UTM | ROBOCON Team

UNIVERSITI TEKNOLOGI MALAYSIA

Department of Programming

ROBOCON CODING STANDARDS

Programming and Organization Guideline

Ng Khin Hooi
August 29, 2010

© 2002 - 2011
UTM ROBOCON

FINAL

This page was intentionally left blank.

© 2002 - 2011
UTM ROBOCON

TABLE OF CONTENT

© 2002 - 2011
UTM ROBOCON

This page was intentionally left blank.

© 2002 - 2011
UTM ROBOCON

Chapter 1 Introduction

1. This document describes the standards used in the ROBOCON programming.
2. The standard is divided into two parts namely the programming coding standard and the project organization standard.

1.1 Programming Coding Standard

1. The programming source code should be arranged in a manner that signifies the standard of ROBOCON programming.
2. The naming and format should also be standardized.

1.2 Project Organization Standard

1. Programming usually consists of normal target and library target.
2. Programming is also classified under robot programming, module programming and peripheral programming.
3. All these types of programming should be arranged in a proper manner for better library arrangement.

1.3 Rational of Standard

1. Standard will help programmers in ROBOCON to arrange their code in a neat way.
2. It also helps a programmer to read other programmer's code in an easier way.
3. Library can be arranged neatly in the repository.
4. Documentation will help other programmer of following years to understand the code.
5. Development by programmers of following years can be done faster and more organized.

© 2002 - 2011
UTM ROBOCON

This page was intentionally left blank.

PART A: PROGRAMMING CODING STANDARD

© 2002 - 2011
UTM ROBOCON

This page was intentionally left blank.

Chapter 2 Indentation

1. Indentation will be four (4) spaces as the default in MDLAB IDE.
2. Use tab instead of four spaces.

2.1 Indentation for switch-case

1. Switch and case should be on the same column.
2. Statement will be indented once.

```
switch (variable){
case ‘a’:
    statement_1;
    statement_2;
    break;
default:
    statement_3;
    break;
}
```

Chapter 3 Columns

1. Maximum number of columns must be 80.
2. This will improve readability without the need to scroll sideways.
3. A good practice is to have a ruler in MPLAB IDE as a guide.
4. To break a long line, return the last word that touches the 80th column line and tab until the remaining line reach the 80th column. Do not keep the remaining line at the beginning of the row.

© 2002 - 2011
UTM ROBOCON

Chapter 4 Braces for Blocks

1. Curled braces for different blocks are place at different area.
2. Two significant differences are the block for program flow control and function.

4.1 Braces for program flow control

1. Opening braces will be placed on the same row of the keyword.
2. Closing braces will be placed on a new line at the first column.

4.1.1 Braces for if-else with single statement

1. Do not use braces for single statement.

```
if (condition)
    statement_1;
else
    statement_2;
```

4.1.2 Braces for if-else-if with multiple statement

1. Use braces for all branch in if-else even there is a single statement.

```
if (condition){
    statement_1;
} else if (condition){
    statement_2;
    statement_3;
} else {
    statement_4;
}
```

4.1.3 Braces for switch, while, do-while, and for

1. Braces for program flow control will retain.

```
switch (variable){
case 'a':
    statement_1;
default:
    statement_2;
}
```

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

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

```
for (; ; ){
    statement;
}
```

4.2 Braces for functions

1. Opening braces will be at a new line.
2. Closing braces will also be at a new line.

```
void function(void)
{
    statement;
}
```

Chapter 5 Spaces

1. Spaces must be used in a manner that will make code looks neat.
2. Different place will invoke different spaces rules.
3. Note that this section of rule is optional but encouraged to follow.

5.1 Spaces for program control flow keyword

1. Use a space after each keyword.

```
if (condition)
switch (variable)
```

5.2 Spaces for functions and macros

1. Do not use space for functions and function-like macros.

```
a = addition(b, c);
```

5.3 Spaces for assignment, binary and ternary operators

1. Use a space at each side for these operators.

```
= + - <> * / % | & ^ <= >= == != ? :
```

5.4 Spaces for unary operators

1. Do not use space for these operators.

```
& * + - ~ ! ++ --
```

© 2002 - 2011
UTM ROBOCON

Chapter 6 Naming

1. Naming will be different for each type of keyword.

6.1 Define preprocessors

1. Define will use all uppercase.
2. Separate each word with underscore.

```
#define MAX_NUMBER 100
```

6.2 Enumerations

1. Enumeration will use all uppercase.
2. Separate each word with underscore.

```
enum {START = 0, STOP, REPEATED_START};
```

6.3 Type definition

1. For type declaration, use all lowercase. Separate each word with underscore. End the type with `_t`.
2. For variable declaration, use all lowercase. Separate each word with underscore.
3. For each members of the structure, use lowercase, Separate each word with underscore.
4. Note that braces like in *4.1 Braces for program flow control*will be used.
5. This is applicable to structure and union.

```
typedefstruct{
intquot;
int rem;
} div_t;

div_t number_1, number_2;
```

6.4 Variables

1. Global and local variable will use different naming.

6.4.1 Global variables

1. Use Hungarian notation.
2. Start variable with the type. For example uc for `unsigned char`.
3. Capitalize each word.
4. Do not use underscore for each word.

```
unsigned char ucFirstNumber = 100;
signed intsiSecondNumber = 101;
```

6.4.2 Local variables

1. Do not use Hungarian notation.
2. Use all lowercase.
3. Separate each word with underscore.
4. Use short words if possible.

```
i, tmp, second_no, pos, speed
```

© 2002 - 2011
UTM ROBOCON

6.5 Functions

1. Naming for function will be divided into two types.

6.5.1 Internal functions

1. Usually these kinds of functions are for the main program to use.
2. Use all lowercase.
3. Separate each word with underscore.

```
addition(); get_position(); calculate_speed();
```

6.5.2 Modules and peripherals functions

1. These kinds of functions will interface the main program with modules through communication protocols or to setup and use peripherals.
2. Start with all uppercase of the module or peripheral.
3. Use capitalized words.
4. Do not separate words.

```
MCMForward(); LCDPutsChar(); I2CSend(); ADCStart();
```

© 2002 - 2011
UTM ROBOCON

Chapter 7 Comments

1. Use only slash-star comments for describing the statements or functions
2. Comments should be put above of a statement if it is long or at the right side of a statement if it is short.
3. Use comment blocks for multiple line of comments
4. This rule is exception for 'hiding' codes where double slash can be utilized.

```
/*
 * This is a block comment
 *where comments come in multiple lines
 *to describe a statement or function
 */
statement;
```

```
/* This is a single line comment describing statement or function */
statement;
```

```
statement;  /* Short description */
```

```
//statement; /* This is to hide a statement for troubleshooting */
```

Chapter 8 Title Block

1. Every source code or header code will need to have a title block.
2. Include title, author, current version, and date.
3. Include also description, version history and current bug.
4. Block patterns are free but please stick to *Chapter 3 Columns* and preferably *Chapter 7 Comments*.

```
/*****************************************************************
 * Title    : Motor Control Module
 * Author   :Ng Khin Hooi
 * Version  : 3.32
 * Date     : August 2010
 *****************************************************************
 * Description:
 * - Introduced a new function
 *
 * Version History:
 * 3.30
 * - Fixed bug
 * 3.00
 * - Added function_1
 *
 * Bugs:
 * - Variable assignment at certain area.
 *****************************************************************/
```

8.1 Blocks Elements

1. Title – The topic of the code, either a robot program, module program or a peripheral program.
2. Author – Use full to write the author's name. May include multiple authors.
3. Version – Use a number with two decimal places only.
4. Date – Month and year only
5. Description – Describe the current version
6. Version History – Describe the previous versions (as in the description for previous versions).
7. Bugs – Current known bugs that have yet to be solved. Include the solution in version history.

© 2002 - 2011
UTM ROBOCON

This page was intentionally left blank.

PART B: PROJECT ORGANIZATION STANDARD

This page was intentionally left blank.

Chapter 9 Versioning

1. Version format will be one number with two decimal places.

```
Version 3.01
V3.01
```

2. First version will start with V1.00.
3. Each revision will increment one significant figure depending on the changes made.
4. Add 0.01 for slight changes, bug fix, typo error, and the likes.

```
Version 3.01 to Version 3.02
```

5. Add 0.10 and floor rounding for major changes, adding of functions, and the likes.

```
Version 3.02 to Version 3.10
```

6. Add 1.00 and floor rounding for rewriting the whole code for the same program, module or peripheral.

```
Version 3.12 to Version 4.00
```

7. Note that if adding any number that will affect the next significant figure, just let it be.

```
Version 3.09 + 0.01 will produce Version 3.10
Version 3.90 + 0.10 will produce Version 4.00
Version 3.99 + 0.01 will produce Version 4.00
```

8. Version history should be written from bottom to top in the title block as in *Chapter 8 Title Block*.

```
 * Version History:
 * 3.30
 * - Fixed bug
 * 3.00
 * - Added function_1
 * 2.00
 * - Some function added
 * 1.00
 * - Introduce module into library
```

© 2002 - 2011
UTM ROBOCON

Chapter 10 Project Types and Naming

1. Project can be divided into several types under different category.
2. First category is function.
3. Second category is target.

10.1 Category by Function

1. Components under this category are robot, module and peripheral.
2. Robot programming is for a sequential flow of instructions. It may include target from other project type.
3. Module programming can be either a sequential flow of code in another microcontroller or it can be a library target built in a project.
4. Peripheral programming is used to initialize a peripheral and to utilize a peripheral. If possible, use the Microchip C Compiler built-in library.
5. Module programs and peripheral programsare classified under ROBOCON library.

10.2 Category by Target.

1. Components under this category are normal and library.
2. Normal target is an executable program, usually contains sequence of instruction to execute in a `main()` program.
3. Library target is a group of functions to execute a task. May or may not have a dedicated project.

© 2002 - 2011
UTM ROBOCON

10.3 Project Naming

1. A certain programming can have two categories.
2. Most source code naming will be similar to the project naming.

	Normal Target	Library Target

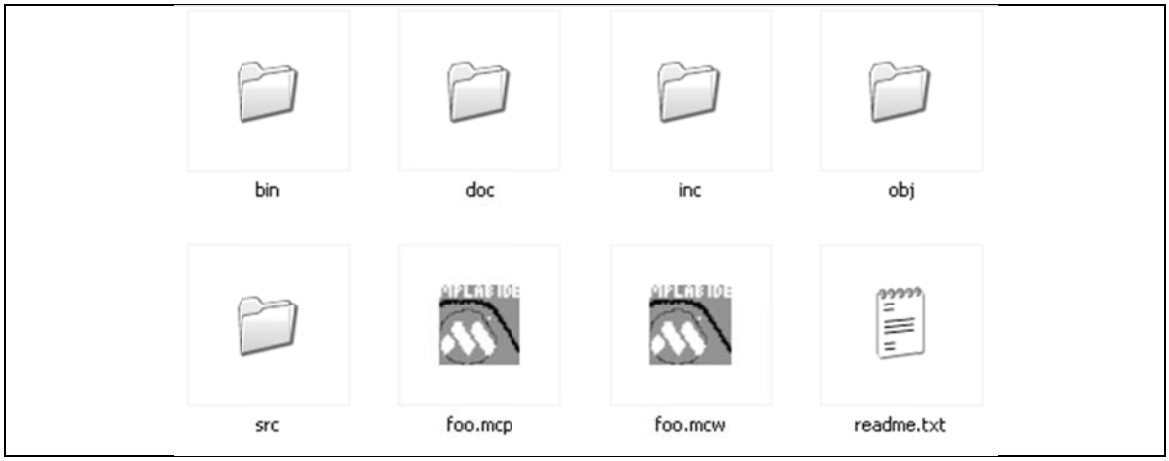
Robot program	- Program for robots.
	- Project name should be lowercase, separate words with underscore.
	
	`mankaura.mcp`
	`auto_robot.mcp`
		- Robot specific functions.
		- Project name or source code should be lowercase, separate words with underscore.
		
		`turn.c`
		`line_following.c`

Module program	- Program in another microcontroller for specific purpose.
	- Project name should be uppercase depicting the function of module.
	
	`MCM.mcp`
	`LDR.mcp`
	
	- Naming for module `main()` source code should contain a `_main` suffix.
	
	`MCM_main.c`
	`LDR_main.c`
		- Extension of module program in normal target or module interface program.
		- Naming of project or source code, start with the module or peripheral abbreviate all in uppercase, continue with an underscore then use all lowercase separate words with underscore.
		
		`MCM_system.c`
		`MCM_interface.mcp`
		
		- Application based
		- Project name should be uppercase depicting the function of module.
		
		`PID.mcp`
		`SR.mcp`

Peripheral program		- Using peripherals.
		- Project name should be uppercase depicting the function of module.
		
		`ADC.mcp`
		`QEI.mcp`

Chapter 11 Project Files Organization

1. Each project should be put in a folder.
2. In the folder, there will be output folder (bin), documentation folder (doc), include folder (inc), object folder (obj), source code folder (src), MPLAB project file (.mcp), MPLAB workspace (.mcw), and an optional readme file (.txt).



11.1 Output folder (bin)

1. This folder consists of three file types generated from MPLAB IDE after building a project.
2. .hex, .cof, and .map.

11.2 Documentation folder and References (doc)

1. Documentation will be done in Microsoft Word according to the Department of Programming Formal Documentation template.
2. For any programming, two formal documentations are needed.
3. The first is the technical report documentation.
   3.1. Documentation will follow the UTM thesis writing format which includes:
      3.1.1. Abstract
      3.1.2. Introduction
      3.1.3. Literature review
      3.1.4. Methodology
      3.1.5. Result and discussion
      3.1.6. Conclusion and recommendation
4. The second is the instruction manual documentation which explains the methods to use the program.
   4.1. Instruction manual should include:
      4.1.1. Introduction
      4.1.2. Variables and Structures
      4.1.3. Functions and Macros
5. This folder should include softcopy of references or any link to the following topic.
6. This folder should also include the files used in MPLAB tools like dsPIC Filter Design, Matlab and so on.

© 2002 - 2011
UTM ROBOCON

11.3 Include folder (inc)

1. Header files for the project.

```
auto_1.h, mainboard_1.h, MCM_system.h, MCM_common.h, I2C_sending.h
```

11.4 Object folder (obj)

1. Object file generated from building the project.
2. Project should include this object file instead of source code file.

11.5 Source code folder (src)

1. The source code for the project, consist of .c or .s

```
auto_1.c, mainboard_1.c, MCM_system.c, MCM_common.c, I2C_sending.c
```

11.6 MPLAB project file (.mcp) and workspace (.mcw)

1. The MPLAB project file and workspace file.

11.7 Read me file (.txt)

1. Readme.txt will state the content of each folder.

© 2002 - 2011
UTM ROBOCON

Chapter 12 Project Source Code and Header File Format

1. Source code and header file from different target should contain different elements.
2. Format for source code and equivalent header file for normal target and library target can be flexible.

	Normal Target Source Code (.c)	Normal Target Header File (.h)	Library Target Source Code (.c)	Library Target Header File (.h)
Title block	✓	✓	✓	✓
Include Header	✓	✓	✓	✓
Define		✓		✓
Enumerator		✓		✓
Variable	✓	✓	✓	✓
Function Prototype	✓	✓		✓
Main Function	✓			
Subroutine Function	✓		✓	

© 2002 - 2011
UTM ROBOCON

Back Matter

This coding standard is composed based on agreement of the programmers for ROBOCON Team 2011. The standard also adapts the standard from previous years. Standards from various sources too are adapted like Linux Kernel Coding Standard and Microchip C Compiler Library. Acknowledgements are given to Patrick Chin Jun Hua, Low Yee Ling and Tam Lih Ting who participated in the development of this coding standard and to Tan Hui Fen and Teoh Han Jie who reviewed it.

© 2002 - 2011
UTM ROBOCON