コーディング規約

Middleware of HAL

目的

コードスタイル、拡張性はもちろんのこと、テストのしやすさも現在は重要である。ここに定める規則はそれらを考えて作られたものである。また、組み込み系においてはリアルタイム性も非常に重要である。これは現在求められている制御のレベルが高いことが理由である。そこで以下の点に注意して作成された。

- 組み込み系本来の性能を殺さないこと
- リアルタイム性を重視し、あらゆる機能を拡張できるようにすること。

用語の説明

本規約にて以下は読みえ変えておくこと。

・Src は Source、ソースファイルのことを指す。

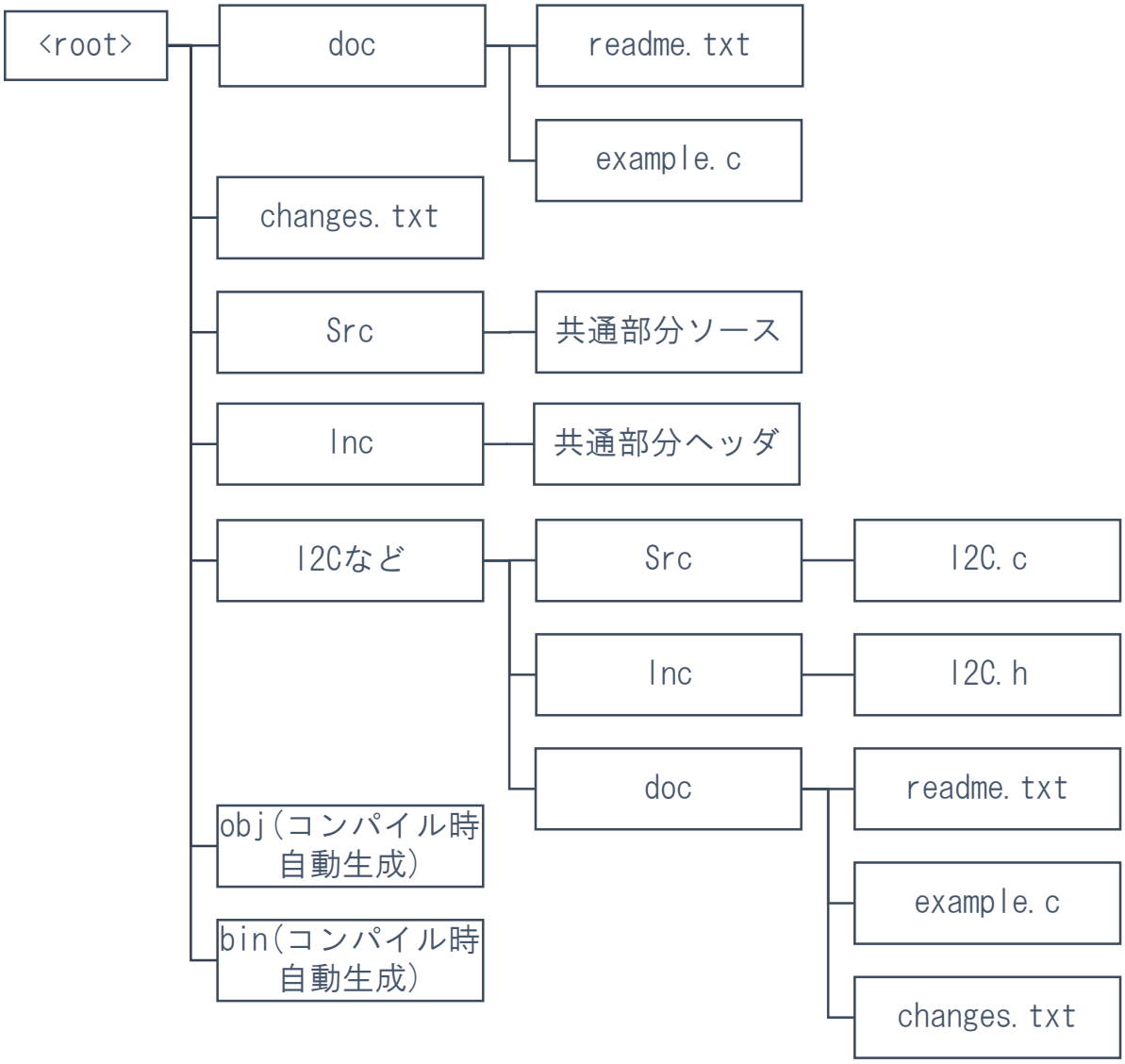
・Inc は Include、すなわちヘッダ等を指す。

・ワイルドカードを本文中でも用いる。*.c と書けば C ファイルのことを指す。

本プロジェクト構造

本プロジェクトの構造をいかに示す。作られるすべてのファイルは以下の構造で保存される。このうちコンパイルされるものは<root>もしくはサブディレクトリの<Src/>に存在する*.c が対象となる。これらは自動認識され、必要により再コンパイルされる。もちろんヘッダ等の依存関係も解決する必要がある。同様に<Inc/>に存在するファイルはすべて IncludePath として認識する。この IncludePath は自動的に加えられる。

Doc にはドキュメントを入れる。Readme.txt には基本的な使い方などを記述し、example.c も必要に応じて作成する。なお、サンプルコードは必ず検証コードとすること。changes.txt には変更点をまとめておく。



新規作成時のファイルヘッダ添付情報

新規にファイルを作成する場合、以下の情報を入れておくこと。

```
/* **KNCTRBKN Middleware of HAL.**
 * (c)2016 KNCT RBKN
 *
 * autor:****
 * version:
 * last update:
 *
 * **overview**
 * <概要>
 */
```

.c,.h のほかに、Makefile,readme.txt,changes.txt もこれに従う。
それに加えてヘッダファイルにはインクルードガードをつける。
#ifndef __<filename>_H
#define __<filename>_H
<ここに記述>
#endif

コーディングスタイル

コーディングスタイルは Uncrustify でフォーマットした結果に従う。ここでは特段指定はしない。

コード規約

コードを書く上でのコード規則をここに定義する。
・関数のプロトタイプ
関数のプロトタイプは関数に合わせて設定するが、引数は 4 個以上もちいらない。これは速度を殺し、性能を殺すことになるためであるが、構造体を用いることで回避できる。
また、通常返り値は成功、不成功を普通は返し、さらにコードでの例外処理は別途与えるものとする。

・関数の設計
関数はローコストである。ゆえに細かいところで関数を作成すべきである。その関数について幾つか注意点を上げる。

・再帰処理はもちいらない。ループに展開すべき
・いかなる処理も必ず終わるようにする。ただ例外処理やエラー処理はこれに限らない。
・細かい処理の中で速度が必要と思われるものや、ループ中で多く呼び出されるものは"inline"を使って展開すべきではあるが、いつも早いとは限らず引数が多い場合などにはかえってよくないこともあることに注意する。

・条件分岐
switch 文の中ではブロック文で括ることを推奨する。これは変数宣言も可能とするためである。このとき、ブロック文の最後に break を挿入する。default の例外処理も必ず入れること。但し同じ処理を回す場合は可読性が失われない範囲でこれには限らない。

```
switch(**){
case 1:
{
```

```
}break;
case 2:
{

}break;
default:
{

}break;
}
```

・変数の型
型は int 型を基本とする。高速に作動可能であるためである。但し大きな数(2^15-1~)を扱う場合は int32_t のように型を明確に示すこと。
int,unsigned 以外の short,long, 等の方の大きさが保証されないものは禁止する。int16_t,int64_t といったものを用いる。符号なしは特別理由がない場合は使用しない。

例
・ループカウンター
　int 型
・オーバーサンプリング処理(大きい数を用いる時)
　int32_t 型
・符号つきの 8bit 型
　int8_t 型

・定数
定数はできる限り const で示す。型をはっきりさせることができ、ミスも減るだろう。但し必ずこれに従うものではない。

・マクロ関数
マクロ関数は使用すべきではないが、組み込むことを明示的に示すものとしての使用は可能である。これは関数と同じ命名規則にのっとる。

・関数やグローバル変数の保護
外部に公開しないグローバル変数、関数は static を付加させ、必ず保護すること。

・禁止事項
goto は禁止。コードが読みにくくなるため。continue,break を用いること。
深いネストも禁止。関数に分けてネストの数を減らすこと。読みにくいため。

命名規則

いか関数名、型名、変数名、定数名について示す。

・関数名
関数名の最初に MX_を付加させておく。
関数名はキャメル記法で書く。

MX_analogRead

その上で略語が続いて読みづらいと判断した場合や統一性を目的としたアンダーバーの使用は可とする。

MX_UARTWriteViaDMA

関数名が短い場合は略語をもちいらない。

MX_UARTWriteByte
など複数の返り値を持つ、似たような関数も勿論返り値が複数考えられるようなものは最後に型を書いておく。
set,get などで変数の情報を得る場合はその変数名と関数名の一部を同じにする。

また失敗時にクラッシュするものは OrDie をつけて明示しておくこと。
MX_UARTInitOrDie

・変数名命名規則
変数名はアンダースコア[_]を区切りとし、命名の基本は関数名と同様 である。
特に bool を使用する場合は is-,can-,has-といったものを使うこと。基本的に loop は i で書き、i1,i2 のように書き足す。

```
int i;
bool has_started;
bool can_start;
```

基本的にグローバル変数自体使わない。ではあるがどうしても使う場合は g_を付加する。
外部に公開しないとき(static の付加時）は必ずではない。(後述)
int g_error_number;

いかに略語の例をあげる。
Init...Initialize
disp...display
num...number(must be integer
ch...charactor
by...byte
i...integer(loop counter に使用)