simulator to test the function.
Note that it's not possible to change the switch when running the program so you must use the **debugger mode** in which you can change the switch before step further in the program.

When ok you just copy the code to any editor and add necessary information in the program head, save it and hand it in as usually together with the other solutions.

// Program skeleton for the exercise 4.3 AT

```
// You can this file at course home page, Files/ Program example…/Course week #4
// ------- Program skeleton for exerc 4.3 AT ----
// ------- File name :   skeleton_4_3_AT.txt ----------------
//#include <avr/io.h>   Not needed
#include <util/delay.h>

#define REG8(x) *((unsigned char *) (x))     // A macro to use for IO R/W
#define OUT_PORTD          0x2B
#define IN_PORTB           0x23
#define DDR_D              0x2A
#define DDR_B              0x24
#define BLINK_DELAY_MS     800


int main (void)
{
   unsigned char code;
   unsigned char incode;
   // If use of simple pointer and not macro REG8
   //unsigned char *portd;
   //portd=(unsigned char *)0x002B; // Must do as this
   //*portd=0;  If use of pointer for IO R/W
   //  set  PORTD for output, set bit gives out
     REG8(DDR_D)=0xFF;
  //  set  PORTB for input, clr bit gives inbit
     REG8(DDR_B)=0x00;

  while(1){


  _delay_ms(BLINK_DELAY_MS); // In util/delay.h

 }
}
```

Exerc_4_4_AT (File name exerc_4_4_at.c) (2p)

In this exercise you shall design a general C-program that intends to read a keyboard in the same way as described in exercise 4_4_XCC.
The program reads a keyboard register with a specific address. The value (0-15) of a pressed key is the value of bit 0 – 3 in the register. If a key is down when the register is read bit 7 is zero otherwise the bit 7 is set (1).

To simulate this reading of the keyboard with a general C-program we will use some C-programs help-functions. You can find these at the course homepage in Files. / Program examples… ../Course week #4

In the file : bit_manage_func.c you can find the following functions:

void f_delay(int tenth_sec) : Gives a time delay for a number of tenth of seconds.

unsigned char random_inport(void) : Generates a random value 0 – 255 .

This will simulate the read value from an 8 bit inport in a system.

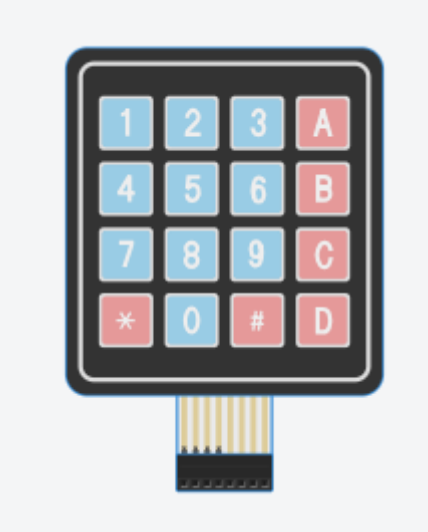
void printport(int portvalue) : Prints out the binary pattern and the decimal value for a char of value 0 – 255. (Not used in this exercise).

The keyboard is controlled by a hardware and to read a pressed key number you read the 8-bit value in a register at a specific address. In our case we simulate this by call the function ***random_inport()***. The value in register should be interpreted as follows: If bit no 7 is zero there is a key pressed and the key nr (0 – 15) can be read as bit 0 to 3 in the value. The value on bit no 4 to 6 is of no interest and its value should not affect the program function.

Write a program that periodical (once every half second) reads the keyboard register (the random number). If a key is pressed (bit-7) read the key nr and print it out on the console in hexadecimal form 0 – 9, A- F. If no key pressed there shouldn't be any printout.

Exerc_4.5 AT (Filename: exerc_4_5_at.c): Keyboard scanning. (3p)

Some cheap keyboards are not connected to a supporting hardware with register for reading the key number. This simpler device is a bit harder to read. The XCC ML2 Keyboard can act in this simple mode. In TinkerCad you can connect such a simple keyboard to Arduino Uno and then design a program that reds a pressed key number.



The principle for the keyboard and reading a key.

The keyboard is built up by four row connectors and four column connectors. At the 16 crosses between raw and column there is normally no electrical connection. At each cross there is a key and if a key is pressed there will be an electrical connection between the column and raw.

The rows (row 1 to row 4) is possible to connect to anything via the four left bits in the flat connector. On the four right bits the four columns are possible to connect to anything.

The keyboard can be connected to the Arduino system bit PD0 to PD7. The PORT_D can then be configured so that PD7-PD4 are out-bits connected to the rows and PD3-PD0 are in-bits connected to the columns.

One way to detect a pressed key is to set a row to low (0) and the other row bits to high (1). If any key in that row now is pressed it will give the corresponding column bit a clear value (0) and the other bits high values. This can be detected by reading the column INPORT. If no key in the specific row is pressed the value from the column-bits will be high on all four bits.

To read the keyboard you must in a loop set one rows to low and the other to high and check if column INPORT is not equal to four high bits. If still four bits high set next row to 0 and check the columns value. As soon as you find a column value with not four set bits you have detect a pressed key. If so, you know which row and column it is, and you can from that calculate which key number is pressed. The pattern for these two nibbles (nibble = four bits) is unique for each key. If you check all four rows without finding any column INPORT value not equal to four clear bits there is no key pressed for the moment.

We can assume that Key is numbered from top row from left 0 , 1 , 2 , 3 , 4 and next row 5 , 6 , 7 , 8 etc.

To do :

Start TinkerCad, open Gallery/Circuits and look for the example Calculator. It's a system with Arduino, a keyboard and an LCD display. Copy this to you self so you can develop and simulate programs for that system.

Study the program written with support of a lot library function. Erase most of the program and keep only the needed part to write characters on the LCD display.

Develop a program that has the structure as below:

- configure the IO port
- in an infinity loop
  Call a function checking if any key is pressed and if read it's value
  Prints out the key value (0-9 , A-F) on the LCD display.
  Delay for one second.

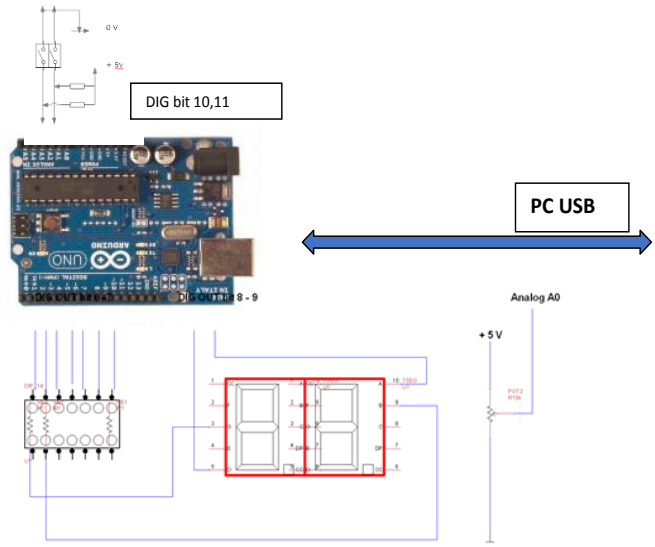
Demonstrate and hand in the solution as for 4.3 at.

Work package nr 5/ A simple embedded system for AD converting
Exerc_5_ 1 : (Filename exerc_5_1.c)

In this exercise you should build a Temperature meter based on an Arduino Computer Board and some additional components and a program developed with the Arduino environment version of C.

The system should be built as the figure to right shows the principal for.

Note: The circuit figure is just viewing the principle for the real circuits and the symbol for the display is not exactly the used one. For details see data sheets for individual components.



Components (If needed Data sheets Files on course homepage**)**

1 pc	Arduino Uno CPU board
1 pc	Dual 7 segment digit display
1 pc	Resistance net in a DIL (7x0.68 kOhm)
1 pc	Temperature sensor
1 pc	2 bits dip switch
2 pc	Resistors 1 kOhm

Very draft description of the function: (Feel free to input personal ideas)

a) Basic version
In a loop structure:

- Read the analog value from temp sensor through AD channel A0
- Calculate the temperature and then the pattern for the digits.
- Check dipswitch if Normal or Max mode
- Check and if temp more than max save to max
- Write out to one digit
- Delay.
- Write out to the other digit
- Delay

You should use one digital Out bit for each of the digits segments (#7) . Set bits pattern for the digit nr (0-9). High bit value will light up the actual segment. Chose digit to light up by the two additional out bits (8-9). One of these two bits low will turn selected digit on.

Try to read the analog input at least with a frequency of 10 Hz and to find a frequency for the displaying of each digit to give a readable value.
The 2 bit DIP switch should be used to select Normal mode or Max mode. In Normal mode the system views the actual temperature and in Max mode it will view the stored max temperature. Connecting the DIP switch to digital in its 10 and 11. The connection of the resistors will give the value high (1) to the inbit if switch not closed and low (0) if closed.

Components and construction of the system:
For the project we have components for up to 12 groups. You must combine 2-3 projects groups to create one Arduino group of about 6 students. This group can get a box with all needed components, wires and tools for building a system. One of the group members is responsible for the hardware and had to sign a paper when fetching out from the TA or the teacher.
In the group you must jointly decide how to build the system, and then build it up. After that you can go on together and develop the program or develop different programs in the smaller groups.

To hand in at the deadline (As defined at the homepage)

- The **program code** for your solution. (exerc_5_1.ino as a .zip file) (6 p)
  as assignment : **Work_package nr 5_1** at the course homepage.
  All small groups hands in a version even if common solution.
  List all names that participated in the solution.

- **A written report (.pdf document)** . (4p)
  One written report from each small group as assignment: **Work_package nr 5_2** at the course homepage.

The **written report** should approximately cover :
- A short description of the system. (For a picture of the system you can download a .docx version of the document : DIT165 Exercises_P2_v1_1402xx
- Some short of the ADC function in Arduino. If possible try to describe the function analogRead().
- Short description of your program structure.

Size : Approximately 2-3 pages

Work package nr 6 : (Total points 10 p)

Wil be delivered in a separate pdf DIT632 Exercises_P3_v19xxxx.pdf later on .