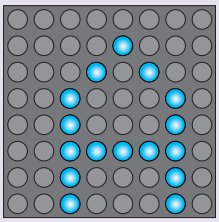
26 STEP

タイマ割込み

STEP25 では、外部割込みの 1 つであるキー入力割込みを学びました。
この STEP では内部割込みの 1 つであるタイマー割込みについて実習します。

課題　26-1

コンペアマッチによる割込みで LED ドットマトリクスに
A をダイナミック点灯させる。



STEP 26 タイマ割込み

タイマ割込みには、STEP22 で軽く触れた ITU（インテグレーテッドタイマユニット）のコンペアマッチを使います。
コンペアマッチとは、あらかじめ好きなカウント値を設定しておき、そのカウント値とタイマカウンタの値をコンペア（比較）してマッチ（一致）したら割込みを発生させたり、レジスタのフラグを立てたりすることを言います。
カウント値を設定を行うレジスタが STEP22 でも名前が出てきた、ジェネラルレジスタです。

26.1 ジェネラルレジスタ【 GRA, GRB 】

GR(ジェネラルレジスタ)は 16 ビットのレジスタで各チャネルに 2 本(GRA、GRB)あります。
ジェネラルレジスタの値は常にタイマカウンタと比較され、このジェネラルレジスタの値とタイマカウンタの値が一致することをコンペアマッチと言います。

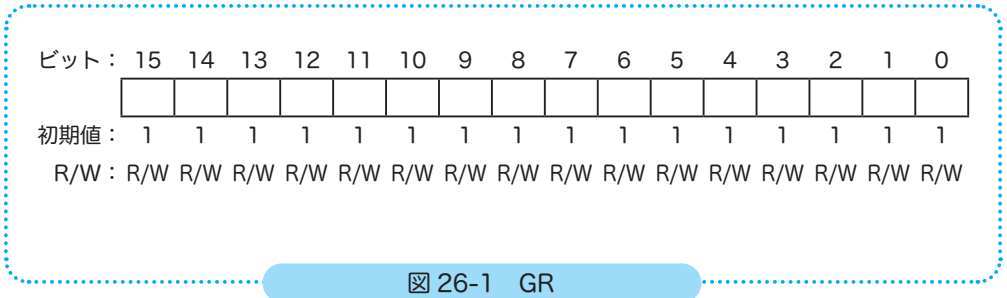


図 26-1 GR

チャネル	レジスタ名	機能
0	GRA0, GRB0	アウトプットコンペア／インプットキャプチャ兼用レジスタ
1	GRA1, GRB1	
2	GRA2, GRB2	
3	GRA3, GRB3	アウトプットコンペア／インプットキャプチャ兼用レジスタ。バッファレジスタ（BRA、BRB）と組み合わせることにより、バッファ動作設定可能
4	GRA4, GRB4	

26.2　タイマステータスレジスタ　【 TSR 0 ～ 4 】

TSR（タイマステータスレジスタ）は、STEP22 で既に出てきましたが再掲載します。この STEP では IMFB、IMFA を使います。

ビット：	7	6	5	4	3	2	1	0
	—	—	—	—	—	OVF	IMFB	IMFA
初期値：	1	1	1	1	1	0	0	0
R/W：	—	—	—	—	—	R/(W)※	R/(W)※	R/(W)※

※フラグをクリアするための 0 ライトのみ可能です。

図 22-3（再掲）　TSR

【 bit 2　OVF 】オーバフローフラグ

TCNT がオーバフロー、またはアンダフローしたときに 1 が書き込まれます。初期値は 0 です。つまり、OVF を見てやればタイマカウンタがオーバフロー (アンダフロー) したか確認できるということです。

【 bit 1, 0　IMFB, IMFA 】インプットキャプチャ／コンペアマッチフラグ

IMFB、IMFA はジェネラルレジスタ B(GRB)、ジェネラルレジスタ A(GRA) のコンペアマッチを知らせるフラグで、コンペアマッチすると 1 が書き込まれ、それ以外では 0 が書き込まれます。

0	〔クリア条件〕 IMFB ＝ 1 の状態で、IMFB フラグをリードした後、IMFB フラグに 0 をライトしたとき
1	〔セット条件〕 (1) GRB がアウトプットコンペアレジスタとして機能している場合、TCNT ＝ GRB になったとき (2) GRB がインプットキャプチャレジスタとして機能している場合、インプットキャプチャ信号により TCNT の値が GRB に転送されたとき

※ IMFA も同条件でクリア / セットされます。IMFA の場合は GRB を GRA に読み替えてください。

コンペアマッチのタイミングで行う動作を周期カウンタ動作と言います。
コンペアマッチの確認はポーリングではなく割込みで発生するようにしましょう。割込みの許可を行うレジスタがタイマインタラプトイネーブルレジスタ (TIER) です。

26.3 タイマインタラプトイネーブルレジスタ【 TIER 0 ～ 5 】

TIER（タイマインタラプトイネーブルレジスタ）は、8 ビットのレジスタです。各チャネルに 1 本、計 5 本の TIER があります。

ビット：	7	6	5	4	3	2	1	0
	—	—	—	—	—	OVIE	IMIEB	IMIEA
初期値：	1	1	1	1	1	0	0	0
R/W：	—	—	—	—	—	R/(W)	R/(W)	R/(W)

図 26-2 タイマインタラプトイネーブルレジスタ

【 bit 2 OVIE 】オーバフローインタラプトイネーブル

OVF が 1 になった時に、割込みを発生させるかどうかを設定するビットです。
ということは、STEP22 のポーリングによる OVF の確認ではなく、割込みによって行うこともできるということになります。

0	OVF フラグによる割込み（OVI）要求を禁止
1	OVF フラグによる割込み（OVI）要求を許可

【 bit 1, 0 IMIEB, IMIEA 】インプットキャプチャ／コンペアマッチインタラプトネーブル

IMIEB は、ジェネラルレジスタ B とのコンペアマッチで IMFB が 1 になった時に割込みが発生させるかどうかを設定するビットです。0 が割込み禁止で、1 が割込み許可です。
IMIEA も IMIEB と同様の機能です。

0	IMFB フラグによる割込み（IMIB）要求を禁止
1	IMFB フラグによる割込み（IMIB）要求を許可

フローチャートはこのようになります。
各処理の記述方法については、これまでのレジスタ解説と「プログラム例 26-1」を参照してください。

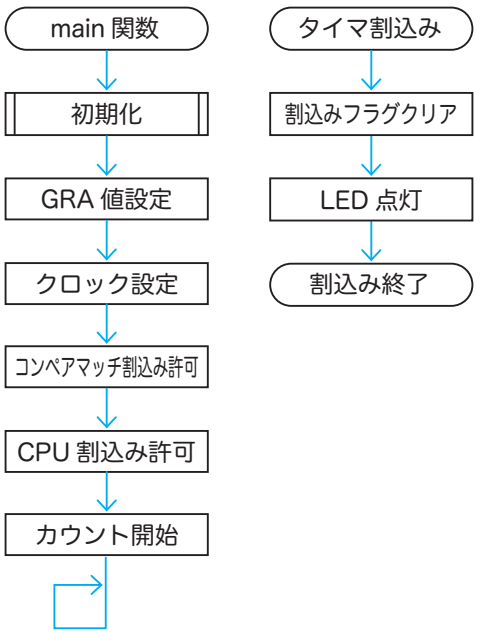


図 26-3　フローチャート

プログラム例　26-1

```
06  #define IMIA0_INT_HANDLER   // IMIA0 割込みを使用することを宣言
07  #include <3052f.h>          // 3052F 固有の定数
08  #include <ascii_x.h>        // キャラクタコード
09  #include <interrupt.h>      // 割込みベクタテーブルの定義

    中略

27  /*
28   *  タイマ割込みハンドラ 1[ms] 毎に呼び出される
29   */
30  void __attribute__((interrupt_handler))
31  IMIA0_INT(void)
32  {
33      if (ITU0.TSR.BIT.IMFA)     // IMFA フラグがセットされていれば
34          ITU0.TSR.BIT.IMFA = 0; // (リードした後) IMFA フラグクリア
35
36      // index+1 行目を点灯
37      PB.DR.BYTE = ~ASCII['A'][index]; // ~ 無しにすると文字の明暗が反転
38      P4.DR.BYTE = p4_array[index];
39
40      index++;       // LED ドットマトリクスの点灯行が 1 行下にずれるように添え字の変更
41      if (index > 7) // index 値の補正
42          index = 0;
43  }
44
45  /*
46   *  main 関数
47   */
48  int main(void)
49  {
50      initIO(); // 初期化関数の呼び出し
51
52      // タイマ割込みにチャネル 0 の GRA コンペアマッチを使用し割込みの許可
53      ITU0.GRA = 2000;           // GRA 設定値
54      ITU0.TCR.BYTE = 0xA3;      // クロックの設定 (25MHz/8) 1010 0011
55      ITU0.TIER.BIT.IMIEA = 1;   // GRA コンペアマッチによる割込み許可
56      asm("andc.b #0x7f,ccr");   // CPU の割込み許可
57
58      ITU.TSTR.BIT.STR0 = 1; // タイマカウンタ TCNT0 をスタート
59
60      while (1)
61          ;
62
63      return 0;
64  }
```

プログラム例　7-2

```
/*
 *  main 関数
 */
int main(void)
{
    initIO(); // 初期化関数の呼び出し

    // 永久ループ
    while (1)
    {
        int index = 0; // 両配列に使う添え字の変数

        for (index = 0; index < 8; index++)
        {
            P4.DR.BYTE = a_p4[index];
            PB.DR.BYTE = a_pb[index];

            waitMs(1); // 待ち時間関数の呼び出し
        }
    }

    return 0;
}
```

実行の結果、見た目には STEP07「ループによるダイナミック点灯」と変わりありません。では、何が変わったのでしょう。「プログラム例 7-2」と比べてみてください。特に注意して欲しいのが main 関数の中身です。「プログラム例 7-2」ではドットマトリクス LED を制御するための処理を記述してありましたが、「プログラム例 26-1」では割込み設定をした後に無限ループがあるだけで、中心となる処理は何もありません。割込みイベントが発生すると、自動的に割込みハンドラが呼び出されるからです。main 関数にプログラムが無いということは、ソフトによる CPU の負荷を軽減できるということなのです。

STEP 26 タイマ割込み

ascii_x.h に、キャラクタコード 20h ～ 3Fh までの文字パターンを用意しました。このヘッダファイルの使用方法は以下のとおりです。

```
moji = ASCII[code][row];
```

このように記述したとき、変数 moji には、キャラクタコード code の上から row 行目の点灯パターン（８ビット）が代入され、１が点灯、０が消灯に対応しています。

例えば、コード 0x30（'A'）の点灯パターンは、

ASCII['A'][0] ← 0x00
ASCII['A'][1] ← 0x08
ASCII['A'][2] ← 0x14
ASCII['A'][3] ← 0x22
ASCII['A'][4] ← 0x22
ASCII['A'][5] ← 0x3e
ASCII['A'][6] ← 0x22
ASCII['A'][7] ← 0x22

のように格納されています。

実際のプログラムでの使用方法は「プログラム例 26-1」を参照してください。
なお、0 ≦ code ≦ 0x7f 0 ≦ row ≦ 7　の範囲を超えて参照した場合の結果は保証されませんのでご注意ください。

最後になりましたが、すべての実習の配線をまとめた図を掲載しておきます。
なお、スピーカは実習内容によって、$P6_0$ もしくは $P7_6$ に配線してください。

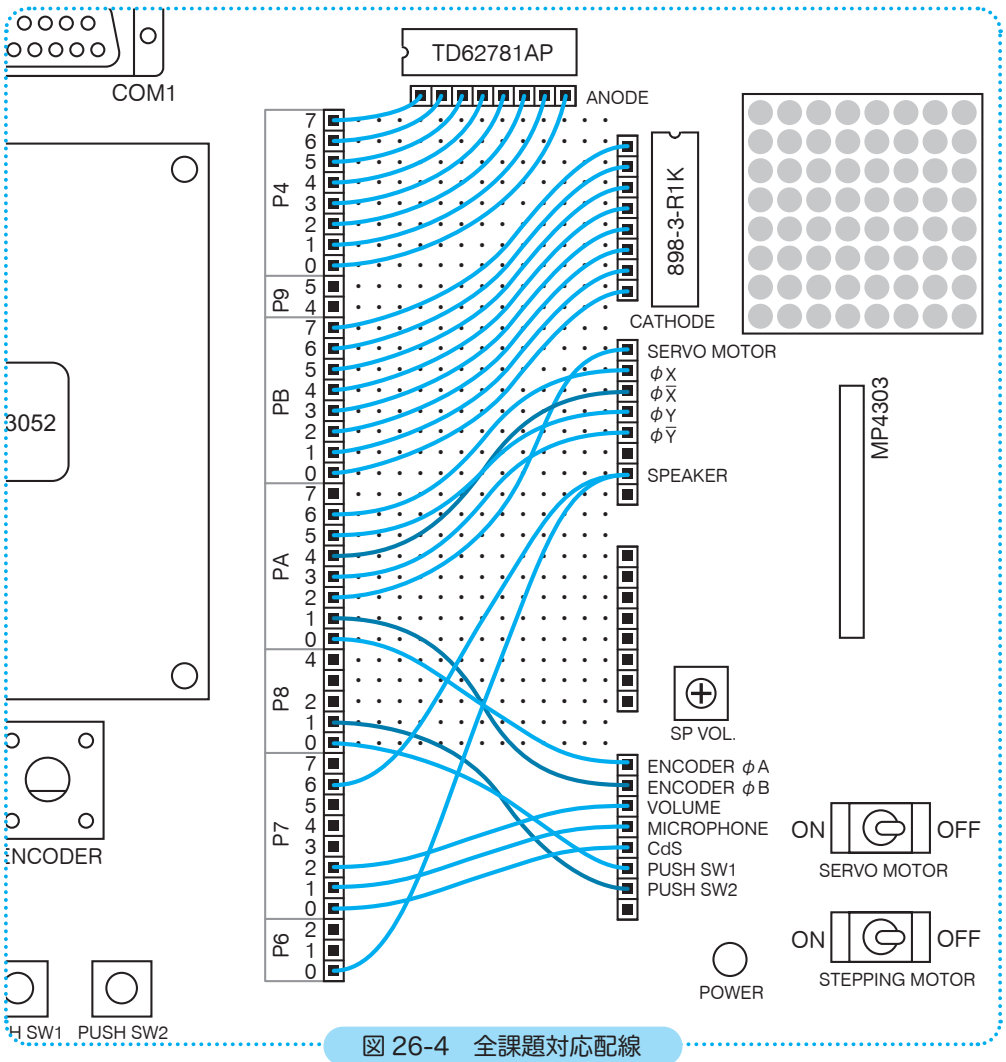


図 26-4　全課題対応配線

本教材の実習は以上です。
ここからは、これまで作成したプログラムの「ループ待ち時間関数」をタイマや割込みを使って改良したり、課題を組み合わせてオリジナル課題を設定して挑戦してみるのもいいでしょう。本教材がマイコン学習のきっかけになれば幸いです。