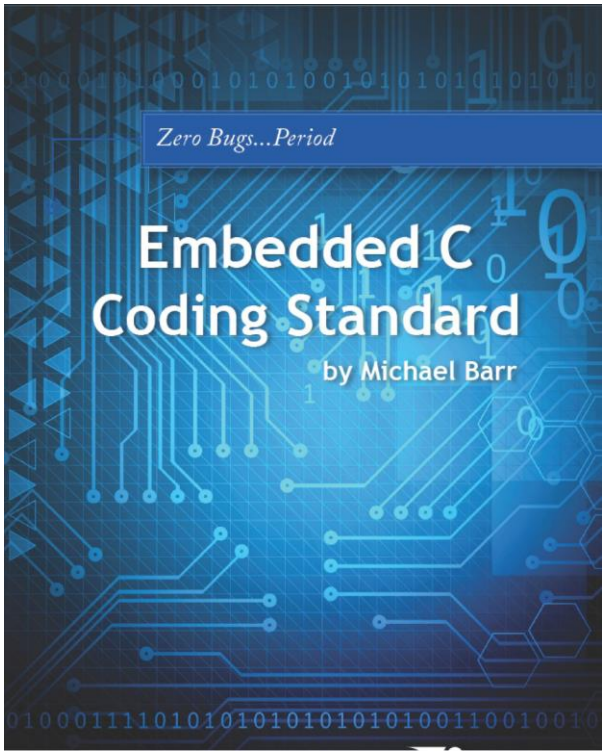
코딩 가이드라인



https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard

Braces

- 중괄호 사용
  - If, else, switch, while, do, for 문장 뒤에 반드시 사용
  - Single statement나 empty statement에도 반드시 사용
- 사용 방법
  - Left brace('{')는 블록이 시작하는 다음 줄에 '{' 만 표시
  - Right brace('}')는 블록이 끝나는 다음 줄에 '}' 만 표시

```
while (idx_loop == 0) ;
```

```
while (idx_loop == 0)
{
}
```

```
while (idx_loop == 0)
{
// empty statement
}
```

Parentheses

- C 언어의 연산자 우선 순위 사용 금지
  - 우선 순위가 명확하더라도 괄호를 사용

```
if ((speed_from_abs > 0) && (speed_from_abs < 255))
{
    vehicle_speed = speed_from_abs;
}
```

형변환 (Type Casting)

- 형변환을 사용할 경우 반드시 주석에 명시
  - 형변환은 위험한 작업이므로 이유에 대하여 명시를 해야 함
  - 코드 리뷰 과정에서 활용

```
uint16_t  capture_input = get_timer (TIMER0);
result = abs((int) capture_input);   // 32-bit int assumed.
```

키워드 *static*

- 규칙
  - 선언되어 있는 모듈 밖에서 사용될 필요 없는 함수나 변수에는 반드시 *static* 키워드를 사용
  - Cf: 전역변수와 정적변수

```
int convert_float_fix (float32_t num_float)
{
…
}
```

```
static int convert_float_fix (float32_t num_float)
{
…
}
```

키워드 *const*

- 규칙
  - 적절한 상황 마다 변수에는 반드시 *const* 키워드를 사용
    - 초기화 후 변경되지 않는 변수를 선언할 때

      ```
      double const pi = 3.141592;
      ```

    - 변경되지 않아야 하는 call-by-reference 함수의 매개변수를 정의할 때

      ```
      int example_fun(char const * param1, char * const param2);
      ```

    - 변경되지 않아야 하는 구조체와 공용체에 있는 필드를 정의할 때

    - 상수 정의를 위한 #define 의 대체로 사용할 때

      ```
      #define PI 3.141592
      const double pi = 3.141592;
      ```

키워드 *volatile*

- 규칙
  - 적절한 상황 마다 변수에는 반드시 *volatile* 키워드를 사용
    - Interrupt service routine에서 사용할 수 있는 전역변수를 선언할 때
    - 두 개 이상의 태스크에서 사용할 수 있는 전역변수를 선언할 때
      - Semaphore 등의 고려 필요
    - Memory-mapped I/O 주변장치의 레지스터의 포인터를 선언할 때
    - Delay loop counter를 선언할 때

주석

- 규칙
  - C++ 스타일의 Single-line comments // This is comment
  - 기존의 C 스타일의 주석 /* … */
  - 주석은 중첩되지 않아야 함 /* comment1 /* comment2 */ */
  - 주석은 일시적으로라도 코드의 블록을 disable하기 위해 사용해서는 안됨

```
/*
if (vehicle_speed > 30)
{
    abs_control_index = 0;
}
*/
```

```
#if 0
if (vehicle_speed > 30)
{
    abs_control_index = 0;
}
#endif
```

Blank Lines

- 규칙
  - 한 문장 이상이 코드의 한 줄에 포함되면 안됨

    vehicle_speed = 30, abs_speed = 30;

  - 블록 사이에 한 줄을 띄움
  - 소스 파일의 제일 마지막 줄에 비어있는 줄이 있어야 함

Indentation

- 규칙
  - 한 모듈 내에서 indentation은 4 space를 사용함
  - switch 문장에서 각각의 case 문은 들여쓰기를 해야 하며, case 블록의 내용은 한 번 더 들여쓰기가 되어야 함
  - 한 줄에 표시하기에 문장이 너무 길 경우 가능한 한 읽기 편한 방식으로 다음 줄을 들여쓰기 해야 함
  - Tab 사용 금지

```
sys_error_handler(int err)
{
    switch (err)
    {
        case ERR_THE_FIRST:
            …
        break;

        default:
            …
        break;
    }

    if ((first_very_long_compariaon_here
        && second_very_long_comparison_here)
        || third_very_long_compariaon_here)
    {
        …
    }
}
```

Naming Conventions

- 모든 모듈 (파일) 이름은 영문 소문자, 숫자, underscore (_)로 구성되어 있어야 하며 공백을 사용해서는 안됨

- 모든 모듈 이름은 첫 8자가 서로 달라야 하며 헤더 파일과 소스 파일에 각각 .h 와 .c 의 확장자를 가져야 함

- 모듈 이름은 표준 라이브러리 헤더 파일의 이름과 공유해서는 안됨 (ex: ‘stdio’, ‘math’)

- main() 함수를 포함하는 모듈은 파일의 이름에 “main” 단어를 포함해야 함

헤더 파일

- 규칙
  - 하나의 소스 파일을 위한 하나의 헤더 파일을 가져야 하며 파일 이름은 동일해야 함
  - 각 헤더 파일은 여러 번 포함 되는 것을 방지하기 위해 전처리기 가드를 포함해야 함

```
#ifndef _ADC_H
#define _ADC_H
...
#endif
```

  - 헤더 파일에는 다른 모듈에서 알아야 하는 프로시저, 상수, 데이터 타입 만을 기술해야 함
  - 헤더파일은 #include 문장을 포함하지 않아야 함

소스 파일

- 규칙
  - 각 소스 파일은 적절한 제어 단위의 동작만을 포함해야 함
  - 각 소스 파일의 구조는 다음의 섹션 등이 일관된 순서로 나타나야 함
    - 코멘트 블록, include 문장, 데이터 타입, 상수, 매크로 정의
    - 정적 데이터 선언, private function prototype, public function body, private function body
  - 각 소스 파일은 같은 이름의 헤더 파일을 include 해야 함
    - Public 함수의 원형과 바디의 매치
  - 절대 패스는 include 파일 이름에 사용하지 않음
  - 각 소스 파일은 사용하지 않는 include 파일을 쓰지 말아야 함
  - 소스 파일은 다른 소스 파일을 #include 해서는 안됨

파일 템플릿

- 규칙
  - 헤더 파일과 소스 파일에 대한 템플릿 모음은 프로젝트 단계에서 유지되어야 함

```
/** @file module.h
*
* @brief A description of the module's purpose.
*
* @par
* COPYRIGHT NOTICE: (c) 2014 Hyundai Motor Cor.
* All rights reserved.
*/

#ifndef  _MODULE_H
#define _MODULE_H

#ifdef __cplusplus
extern "C" {
#endif

int8_t max8(int8_t num1, in8_t num2);

#ifdef __cplusplus
}
#endif

#endif   /* _MODULE_H *?
```

```
/** @file module.c
*
* @brief A description of the module's purpose.
*
* @par
* COPYRIGHT NOTICE: (c) 2014 Hyundai Motor Cor.
* All rights reserved.
*/

#include "module.h"

/*!
* @brief Identify the larger of two 8-bit numbers
* @param[in] num1 The first number to be compared.
* @param[in] num2 The second number to be compared.
* @return int8_t
*/
int8_t
max8 (int8_t num1, int8_t num2)
{
    return ((num1 > num2) ? num1 : num2);
}
```

데이터 타입

- 규칙
  - 모든 새로운 데이터 타입의 이름은 소문자와 내부에 underscore로 이루어져 있고 '_t' 로 끝나야 함
  - 모든 새로운 구조체, 공용체, 열거형은 typedef를 통해 이름을 붙여야 함

```
typedef struct
{
    uint16_t  count;
    uint16_t  max_count;
    uint16_t  _unused;
    uint16_t  control;
} timer_t;
```

Fixed-Width Integers

- 규칙
  - 정수를 사용할 때 값의 범위에 따라 적절한 데이터 타입 char, short, int, long, longlong을 사용해야 함

Integer Width	Signed Type	Unsigned Type
8 bits	int8_t	uint8_t
16 bits	int16_t	uint16_t
32 bits	int32_t	uint32_t
64 bits	int64_t	uint64_t

[C99] stdint.h header file

  - 키워드 short와 long은 사용하지 않아야 하며 문자 연산을 제외하고 char을 사용하지 않아야 함

Signed Integers

- 규칙
  - Bit-field는 부호 있는 정수형으로 정의해서는 안됨
  - Bit-wise 연산 (&, |, ~, ^, <<, >>)은 부호 있는 정수 데이터를 연산해서는 안됨
  - 부호 있는 정수는 비교 또는 연산에서 부호 없는 정수와 결합 되어서는 안됨

```
uint8_t  a = 6u;
int8_t   b = -9;

if (a + b < 4)
{
    // This correct path should be executed
    // if -9 + 6 were -3 < 4, as anticipated.
}
else
{
    // This incorrect path is actually executed,
    // as -9 + 6 becomes (0xFF -9) + 6 = 252.
}
```

Floating Point

- 규칙
  - 가능한 한 부동 소수점 상수와 변수의 사용은 피함
    - 고정 소수점 연산으로 대체
  - 부동 소수점 연산이 필요할 경우
    - [C99] 의 float32_t, float64_t, float128_t 의 타입을 사용
    - Single-precision 상수에 'f'를 추가 (Ex: pi = 3.1415927f)
    - 컴파일러에서 double-precision을 지원하는 지 확인
    - 부동 소수점 값을 등호나 부등호로 테스트 하지 않음

```
//Ensure the compiler supports double-precision.
#include <limits.h>
#if (DBL_DIG < 10)
    #error “Double precision is not available!”
#endif
```

구조체와 공용체

- 규칙
  - 컴파일러가 삽입하는 패딩에 대한 주의가 필요함
  - Bit-field 내에서 bit의 순서를 컴파일러가 변경하지 않도록 주의가 필요함

```
struct {
    char a;
    int b;
    char c;
    short d;
}
```

주소

+0	pad	pad	pad	a
+4	b[31, 24]	b[23, 16]	b[15, 8]	b[7, 0]
+8	d[15, 8]	d[7, 0]	pad	pad

```
struct {
    char a;
    char c;
    short d;
    int b;
}
```

주소

+0	d[15, 8]	d[7, 0]	c	a
+4	b[31, 24]	b[23, 16]	b[15, 8]	b[7, 0]

Procedures – Naming Conventions

- 규칙
  - 프로시저의 이름에 C 언어의 키워드나 C 표준 라이브러리의 함수 이름과 겹쳐서는 안됨
  - 프로시저의 이름은 underscore로 시작해서는 안됨
  - 프로시저의 이름은 31자 보다 길면 안됨 (MISRA)
  - 함수의 이름은 대문자를 포함해서는 안됨
  - 매크로의 이름은 소문자를 포함해서는 안됨

Procedures – Naming Conventions

- 규칙
  - Underscore는 프로시저 이름에서 단어를 나눌 때 사용됨
  - 각 프로시저의 이름은 그 목적을 기술할 수 있게 작명해야 함
    - Ex. adc_read(), led_is_on()
  - 모든 public 함수의 이름은 그 모듈의 이름을 prefix로 가져야 함
    - Ex. force_read() in force.c file

함수

- 규칙
  - 각 함수의 길이는 50 ~ 100 라인을 유지하도록 함
    - 한 페이지에 프린트 가능함
  - 모든 함수는 가능한 한 프린트 한 페이지의 최상에 위치하도록 함
    - 한 페이지에 여러 작은 함수를 넣을 수 있는 경우 제외
  - 모든 함수는 하나의 exit point를 가져야 하며 함수의 최하단에 있어야 함
    - 즉, 키워드 *return*은 함수의 제일 마지막에 한번만 나타나야 함
  - Public 함수의 프로토타입은 모듈의 헤더 파일에 정의 되어야 함
  - 모든 private 함수는 *static*으로 정의 되어야 함
  - 각 매개변수는 명시적으로 선언되어야 하고 의미 있는 이름을 가져야 함

함수

```
int
state_change (int event)
{
int result = ERROR;
if (EVENT_A == event) {
result = STATE_A;
return (result);
} else {
result = STATE_B;
return (result);
}
}
```

```
int
state_change (int event)
{
    int result = ERROR;

    if (EVENT_A == event)
    {
        result = STATE_A;
        // Don’t return here.
    }
    else
    {
        result = STATE_B;
    }

    return (result);
}
```

Function-Like Macros

- 규칙
  - Inline 함수를 사용할 수 있는 경우 매개변수가 있는 매크로는 사용하지 않음
  - 만약 매개변수 있는 매크로를 사용해야 한다면
    - 괄호를 이용해 매크로 전체 바디를 감싸야 함
    - 괄호를 이용해 매개변수를 사용하는 모든 부분을 감싸야 함
    - 각 매개변수는 의도하지 않은 side effect를 피하기 위해 한 번 이상 사용하지 않음

```
// Don’t do this …
#define MAX(A, B)  ((A) > (B) ? (A) : (B))

// … if you can do this instead.
inline int max(int num1, int num2)
```

Tasks

- 규칙
  - 태스크를 의미하는 모든 함수는 "_task"로 이름이 끝나야 함

```
void
alarm_task (void * p_data)
{
    alarm_t    alarm = ALARM_NONE;
    int        err   = OS_NO_ERR;

    for (;;)
    {
        alarm = OSMboxPend(alarm_mbox, &err);
        // Process alarm here.
     }
}
```

Interrupt Service Routines

- 규칙
  - ISR은 일반적인 함수와는 다르기 때문에 #pragma 또는 "__interrupt" 와 같은 키워드를 통해 컴파일러에 알려줘야 함
  - ISR을 구현하는 모든 함수에 "_isr"로 끝나도록 이름을 부여해야 함
  - ISR은 다른 소프트웨어 파트에서 절대로 불리면 안되기 때문에 프로토타입을 선언하지 않고, static으로 선언하며, 관련 드라이버 모듈의 가장 마지막에 위치시킴
  - Default ISR은 처리되지 않는 인터럽트 소스를 위해 벡터 테이블에 인스톨되어야 함
  - ISR에서 전역 변수를 공유할 경우 태스크에서 그 전역변수를 보호해야 함
    - Ex. Semaphore, Mutex 등

Variables – Naming Conventions

- 규칙
  - C 또는 C++의 키워드를 이름으로 사용해서는 안되며 C 표준 라이브러리에 있는 이름을 사용해서도 안됨
  - Underscore로 시작하는 이름을 사용하면 안됨
  - 31자 보다 이름이 길면 안됨
  - 루프 카운터를 포함하여 3자 이하의 이름을 사용해서는 안됨
  - 변수 이름에 대문자를 포함해서는 안됨
  - 변수 이름에 숫자값을 포함해서는 안됨
  - 변수 이름에서 단어를 분리하기 위해 underscore를 사용해야 함

Variables – Naming Conventions

- 규칙
  - 각 변수의 이름은 사용 목적에 따라 기술해야 함
  - 모든 전역 변수의 이름은 'g' 문자로 시작해야 함 (Ex. g_zero_offset)
  - 모든 포인터 변수의 이름은 'p' 문자로 시작해야 함 (Ex. *p_led_reg)
  - 모든 pointer-to-pointer 변수의 이름은 'pp'로 시작해야 함
    - Ex. gpp_vector_table
  - Boolean 정보를 포함하는 모든 정수 변수는 'b' 문자로 시작해야 함
    - Ex. b_done_yet, gb_is_buffer_full

초기화

- 규칙
  - 모든 변수는 사용 전에 초기화해야 함
  - 변수는 필요할 때 정의하는 것이 좋음
    - C90에서는 모든 변수는 함수의 시작에서 정의해야 함

```
for (int loop = 0; loop < MAX_LOOPS; loop++)
{
    …
}
```

변수 선언

- 규칙
  - Comma (',') 연산자는 변수 선언에서 사용되지 않아야 함

```
char * x, y;    // Is y supposed to be a pointer?
```

If-Else 문

- 규칙
  - if 와 else if 에서 짧은 코드가 먼저 위치 해야 함
  - 중첩된 if-else 문장은 two-level 보다 깊으면 안됨
    - 복잡성을 줄이고 이해를 돕기 위해 함수 호출이나 switch문 사용
  - if 또는 else if 조건문에서 대입이 이뤄지면 안됨
  - else if 구를 갖고 있는 if 문은 else로 끝나야 함

```
if (NULL == p_object)
{
    result = ERR_NULL_PTR;
}
else if (p_object = malloc(sizeof(object_t)))    // NO!!!
{
    ...
}
else
{
    // Normal processing steps,
    // which require many lines of code.
}
```

Switch 문

- 규칙
  - 각 case의 *break*는 블록의 내용이 아니라 대응되는 *case* 와 align 되어야 함
  - 모든 *switch* 문은 *default* 블록을 포함해야 함

```
switch (err)
{
    case ERR_A:
        …
    break;

    case ERR_B:
        …
    break;

    default:
        …
    break;
}
```

Loop

- 규칙
  - *while* 또는 *for* 루프의 종료를 테스트하기 위해 숫자를 직접 사용하지 않아야 함
  - *for* 문의 루프 카운터를 초기화 하는 경우를 제외하고 루프 제어문에서 대입을 사용하면 안됨
  - 무한 루프는 "*for (;;)*"를 통해 구현해야 함
  - 비어진 바디를 갖고 있는 각 루프는 이유에 대하여 주석을 포함하여 중괄호를 사용해야 함

```
// Don't use a magic number ...
for (int row = 0; row < 100; row++)
{
    // ... when you mean a constant.
    for (int col = 0; col < MAX_COL; col++)
    {
        ...
    }
}
```

Unconditional Jumps

- 규칙
  - *goto*, *continue*, *break* 와 같은 무조건 분기는 사용하지 않아야 함

Equivalence Tests

- 규칙
  - 변수와 상수 값의 등호나 부등호 평가를 할 때 상수 값을 비교 연산자의 왼쪽 오퍼랜드로 사용해야 함

```
if (NULL == p_object)
{
    result = ERR_NULL_PTR;
}
else if (p_object = malloc(sizeof(object_t)))    // NO!!!
{
    …
}
else
{
    // Normal processing steps,
    // which require many lines of code.
}
```

Functional Overview

- Three mechanisms
  - Alive supervision (Heart-beat monitoring)
    - For supervision of timing of periodic software
  - Deadline monitoring
    - For aperiodic software
  - Logical monitoring (Control-flow monitoring)
    - For supervision of the correctness of the execution sequence