第3章 演算能力を必要とする処理を並列実行してマルチコア化の効果を測定する

SMP方式「Try Kernel-S」を動かしてみる

豊山 祐一

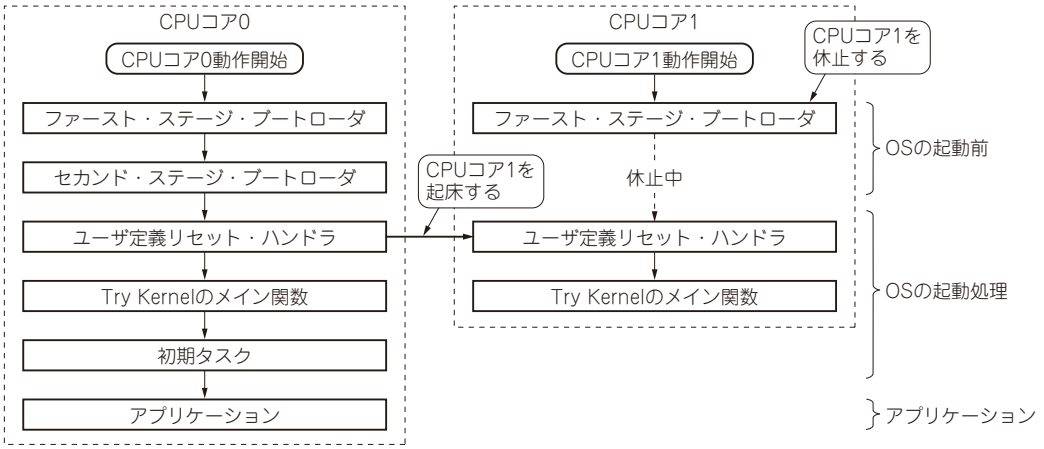
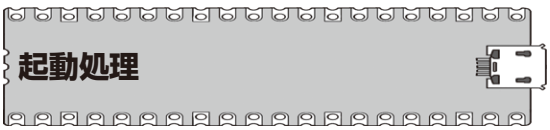


図1 Try Kernel-Sの起動処理の流れ

ここまで作成してきたSMP方式のリアルタイムOS Try Kernel-SをPicoで実行し，SMP方式のマルチコアの効果を検証します．



Try Kernel-Sの起動処理の流れを**図1**に示します．ユーザ定義リセット・ハンドラが実行されるまでは，AMP方式のTry Kernel-Aと同じです．ユーザ定義リセット・ハンドラでは，Try Kernel-Aの場合はそれぞれのCPUコアで初期タスクやアプリケーション・プログラムが実行されたのに対し，アプリケーション・プログラムがシステム全体で1つであるTry Kernel-Sでは，CPUコア0のみで実行します．

● ユーザ定義リセット・ハンドラ

CPUコア0用のユーザ定義リセット・ハンドラの処理のソースコードを**リスト1**に示します．基本的にはTry Kernel-Aと同じ処理です．マルチコア関連の処理として，CPUコア間スピンロックの初期化とCPUコア1の起床を行っています．

CPUコア1用のユーザ定義リセット・ハンドラのソースコードを**リスト2**に示します．Try Kernel-Aと比べると，システム・クロックの初期化処理がなくなっています．システム・クロックは，Try Kernel-Sとして1個であり，CPUコア0でのみ動作します．

● 各CPUコアのメイン関数

ユーザ定義リセット・ハンドラから，各CPUコアのメイン関数`main_c0`と`main_c1`が実行されます．メイン関数のソースコードを**リスト3**に示します．

どちらのCPUコアでも，CPUコア間割り込みの初期化と割り込みハンドラの登録が行われます．そして，CPUコア間のスピンロックを利用して，CPUコア間の起動処理の同期をとります．同期の方法はTry Kernel-Aと同じです．

ここまでで起動処理は終わり，CPUコア0は初期タスクを生成し実行します．一方，CPUコア1は実行するタスクが現れるまで無限ループに入って待ち続けます．

このCPUコア1の無限ループは，最初のタスクが実行された以降はもう実行されることはありません．その後は，ディスパッチャのアイドル・ループでタスクの実行待ちが処理されます．