

第19章 DMAでパルスモーターを制御する

パルスモーターを学習するのはこれで3回目です。最初は、パルスモーターを回転させるのにウェイト関数を使っていました。この方式ではパルスモーターが回転している間、CPUは無駄なループ文を実行しているだけで、関数から戻ってこない仕様でした。これではCPUは他の仕事ができないので、とても実用的ではありませんでした。

そこでパルスモーターの2回目では、タイマー割り込みを使用して改善しました。割り込み関数を実行していなければ、メインのルーチンでCPUに仕事をさせることができます。

しかし、いつでもこの方法が有効とは限りません。モーターの回転を上げようとすると、タイマー割り込みの周期が短くなります。そうすると、割り込み処理の時間が増えていき、CPUの負荷が増大します。最悪の場合は割り込み処理だけになってしまいます。

したがって、モーターの回転があまり速くない場合だけ有効な処理と言えます。

CPUに負荷をかけずに、モーターを回すことはできないのでしょうか？ マイコンにはCPU以外にも様々なハード的な機能が内蔵されていて、これを利用すれば可能です。

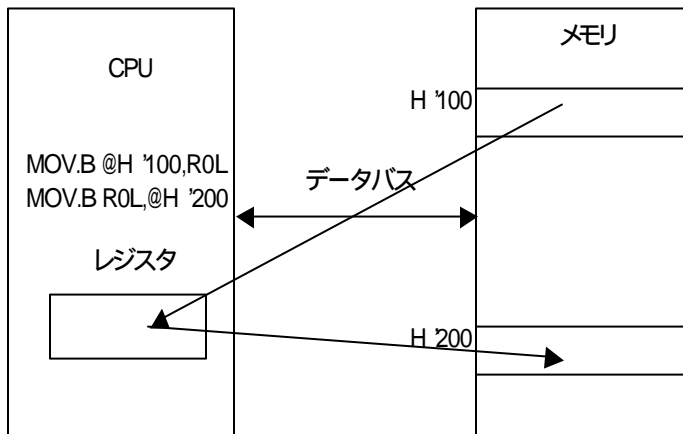
例えば、タイマーの機能にパルスを出力できる機能があります。これを使えば、タイマー機能がモーターを回し続けてくれますのでCPUは一切ノータッチです。

しかし、欠点もあって、モーターに接続するCPUの端子が特定されていることです。マイコンシステムには、モーター以外にも様々なデバイスが接続されているのが普通です。学習ボードを見れば分かるように、液晶やキースイッチなどいろいろなものを接続しなければなりません。そうすると、パルスモーターだけ最優先させて端子を割り当てるわけにもいかなくなります。（ちなみにパルスを出力できる端子には、オプションのDCモーターを接続します。パルスモーターよりDCモーターを優先させたわけです。）

そこで、CPUが内蔵しているDMAという機能を使うことにします。これを使えば、CPUに負荷をかけずにモーターを回すことができるのです。そこで、最初にDMAについて学習します。

DMAとは

DMAとはダイレクト・メモリ・アクセスの略で、CPUを介さずにメモリやIOレジスタ間の転送を行うものです。DMAを使わない、普通のメモリ間転送を見てみましょう。

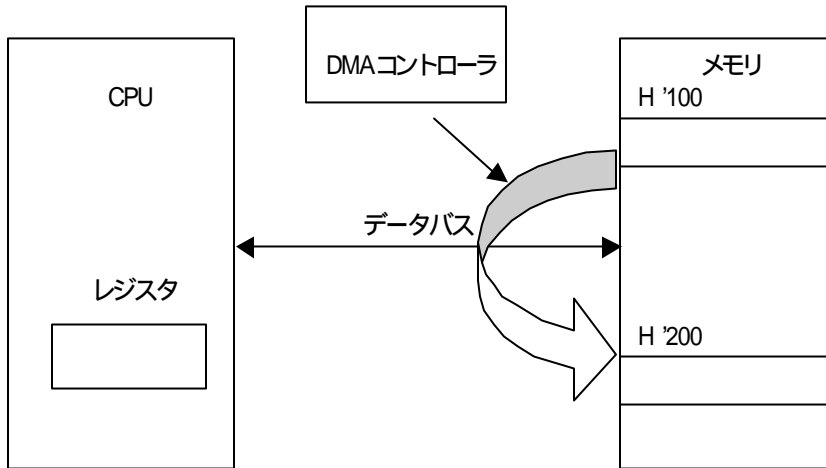


メモリ上のH'100番地のデータをH'200番地に転送する場合があります。この場合、CPUは以下の2命令を実行しなければなりません。

```
MOV.B  @H'100,R0L
MOV.B  R0L,@H'200
```

したがって、CPUに仕事をさせないと転送は起こりません。またデータの流れとしてはH'100番地のデータがデータバスを介して、一旦CPU内部のレジスタに格納され、そこからまたデータバスを介してH'200番地に格納されます。

DMAの場合は次のようになります。



マイコンに内蔵されたDMAコントローラが、メモリからメモリへ直接データを転送します。CPUは一切ノータッチで何の命令も実行しません。

ただし、この転送でもデータバスが使われますので、一瞬ですがバスが占有された状態になりますので、CPUはその間メモリアクセスができなくなります。しかし、まったく命令は実行されませんので通常の場合に比べてはるかにCPUの負荷は減ります。

メモリで説明しましたが、IOでも同じことが言えます。メモリからIOへ、IOからメモリへ、IOからIOへの転送も可能です。

DMAコントローラについて

H8/3048Fマイコンに内蔵されたDMAコントローラの使い方について説明します。このDMAコントローラには、ショートアドレスモードと、フルアドレスモードがあります。

ショートアドレスモード

転送元、転送先の番地の一方がショートアドレス、一方がフルアドレス

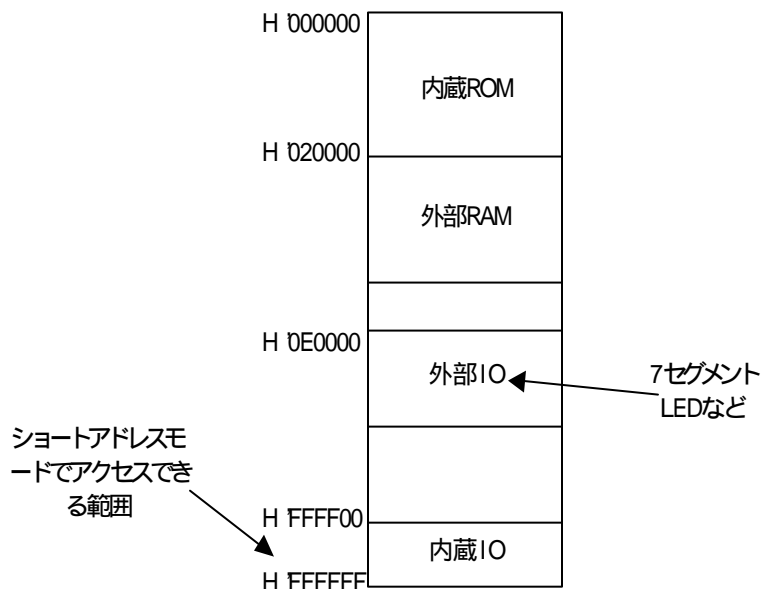
フルアドレスモード

転送元、転送先の番地が両方ともフルアドレス

ショートアドレスとは、アドレスを8ビットで指定することです。上位のビットは自動的にFFFFになります。例えば、番地H'D3を指定したとすると自動的にH'FFFFD3を指定したこと

になります。つまりショートアドレスで指定できる番地の範囲はH'FFFF00~H'FFFFFFとなります。

この範囲は、通常マイコン内蔵のI/Oレジスタの範囲になります。つまり、ショートアドレスモードは、一方が、内蔵I/Oレジスタの場合に有効なモードと言えます。



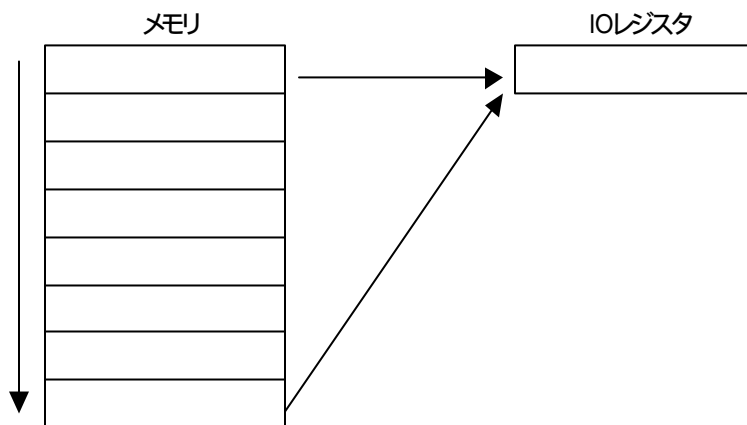
これに対して、フルアドレスモードは番地を全桁指定しますので、メモリ、内蔵I/Oの区別なく自由に設定できるモードと言えます。

今回は、転送先は、モーターが繋がれているポートのデータレジスタPADRであるので、ショートアドレスモードを使うことにします。

ショートアドレスモードにはさらに3つの動作モードがあります。

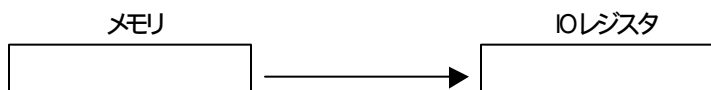
I/Oモード

このモードは、メモリ上に並んだデータ列を順次、指定のI/Oレジスタに転送します。



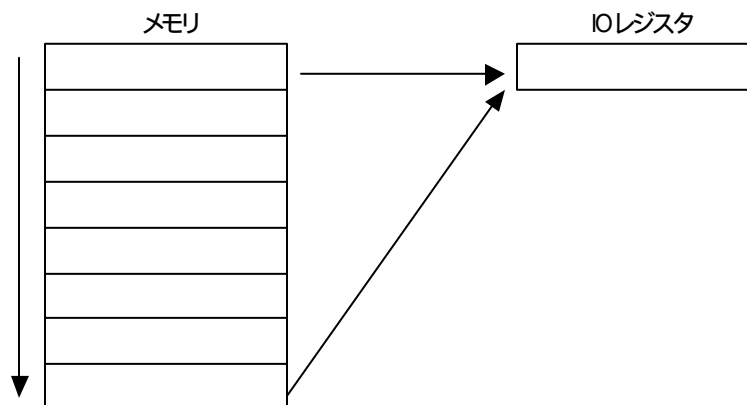
アイドルモード

このモードは、メモリ上の指定番地のデータを指定回数だけ指定のIOレジスタに転送します。同じデータを何度も出力する場合などに有効です。



リピートモード

このモードは、I/Oモードと同じですが、指定回数転送したら再度、同じ事を繰り返します。意図的に止めるまで永遠に繰り返します。



パルスモーターを回す場合、4つのデータH'48、H'60、H'A0、H'88を順々に出力すれば良いのと、意図的に止めるまで回し続けますのでリピートモードが最適です。今回はリピートモードを使用します。

DMAの起動要因

DMA転送のタイミングは、どのように決まるのでしょうか？ 実は、これは割り込みと似ています。割り込み要因がDMAの転送要因になるのです。

ただし、すべての割り込みが許されているわけではなく、許されているのは、16ビットタイマーの割り込み、あるいはシリアル受信割り込み、その他です。

割り込みと同じ手順でDMAを起動させるのです。例えば、今まで何度か出てきた10ms間隔の割り込みを思い出して下さい。通常ですと10ms間隔で割り込み要求が出て、割り込み関数が実行されます。

しかし、DMAの転送設定をしていると、10ms間隔で割り込み要求が出るのは同じです。しかし、割り込み関数が実行されるのではなく、DMA転送が起こるのです。

「割り込み関数の実行」が「一回のDMA転送」に置き換わったと思えば分かりやすいでしょう。

DMAコントローラの制御方法

DMAコントローラは、マイコン内蔵の機能ですからI/Oレジスタを使って制御します。したがって、I/Oレジスタの使い方を説明します。なお、DMAコントローラは2チャンネル内蔵されています。チャンネル0と、チャンネル1です。また、それぞれAとBのグループがあります。今回はチャンネル0のグループAを使いますので、レジスタ名称の最後に必ず0Aがつきます。

MAR0Aレジスタ

このレジスタは32ビットのレジスタで、転送元の番地を格納しておくレジスタです。(ハードウェアマニュアルでは8ビットレジスタ4本として説明されていますが、32ビットレジスタ一本と考えて問題ない)

このレジスタは32ビットレジスタですから#defineで番地を定義する場合、以下のようにcharではなくlongで定義することに注意して下さい。

```
#define MAR0A (*(volatile unsigned long *)0xFFFF20)
```

さて、今回は4つのデータH'48、H'60、H'A0、H'88を繰り返し、PADRに出力するわけですから、次のような配列を定義します。

```
char cw_data[] = {0x48, 0x60, 0xA0, 0x88};      //正回転
char ccw_data[] = {0x88, 0xA0, 0x60, 0x48};    //逆回転
```

転送元の番地は、この配列の先頭番地に他なりません。したがって、次のようにMAR0Aレジスタを初期化します。

```
MAR0A = (unsigned long)cw_data;      //正回転の場合
MAR0A = (unsigned long)ccw_data;    //逆回転の場合
```

配列の先頭番地はポインタ型です。一方レジスタはunsigned long型ですので、(unsigned long)でキャスト(強制型変換)しています。

IOAR0Aレジスタ

このレジスタは8ビットのレジスタで、転送先の番地を格納します。今回、転送先はPADRなので、PADRの番地を代入します。

```
IOAR0A = (unsigned char)&PADR;
```

8ビットレジスタへの代入なので(unsigned char)でキャストしています。この場合、上位のビットは切り捨てられます。PADRの番地はFFFFD3ですから、実際にIOAR0Aレジスタに格納されるのはD3だけです。しかし、DMAコントローラは上位のビットはFFFFと解釈しますので何も問題はありません。

ETCR0Aレジスタ

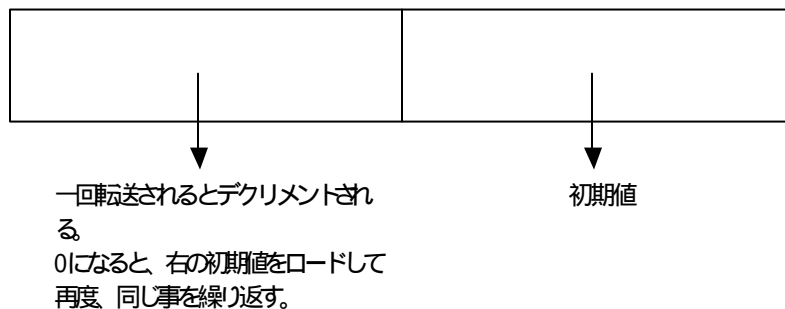
このレジスタは16ビットレジスタで、繰り返しの回数を指定するレジスタです。16ビットレジスタですから、#defineで番地定義する場合はcharではなくshortであることに注意して下さい。

```
#define ETCR0A (*(volatile unsigned short *)0xFFFF24)
```

このレジスタに繰り返し回数を設定しておく、一回転送されるたびにデクリメントされ0になると転送が終了します。16ビットレジスタなので、繰り返し回数の最大値は65536です。

なお、今回利用するリピートモードでは、このレジスタは上位8ビットと下位8ビットに分かれ、上位8ビットが繰り返しカウンタになります。そして下位8ビットが繰り返しカウンタの初期値になります。

ETCR0A



今回、4つのデータを繰り返し出力するので、繰り返し回数は4になります。このレジスタに0x0404を代入します。

```
ETCR0A = 0x0404;
```

DTCR0Aレジスタ

このレジスタは、DMA転送の様々な制御設定をするレジスタです。

DTCR

ビット	7	6	5	4	3	2	1	0
	DTE	DTSZ	DTID	RPE	DTIE	DTS2	DTS1	DTS0
初期値	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ビット7	説明
DTE	
0	データ転送禁止。
1	データ転送許可

ビット6	説明
DTSZ	
0	バイトサイズ転送
1	ワードサイズ転送

ビット5	説明
DTID	
0	データ転送後MARQAをインクリメント
1	データ転送後MARQAをデクリメント

ビット4	ビット3	説明
RPE	DTIE	
0	0	I/Oモードで転送(DTIEによる割り込み要求を禁止)
	1	I/Oモードで転送(DTIEによる割り込み要求を許可)
1	0	リピートモードで転送
	1	アイドルモードで転送

ビット2	ビット1	ビット0	説明
DTS2	DTS1	DTS0	
0	0	0	ITU0のコンペアマッチ割り込みで起動
		1	ITU1のコンペアマッチ割り込みで起動
	1	0	ITL2のコンペアマッチ割り込みで起動
		1	ITL3のコンペアマッチ割り込みで起動
1	0	0	SCIOの送信割り込みで起動
		1	SCIOの受信割り込みで起動
	1	0	DREQ端子の立ち下がりがリッジで起動
		1	DREQ端子のLOWレベル入力で起動

まず、ビット7のDTEビットから見て行きましょう。このビットを1にするとデータ転送が許可されます。すなわち、設定した割り込み要求があると転送が始まります。

当然1に設定します。

ビット6のDTSZビットは、転送がバイトサイズかワードサイズかの選択をします。バイトサイズなので初期値のままです。

ビット5のDTIDビットは、転送ごとにMAR0Aレジスタをインクリメントするか、デクリメントするか選択します。デクリメントに設定すると、データ列を後ろから順に出力することができます。今回のプログラムではインクリメントなので初期値のままです。

ビット4、ビット3のRPE、DTIEビットで転送モードを指定します。今回はリピートモードなので、RPE:DTIE=1:0に設定します。

ビット2～ビット0のDTS2～DTS0ビットで、DMA転送の起動要因となる割り込みを選択します。タイマーチャンネル3 (ITU3)を使用しますので0:1:1に設定します。

以上、まとめるとDTCRレジスタに設定する値は0x93ということになります。

ただし、ハードウェアマニュアルのビット7(DTEビット)のところの説明を良く読むと次のようなことが書かれています。

「...DTEビットは、DTE=0の状態をリードした後、1をライトしたときに1にセットされません。」

こういったハードウェアマニュアルの些細な記述も見逃してはいけません。一度値を読んだ後に、書き込みをしなければなりません。これは、タイマー割り込みのところの説明した割り込み要求フラグと同じです。値を読み込んだ後に書き込みする場合は、演算代入(|=または&=)を使えば良いのでした。したがって、

```
DTCR0A |= 0x93;
```

となります。

以上で基本事項の説明は終わりです。サンプルプログラムに移ります。

プログラムの実行方法

プロジェクトは[PulseMotor3]フォルダの[PulseMotor3.yip]です。
このプログラムを実行すると、モーターが正方向に回転し続けます。
テンキーから

- [7]を入力するとモーターは停止します。
- [8]を入力するとモーターは正方向に回転します。
- [9]を入力するとモーターは逆方向に回転します。

サンプルプログラム解説

基本的な構造は、PulseMotor2のサンプルプログラムと同じです。PulseMotor2では10ms秒ごとにタイマー割り込み関数を実行して、割り込み関数内でパルスの出力をしていました。

このサンプルプログラムでは、割り込み関数はまったく必要ありません。したがって、PulseMotor2から割り込み関数を削除し、以下に説明する関数を追加します。

```
/* DMA */
#define MAROA (*(volatile unsigned long *)0xFFFF20)
#define ETCROA (*(volatile unsigned short *)0xFFFF24)
#define IOAROA (*(volatile unsigned char *)0xFFFF26)
#define DTCROA (*(volatile unsigned char *)0xFFFF27)

/* パルデータの配列 DMAで転送する */
char cw[] = {0x48, 0x60, 0xA0, 0x88}; //正回転データ
char ccw[] = {0x88, 0xA0, 0x60, 0x48}; //逆回転データ
```

DMAコントローラのレジスタの番地定義と正回転、逆回転のデータ列を配列で定義します。

```

/*****
*****   パルスモータをDMAで正方向に回転させる           *****
*****           引数   なし                               *****
*****           戻り値 なし                               *****
*****/
void cw_pulse_motor(void)
{
    MAROA = (unsigned long)cw;      //転送元アドレス設定
    IOAROA = (unsigned char)&PADR; //転送先アドレス設定
    ETCROA = 0x0404;               //繰り返し回数設定
    DTCROA |= 0x93;                //リピートモード、ITU0割り込みで起動、DMA転送許可
    TSTR |= 8;                     //タイマカウンタスタート
}

```

この関数は、パルスモータを正方向に回転させる関数です。DMAコントローラの設定をした後に、タイマカウンタをスタートさせています。

```

/*****
*****   パルスモータをDMAで逆方向に回転させる           *****
*****           引数   なし                               *****
*****           戻り値 なし                               *****
*****/
void ccw_pulse_motor(void)
{
    MAROA = (unsigned long)ccw;    //転送元アドレス設定
    IOAROA = (unsigned char)&PADR; //転送先アドレス設定
    ETCROA = 0x0404;               //繰り返し回数設定
    DTCROA |= 0x93;                //リピートモード、ITU0割り込みで起動、DMA転送許可
    TSTR |= 8;                     //タイマカウンタスタート
}

```

この関数は、パルスモータを逆方向に回転させる関数です。cwがccwになっただけで正方向の場合と同じです。

```

/*****
*****   パルスモータを停止させる                               *****
*****           引数   なし                               *****
*****           戻り値 なし                               *****
*****/
void stop_pulse_motor(void)
{
    TSTR &= ~8;           //タイマーカウンタをストップ
    TSR3 &= ~1;           //割り込み要求フラグをクリア
    DTCROA = 0;           //DMA転送を禁止
}

```

この関数は、パルスモータを停止させる関数です。最初にTSTR &= ~8でタイマーカウンタを停止させます。次に割り込み要求フラグをクリアします。最後にDMA転送を禁止にします。

割り込み要求フラグのクリアが必要なわけは次のとおりです。割り込み要求は10ms間隔ごとに発生しています。しかし、DMA転送が許可されているので割り込みは発生しません。代わりにDMA転送が実行されるのです。DMA転送を禁止にした場合、その制約がなくなるので、割り込みが発生してしまいます。

割り込みが発生してしまうと、割り込み関数がないので、また割り込みベクタの登録もしていないので、プログラムは暴走してしまいます。

「割り込み関数がないから、割り込みは発生しない」と考えてはいけません。CPUは割り込み関数があるうがなかろうが（そんなことはCPUには分かりません）、割り込み要求があったら、ベクタテーブルから番地を読み出して、そこへジャンプしようとします。割り込みベクタの登録も割り込み関数の記述もないわけですから、デタラメな番地にジャンプして、プログラムが暴走することになります。

それをさけるために、割り込み要求フラグのクリアをしているのです。

```

/*****
*****   パルスモーターテストプログラム3           *****
*****/
void main(void)
{
    puts("パルスモーターテストプログラム3");

    init_timer_int();           //タイマー割り込み初期化
    init_pulse_motor();        //パルスモーター初期化
    cw_pulse_motor();          //モータを正方向に回転

    while (1) {
        char c;
        c = key_getc();         //キー入力待ち
        if (c == '7') {
            //停止
            stop_pulse_motor();
        }
        else if (c == '8') {
            //正方向
            stop_pulse_motor();
            cw_pulse_motor();
        }
        else if (c == '9') {
            //逆方向
            stop_pulse_motor();
            ccw_pulse_motor();
        }
    }
}

```

最後にメイン関数です。PulseMotor2の場合と違って、前記で説明した関数を呼び出すように変更してあります。

課題

サンプルプログラムではモーターはずっと回り続けます。これを修正して、一回転したら止まるように変更せよ。ただし、DMAコントローラのI/Oモードを使え。

注意

誤ってパルスモーターに異常なパルスを与え続ける状態が続くとモーターが発熱して破損する恐れがあります。もし、プログラム実行中にモーターが異常な動作をしたらすぐ電源を切るようにして下さい。

ヒント

次のデータ配列を利用する

これは0x48,0x60,0xA0,0x88を12個並べた配列である。

char

```
cw[]={0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88,0x48,0x60,0xA0,0x88};
```

繰り返し回数レジスタETCROAに繰り返し回数48を設定する。ETCROA=48;

DTCROレジスタの設定を変更してI/Oモードにする。