

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。

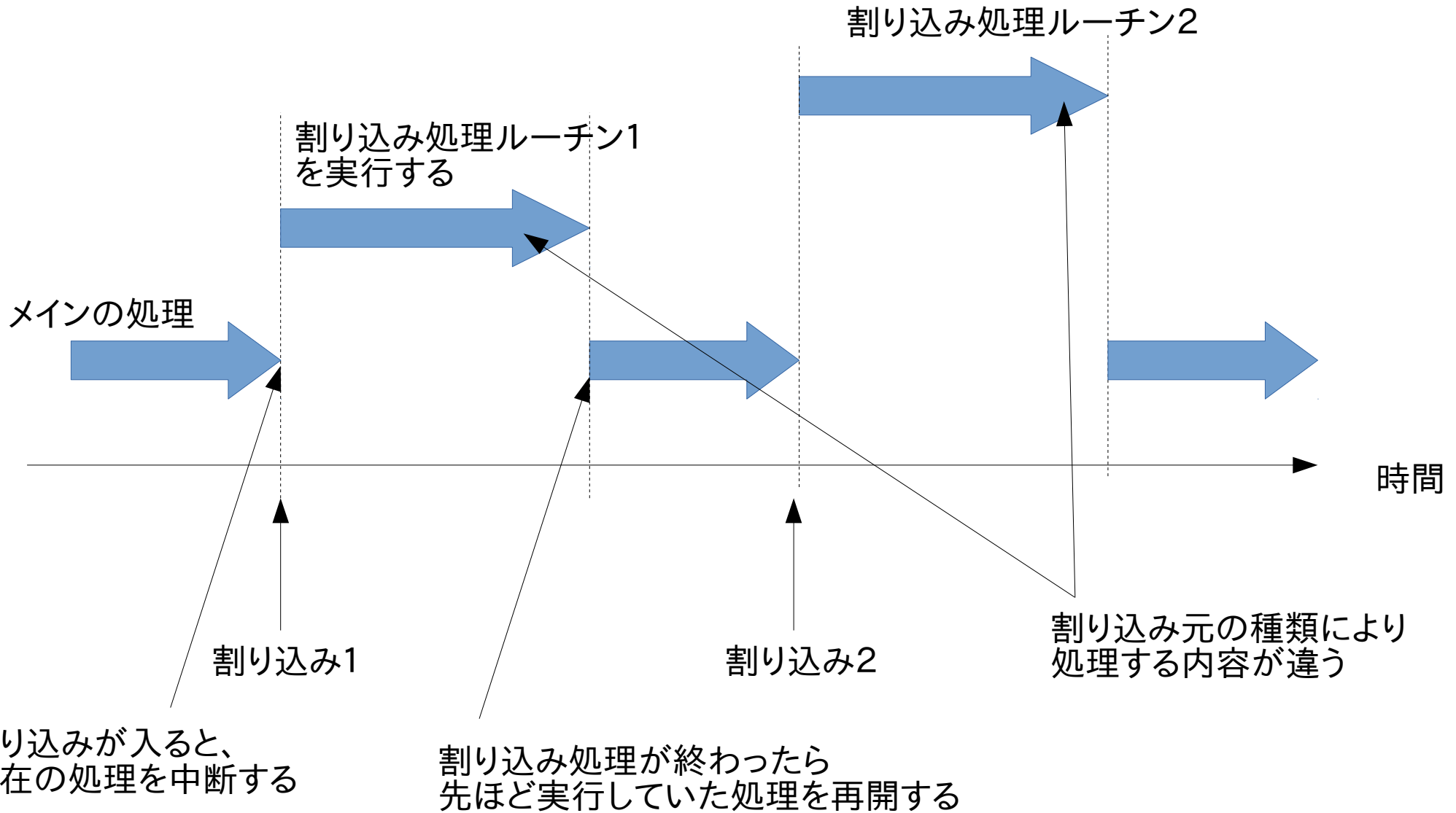
例えば、
時間のかかる周辺機器を使う場合、その周辺機器が動作を終了するまで、CPUは待たなければいけない。

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



実際の割り込みルーチンの例 (PIC24Fシリーズ)

PIC24シリーズの場合は、それぞれの割り込み要因ごとにISRを設定できる。(ベクタ方式)

ADC割り込みの場合

```
void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void){  
    :  
    IFS0bits.AD1IF= 0; 終了前にはフラグをクリアする  
}
```

タイマー(TMR1)割り込みの場合

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void) {  
    :  
    IFS0bits.T1IF = 0; 終了前にはフラグをクリアする  
}
```

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。
PIC24FJ32GA002には、5つのタイマー(TMR1~5)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、TMR1(16bitモード)について説明する。

関連する主なレジスタ

T1CON

TMR1

PR1

データシートより転載

FIGURE 11-1: 16-BIT TIMER1 MODULE BLOCK DIAGRAM

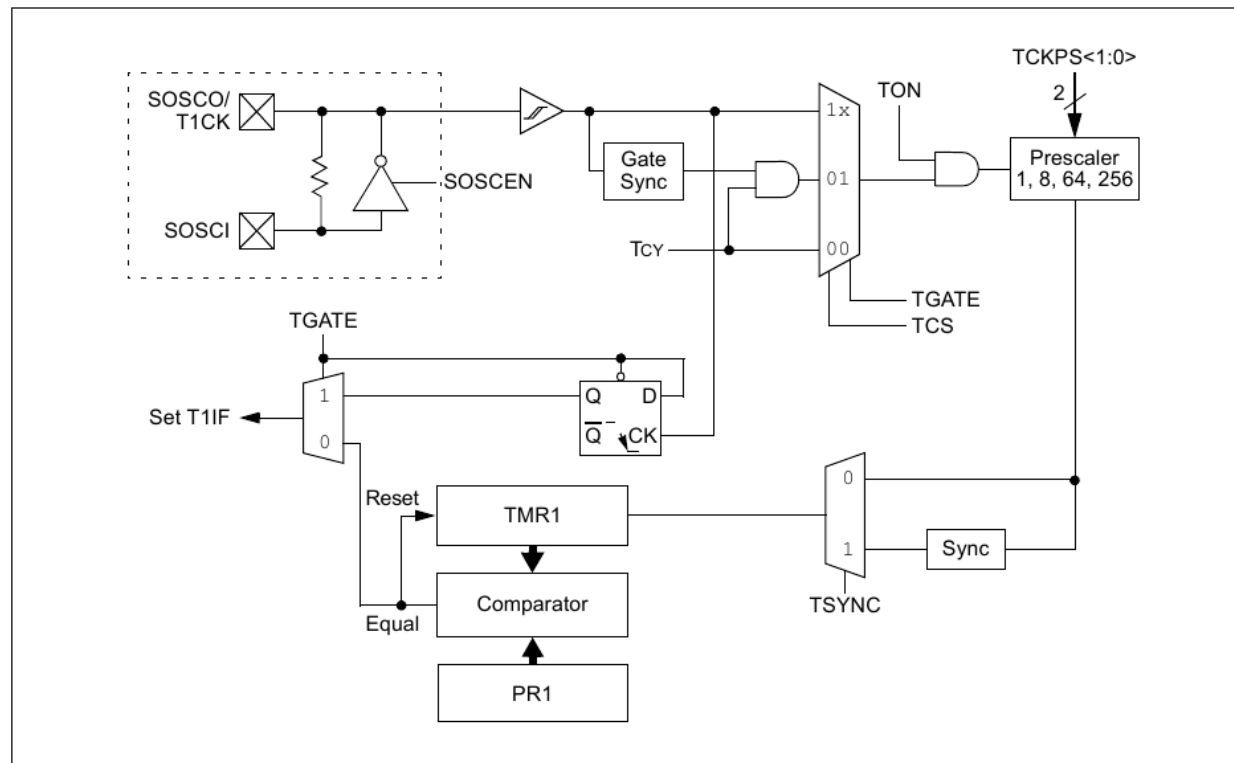
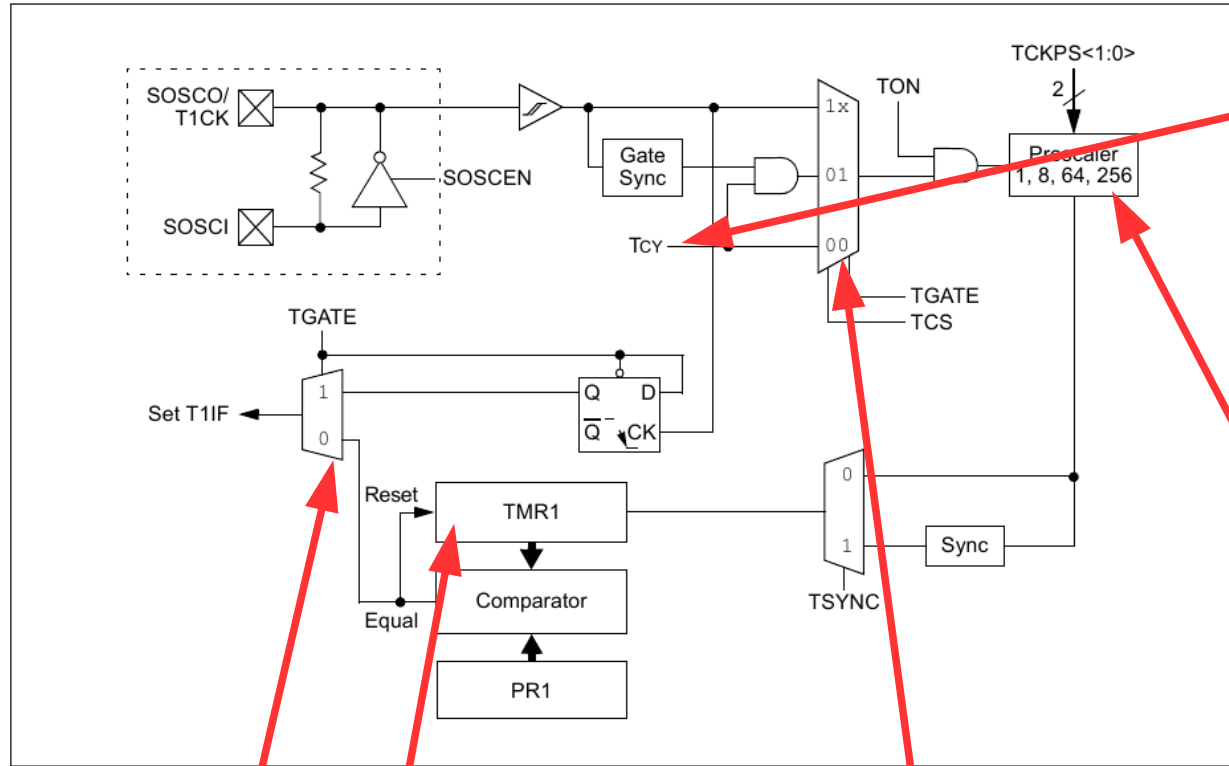


FIGURE 11-1: 16-BIT TIMER1 MODULE BLOCK DIAGRAM



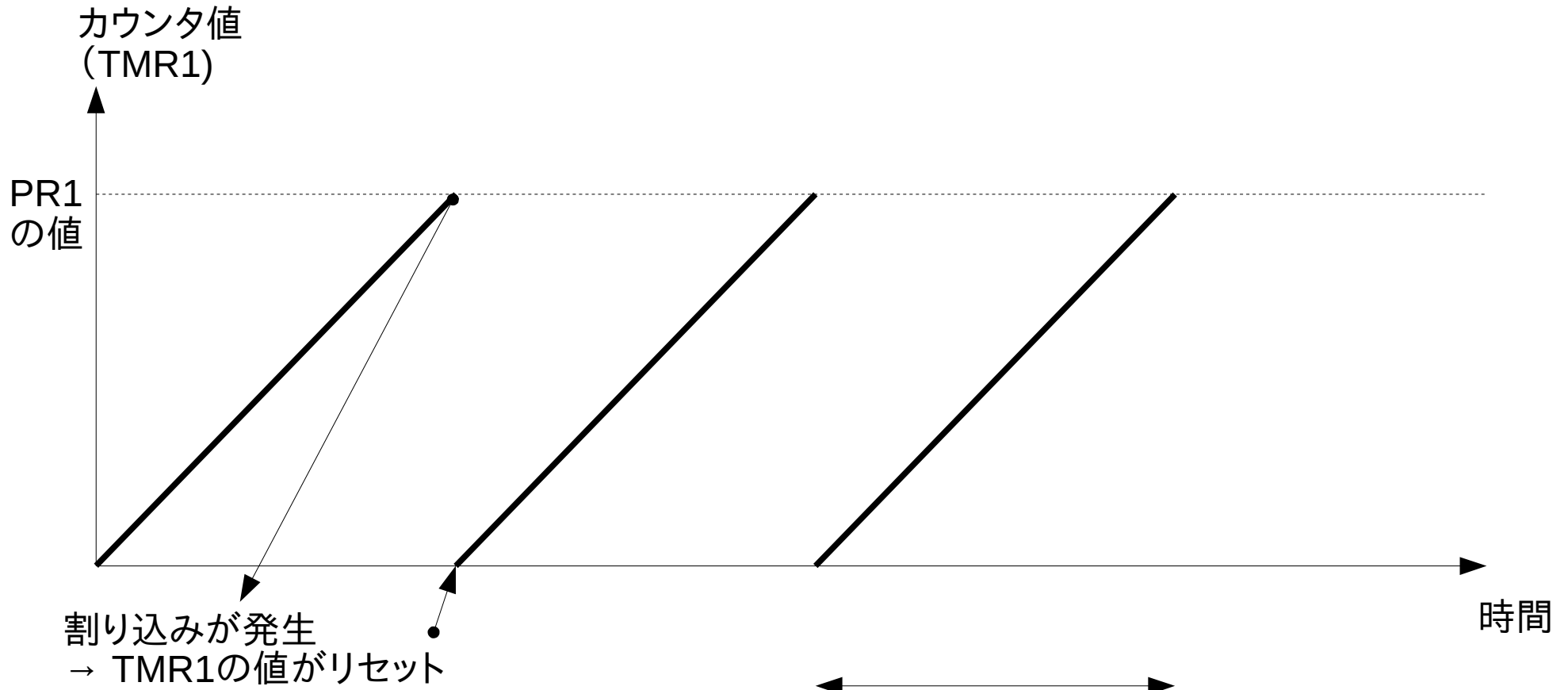
クロック信号(の1/2)

プリスケaler
(分周器)

カウンタ値がPR1の値と等しく
なったら、割り込みが発生

カウンタ(16bit)

カウントアップのための信号選択



割り込みの周期は、
 $(PR1の値) / \text{タイマクロック}$
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...)

```
int main(){
    init();

    LED0_off();
    LED1_off();

    while(1)
        ;
    //Never comes here
    return 0;
}
```


初期化関数

```
void init(){
```

```
    AD1PCFG = 0xffff; //All digital IOs
```

```
    EnableLED0();
```

```
    EnableLED1();
```

```
//oscillator
```

```
    CLKDIV = 0x00; // CPU clock 1:1, FRC PS = 1, PLL enabled (i.e. 32MHz),  
    //f_CPU = f_peripheral = 32MHz
```

```
//TMR1 (1ms)
```

```
    T1CON = 0b00000000000010000; //Fcy=Fosc/2(=16MHz), Prescaler(1/8) ->  
    2MHz
```

```
    PR1 = 200; // 200/2MHz = 0.1ms
```

```
    TMR1 = 0;
```

```
    IPC0bits.T1IP = 7; //Interrupt priority = 7
```

```
    IEC0bits.T1IE = 1; //Enable Interrupt
```

```
    T1CONbits.TON = 1; //start
```

```
}
```

すべてのIOポートをGPIOで使用する

システムクロックの設定(データシート参照)
これで、PICは32MHzの内蔵クロックで動く

タイマーの設定(詳しくはデータシート参照)
入力はシステムクロックの1/2, プリスケアラは1/8
つまり $32\text{MHz} / 2 / 8$ で2MHzでタイマーが駆動される

タイマー周期を設定する(今回は0.1msにしてある)

タイマーカウンターをリセット

割り込みの優先度を"7"にする.

割り込みを許可する

タイマーをスタートさせる

割り込みルーチン

タイマー1の割り込みハンドラ (ISR)

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void) {  
    static unsigned int 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。じゃないと関数を出ると値は破棄される)

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

```
if( ++counter > 20000 ){ //2.5 sec
```

カウントアップしつつ、値を比較

```
    counter = 0;
```

20000カウント(=2秒経過)したら、LEDとカウンターをリセット

```
    LED0_off();
```

```
}else if( counter > 10000 ){ //2.0 sec
```

```
    LED0_on();
```

10000カウント(=1秒経過)したら、LEDを点灯させる

```
}
```

```
IFS0bits.T1IF = 0;
```

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.

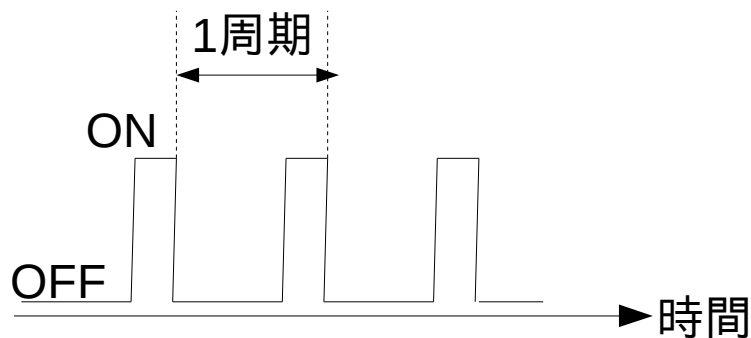
そのような目的のために, PICにはPWMモジュールが組み込まれている.

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

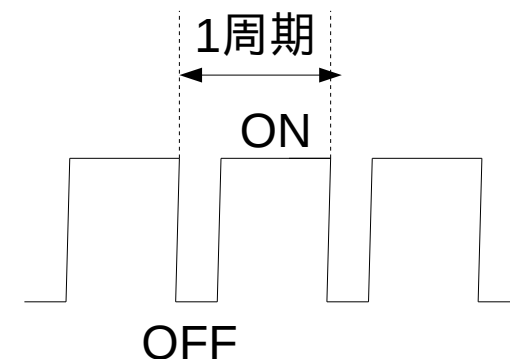
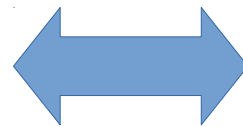
→ただし, ソフトウェア処理に多少の時間がかかる(あまり早いON, OFFは難しい)
(人間の目をごまかすくらいの十分な早さは出せる)



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)
unsigned char LED0_duty;
#define LED0_PWM(val) LED0_duty = (val)
```

LED点灯の周期を設定(これで10msに設定される)

各LEDのON時間を保存しておくための変数

ON時間を設定するためのマクロ

```
//TMR1 Interrupt
void __attribute__((interrupt, no_auto_psv))
_T1Interrupt(void) {
    static unsigned int LED_timer;
```

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();

if(--LED_timer == 0){

LED_timer = LED_PERIOD;

LED0_off();

}

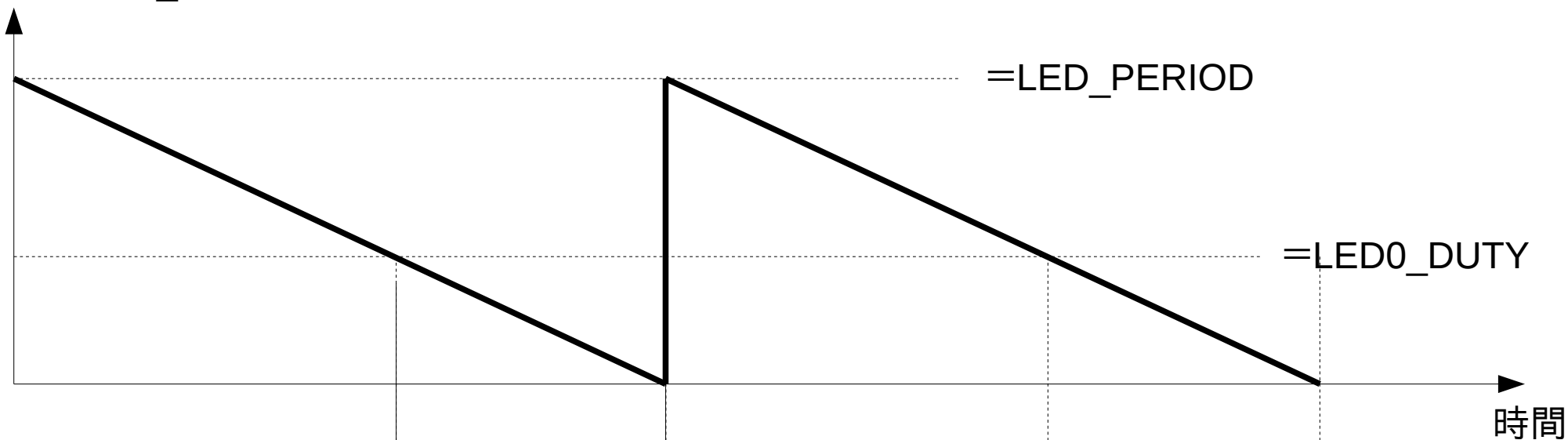
IFS0bits.T1IF = 0;

}

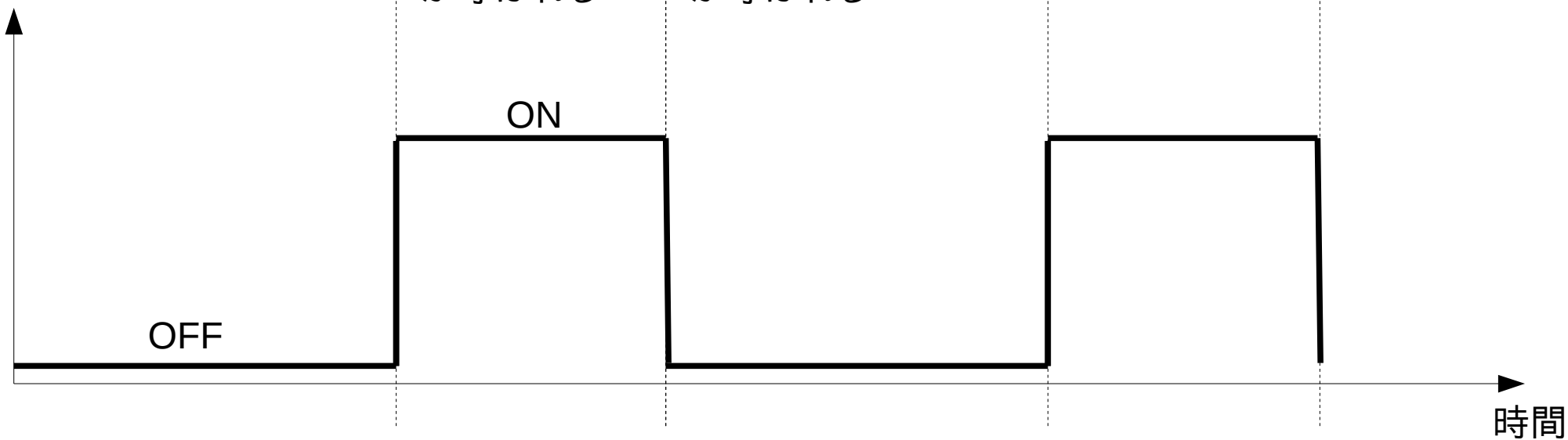
← カウンタ(LED_timer)と設定値を比較
LEDx_dutyの値に応じてLEDを点灯

← 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

変数"LED_Timer"の値



LED0の出力



LED0_ON()
が呼ばれる

LED0_OFF()
が呼ばれる

ON

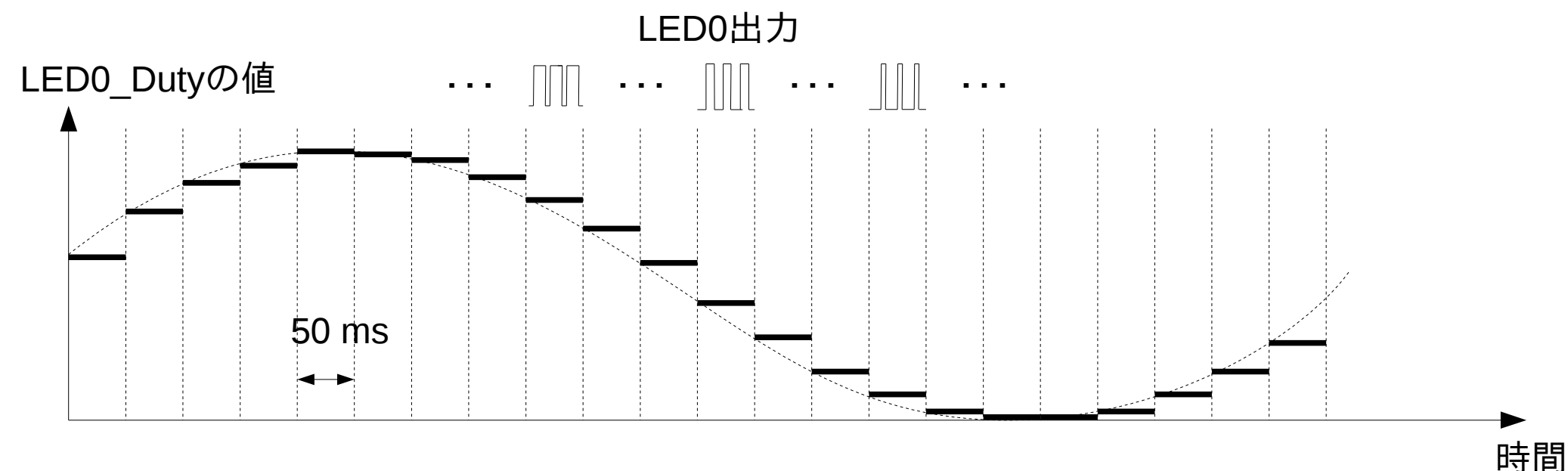
OFF

main.c

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