**Sprints**

**Coding Guidelines**

**Version 1.0**

**Hossam Adel**

## Contents

[2. Version History 3](#_Toc23538264)

[3. Coding Rules 4](#_Toc23538265)

[3.1 Code Line Width: 4](#_Toc23538266)

[3.2 Multiple Inclusion Protection 4](#_Toc23538267)

[3.3 Signed Operations and Mathematical 4](#_Toc23538268)

[3.4 Pointer Basic Rule 4](#_Toc23538269)

[3.5 Local Variable: 4](#_Toc23538270)

[3.6 Global Variable 4](#_Toc23538271)

[3.7 Type Definition 5](#_Toc23538272)

[3.8 Inclusion Sequence 5](#_Toc23538273)

[3.9 Inclusions General Rule 5](#_Toc23538274)

[3.10 Code Template 5](#_Toc23538275)

[3.11 Error Handling 6](#_Toc23538276)

[3.12 Explicit Casting is Mandatory 7](#_Toc23538277)

[4. Files naming and Folder Structure 7](#_Toc23538278)

5. Code commenting and Doxgen Format : 8

# Version History

|  |  |  |
| --- | --- | --- |
| Version 1.0 | Initial Draft | 1/11/2019 |

## Coding Rules

## Code Line Width:

The code line have to between 80 to 120 character to fit to any type of screen.

## Multiple Inclusion Protection

All the include files MUST have a header protection using the same name of the file but CAP latter with \_ between the words and two \_ in the before and after the name.

Ex: BCM\_PbCfg.h should have

#ifdef \_\_BCM\_PBCFG\_\_

#define \_\_BCM\_PBCFG\_\_

/\*Your code \*/

#endif

## [Signed Operations and Mathematical](#_Toc477365546)

Don’t use shifting operation with the sign variables, also don’t expect the ordering of the operation always use the Braces ().

## Pointer Basic Rule

Always check the pointer before using it.

If (ptr != NULL\_PTR)

{

\*Ptr = 5 ;

}

## Local Variable:

Name should be unique in the function and not the same as to any global.

Use the same examples below but remove **g** Ex: au8\_BcmCounter.

## Global Variable

Naming should be unique and prefer to be descriptive as his scope, type and functionality

Ex1, gpstr\_BcmJobBuffer : when I see it I will know this’s global pointer struct for bcm job buffer.

Ex2, gau8\_BcmConfig : global array uint8

Ex3, gu8\_BcmTxStatus, gb\_Bcm, gstr\_BcmTxBuffer and so on

## Type Definition

Each type define should ended by \_t .

* Don’t use the native data type like int always re type them under compiler.h

Typedef unsigned short uint16\_t;

* When define a complex datatype always define as StrName\_t, EnuName\_t, UniName\_t.
* You can use gStrName\_t if this’s a global struct (user can use it), if not just keep it StrName\_t also for Enum and Union.
* If you need to define pointer datatype use PtrName\_t/gPtrName\_t.

NOTE: for the module typedef use the module name after the type like gStrBcmConfig\_t

## Inclusion Sequence

Standard include should done first then other module in sequence from MCAL to APP.

## Inclusions General Rule

* Don’t define any variable inside the header file.
* Don’t include any .c file.

## Code Template

You files should be structured as follows:

* Any “.h” file should be partitioned and ordered into the following sections:

|  |
| --- |
| #ifndef \_\_BCM\_H\_\_  #define \_\_BCM\_H\_\_  /\*- INCLUDES -----------------------------------------------\*/  /\*- CONSTANTS ----------------------------------------------\*/  /\*- PRIMITIVE TYPES ----------------------------------------\*/  /\*- ENUMS --------------------------------------------------\*/  /\*- STRUCTS AND UNIONS -------------------------------------\*/  /\*- FUNCTION-LIKE MACROS -----------------------------------\*/  /\*- FUNCTION DECLARATIONS ----------------------------------\*/  #endif /\*BCM.h\*/ |

|  |
| --- |
| /\*- INCLUDES ----------------------------------------------\*/  /\*- LOCAL MACROS ------------------------------------------\*/  /\*- LOCAL FUNCTIONS PROTOTYPES ----------------------------\*/  /\*- GLOBAL STATIC VARIABLES -------------------------------\*/  /\*- GLOBAL EXTERN VARIABLES -------------------------------\*/  /\*- LOCAL FUNCTIONS IMPLEMENTATION ------------------------\*/  /\*- APIs IMPLEMENTATION -----------------------------------\*/ |

* Any “.c” file should be partitioned and ordered into the following sections:

## Error Handling

* Each module should have list of errors which are declared already in the file “module \_retval.h”.
* This file, “module \_retval.h”, should be included in the final file “module.h” of the module.
* Errors should be formatted as the following example:

|  |
| --- |
| /\*\*  \* @brief: This return value indicate that the function could  \* run successfully and it returns no errors.  \*/  #define BCM\_SUCCESS ( 0 )  /\*\*  \* @brief: This return value indicate that the selected resource  \* not found in the platform.  \*/  #define BCM\_ERROR\_RESOURCE\_NOT\_FOUND ( -1 )  /\*\*  \* @brief: This return value indicate that the function tries  \* to use an uninitialized module.  \*/  #define BCM\_ERROR\_NOT\_INITIALIZED ( -2 ) |

## Explicit Casting is Mandatory

In assignment or mathematical formula use explicit casting with the constants to ensure they are stored in the correct format, giving the correct result.

## Files naming and Folder Structure

SprintW4

* App
  + Project1
    - App.c
  + Project2
    - App.c
  + And so on ….
* Firmware
  + BCM
    - Bcm.c
    - Bcm.h
    - Bcm\_PbCfg.h
    - Bcm\_PbCfg.c
  + TMU
    - Tmu.c
    - Tmu.h
    - Tmu\_Lcfg.c
    - Tmu\_Lcfg.h
    - Tmu\_Cfg.h
  + And so on ….
* Mcal
  + Timer
    - Timer.c
    - Timer.h
  + Uart
    - Uart.c
    - Uart.h
  + And so on ….
* Infrastructure
  + Compiler.h
  + Retval.h

And so on ….

## Code commenting and Doxgen Format