CTO RISC-V Firmware Coding Convention

Western Digital ® Version 1.0

Table of Contents

Revision History

Revision	*Date*	*Contents*	*Author(s)*
0.1	Dec 26, 2018	Initial revision	Ofer Shinaar
1.0	Aug 30, 2021	Converted form .docx to .adoc. Fixed minor typos.	Pasha Tikhonov

List of Figures

List of Tables

Reference Documents

Item #	Document	Revision Used	Comment

Abbreviations

Chapter 1. Overview

Firmware code convention is a guideline for how to syntaxes the firmware code in C and ASM languages.

The basic rule for the FW engineer is “keep the common ground syntaxes”, meaning, if you port your code and you have different convention then you don’t have to modify it to this convention as long as your code have common ground syntaxes.

Uniformity is the key for good, readable convention

Following conventions are for new code written for RISCV firmware and Tool chain, for C and ASM languages.

Chapter 2. C code convention

- All Prefixes/postfixes in lower case

2.1. Typedef

2.1.1. Typdef header

- All Defines & Macros must have header - One line header

2.1.2. Syntax

- Postfix ***'{type name}_t'*** in the end of new declaration

Ex:

```
typedef BaseType_t
```

2.2. Structures:

2.2.1. Structure header

- Must have header – One line header

2.2.2. Syntax

- Must have a name
- On structure usage the new var will have a prefix of ***'_t{newVarName}'***

Ex:

```
Typedef struct myStruct
{
} myStruct_t;
```

Usage:

```
myStruct_t tGrooveDscr
```

2.3. Enum

- Avoid enums usage as possible, so you will not depended on the compiler size translation.

2.3.1. Header

- All Enums must have header - One line header

2.3.2. Syntax

- Enum must have a meaningfull name and end with Prefix ***'e{name}'***

Ex:

```
Typedef enum myHugeEnum
{
} eMyHugeEnum_t;
```

2.4. Variables and Functions:

- Using "System Hungarian notation"

2.5. Vars

- After Prefix the Var name will start with capital letter.
- Variable after prefix will be with higher case

2.5.1. Var Syntax

- ***u{x} - unsigned*** + ***type*** {char, int, long, double, etc...}

Ex:

```
unsigned char ucMyunsigned
```

- ***{x} - signed type*** {char, int, long, double, etc...}

Ex:

```
char cMysigned
```

- ***{st} - struct type***

Ex:

```
myStruct_t stGrooveDscr
```

- ***{g_} – Global varibale***

Ex:

```
unsinged int g_uiMyglobal;
```

2.5.2. Pointers Syntax

- Prefix ***'p{pointer name}'*** and the type does not matter.

Ex:

```
void pMyPointer;
myStruct_t pGrooveDscr; //this is pointer with struct type
```

- Function pointers start with
  prefix ***'fptr{function name starts with the capital case}'***

Ex:

```
int myGrooveFunction(char);
int (* fptrMyPointerToGroove)(char) = myGrooveFunction;
```

2.5.3. Variables names

typedef signed char	*s8_t*
typedef signed short	*s16_t*
typedef signed int	*s32_t*
typedef signed long long	*s64_t*
typedef unsigned char	*u8_t*
typedef unsigned short	*u16_t*
typedef unsigned int	*u32_t*
typedef unsigned long long	*u64_t*

2.6. Defines & Macros

2.6.1. Defines & Macros header

- Must have header - One line header

2.6.2. Define Syntax

- All capital letters
- Define prefix ***'D_{define name}'***

Ex:

```
#define D_MY_DEFINE
```

2.6.3. Macro Syntax

- Marco prefix ***'M_{macro name}'***

Ex:

```
#define M_MY_MACRO(_X_,_Y_)
```

2.6.4. Long Marcos and define

- Should be in multi lines

Ex:

```
#define M_MY_MACRO(_X_,_Y_) \
{                           \
    Syntax                  \
}
```

2.7. Functions

2.7.1. Function Header

- All functions must have header - Function Header
- Function declaration should be in one line

2.7.2. Parenthesis

- Start in a new line

2.7.3. Function name

- Start with lower case
- Should be meaningful

Ex:

```
u32_t myGrooveFunction(void);


u32_t myGrooveFunction(void)
{
////
}
```

2.8. Get and Set functions

- Must have a postfix of Set/Get

Ex:

```
u32_t myGrooveFunctionGet();

u32_t myGrooveFunctionSet(u32_t myArg);
```

2.9. C files

2.9.1. C files header

- must start with header - C file header

2.9.2. C file name

- If the c file is a part of reprehensive module it should hold the module name as a prefix

Ex:

```
src\rtos\

rtos_mutex.c

rtos_sema.c



src\spi\

spi_api.c

spi_eng.c
```

2.10. Header files

2.10.1. Header

- All h files must start with header - C file header

2.10.2. Nesting

- Should ***avoid nesting*** as much as possible

2.10.3. Naming

- If the h file is a part of reprehensive module it should hold the module name as a prefix

Ex:

```
\rtos\inc

rtos_mutex.h

rtos_sema.h



spi\inc\

spi_api.h

spi_eng.h
```

2.11. Comments

- Try using C89 comments

```
/**/
```

- Try not mixing C comments in ASM files. In ASM files use ASM comments.

2.12. One line header

- Use the following for: Marcos, Defines, and sections.

```
/*
* your syntax here
*/
```

2.13. Sections

- Keep C files sections as is, even if its empty

```
/*
*   Include Files
*/

/*
*  Macro definitions
*/

/*
* Enumeration declarations (enum)
*/

/*
*  Type definitions
*/

/*
*  Structure declarations
*/

/*
*  External prototypes
*/

/*
*  Function prototypes
*/

/*
*  Global Variables
*/

/*
*  Globals
*/
```

- Header files will have the same except globals

2.14. C file header

```
/*
* Copyright (c) 2010-2016 Western Digital, Inc.
*
* SPDX-License-Identifier: Apache-2.0 (OS-TBD)
/

/
* @file file name
* @Author Author name
* @Created date Date creted
* @brief Short brief
*/
```

2.15. Function header

```
/*
* @brief Short brief
*
* More description if needed.
*
* @param param1
* @param param2
* @param paramN
*
*/
```