特設

第5章 定番Lチカからスイッチ入力，ロータリ・エンコーダ用ドライバまで

ダウンロード・データあります

PIOプログラミング［事例集］

角 史生

MicroPythonを使ってステート・マシンを制御するプログラムの流れを**図1**に示します．プログラマブルI/O（PIO）命令セットを用いて，プログラマブルI/O用のソースコードを作成して，アセンブラによってプログラマブルI/O用の機械語に変換します．ステート・マシンに必要な情報を設定して，起動することで利用可能になります．

● ステート・マシンとStateMachineを使い分ける

MicroPythonの場合，PIO命令を機械語に変換するためのアセンブラpio_asmを利用可能です．また，1つのMicroPythonソースコード内に，プログラマブルI/O用ソースコードとMicroPythonソースコードをまとめることが可能です．以降の説明では，pio_asmによって変換された機械語列をバイナリ・コードと呼びます．

以降の説明で，ステート・マシンと記載する場合と，StateMachineと記載する場合とがあります．ステート・マシンの機能を述べる際やハードウェア・ブロックを示す場合はカタカナ表記を用い，MicroPythonにより定義されているStateMachineクラスやオブジェクトを述べる際は英語表記とします．

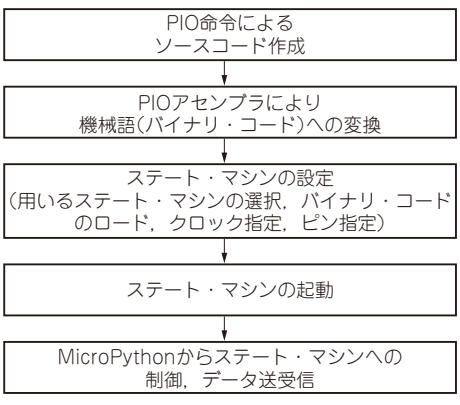

図1　ステート・マシン制御プログラムの流れ

また，プログラマブルI/Oは2系統あり，ステート・マシンは各プログラマブルI/Oに4台入っています．*N*番目のプログラマブルI/Oを指定する場合はPIO［*n*］と表記し，*N*番目のステート・マシンを示す場合はStateMachine［*n*］と表記します（*n*は0スタート）．

プログラム① NOP命令だけを実行する

PIOプログラミングを説明するに当たり，まず最も簡単なNOP命令だけを実行するプログラムを示します（**リスト1**）．

● 1．ステート・マシン用ソースコードの定義とアセンブル

nop()を指定してNOP命令だけによるプログラマブルI/O用ソースコードを定義します．

PIOアセンブラを起動するため，@rp2.asm_pio()とデコレータを設定します．これにより，def asm_nop():からの行はPIOアセンブラのソースコードとして扱われ，プログラマブルI/Oの機械語列（バイナリ・コード）に変換されます．変換されたバイナリ・コードは変数：nop_progに格納されます．REPLで変数nop_progを表示させると，変換されたバイナリ・コードの内容を確認できます．

リスト1　最も簡単なNOP命令だけを実行するプログラム

```
import rp2

# ステート・マシン用コードの定義とアセンブル
#------------------------------
@rp2.asm_pio()   # デコレータasm_pioを設定することで
def nop_prog():  # アセンブラ（asm_pio）を呼び出し
    nop()        # NOP命令だけからなるソースコード
#------------------------------

# ステート・マシンの設定とオブジェクト生成
#   PIO[0]のStateMachine[0]を指定，
                              バイナリ・コードはnop_progを指定
sm = rp2.StateMachine(0, nop_prog)

# ステート・マシンの起動
sm.active(1)
```