Exercises and assignments for the course DIT165 Development of Embedded systems _ Part 2 of 3

Bonus points : For all assignments in this course you will receive bonus points added to the points in the written exam. The bonus points are only for the ordinary exam and the two re-exams during the same year.

For each of the handed in and approved exercises you get a number of points from zero up to a maximum specified points. You can get a total sum of 53 points. Your total sum of points will be divided by 10 and then rounded to an integer following the usual rules. This will give you 0 - 5 bonus points to add to the written exam points. The written exam will have a maximum of 30 points without included bonus points. Pass level is 15 points.

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: 2017-mm-dd
Group Number:
Members That contributed:
xxxxxxx xxxxxxxx
yyyyyy YYYYYYYYYY
zzzzz ZZZZZZZ
Demonstration code: [<Ass code 1-5> ]        Important , No code no bonus !
====================================== * /
```

All submissions must be handed in by one of the group members via GUL. Before handed in all programs should be demonstrated for a course assistant and if the TA approve the program you will get a specific unique demonstration code <xxxxx> to type in to the programs 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 a IDE 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 for study low level C-programming. We use this environment because it includes a simulator which gives us a good opportunity to study what happens when we control individual bits of a byte in I/O-ports.

The XCC IDE exists only for Windows environment. Hopefully there is someone in every group that are Windows user. If you not have the possibility to install XCC you can do alternative exercises for all of them or install the VM package with Windows 10 included as pointed out from the lecture slides for lecture 3_2 and 4_1.

For those who has Windows computers it is very easy to fetch and install the program. You can fetch an installer program from the course homepage in Documents/ Materials for exercises and assignments . In the same catalog you can find a short quick guide for start using XCC, Intro XCC12 Quick Guide_12.pdf

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
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 nr 0

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 of 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
gear_pos	2
key_pos	2
brake1	1
brake2	1

Pack these values together in a byte (unsigned char) as an integer and print it out to stdout in hexadecimal form , in this example it should be 'AB' corresponding to bits '10101011. After this your program should return 0. If your program finds anything wrong (too many/few arguments, faulty input values..) your program should print error and return a not zero value.

Exerc_4_ 2 (decode.c) (2p)

Write a program ***decode.c*** that takes 1 argument , in the example 'AB' (less or more should be treated as an error). The arguments should correspond to the byte as printed by your code program. If your program finds anything wrong (too many/few arguments, faulty input values..) your program should print error and return a not zero value.

The program should unpack the bytes according to the specification above. Print out the result like this:
Start program in consol windows : ***decode AB***
Should give this printed out.

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

The exercises 4.3 – 4. 5 are aimed to be developed and tested in the XCC IDE and its´ including simulator.

If you **not have a Windows computer** or not willing to spend some extra time for a possible better understanding of IO management by install and understand this IDE you just **chose to solve the alternative exercises 4.3 alt – 4.5 alt** that you find further on in this document.

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 making it possible to test program in combination with a simulated part of the IO unit without having access to the real hardware.
Unfortunately, XCC12 IDE is just available for Windows computers. It is very good for illustrate program developments dealing with reading / writing to IO units so I give you who have access to a Windows computer the opportunity to solve and test a couple of exercises (4.3 – 4.5) with help of this environment. There will later on also be a couple of exercises in WP 6 that could be tested in this IDE.

To be able to solve the exercise 4.3 – 4.5 read the information of and use 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 Windows computer which is very easy to do. You find the installer at the same place as the Quick Guide.
You can also find a *firstprog.c* file with a short test program that you can try to compile and run. The program will do a printout in the console (standard IO for the system). To attach the console to the system in the simulator mode you need to read the Quick Guide about testing this demo program.

Exercises for the XCC12 environment.
Exerc_4_ 3 (Filename exerc_4_3.c) (2p)

Write a program with the function described below that uses the 8 bit I/O 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:

At start of program write 3 (binary 00000011) to the OUT port. Then in an infinitive loop check bit No 4 on INPORT and if bit No 4 is set left rotate the content on the OUT port one bit.
As long as bit No. 4 on the INPORT is set periodically left rotate around the content on OUTPORT one bit per shift. The rotation should stop as soon as the bit No. 4 on INPORT is reset. The frequency of bit shift can be chosen arbitrary and adapted till simulator by using a simple delay with a for loop.
Write the program and test it in the XCC simulator.
In simulator connect the ML4 IO unit at address 0X400 which gives dipswitch input address 0x400 and parallel output address 0x402. Information in XCC help, IO simulator ML 4 devices.

Exerc_4_ 4 (Filename exerc_4_4.c) / **Keyboard reading** (2p)
Write a program with the function described below that uses the ML 2 Keyboard (**ML15** mode, address 0x09C0) and the console connected in XCC.

The keyboard reading is manage by some hardware device and to read a press the key No you just read a value in a register with a specific address. To read a key you can either check if a key is pressed for the moment or just wait for an interrupt signal which can occur for every time a key is pressed if some initiation is done. At both methods you read the value from a specific register in the device.

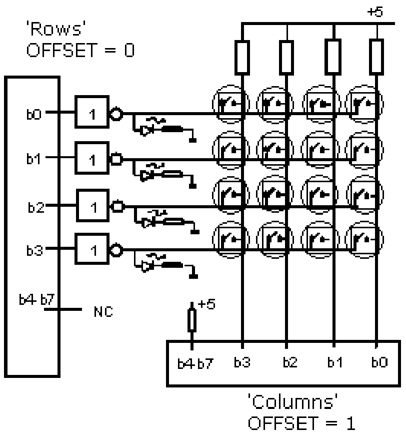
In this exercise you should develop program that periodically check the keyboard and if a key is pressed, read the value in the register and convert it to corresponding Hexadecimal integer 0 – 9, A-F and print it out in the console window. If no key is pressed when checking no printing is done.

Note: The program should consist of appropriate number of function with well specified tasks.

Exerc_4_ 5 (Filename exerc_4_5.c) / **Keyboard scanning** (3p)

Some cheap keyboards are not connected to an electronic device and is harder to read if you attach the keyboard to a system through the systems INPORT and OUTPORT.

You can see the principle for this in the figure besides.



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 column is connected to the four least significant bits (b3 - b0) on the INPORT to the system.

All four rows are through an inverter connected to the least significant bits (b3 – b0) on the OUTPORT of the system.

The idea for this keyboard connection is as follows.

If no key is pressed the value on INPORT[3..1] will be high on all bits . If value on all rows are high (what will happen if the OUTPORT[3..1] all are zero (due to the inverter between) any key pressed will not change the situation and all BITS on INPORT will still be high. The only way to detect a key pressed is if to set one row to low (set the row bit in OUTPORT) and if any key in that row is pressed it will give the corresponding column a zero value that can be detected by reading the INPORT.

To read the keyboard you have to in a loop clear one rows value (set the OUPORT bit) and check if INPORT is not equal to four set bits. If still four set bits you clear next row and check column. As soon as you find a column value not four set bits you have detect a pressed key. If so you know what key is pressed due to the value of the rows (one is zero) and the columns (one is zero) . The pattern for this two nibbles (nibble = four bits) is unique for each key. If you go through all four rows without finding any INPORT not equal to four set bits there is no key pressed for the moment.

Key is numbered as 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 address 0x09C0 which gives row register the address 0x09C0 and the column register 0x09C1). Connect also the console in XCC.

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) and 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 pushed when checking no printing is done.

Alternative exercises 4.3 alt – 4.5 alt if not use of the XCC12 environment.
The following exercises number 4.3a – 4.5a is for those not having access to a Windows computer which is necessary for the previous exercises 4.3 – 4.5.

Some C-programs files with some help-functions to use when solving this exercise can be found at the course homepage in the documets: Documents/ Program examples ../Programs for WP4

In the file : bit_manage_func.c you can find these 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.

The program also includes a main() that gives you possibility to test the functions in your environment.

Exerc_4_3 alt (Filename: exerc_4_3a.c) (2p)

Imagine that we have one eight bit INPORT and one OUTPORT.

You shall develop a program with a function as describes below.

At start program should write out the value 3 to the OUTPORT. In this case we simulate by use of printport() which had to be a little changed.

The INPORT should then be read in an infinitive loop every half a second. In this case we simulate this by reading the value from random_inport(). If bit 4 on INPORT is set a new value should be written out on OUTPORT where the bits should be rotated left one step. If bit 4 on INPORT is zero nothing new should happen to OUTPORT. In a rotation bit 7 is shifted to bit 0.

The print out should be close to as follows:

```
Inbit 4 is 0  OUT  0 0 0 0 1 1 0 0  ----------- Port value 12

Inbit 4 is 1  OUT  0 0 0 1 1 0 0 0  ----------- Port value 24
```

Exerc_4_4 alt (Filename: exerc_4_4a.c) : Keyboard reading (2p)

Imagine that we have a keyboard with 16 keys on and a console for printing out attached to a system.

The keyboard is controlled by a hardware and to read a pressed key number you read an 8 bit value in a register at a specific address. In our case we simulate this by reading ***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 register. Value on bit No 4 to 6 is not of interest and should not affect the program.

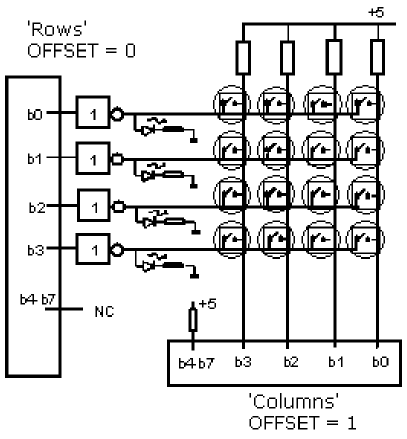
Write a program that periodical (once every half second) reads the keyboard register. If a key is pressed read the key nr and print it out on the console in hexadecimal form 0 – 9 , A- F. If no key pressed there shouldn't be any printout.

Exerc_4_5alt (Filename: exerc_4_5a.c) : Keyboard scanning. (Not possible to test) (1-3p)

Some cheap keyboards are not connected to an electronic device and is harder to read if you attach the keyboard to a system through the systems INPORT and OUTPORT.

You can see the principle for this in the figure besides.

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 column is connected to the four least significant bits (b3 - b0) on the INPORT to the system.

All four rows are through an inverter connected to the least significant bits (b3 – b0) on the OUTPORT of the system.

The idea for this keyboard connection is as follows.

If no key is pressed the value on INPORT[3..1] will be high on all bits . If value on all rows are high (what will happen if the OUTPORT[3..1] all are zero (due to the inverter between) any key pressed will not change the situation and all BITS on INPORT will still be high. The only way to detect a key pressed is to set one row low (set the bit in OUTPORT) and if any key in that row is pressed it will give the corresponding column a zero value that can be detected by reading the INPORT.

To read the keyboard you have to in a loop clear one rows value (set the OUPORT bit) and check if INPORT is not equal to four set bits. If still four set bits you clear next row and check column. As soon as you find a column value not four set bits you have detect a pressed key. If so you know what key is pressed due to the value of the rows (one is zero) and the columns (one is zero). The pattern for this two nibbles (nibble = four bits) is unique for each key. If you go through all four rows without finding any INPORT not equal to four set bits there is no key pressed for the moment.

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

To do :

Write a draft for a C-function *unsigned char check_keyboard(void)* that scans a keyboard of the type as above and that returns the key nr (0 – 15) if any key pressed and if no key pressed return FF_{hex}. The function should scan the keyboard once end then return the result.

Write the function in your editor without testing it though it´s hard to test without the hardware or a simulator for it. Show it and explain the solution for a TA. You will get an exam code from the TA if the TA think it seems to be close to a solution. Then you hand in the code together with the other programs in WP 4.

Exercise : WP4_Extra_1 (File name wp4_extra_1.docx , or .txt) (1-2p)

Discuss in the group and see if you can come up with any ideas of possible ways of test an algorithm as above exercise 4.5 without access of hardware and or access of a simulator including the IO unit. It should be as simple so it´s possible to implement and use it for an exercises like this one.

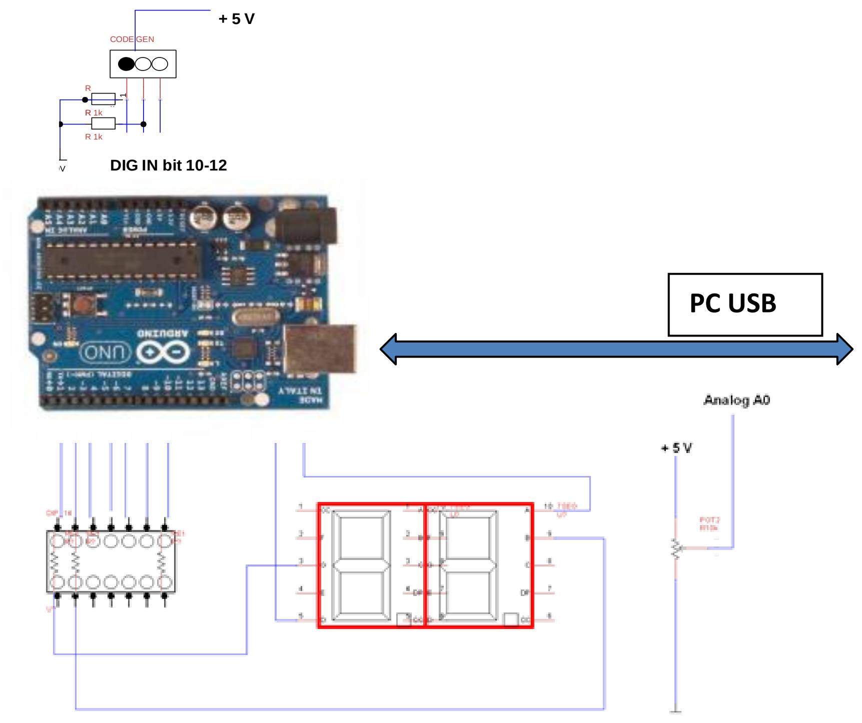
If you have any good idea describe it in a text document and hand it in together with the programs. You can do this regardless whether you have done exercises nr 4.3 – 4.5 or the alternative ones.

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 in GUL **)

1 Arduino Uno CPU board

1 Dual 7 segment digit display

1 Resistance net in a DIL (7x0.68 kOhm)

1 Temperature sensor

1 4 bits code switch (For extended function)

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.
- 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.

b) Extended version
Try to extend the function with the following:
By use of a 4 bit code switch (we can use less than four) connected to some digital in bits (10 - 12) we can read a in code depending on the selection from user binary 0 to F (if four bits). We can use the in value to select one of several possible function modes. In our case we just will have two modes; code zero basic mode and code x max temp mode.
Extend the basic version of the program with this extended mod.
Note: The code switch connect the c-pin to the pins (1,2,4,8) that have a zero value for the corresponding bit pattern. (Ex: for position 3 there is connection between c and 4 and 8.

Components and construction of the system:
For the project we have components for 10 groups. You have to combine 3-4 projects groups to on construction group. This group can get a box with all needed components, wires and tools for building a system. One of you is responsible for the hardware and had to sign when fetching out from the TA.
In the group you have to jointly decide how to build the system , and then build it up. After that you can go on together and develop the program or develope 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) (4 p basic ,+2p extended)
  as assignment : **Work_package nr 5_1** in GUL
  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** in GUL.

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 alalogRead().
- Short description of your program structure.

Size : Approximately 3 pages

Work package nr 6 : Event driven door automat (for XCC users) (Total points 10 p)

Wil be delivered in a separate pdf DIT165 Exercises_P3_v17xxxx.pdf later on .