第1章 main関数の代わりにLEDを点滅させるまで

起動処理のプログラムを作る

豊山 祐一

ここではTry Kernelの起動処理プログラムを作成していきます．ここで作成するプログラムのファイルは，特に指定のない限りTry Kernelのソースコードのbootディレクトリに置きます．

1 セカンド・ステージ・ブート・ローダ

セカンド・ステージ・ブート・ローダは，外部フラッシュ・メモリの先頭256バイトの領域に配置します．プログラムの処理内容もメモリ・サイズも決まっていますので，Pico公式のC/C++ SDKのものをそのまま使用します．

公式SDKの中にはセカンド・ステージ・ブート・ローダのアセンブリ言語によるソースコードが含まれています．ただし，プログラム領域の末尾にチェックサムを加えるために，アセンブリ言語によるソースコードではなく，バイナリのデータとしてC言語で記述します．

リスト1にセカンド・ステージ・ブート・ローダのC言語のソースコードを示します．UB型（符号なし8ビット整数）の配列boot2が，セカンド・ステージのバイナリ・データです．

このプログラムを特定のアドレスに配置するために，__attribute__によってセクション名を指定しています．セカンド・ステージ・ブート・ローダのセクション名はboot2です．

リスト1　セカンド・ステージ・ブート・ローダ（boot¥boot2.c）

```
const unsigned char boot2[]__attribute__((section(
  ".boot2"))) = {
0x00, 0xb5, 0x32, 0x4b, 0x21, 0x20, 0x58,
0x60, 0x98, 0x68, 0x02, 0x21, 0x88, 0x43,
0x98, 0x60, 0xd8, 0x60, 0x18, 0x61, 0x58,
/* 一部省略 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x74, 0xb2, 0x4e, 0x7a,
};
```

2 例外ベクタ・テーブル

外部フラッシュ・メモリのセカンド・ステージ・ブート・ローダの後ろに例外ベクタ・テーブルを配置します．

Arm Cortex-Mでは例外ハンドラをC言語の関数として記述できるので，ベクタ・テーブルはC言語で関数ポインタの配列として記述します．Try Kernelの例外ベクタ・テーブルを**リスト2**に示します．

ベクタ・テーブルの実体は関数ポインタの配列であり，セカンド・ステージ・ブート・ローダと同じように，特定のアドレスに配置するために，__attribute__によってセクション名を指定します．例外ベクタ・テーブルのセクション名はvectorです．

テーブルの1番目の要素INITIAL_SPはスタッ

リスト2　例外ベクタ・テーブル（boot¥vector_tbl.c）

```
void (* const vector_tbl[])() __attribute__((
                        section(".vector"))) = {
    (void(*)())(INITIAL_SP),    // 0: Top of Stack
    Reset_Handler,              // 1: Reset
    Default_Handler,            // 2: NMI
    Default_Handler,            // 3: Hard Fault
    0,                          // 4: 未使用
    0,                          // 5: 未使用
    0,                          // 6: 未使用
    0,                          // 7: 未使用
    0,                          // 8: 未使用
    0,                          // 9: 未使用
    0,                          // 10: 未使用
    Default_Handler,            // 11: Svcall
    0,                          // 12: 未使用
    0,                          // 13: 未使用
    Default_Handler,            // 14: Pend SV
    Default_Handler,            // 15: Systick
    Default_Handler,            // IRQ 0
    Default_Handler,            // IRQ 1
    Default_Handler,            // IRQ 2
            /* 一部省略 */
    Default_Handler,            // IRQ 31
};
```