SDU

EMP C Code Standard

General notes

- You are expected to follow these guidelines in your projects
- Originally developed by Karsten Holm Andersen and Morten Hansen

Good Programming Practice

- Think in abstraction levels
- Limit module size
- Limit function size
- Comment!
- Use proper indentation
- Use expressive file names
- Avoid: GOTO and CONTINUE
- Be aware of : RETURN and BREAK
- Avoid code repetition (copy/paste)
  - If it repeats, should it be a function?

SDU

Think in abstraction levels

- Always be sure that structures, functions modules and processes have a **well-defined purpose and interface**
- Avoid the use of handyman-variables, functions and modules with a **mixed purpose**.

SDU

Limit module size

- **Big modules** with a highly complex interface e.g. many global variables should be avoided
- Big modules are **difficult to understand** and make it nearly impossible to create a thorough module test
- Even though it is exactly the big modules, that demands module tests the most

SDU

Limit function size

- To archive a high degree of readability and transparency the **size of functions** and number of branching should be **limited**

- If a function contains more than 5 or 6 levels of indentation or occupy more lines than presented on the screen, it should be considered to **break the function into sub functions**.

SDU

Comment your code!

- Write comments for each line of code or each section of code which is not absolutely self-evident
- After one month many lines of code which you write will become non-self-evident

```
/**
   multi-line comments
   in C
 */
```

```
// single line comments
GPIO_PORTF_DEN_R = 0x1E;      // enable the GPIO pins for digital function (PF1 - PF4)
GPIO_PORTF_PUR_R = 0x10;      // enable internal pull-up resistor for switch (PF4)
```


SDU

Use proper indentation

- Indentation makes it much easier to understand your or others' code
- Use the TAB key to indent your code
- Indentation should increase inside each control structure (usually after the start of curly braces)

```
while(1)                                        // loop forever
{
    while( !ticks );

    // The following will be executed every 5mS
    ticks--;

    if( ! --alive_timer )
    {
      alive_timer        = TIM_1_SEC;
      GPIO_PORTF_DATA_R ^= 0x04;
    }
}
return 0;
```


SDU

Use expressive file names

- The **file name** of the module must relate to the well-defined **functions** of the module
- In C code the name of the module equals the name of the source file without the .c extension
- The name of the definition file is constructed by the module name with a .h extension
- For Compatibility reasons to the MS/PC-DOS operating system, no module name should exceed **8 characters**
- As part of the documentation for the project, a survey of the files and their contents/functions should be created.

SDU

Avoid GOTO and CONTIUNE - pay attention to RETURN and BREAK

- It is always possible to avoid GOTO, CONTINUE and BREAK in the code
- BREAK is though used in the switch statement to end each case
- The RETURN instruction is used to return from a function. It must be the last instruction of the function and it should always be placed here
- Use state variables instead of several return statements in a function
- By restricting RETURN to the end of the function, it is secured that the function has a single entrance and a single exit
- This ensures a readable structure

SDU

Avoid code repetition (copy/paste)

- The same piece of code (coherent program lines) should not occur more than once
- This also applies to code pieces that in principle are identical but work on different variables
- Use functions, constants, definitions and local variables instead of identical pieces of code

SDU

Programming Rules

- Rules about upper and lower case letters
- Use expressive identifiers
- Use English
- Underscore in identifiers
- Avoid numeric values in the code – magic numbers
- Indentation and the use of {}

SDU

Rules about upper and lower case letters

- C is case sensitive
- Type definitions, enumerations, macros, #define's and constants are written in upper case letters
- Hexadecimal numbers are written as 0xFF
- All other words in the source code are written in lower case letters
- Comments and other explanations may contain both upper and lower case letters

```
#define TASK_SW_TIMERS 0x1A
#define TASK_MAIN 0xFF
```


SDU

Use expressive identifiers

- The names for variables, constants, types, functions and so on should be expressive – explaining their purpose/function
- Abbreviations must be avoided, as they often would only destroy the readability
- According to the ANSI-C specification the compiler must support at least 32 characters in local declarations, and 8 characters in global declarations

```
static INT8U flash_state = 0;
 INT8U led_action;
```


SDU

Use English

- Nothing is worse than source code where several languages are mixed
- C is English, and that is argument enough for writing all the code and comments in English
- Yours truly does not fully understand Danish, but it would be good for you if I do understand your code

SDU

Underscore in identifiers

- To help with readability, use underscores to separate words, for example counter_value

SDU

Avoid numerical values in the code

- Use #define or const declaration instead of **magic numbers** (numerical values) in the code
- There are however special situations where numerical values might be clearer, e.g. assignment of the start value = 0, in a counting loop counting from 0 to a final value
- Use of magic numbers is wrong because you will forget what they stand for, use of descriptive macros is much better

```
#define TIM_1_SEC     200
...
//  int alive_timer = 200;       // MAGIC NUMBER - wrong
    int alive_timer = TIM_1_SEC;
```


SDU

Indentation of source code in connection with structure

- Statements between { } or a single line after a control structure indents with 1 TAB
- The overall philosophy is that { and } belongs to the same level and thus shall have the same indentation
- { is always alone, in other words { is always followed by a line break
- In type definitions or do while loops } is followed by the type name or an expression, else } is alone on the line

```
while( expression )
{
  statement;
};
```


SDU

Numeric and Boolean expressions

- In numerical expressions the arithmetic operators like +, -, * and / must be separated by spaces on each side
- Equal signs (‘=’) are followed by exactly one space and are preceded by at least one space

```
variable_1 = (variable_2 * variable_3) / variable_4;
variable_10 = -(variable_3 + variable_4) / 2;
variable_6 = (variable_6 | variable_4) & (variable_1 | variable_2);
variable_7 = (variable_5 * variable_5) + (variable_2 / variable_1);
variable_11 = variable_1;
if ( ( variable_1 == (variable_2 & DEFINE_1) && ( variable_3 == MAX_LIMIT ) )
{
        variable_12 = variable_3;
}
```


SDU

Programming Rules III

- Rules for portability
  - Use data types with fixed data width, to replace the predefined data types in C. See the supplied header file containing type definition: emp_type.h

```
typedef unsigned char             BOOLEAN;
typedef unsigned char             INT8U;     /* Unsigned  8 bit quantity           */
typedef signed   char             INT8S;     /* Signed    8 bit quantity           */
typedef unsigned short            INT16U;    /* Unsigned 16 bit quantity           */
typedef signed   short            INT16S;    /* Signed   16 bit quantity           */
typedef unsigned long             INT32U;    /* Unsigned 32 bit quantity           */
typedef signed   long             INT32S;    /* Signed   32 bit quantity           */
typedef unsigned long long        INT64U;    /* Unsigned 64 bit quantity           */
typedef signed   long long        INT64S;    /* Signed   64 bit quantity           */
typedef float                     FP32;      /* Single precision floating point    */
typedef double                    FP64;      /* Double precision floating point    */
```


SDU

Programming Rules IV

- WHILE statement

```
while( expression )
{
    statement;
};
```

- DO-WHILE statement

```
do
{
    statement;
} while( expression );
```


SDU

Programming Rules V

- FOR statement

```
for( counter = 0; counter < MAX_VALUE; counter++ )
{
   statement;
};
```


SDU

Programming Rules VI

- SWITCH statement

```
switch (select_expression)
{
    case CASE_0:
        statement_0_0;
        -
        statement_0_n;
        break;
    case CASE_1:
        statement_1_0;
        -
        statement_1_n;
        /* no break */ /* execute statements in case m */
    case CASE_M:
        statement_m_0;
        -
        statement_m_n;
        break;
    default:
        statement do_this_if_no_case_match;
};
```


SDU

SWITCH statement

- The switch structure should always be used instead of 3 or more branches on the same variable
- This means that whenever writing an “if - else – if” statement, you should consider to using a switch statement instead

SDU

File structure

- Function header
  - Every function declaration must use a standard function header
  - This header must contain information about the input, the output and a description of the function

```
Void test1(void)
/*********************************
*  Input   :
*  Output  :
*  Function:
*********************************
{
      dummy1++;
}
```


SDU

File structure II

- Standard module header
  - All the *.c and *.h files must start with a standard module header
  - This contains information about the modules name, the project it belongs to, and a version log

```
/*********************************************
* Univeristy of Southern Denmark
* Embedded Programming (EMP)
*
* MODULENAME: emp.c
* PROJECT: EMP
* DESCRIPTION: See module specification file (.h-file)
* Change log:
*********************************************
* Date of Change
* YYMMDD
* ---------------
* 190128 PO Module created.
*
*********************************************
```


SDU

File structure III

- Overall structure in .c files
  - The contents of the module file (*.c file) is split up into **6 sections**: Headline, include files, definitions, constants, variables and functions
  - The individual sections are **separated by a comment line**, and all the sections must be included even though they are empty
  - This ensures a uniform structure, and prevents wasted time if the section is needed later

SDU

File structure III

- Overall structure in .c files

```
/****************** Header *********************/
/****************** Include files **************/
#include ”emp_type.h”
/****************** Defines ********************/
/****************** Constants ******************/
/****************** Variables ******************/
INT8U dummy1;
INT16S dummy2;
/****************** Functions ******************/
void test1(void)
/*********************************************
* Input:
* Output:
* Function:
*********************************************/
{
    dummy1++;
}
/****************** Enf of module **************/
```


SDU

File structure IV

- Overall structure in .h files
  - The contents of the specification file (*.h file) is split up into **5 sections**: Headline, include files, definitions, constants and functions
  - As in the module file, the individual sections are separated by a comment line, and all the sections need to be included even though they are empty
  - This ensures a uniform structure, and prevent wasted time if the section later is needed.

SDU

File structure IV

- Overall structure in .h files

```
/****************** Header *********************/
/****************** Include files **************/
#include ”emp_type.h”
/****************** Defines ********************/
#define MY_DEF 12
/****************** Variables ******************/
extern INT8U dummy1;
extern INT16S dummy2;
/****************** Functions ******************/
extern void test1(void)
/***********************************************
* Input:
* Output:
* Function:
***********************************************/
/****************** Enf of module **************/
```


SDU

File structure V

- Module check in header file
  - In order to prevent compile error when including files, the specification file must check if the module already has been included ones.

```
#ifndef _MODULENAME_H
#define _MODULENAME_H

/* The contents of the specification file */

#endif
```


SDU