平成 24 年 6 月 27 日

# 実験 12. H8 マイコンを用いた入出力制御

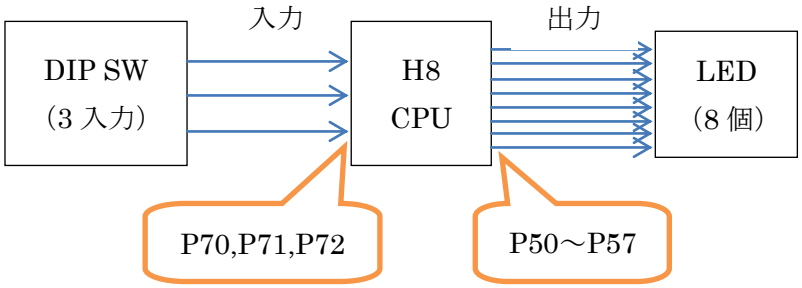
## １．目的

この実験では，ルネサステクノロジー社製 H8 CPU（以下，マイコンと呼ぶことにする）を用いて，DIP スイッチからの入力信号に応じて、LED の点灯を制御するプログラムを作成し、実験 11 と併せてマイコンによる入出力制御方法について学ぶ.

## ２．実験

### 2.1　概要

H8CPU のポート 7 （P70, P71, P72）に接続された DIP スイッチの 3 つの入力信号を読み取り、それらを 3 ビットの二進数と見なして、対応したポート 5 の各ビット（P50～P57）に接続された 8 個の LED を点灯させる。



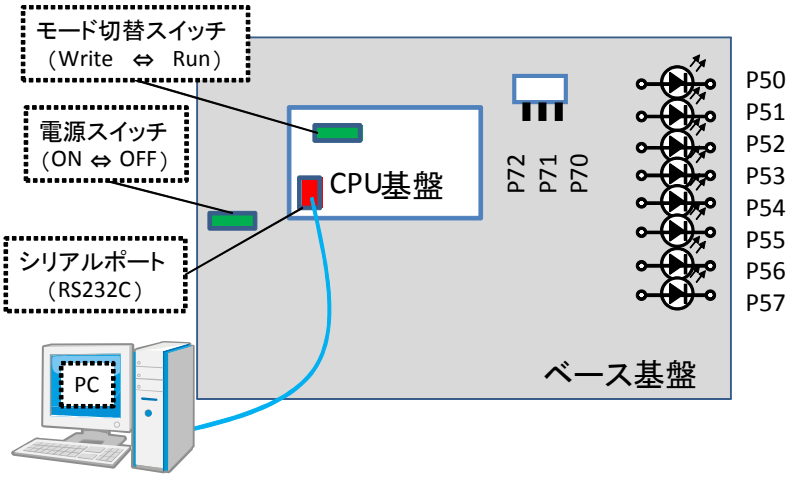




図　実験の入出力構成と配線

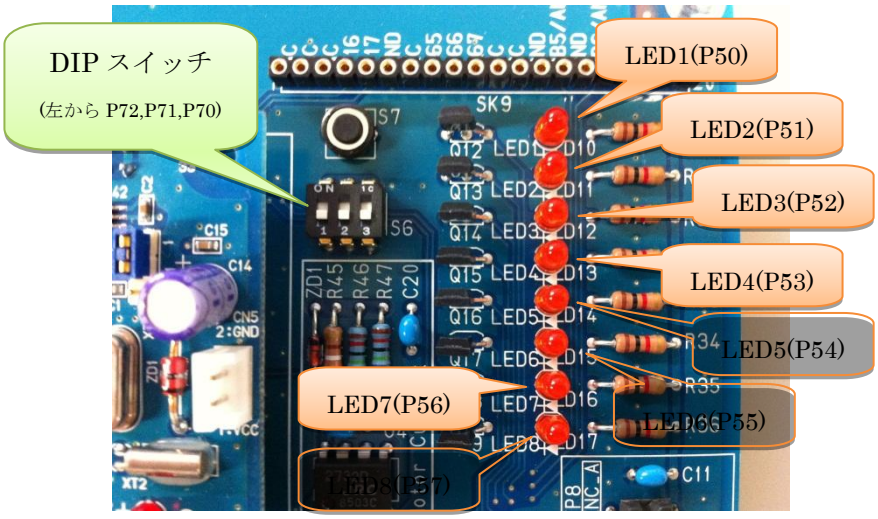



図　DIP スイッチ（Port7 に接続）と LED（Port5 に接続）

実験の流れを以下に示す.

A：新規プロジェクトの作成

新規プロジェクトを作成し，プログラミング環境を準備する.

B：プログラムの作成

C 言語で LED 点滅プログラムをコーディングする.

C：ビルド（コンパイル作業）

作成したソースファイルをコンパイルし，H8 CPU が理解できる形式に変換し，「実行ファイル」を生成する.

D：実行ファイルをマイコンに書き込む

PC と H8 マイコンを接続し，生成した実行ファイルをマイコンに書き込む.

E：実験（動作確認）

電源を入れ，プログラム通りに DIP スイッチの入力によって LED の点灯を制御できることを確認する.

これらの手順のうち、A～D は実験 10 および 11 と同じであるため、詳細は前回までの実験資料を参考にすること。

## 2.2　装置

### （1）　DIP スイッチ

今回の実験で使用するスイッチは、DIP スイッチと呼ばれるものであり、つまみをスライドさせることで、ON／OFF を切り替えることができる。図のようなスイッチの場合、1 つのつまみが、1 つのスイッチになっているため、4 ビットのスイッチとなる。e-nuvo BASIC のベース基盤に装着されているスイッチは、3 ビットのスイッチである。これらのスイッチ

はそれぞれマイコンのポート 7 のビット 0(P70)、ビット 1(P71)、ビット 2(P72)につながっており、これらのビットの入力信号を取得できれば、スイッチの ON／OFF 状態を判断することが可能となる。

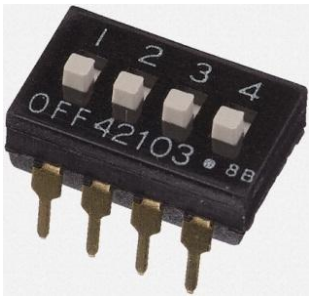



図　4bit-DIP スイッチ

（2）　LED

本実験では 8 個の LED を利用する。各 LED は、マイコンのポート 5 のビット 0（P50）からビット 7（P57）に接続されており、これらのビットへの出力を High／Low にすることで LED の点滅を制御することができる。

2.3　実験の手順

A. 新規プロジェクトの作成

デスクトップの「プログラミング数値処理関連」から「High-Performance Embedded Workshop」のショートカットをダブルクリックし、HEW を起動する。HEW の起動が完了したら、この実験用の新規プロジェクトを作成する。

- プロジェクトタイプ：Application
- ワークスペース名：DIPSW_TEST
- プロジェクト名：DIPSW_TEST
- ディレクトリ：z:¥DIPSW_TEST
- CPU 種別：H8S,H8/300（初期値のまま）
- ツールチェイン：Renesas H8S,H8/300 Standard（初期値のまま）

プロジェクト作成において，注意するところは以下の 2 か所である．それ以外は変更点はない.

（1）「1/9　CPU の選択」

CPU シリーズ　：　300H

CPU タイプ　　：　3687

（2）「1/4　標準ライブラリの選択」

「全て無効」ボタンをクリックして，ライブラリを外す.

プロジェクトが完了したら、前回までの実験同様に**ヘッダファイル「3687.h」**を実験用のフォルダ（DIPSW_TEST.c と同じディレクトリ）にコピーする。 3687.h は以下のディレクトリからコピーすること.

| Y:¥HEW によるプログラミング開発¥3687.h |
|---|

さらに、HEW の画面において、先にコピーしたヘッダファイル「3687.h」をプロジェクトに追加する作業も前回の実験と同様に行う（詳細は前回の資料を参照すること）。

B. プログラムの作成

HEW 上で、実験用のプログラムを作成する。大きく分けて、以下の処理を追加修正する。

（1） ヘッダファイル「3687.h」のインクルード
（2） 内部関数 init()の宣言、定義を追加
（3） 内部関数 DIP_SW()の宣言、定義を追加
（4） 内部関数 LED_ON()の宣言、定義を追加
（5） main 関数に、メインルーチンを記述

修正後のプログラム DIPSW_TEST.c を下記に示す。

```
#include "3687.h"

//関数のプロトタイプ宣言
void init(void);
int   DIP_SW(int bit_Num);
void LED_ON(int LED_Num);
void main(void);



//ポートの入出力を設定
void init(void)
{
        IO.PCR5 = 0xFF;//ポート５の入出力設定（LED 出力用）
        IO.PCR7 = 0x00; //ポート７の入出力設定（DIP スイッチ入力用）
}
```

```
//各 DIP スイッチの状態を取得
//引数：スイッチの番号（ビット 0～ビット 2）
//戻り値：スイッチの状態
int DIP_SW(int bit_Num)
{
	switch(bit_Num)
	{
		case 0: return(IO.PDR7.BIT.B0);
		case 1: return(IO.PDR7.BIT.B1);
		case 2: return(IO.PDR7.BIT.B2);
	}
}

//指定した LED を点灯
//引数：点灯させる LED の番号（LED1 から LED8）
void LED_ON(int LED_Num)
{
	switch(LED_Num)
	{
		case 1: IO.PDR5.BYTE = 0x01; break;
		case 2: IO.PDR5.BYTE = 0x02; break;
		case 3: IO.PDR5.BYTE = 0x04; break;
		case 4: IO.PDR5.BYTE = 0x08; break;
		case 5: IO.PDR5.BYTE = 0x10; break;
		case 6: IO.PDR5.BYTE = 0x20; break;
		case 7: IO.PDR5.BYTE = 0x40; break;
		case 8: IO.PDR5.BYTE = 0x80; break;
	}
}

//main 関数
void main(void)
{
	int LED_Num;//点灯させる LED の番号（1～8）
```

```
        //初期環境設定
        init();

        //メインループ
        while(1)
        {
                //DIP スイッチの状態を調べて、点灯させる LED を決める
                LED_Num = 1;
                if(DIP_SW(0) == 1) LED_Num = LED_Num + 1;
                if(DIP_SW(1) == 1) LED_Num = LED_Num + 2;
                if(DIP_SW(2) == 1) LED_Num = LED_Num + 4;

                //選ばれた LED を点灯
                LED_ON(LED_Num);
        }
}
```

D.　コンパイル

メニュー中の「ビルド」から「全てをビルド」を選択し、ビルドを行う。ビルドが成功すると、下図のように HEW 下部のステータス画面に「Build Finished. 0 Errors, 0 Warnings」と表示される。ビルドに成功すると、Debug フォルダの中に「DIPSW_TEST.mot」という実行ファイルが生成されるので、確認せよ。

E　プログラムをマイコンに書き込む

H8 CPU 用のバイナリプログラム「DIPSW_TEST.mot」ファイルが準備できたら、PC と CPU 基盤を接続し、書き込み作業を行う。CPU 基盤と PC をシリアルケーブルで接続し、FDT を使って、バイナリプログラムを書き込め。

F.　実験（動作確認）

接続がすべて完了したら，次の手順に従って動作確認を行う．

Step 1　ベースボードの電源を一度 OFF にする。

Step 2　CPU 基盤の DIP スイッチを「RUN」に変更する。

Step 3　準備が出来たら、ベース基盤の電源スイッチを ON にする。

Step 4　ベース基盤上の DIP スイッチに応じて、点灯する LED が変化すれば成功。

3.4　プログラムの解説

（１）init()関数

この関数では，P50~P57 を出力に設定，P70~P72 を入力に設定している．

```
//ポートの入出力を設定
void init(void)
{
        IO.PCR5 = 0xFF;//ポート５の入出力設定（LED 出力用）
        IO.PCR7 = 0x00; //ポート７の入出力設定（DIP スイッチ入力用）
}
```

ポートの入出力設定は，ポートコントロールレジスタ（PCR）を介して行う（実験 10 で説明済み）．P50~P57（Port5 の BIT0~BIT7）は LED への出力に使う．そのため，このプログラムでは, Port5 全体を出力ポートとして設定している（0xFF は 2 進数で 1111 1111）．

P70~P72（Port7 の BIT0~BIT2）は，DIP スイッチの入力に使う．そのため，このプログラムでは，Port7 全体を入力として設定している（0x00 は 2 進数で 0000 0000）．

（２）DIP_SW()関数

この関数では，指定した DIP スイッチの値（P70，P71，P72 の値）を取得している．

```
//各 DIP スイッチの状態を取得
//引数：スイッチの番号（ビット 0～ビット 2）
//戻り値：スイッチの状態
int DIP_SW(int bit_Num)
{
        switch(bit_Num)
        {
                case 0: return(IO.PDR7.BIT.B0);
                case 1: return(IO.PDR7.BIT.B1);
                case 2: return(IO.PDR7.BIT.B2);
        }
}
```

引数 bit_Num で指定した DIP スイッチの番号に応じて，３つの DIP スイッチの状態を 0（OFF）または１（ON）を返す．

３つの DIP スイッチは，H8 CPU から見ると，それぞれ Port7 の BIT0，BIT1，BIT2 に割り当てられている．ポートの値を参照するには，ポートデータレジスタ（PDR）を介して行う．本実験では Port7 を利用するため，PDR7 を用いることになる．

表　ポートデータレジスタ 7 （PDR7）の仕様

| ビット | ビット名 | 初期値 | 入力(R)／出力(W) | 説明 |
|---|---|---|---|---|
| 7 | － | 1 | － | このレジスタをリードすると，PCR7 がセットされているビットはこのレジスタの値が読みだされます．ビット 7 とビット 3 はリザーブビットです．リードすると，常に 1 が読みだされます． |
| 6 | P76 | 0 | R/W | |
| 5 | P75 | 0 | R/W | |
| 4 | P74 | 0 | R/W | |
| 3 | － | 1 | － | |
| 2 | P72 | 0 | R/W | |
| 1 | P71 | 0 | R/W | |
| 0 | P70 | 0 | R/W | |

（3）LED_ON()関数

この関数では，指定した番号に対応したピンに接続された LED を点灯する．

```
//指定した LED を点灯
//引数：点灯させる LED の番号（LED1 から LED8）
void LED_ON(int LED_Num)
{
        switch(LED_Num)
        {
                case 1: IO.PDR5.BYTE = 0x01; break;      //PDR5.BIT.B0=1
                case 2: IO.PDR5.BYTE = 0x02; break;      //PDR5.BIT.B1=1
                case 3: IO.PDR5.BYTE = 0x04; break;      //PDR5.BIT.B2=1
                case 4: IO.PDR5.BYTE = 0x08; break;      //PDR5.BIT.B3=1
                case 5: IO.PDR5.BYTE = 0x10; break;      //PDR5.BIT.B4=1
                case 6: IO.PDR5.BYTE = 0x20; break;      //PDR5.BIT.B5=1
                case 7: IO.PDR5.BYTE = 0x40; break;      //PDR5.BIT.B6=1
                case 8: IO.PDR5.BYTE = 0x80; break;      //PDR5.BIT.B7=1
        }
}
```

８つの LED は，H8 CPU から見ると，それぞれ Port5 の BIT0~BIT7 に割り当てられている．この関数ではビットごとに LED を ON(1)／OFF(0)を設定するのではなく，８つの LED すべてに対して「一括」で値を設定している．すなわち実験 10 の時は BIT 単位で設定していたが，今回は BYTE 単位での設定となっている．プログラム中の LED_Num に対する処理をまとめると，以下のようになる．

表　引数 LED_Num と Port5 からの出力の対応関係

| LED_Num | PDR5 | | Port5 の状態 | LED の状態 |
|---|---|---|---|---|
| | 16 進数 | 2 進数 | | |
| 1 | 0x01 | 0000 0001 | P50 のみ 1 | LED1 のみ点灯 |
| 2 | 0x02 | 0000 0010 | P51 のみ 1 | LED2 のみ点灯 |
| 3 | 0x04 | 0000 0100 | P52 のみ 1 | LED3 のみ点灯 |
| 4 | 0x08 | 0000 1000 | P53 のみ 1 | LED4 のみ点灯 |
| 5 | 0x10 | 0001 0000 | P54 のみ 1 | LED5 のみ点灯 |
| 6 | 0x20 | 0010 0000 | P55 のみ 1 | LED6 のみ点灯 |
| 7 | 0x40 | 0100 0000 | P56 のみ 1 | LED7 のみ点灯 |
| 8 | 0x80 | 1000 0000 | P57 のみ 1 | LED8 のみ点灯 |

4　課題

先のプログラムを参考に，次の２つの課題を行え．

課題１：下図の①～⑧の順番で，一定の周期で LED1 から LED8 まで点灯させる（図では右から左に明かりが動いているように見せる）プログラムを作成せよ．ただし，LED8 を点灯させた後は，再び最初の状態（LED1 のみ点灯）の状態に戻り処理を繰り返すようにせよ．なお，待ち時間は実験 11 のときに利用した sleep 関数を利用せよ（与える時間は１秒にせよ）．

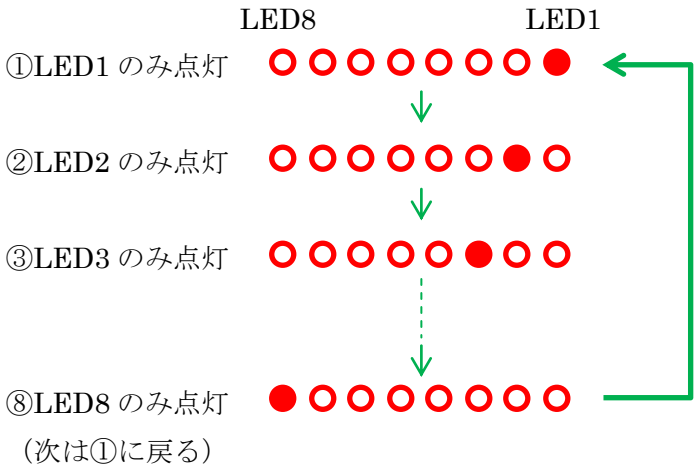



図　LED を順番に点灯させていく様子

課題２： 以下の対応表に従って，P70~P72 の入力に対応して LED1~LED8 を点灯させるプログラムを作成せよ．

図　Port7 の状態と，LED の点灯動作の対応関係

| P72 | P71 | P70 | LED の状態 |
|---|---|---|---|
| 0 | 0 | 0 | 全 LED 消灯 |
| 0 | 0 | 1 | 奇数番号の LED のみ点灯 |
| 0 | 1 | 0 | 偶数番号の LED のみ点灯 |
| 0 | 1 | 1 | LED1～LED4 のみ点灯 |
| 1 | 0 | 0 | LED5～LED8 のみ点灯 |
| 1 | 0 | 1 | 課題１のように LED を点灯 |
| 1 | 1 | 0 | 課題１での点灯順番を反対にして点灯 |
| 1 | 1 | 1 | 全 LED 点灯 |