STEP 08

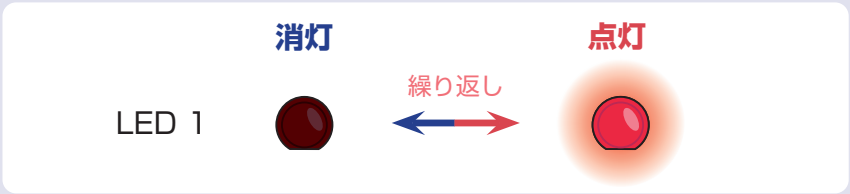
割り込みで LED 点滅

学習内容

LED の点滅は今まで学習した内容で実現できますが，本 STEP では「割り込み」を利用して LED の点滅を実現してみましょう。

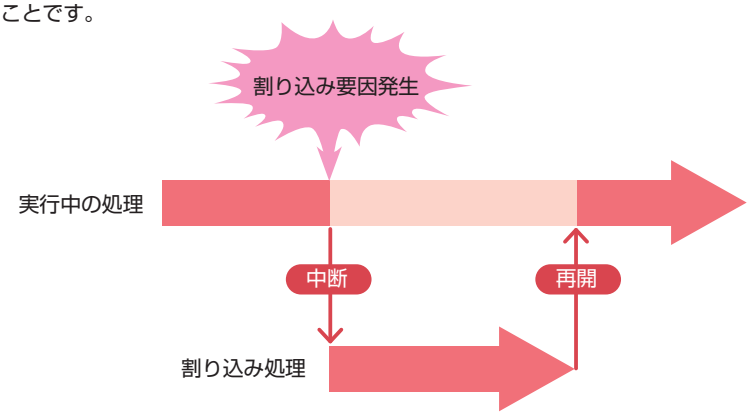
課題 08-1

LED1 を 1 秒周期で点滅させましょう。



割り込みとは

割り込みとは，あるイベント（割り込み要因）が発生した際に，実行中の処理を一時中断して別の処理を行うことです。

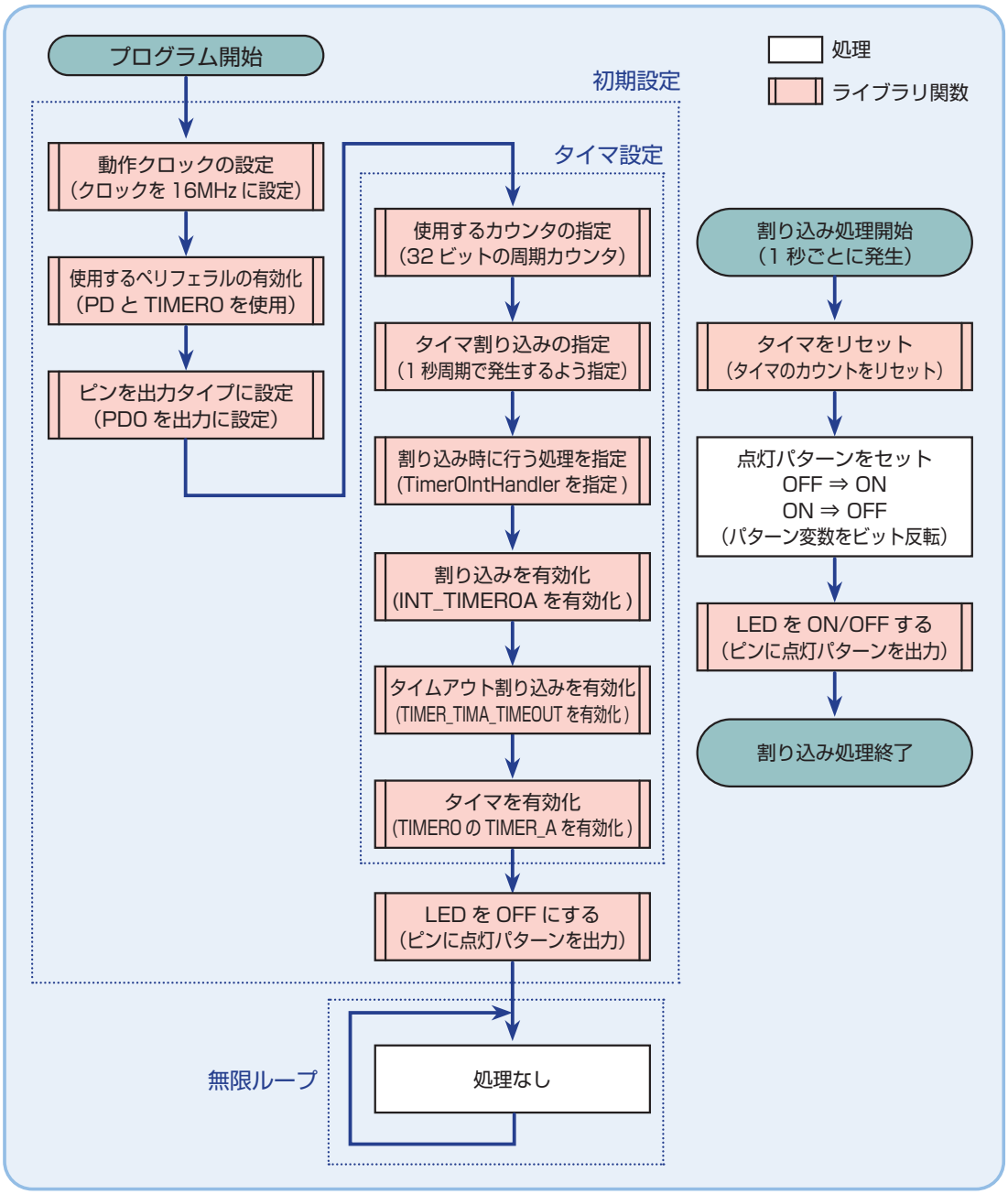


配線 08-1

「配線 07-1」と同じです。

フローチャート　08-1

フローチャートは以下のようになります。



割り込みで LED 点滅

インクルードファイル 08-1

STEP 08-1 で使用するインクルードファイルを解説します。

Tiva ヘッダファイル inc/hw_ints.h

```
#include "inc/hw_ints.h"
```

Tiva のヘッダファイルで，割り込みの割り当てに関するマクロが宣言されています。次に示すライブラリ関数の設定例では，以下のマクロがここで宣言されています。

- INT_TIMER0A

Tiva ヘッダファイル driverlib/interrupt.h

```
#include "driverlib/interrupt.h"
```

Tiva のヘッダファイルで，割り込み制御のためのマクロおよび関数が宣言されています。次に示すライブラリ関数の設定例では，以下の関数がここで宣言されています。

- intEnable(ui32Interrupt)

Tiva ヘッダファイル driverlib/timer.h

```
#include "driverlib/timer.h"
```

Tiva のヘッダファイルで，タイマのためのマクロおよび関数が宣言されています。次に示すライブラリ関数の設定例では，以下のマクロおよび関数がここで宣言されています。

- TIMER_TIMA_TIMEOUT
- TIMER_CFG_PERIODIC
- TIMER_A
- TimerIntClear(ui32Base, ui32IntFlags)
- TimerConfigure(ui32Base, ui32Config)
- TimerLoadSet(ui32Base, ui32Timer, ui32Value)
- TimerIntRegister(ui32Base, ui32Timer, void (*handler)(void))
- TimerIntEnable(ui32Base, ui32IntFlags);
- TimerEnable(ui32Base, ui32Timer)

ライブラリ関数　08-1

STEP 08 で使用するライブラリ関数を解説します。
関数の引数は ui32 ～が「符号無し 32 ビット整数」，ui64 ～が「符号無し 64 ビット整数」です。
ライブラリ関数は TivaWare で提供されています。なお，タイマについては p.61 からの解説もご覧ください。

使用するペリフェラルの有効化　SysCtlPeripheralEnable(ui32Peripheral)

使用したいペリフェラル（ここではタイマモジュール）を有効化する。

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
```

設定項目 例

- SYSCTL_PERIPH_TIMER0：　例はタイマ 0 を使用するときの設定値。

タイマカウンタの設定　TimerConfigure(ui32Base, ui32Config)

使用するタイマのカウンタを設定する。

```
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
```

設定項目 例

- TIMER0_BASE　：設定したいタイマモジュールの指定。例は Timer0。
  ワイドタイマを指定する場合は WTIMER0_BASE のように設定する。
- TIMER_CFG_PERIODIC　：タイマモジュールの全幅（Timer0 の場合は 32 ビット）を周期タイマとして利用。半分の幅（Timer0 の場合は 16 ビット）を持った 2 つの周期タイマとして利用したい時は TIMER_CFG_SPLIT_PAIR ｜ TIMER_CFG_A_PERIODIC ｜ TIMER_CFG_B_PERIODIC のように指定する。

クロック数の取得　SysCtlClockGet( )

マイコンのクロック数（符号無し 32 ビット整数）を返す。

```
SysCtlClockGet();
```

今回，クロックを 16MHz としているので，取得値は 16000000 となる。
設定項目 なし

割り込みで LED 点滅

ライブラリ関数　08-1

タイマ処理周期の設定　TimerLoadSet(ui32Base, ui32Timer, ui32Value)

タイマ処理の発生する周期を設定する。なお，タイマとしてワイドタイマの全幅を利用する場合は TimerLoadSet64(ui32Base, ui64Value) を用いる。

```
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());
```

設定項目 例

- TIMER0_BASE　：設定したいタイマの指定。
  例はタイマ0。
- TIMER_A　：使用するタイマ（カウンタ）の指定。
  タイマモジュールの全幅を利用している場合は A を指定。
  半分の幅を利用している場合は A か B を指定。
- SysCtlClockGet()　：周期を指定する項目。マイコンのクロックを基準に計算される。
  このため，この項目にマイコンのクロック数と同値を設定すると，
  周期が 1 秒になる。このことから以下のようにして周期を指定できる。

（例）

周期 1 秒（1Hz）	SysCtlClockGet()
周期 0.1 秒（10Hz）	SysCtlClockGet() / 10
周期 1 ミリ秒（1kHz）	SysCtlClockGet() / 1000

タイマ割り込みで呼び出される関数の登録　TimerIntRegister(ui32Base, ui32Timer, void (*handler)(void))

タイマ割り込みで呼び出される関数を登録する。

```
TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0IntHandler);
```

設定項目 例

- TIMER0_BASE　：設定したいタイマの指定。
  例はタイマ0。
- TIMER_A　：汎用タイマの指定。A と B のどちらか指定できる。
  例は A を指定。
- Timer0IntHandler　：タイマ割り込みで呼び出される関数を指定する項目。
  ここで，割り込み時の処理が記述された関数名を指定する。
  これにより割り込みが発生したとき，自動的に指定した関数が呼び出される。

割り込みを有効にする　IntEnable(ui32Interrupt)

指定した割り込みを有効にする。

```
IntEnable(INT_TIMER0A);
```

設定項目 例

- INT_TIMER0A　：例はタイマモジュール Timer0 のカウンタ TimerA による割り込みを指定する設定値。Timer0 の全幅を利用する場合も INT_TIMER0A と設定する。ワイドタイマの場合は INT_WTIMER0A のように設定する。

割り込み条件を有効にする　TimerIntEnable(ui32Base, ui32IntFlags)

指定した割り込み条件を有効にする。

```
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

設定項目 例

- TIMER0_BASE　：設定したいタイマの指定。例はタイマ 0。
- TIMER_TIMA_TIMEOUT　：有効にする割り込み条件を指定する項目。例はタイムアウトによる割り込み条件の指定。

タイマを有効にする　TimerEnable(ui32Base, ui32Timer)

指定したタイマを有効にする。

```
TimerEnable(TIMER0_BASE, TIMER_A);
```

設定項目 例

- TIMER0_BASE　：有効にしたいタイマの指定。例はタイマ 0。
- TIMER_A　：汎用タイマの指定。A と B のどちらか指定できる。例は A を指定。

割り込みで LED 点滅

ライブラリ関数　08-1

割り込み条件を初期化する　TimerIntClear(ui32Base, ui32IntFlags)

タイマ割り込み条件を初期化する。

```
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

周期的に割り込みを発生させる場合は必ず必要。
この関数を実行しない場合，最初の 1 回目しか，割り込みが発生しない。

設定項目 例

- TIMER0_BASE　：初期化したいタイマの指定。
  例はタイマ0。
- TIMER_TIMA_TIMEOUT　：初期化したい割り込み条件の指定。
  例はタイムアウトによる割り込みを初期化。

TM4C123GH6PM のタイマについて

タイマとは，マイコンのクロックパルスの回数をカウントすることで時間を計る機能です。
例題のように TM4C123GH6PM を 16MHz で動作させているとします。16MHz ということは，1 秒間に 16,000,000 個（16M = 6 × 10^{6} ）のクロックパルスが出ている，または

$$1\ \mathrm{s} \div 16{,}000{,}000\ \mathrm{Hz} = 6.25 \times 10^{-8}\ \mathrm{s} = 62.5\ \mathrm{ns}$$

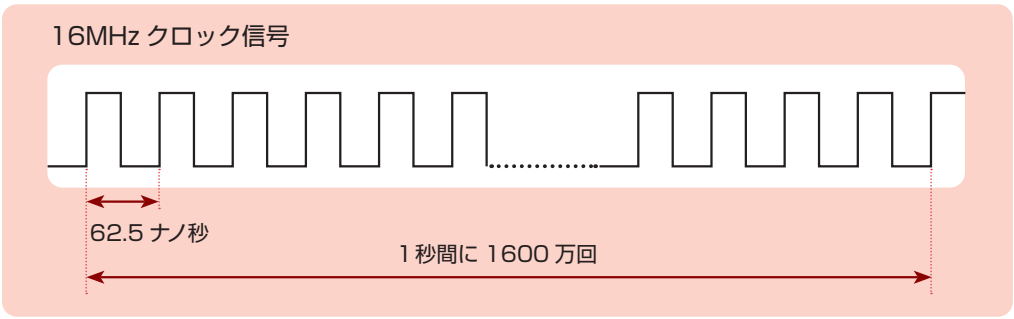
なので，62.5 ナノ秒ごとに 1 パルス出ていることになります。マイコンはこのパルスの回数をカウントすることで時間を計測しています。
例えば，1 ミリ秒（1.0 × 10^{-3} 秒）を計るとします。1 ミリ秒間にクロックパルスは，

$$1.0 \times 10^{-3} \div 62.5 \times 10^{-9} = 16{,}000$$

より，16,000 パルス出るので，クロックパルスを 16,000 回カウントすれば 1 ミリ秒経過したことになります。
　同じように，1 秒を計る場合は，

$$1.0 \div 62.5 \times 10^{-9} = 16{,}000{,}000$$

となるので，クロックパルスを 16,000,000 回カウントすればよいのです。
ライブラリ関数 TimerLoadSet( ) で周期を指定している項目は，このカウント回数を指定しているのです。



割り込みで LED 点滅

TM4C123GH6PM のタイマについて

タイマモジュール	Up/Down カウンタ	偶数 CCP Pin	奇数 CCP Pin
Timer0 (16/32-Bit)	TimerA	T0CCP0	-
	TimerB	-	T0CCP1
Timer1 (16/32-Bit)	TimerA	T1CCP0	-
	TimerB	-	T1CCP1
Timer2 (16/32-Bit)	TimerA	T2CCP0	-
	TimerB	-	T2CCP1
Timer3 (16/32-Bit)	TimerA	T3CCP0	-
	TimerB	-	T3CCP1
Timer4 (16/32-Bit)	TimerA	T4CCP0	-
	TimerB	-	T4CCP1
Timer5 (16/32-Bit)	TimerA	T5CCP0	-
	TimerB	-	T5CCP1
WTimer0 (32/64-Bit)	TimerA	WT0CCP0	-
	TimerB	-	WT0CCP1
WTimer1 (32/64-Bit)	TimerA	WT1CCP0	-
	TimerB	-	WT1CCP1
WTimer2 (32/64-Bit)	TimerA	WT2CCP0	-
	TimerB	-	WT2CCP1
WTimer3 (32/64-Bit)	TimerA	WT3CCP0	-
	TimerB	-	WT3CCP1
WTimer4 (32/64-Bit)	TimerA	WT4CCP0	-
	TimerB	-	WT4CCP1
WTimer5 (32/64-Bit)	TimerA	WT5CCP0	-
	TimerB	-	WT5CCP1

TM4C123GH6PM のタイマ構成

さて，TM4C123GH6PM には Timer0 ～ Timer5，WTimer0 ～ WTimer5 の計 12 個のタイマモジュールが用意されています。また，各タイマモジュールは TimerA，TimerB と呼ばれる2つのタイマ（カウンタ）から構成されています。

Timer0 ～ Timer5 の６つのタイマモジュールは，16/32 ビットタイマと呼ばれており，TimerA，TimerB の 16 ビットカウンタ2つから構成される 32 ビットのタイマとなっています。この 32 ビットというのは，タイマがカウントできる最大値を示しています。32 ビットは

$$2^{32} = 4{,}294{,}967{,}296$$

なので，最大で 4,294,967,296 回カウントできることになります。これは，16MHz のクロックパルスの場合，

$$4{,}294{,}967{,}296 \times 62.5 \times 10^{-9} \fallingdotseq 268.4$$

より，最大で約 268 秒，つまり約４分半まで計ることができます。

各タイマを構成している TimerA，TimerB のふたつのカウンタは，半分の幅（Timer0 ～ Timer5 の場合は 16 ビット）を持った2つのタイマとして使うこともできます。16 ビットのカウンタは，

$$2^{16} = 65{,}536$$

なので，最大で 65,536 回カウントすることが可能です。これは，16MHz のクロックパルスでは，

$$65{,}536 \times 62.5 \times 10^{-9} \fallingdotseq 4.1 \times 10^{-3}$$

より，約４ ミリ秒まで計測可能なことになります。１秒を計ることができないので，課題 08-1 に直接使うことはできませんが，使用可能なタイマの数を倍に増やすことができます。

WTimer0 ～ WTimer5 の６つのタイマモジュールは，32/64 ビットタイマ，あるいはワイドタイマと呼ばれており，Timer0 ～ Timer5 の倍のビット数を持っています。つまり，ワイドタイマは TimerA，TimerB の 32 ビットカウンタ2つから構成される 64 ビットのタイマとなっています。ひとつのワイドタイマの TimerA，TimerB をそれぞれ 32 ビットタイマとして使う場合，16MHz のクロックパルスでは最大で約 268 秒，つまり約４分半まで計測可能です。

また，64 ビットのタイマとして使用した場合，カウントできる最大値は

$$2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$$

になります。16MHz のクロックパルスの場合，

$$18{,}446{,}744{,}073{,}709{,}551{,}616 \times 62.5 \times 10^{-9} \fallingdotseq 1{,}152{,}921{,}504{,}607$$

より，約１兆1500億秒，すなわち３万年以上の計測が理屈の上では可能です。実際にはマイコンのハードウェア的な寿命がこれよりずっと早く来てしまうでしょう。また，長時間の測定では誤差も無視できなくなります。本書のマイコンボードで使われている 16MHz 発振子の精度は± 0.005% とされており，１日あたり数秒の誤差が発生する可能性があります。

割り込みで LED 点滅

コーディング　08-1

フローチャートを元に，ソースを記述してください。ソースが完成したら，実行して動作を確認しましょう。
以下に解答例ソースを示します。解答例やサンプルソースを参考に，皆さんで工夫してみてください。

step08-1.c

CD-ROM の「サンプルソース」フォルダに、各ステップの c ファイルを収録しています

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"

// LED 点灯 / 消灯用マクロ
#define ON  0x00
#define OFF 0xFF
// LED 点灯パターン用変数
char State;

// 割り込み処理
void Timer0IntHandler(void) {
        // タイマリセット
        TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
        // ビット反転
        State = ~State;
        // LED を ON/OFF にする
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_0, State);
}

// LED を周期的（1 秒）に ON/OFF させる
void main(void) {
        // 動作クロックの設定
        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

        // 使用するペリフェラルの有効化
        // : LED 用に I/O ポート D を使用
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
        // : タイマモジュール 0 を使用
        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

        // PD0 を出力タイプに設定
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_0);

```

```
41          // タイマの設定
42          // : タイマモジュール 0 を Full-width(32 ビット ) の周期カウンタで使用
43          TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
44          // : タイマの割り込みが発生する間隔を設定
45          TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());
46          // : 割り込み時に呼び出す処理（関数）を指定
47          TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0IntHandler);
48          // : 割り込みを有効化
49          IntEnable(INT_TIMER0A);
50          // : タイムアウトでの割り込みを有効化
51          TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
52          // : タイマを有効化
53          TimerEnable(TIMER0_BASE, TIMER_A);
54          State = OFF;
55          // LED を OFF にする
56          GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_0, State);
57
58          // 無限ループ
59          while (1) {
60          }
61  }
```

割り込みで LED 点滅

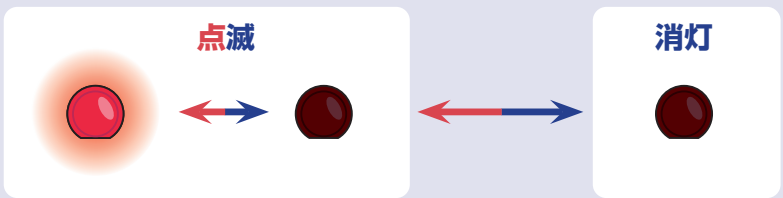
課題 08-2

LED の点滅周期を以下のようにスイッチ（PSW1）で制御してみましょう。

解答は，巻末の解答例集参照

課題 08-3

この課題は応用課題です。時間に余裕がある場合はチャレンジしてみてください。
2つのタイマを使い，LED が点滅と消灯を繰り返すようにしてみましょう。
1 つのタイマで点滅周期を，もう 1 つのタイマで点滅と消灯の周期を制御するようにします。



解答は，巻末の解答例集参照