

第2回

本日の内容
割り込みとは

タイマー

割り込み

今までのプログラムは、順番にそって命令を実行していくのみ。
それはそれで良いが、不便な場合もある。

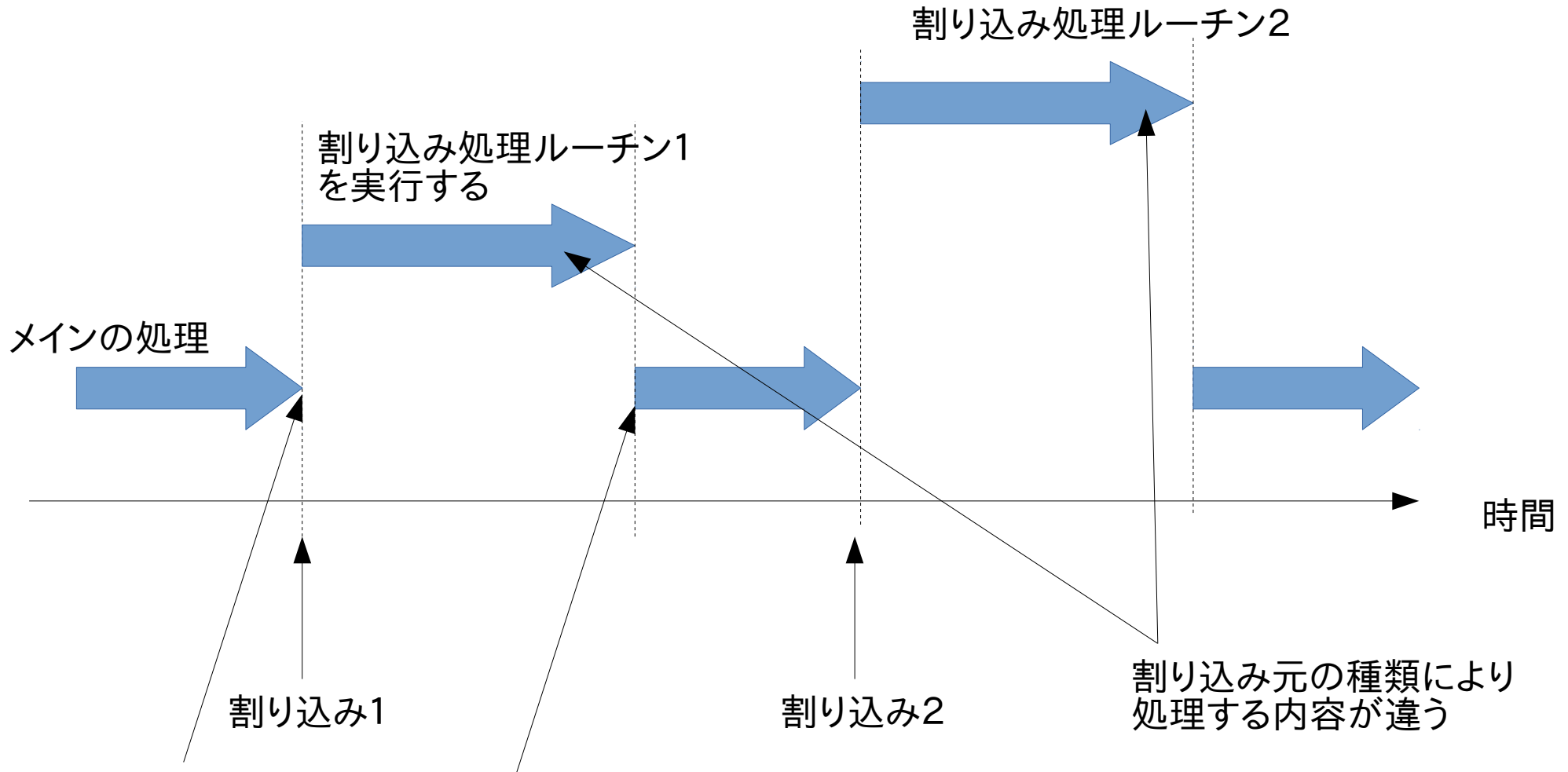
例えば、
時間のかかる周辺機器を使う場合、その周辺機器が動作を終了するまで、CPUは待たなければいけない。

方法1(ポーリング)
一定時間毎に、周辺機器の動作が終了したか調べる。
終了していれば、次の動作に移るし、そうでなければ、また少し待ってから同じことを繰り返す。

めんどくさいし、無駄が多い

方法2(割り込み)
周辺機器に、動作が終了したら通知(=割り込み)してもらうように予め依頼しておく。
CPUは、この周辺機器のことなど気にせず、他の作業をすることができる。
周辺機器からお知らせが来たら、予め決めてある動作(ISR,割り込み処理ルーチン)を行う。

割り込み



割り込みが入ると、
現在の処理を中断する

割り込み処理が終わったら
先ほど実行していた処理を再開する

実際の割り込みルーチンの例 (PIC18Fシリーズ)

PIC18シリーズは、2つの割り込みルーチン(優先度高、低)がある

```
void interrupt High_ISR(){
    int i;

    if(INTCONbits.TMR0IF){
        :
        INTCONbits.TMR0IF=0;
    }
}

void interrupt low_priority Low_ISR(){
    if(PIR2bits.USBIF){
        :
        PIR2bits.USBIF = 0;
    }
}
```

ISRの場合は `interrupt` (or `interrupt low_priority`) を、関数の前につける

ISRの中では、どの要因で割り込みが生じたのか、フラグを見て判断する。(場合分け)

処理を終了する前に、フラグを**必ず**クリアする。

割り込みの設定レジスタ

主な関連レジスタ

INTCON ... 割り込みの許可・禁止の設定

INTCON2

INTCON3

PIR1, 2

PIE1, 2

IPR1, 2

INTCON

GIEH ... 全ての優先度”高”の割り込みの許可・禁止

GIEL ... 全ての優先度”低”の割り込みの許可・禁止

TMR0IE ... TMR0の割り込みの許可・禁止

他にも同様

INTCON2

INTEDGE0 ... 外部割り込みの極性(1なら信号立ち上がり、0なら信号立ち下がりで割り込み発生)

TMR0IP ... TMR0の割り込みの優先度を設定(1なら優先度”高”、0なら優先度”低”)

他にも同様

割り込みの設定レジスタ

INTCON3

INT2IP ... 外部割り込み2 (INT2ピン)の優先度を設定

INT2IE ... 外部割り込み2の許可・禁止を設定

INT2IF ... 外部割り込みが発生したかどうかを示す。割り込みが発生したら自動的に1に設定される。

他も同様

以下のレジスタは、以上と同様。

PIR1, 2 ... それぞれの割り込み要因毎の割り込みフラグ

PIE1, 2 ... それぞれの割り込みの許可・禁止を設定

IPR1, 2 ... それぞれの割り込みの優先度を設定

タイマー

文字通り時間を計測するためのモジュール。
一定間隔で処理をしたいときや、時間を計測したいときなどに使用する。
PIC18F2550には、4つのタイマー(TMR0~3)が内蔵されている。
それぞれ、ビット数や他の周辺モジュールへの接続が異なるので、詳しくはデータシート参照

ここでは、TMR0(8bitモード)について説明する。

関連する主なレジスタ

T0CON

TMR0L

データシートより転載

FIGURE 11-1: TIMER0 BLOCK DIAGRAM (8-BIT MODE)

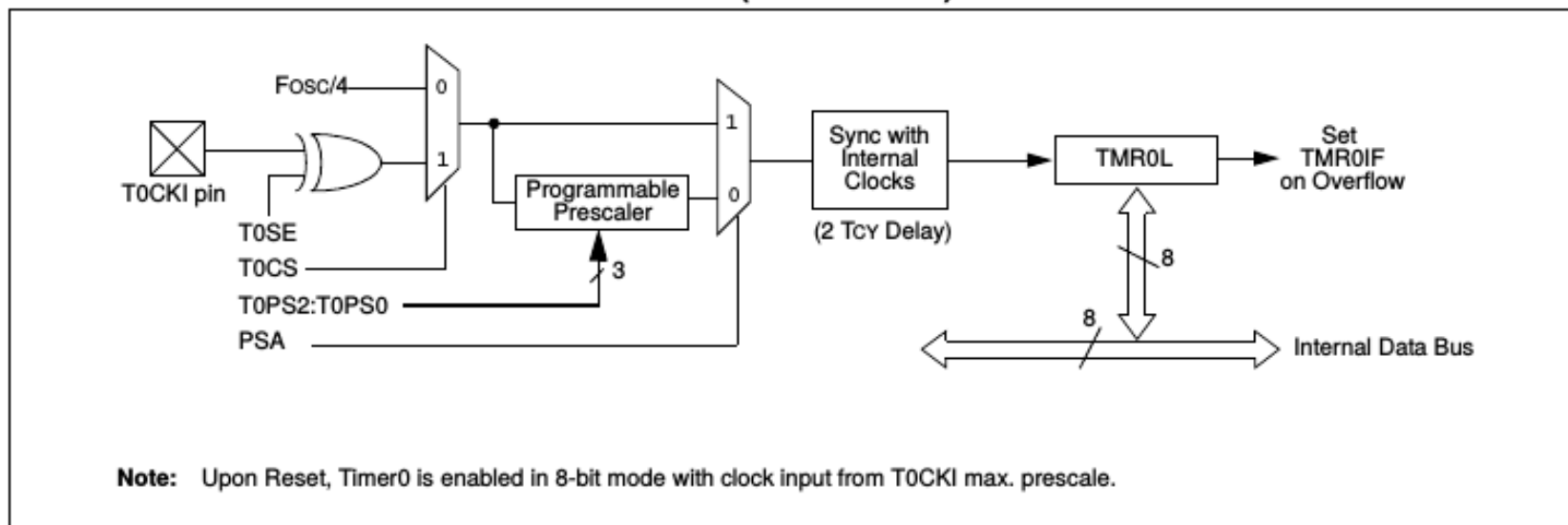
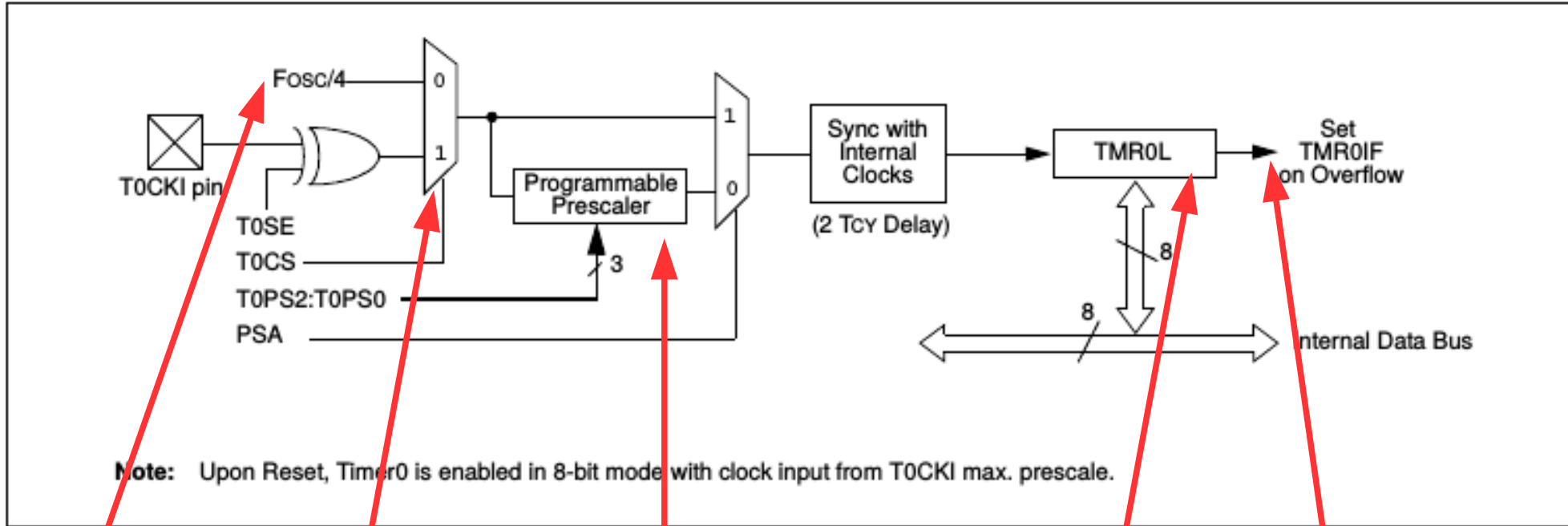


FIGURE 11-1: TIMER0 BLOCK DIAGRAM (8-BIT MODE)

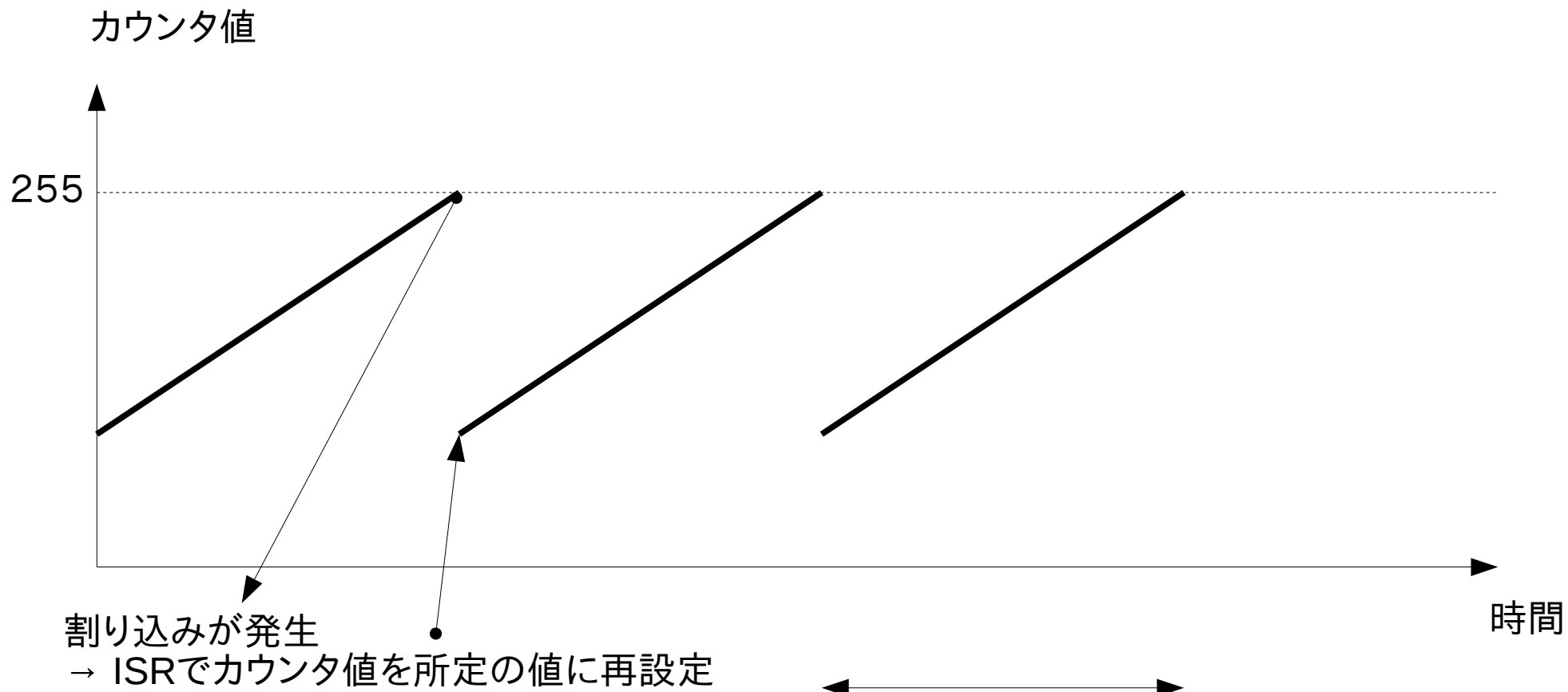


カウントアップのための信号選択
クロック信号(の1/4)

プリスケaler
(分周器)

カウンタ(8bit)

カウンタ値が255(0xff)になったら、割り込みが発生



割り込みの周期は、
 $(255 - (\text{設定カウンタ値})) / \text{タイマクロック}$
になる。

割り込みの実験 (タイマー割り込み)

タイマー (TMR0) を使って一定周期 (0.1ms) で割り込みをかける。
割り込みルーチンが呼ばれた回数をカウントすると、何秒経過したかわかる。
これをつかって、0.5秒ごとに順番にLEDを光らせる

メイン関数 (初期化したあとは、何もしない。これだけ見ると、何もしないように見えるが...)

```
void main(void){  
    InitializeSystem();  
    EnableHighInterrupt();  
    while(1){  
        asm("nop");  
    }  
}
```

初期化関数

```
unsigned long time_counter;  
#define TMR0_PERIOD 105 // (=255-150) TMR0 period: 100us
```

```
void INIT_TMR0(){ タイマーの設定(詳しくはデータシート参照)
```

```
//TMR0 setting [every 100us
```

```
T0CON = 0xd2; //1101_0010 Enable TMR0, 8bit, Internal clk, PSA=1/8
```

```
TMR0L = TMR0_PERIOD; ← タイマー周期を設定する(今回は0.1msにしてある)
```

```
INTCONbits.TMR0IE = 1; //Interrupt enable
```

```
INTCON2bits.TMR0IP = 1; //high priority タイマーがオーバーフロー(0.1秒後)したら  
割り込みを発生させるようにする
```

割り込みの優先度を”高”にする。
これによって、割り込みが起こったら0x08番地に飛ぶ

```
void INIT_LEDs(){
```

```
TRISBbits.TRISB5 = 0; //RB5 -- Output PIN
```

```
LATBbits.LATB5 = 1; //RB5 = 'H'
```

```
}
```

初期化関数

```
static void InitializeSystem(void){  
//GPIO Initialize  
    ADCON1=0x0F;          //All digital IO  
    LATA = 0;  
    TRISA = 0xff;        // Port A : Input  
    LATB = 0;  
    TRISB = 0xff;        // Port B : Input  
    LATC = 0;  
    TRISC = 0xff;        // Port C : Input  
  
    INIT_TMR0();  
  
    INIT_LEDs();  
  
    time_counter = 0;  
}
```



すべてのIOポートを入力で初期化する

割り込みルーチン

優先度”高”のISR

```
void interrupt High_ISR(){  
    static long counter =0;
```

カウンターを定義(staticをつけると、関数を一旦出ても値が保持される. それ以外は関数を出ると値は破棄される)

```
    if(INTCONbits.TMR0IF){
```

この割り込みがTMR0によって引き起こされたかどうかを調べる
もしTMR0割り込みなら, TMR0IFに1が入っているはず

```
        TMR0L = TMR0_PERIOD;    //period: 100 us
```

新たなタイマー周期を設定(一定周期なので, 同じ値を入れる)

```
        //TMR0 ... system time counter
```

```
        if( ++counter > 25000 ){    //2.5 sec
```

```
            counter = 0;
```

```
            LED0_off();
```

```
        }else if( counter > 20000 ){//2.0 sec
```

```
            LED0_on();
```

20000カウント(=2秒経過)したら, LEDをONにする

```
        }
```

```
        INTCONbits.TMR0IF=0;    //Clear Interrupt flag
```

```
    }
```

```
}
```

フラグをクリア <<絶対必要>>

これをしないと, 割り込みがこのルーチンを抜けた瞬間に,
また割り込みが発生してしまう. 無限ループ

タイマーの応用: LEDの明るさ調整

アナログ的にやるのは困難(マイコンのIOは通常デジタル出力)
そこで, すごい早さでON・OFFさせる.
ONとOFFの時間を変えることで, 明るさを変える.

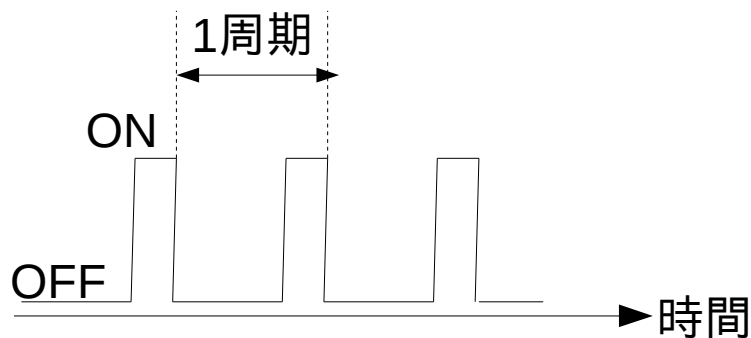
そのような目的のために, PICにはPWMモジュールが組み込まれている.

しかし, PWMモジュールは数が限られている上, 接続するピンが限られてしまう.
(たくさんのLEDの駆動はむずかしい)

そこで, タイマー割り込みをつかって, PWMの機能を実現する.

→ソフトウェア処理なので, 汎用性がある

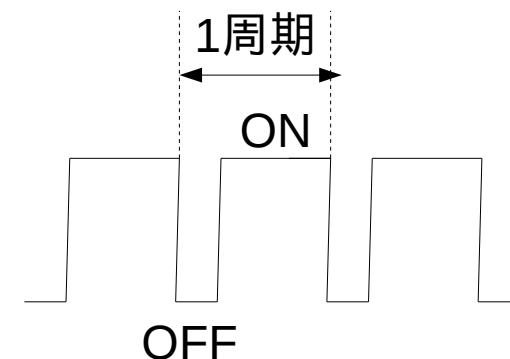
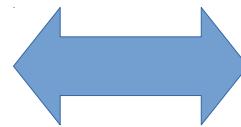
→ただし, ソフトウェア処理に多少の時間がかかる(あまり早いON, OFFは難しい)
(人間の目をごまかすくらいの十分な早さは出せる)



デューティー比 = (ONの時間) / (周期)

デューティー比, 小 = 暗い

連続的に変化



デューティー比, 大 = 明るい

main.c

```
    :  
    #define LED_PERIOD (100)           LED点灯の周期を設定(これで10msに設定される)  
    BYTE LED0_duty;                   LEDのON時間を保存しておくための変数  
    #define LED0_PWM(val) LED0_duty = (val) ON時間を設定するためのマクロ  
    :
```

```
unsigned long time_counter;  
#define TMR0_PERIOD 105  //(=255-150) TMR0  
period: 100us
```

```
void interrupt High_ISR(){  
    static unsigned int LED_timer;
```

```
    if(INTCONbits.TMR0IF){  
        TMR0L = TMR0_PERIOD;    //period: 100 us  
        time_counter++;
```

← システム時間(グローバル変数)を更新

```
        if(LED_timer <= LED0_duty) LED0_on();
```

← カウンタ(LED_timer)と設定値を比較
LEDx_dutyの値に応じてLEDを点灯

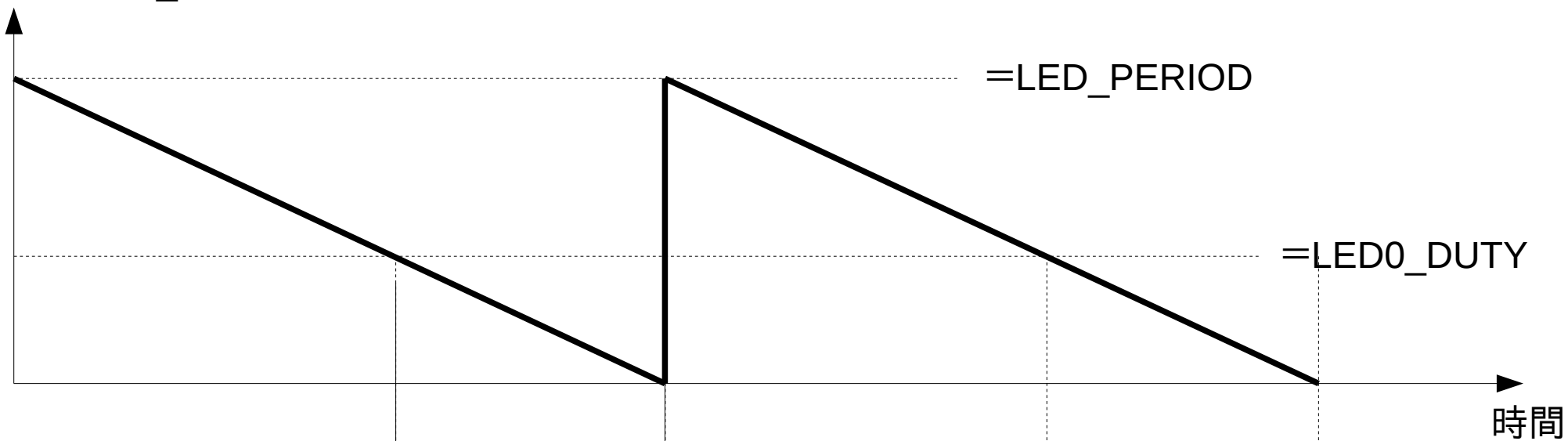
```
        if(--LED_timer == 0){  
            LED_timer = LED_PERIOD;  
            LED0_off();  
        }
```

← カウンタ(LED_timer)が0になったら,
新たな周期を書き込む
すべてのLEDをOFF

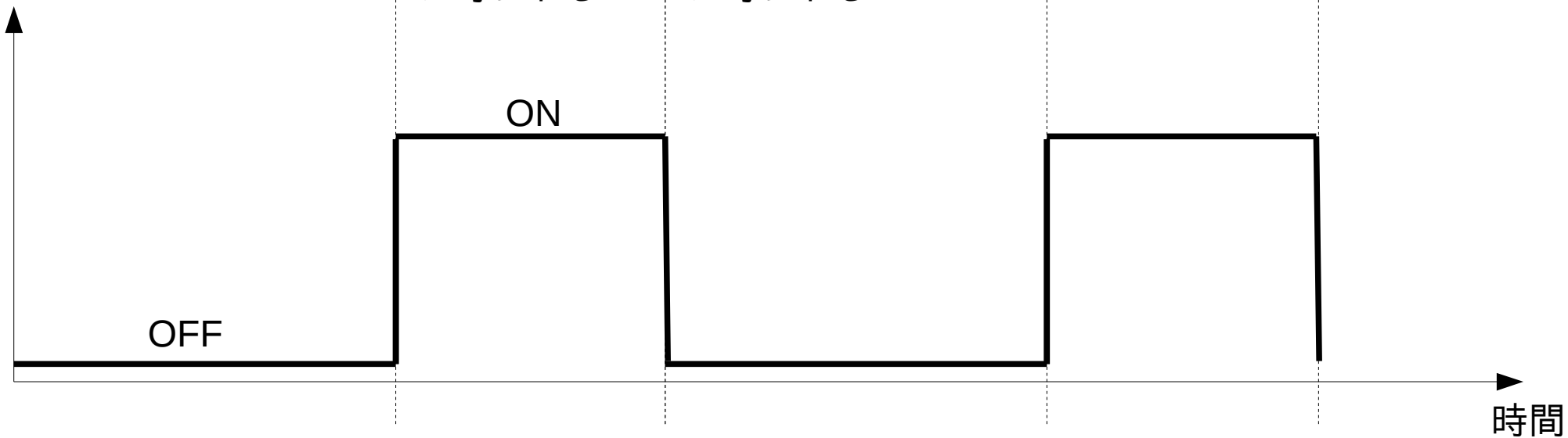
```
        INTCONbits.TMR0IF=0; //Clear Interrupt flag
```

```
    }  
}
```

変数"LED_Timer"の値



LED0の出力

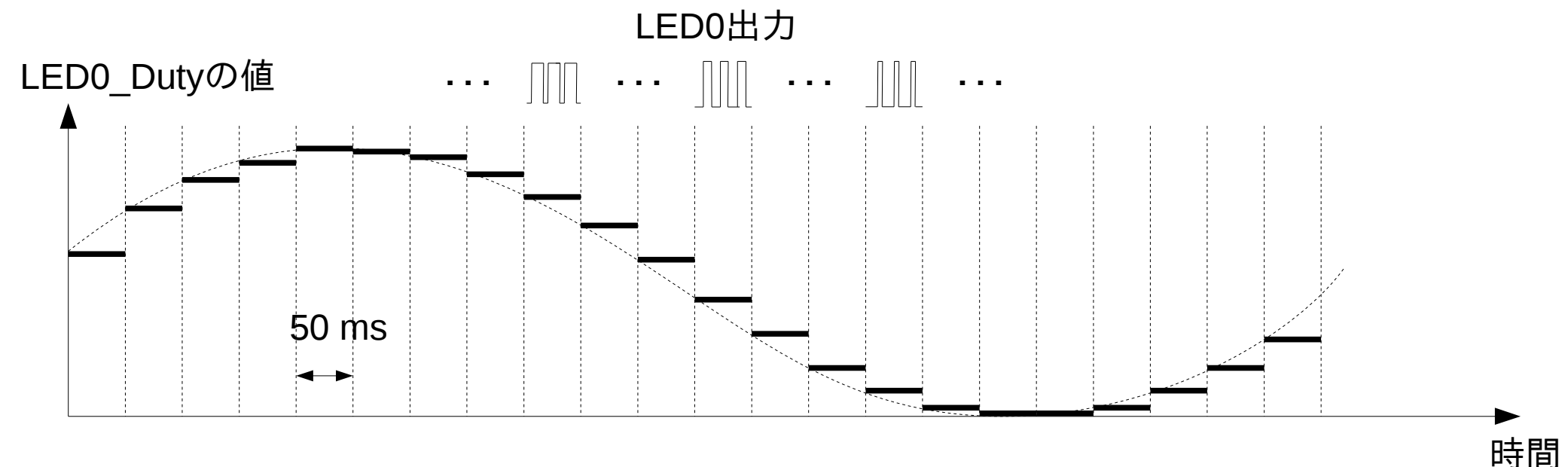


main.c

```
:\nperiod = 10000;\nwhile(1){\nif(time_counter > time_next_change){\n    time_next_change = time_counter + 500;\n    LED0_PWM( (BYTE)(50.0\n        + 50.0*sin( ((double)time_counter/(double)period + 0.00) * 2.0*3.14) ) );\n}\n}
```

システム時間を見て、一定時間ごとに処理を行う
(ここでは50msごと)

LEDの明るさ(PWMのデューティ)をサイン波状に変える



応用

LED_PERIODの値を大きくすれば、遅い点滅も表現できる
(PWMの明るさ制御とまったく同じ。周期が早いと人間の目がごまかされて明るさが変化しているように見えるだけ)

(ラジコン用)サーボモータの制御もPWM変調
→同じ方法で、サーボモータの制御もできる。

フィルタ(広域カットフィルタ, LPF)を通せば、アナログ出力を得られる。
→DAコンバータ(注, 低速)

割り込みを使ったプログラム

問題

ポートB0にスイッチをつなぎ、スイッチを押したら割り込みが発生するようにせよ。

ポートA0にはLEDをつなぎ、割り込みルーチンに入ったら、LEDのON/OFFを切り替えよ

```
void interrupt isr(){  
    if( ...)  
  
    //clear flag  
  
}
```