特設

第3章 メカニズム理解の仕上げ…
LED点滅のプログラムをブート部分から作る

起動プログラム作り…ブート・プログラム/リンカ・スクリプト

桑野 雅彦

LED点滅プログラムの概要

● ベアメタルのサンプルを流用しつつ

LEDの点滅プログラムをブート部分から作ってみます．特集はリアルタイムOSがテーマですので，LED点滅もタイマ割り込み（SysTick割り込み）を使うことにします．

さまざまなマイコンの起動部分からスクラッチで書き起こした（ベアメタルと呼ぶ）サンプルが以下のURLにあります．

`https://github.com/ataradov/mcu-starter-projects/tree/master/rp2040`

この中にRP2040も用意されています．開発環境はgccのツールチェーン（arm-none-eabi-gcc）とmakeがあればよいようになっています．

このベアメタルのサンプルも便利にできているのですが，単にブートからの動きを追うのには少々冗長なので，SysTickを使った必要最小限の機能に絞ります．なるべくあちこちのファイルを参照しなくても良いような，シンプルな表現にしてみました．

ソースコードは，セカンド・ステージ・ブートローダ部分の`startup.c`と，ユーザ・プログラム部分である`main.c`の2つです．それほど難しいものではないと思いますが，プログラムを読む上でポイントとなるところがありますので，ここで少し補足しておきます．

● ブートローダ（`startup.c`）

ファースト・ステージ・ブートローダによって読み込まれ，スタートする部分が`startup.c`で書いた部分です．先頭部分で**リスト1**のように割り込み処理関数が登録されています．

● 割り込み処理関数のプロトタイプ

リスト1の①で割り込み処理の関数をプロトタイプ宣言しています．アトリビュートに，`weak`と`alias`が指定されています．

▶`weak`：ラベルを2カ所で定義

`weak`というのは，ラベルが2カ所で定義された場合に備えたものです．ラベルが2カ所で定義された場合，通常は二重定義のエラーとなりますが，片方に`weak`属性が付いていると，`weak`側は定義されていないのと同じ扱いになり，`weak`が付いていない側のラベルが採用されます．

今回の例では，`isr_SysTick()`がLED点滅用の割り込み関数として`main.c`の中で宣言されていますので，`isr_SysTick`は2カ所で定義されることになりますが，`startup.c`の方が`weak`指定されていますので，エラーになることなく，`main.c`

リスト1　`startup.c`の割り込み処理登録部分（抜粋）

```
//割り込み処理関数のプロトタイプ宣言
#define DUMMY __attribute__ ((weak,
                              alias ("isr_dummy")))←①
DUMMY void isr_NMI(void);
DUMMY void isr_HardFault(void);
DUMMY void isr_SVCall(void);
DUMMY void isr_PendSV(void);
DUMMY void isr_SysTick(void);
/* 省略 */

//ベクタ・テーブルの登録
__attribute__ ((used, section(".vector_table")))
void (* const vector_table[])(void) =←②
{
 0,
 0,
 isr_NMI,
 isr_HardFault,
 /* 省略 */
 isr_SVCall,
 /* 省略 */
 isr_PendSV,
 isr_SysTick,
 /* 省略 */
};

//未登録の割り込みはここに来る
void isr_dummy(void)
{
  while (1)
    ;
}
```