第3章

マルチコア・プログラムを公式SDKを使わずにアセンブラから書き下ろす

Lチカと Hello World! を2コアで並列動作させてみた

中森 章

ラズベリー・パイPico（以降，Pico）のSoCであるRP2040の小さなチップの中には，2個のCPUコア（Arm Cortex-M0+）を内蔵しています．Picoを使うなら，2つのコアを同時に動かしてみましょう．

実際，ソフトウェアの開発キットである公式SDKを使用すれば，2個のコアを同時に動かすことは難しくありません．関数multi_core_launch_core1の引数として，もう一方のCPUコアが実行すべき関数のエントリを指定するだけです．

しかし，筆者の目的は，cmakeやnmakeを使う公式SDKを利用せず，もっと単純に，自前のプログラムをmakeでビルドしたいというものです．

そこで，注目したのがDavid Welch氏のサンプル・プログラム(1)です．これらのプログラムは少し変則的で，RAM上で実行するものですが，「単純でmakeを使う」という点で筆者の期待に合致しています．

公式SDKを利用しないプログラム作り

● 2コアを使うためにmulti_core_launch_core1関数を見てみる

RP2040に搭載されている2つのコアは，共有メモリ上にある，FIFO（メール・ボックス）で通信を行っています．**図1**(2)のような具合です．実際には，このFIFOは第2章で示したブロック図（図1）のSIO（シングル・サイクルI/O）ユニットの中にあります．

まず，RP2040の電源が立ち上がると，2つのコアは同時に動き始めます．ブートROMの中で，2つのコアは同一の命令列である次を実行します．

```
check_core:
      ldr r0, =SIO_BASE
      ldr r1, [r0, #SIO_CPUID_OFFSET]
      cmp r1, #0
      bne wait_for_vector
```

しかし，(SIO_BASE+SIO_CPUID_OFFSET)=0xd0000000というアドレスからは，CPUコア0は0を，CPUコア1は1をリードします．このため，次に書かれたCMP命令とBNE命令により，CPUコア0とCPUコア1の経路が分かれることになります．つまり，CPUコア0はBNE命令の次の命令から実行を継続し，CPUコア1はBNE命令の分岐先のwait_for_vectorに分岐します．

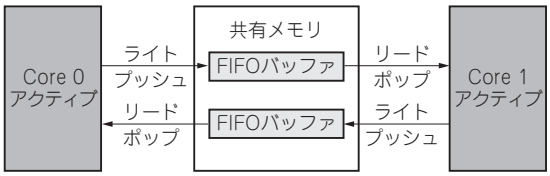

図1　2つのCPUコアは通信用FIFOを介してデータをやりとりする

その後，CPUコア0は，BOOT2ステージの処理やRAMの初期化などを行い，ユーザが書いたmainプログラムを実行します．CPUコア1は，スタンバイ状態になり，CPUコア0が起こしてくれるのを待ちます．

CPUコア1を眠りから起こすために，上述の通信用FIFOを使います．その手順はブートROMの中の命令列を見れば分かるのですが，何をやっているのかは，非常に分かりづらいです．そのような知識を持って，公式SDKのマルチコア部の関数のソースである，pico-sdk/src/rp2_common/pico_multicore/multicore.cを見れば，通信用FIFOに，「それらしい」値を書き込んでいる，**リスト1**の関数がmulti_core_launch_core1関数の実体であると推測できます．

この関数は，通信用FIFOに対して，次を書き込んでいるだけです．

- 0という定数値
- 0という定数値
- 1という定数値
- コア1が使用するベクタ・テーブルの先頭アドレス
- コア1が使用するスタック・ポインタ
- コア1が最初に実行する関数へのポインタ

定数0をFIFOに書き込む前には，受信用のFIFOを空にして，CPUコア1をSEV命令で起動するという操作がありますが，本質的ではないと思います．