割り込み処理 レポート　3I44 吉髙 僚眞

目的

割り込みとは、作業中に外部から要求された別の作業を行い、終了後に元の作業に戻ることをいう。組み込みシステムではリアルタイムな作業が要求されるため、この手法がとられることが多い。ここでは、INT0,INT1 を利用した外部割り込み、およびカウンタを利用した定間隔割り込みについて学習する。

実験方法/結果

3.外部割込

(1) ピン配置

外部割り込み端子 INT0 は 4 番ピンに配置され、PD2 と共用している。

(2) 割り込み許可（ステータスレジスタ SREG I ビット）

SREG レジスタの I ビットは割り込み機能全般を許可するビットである。割り込みを使用するには I ビットを 1 にしておく必要がある。I ビットは割り込みが生じたときに 0 （無効）になるので、（処理終了後）RETI 命令によって 1 （有効）に戻す。また、 SEI 命令で 1 に、 CLI命令で 0 にすることもできる。（C 言語では sei()、cli()である）

(3) 外部割り込み要因（外部割り込み制御レジスタ A EICRA）

INT0 端子の割り込み要因を設定する。例えば、立ち下りエッジを割り込み要因として検出するには、ISC01,ISC00 を、それぞれ1, 0 にすればよい。（立ち下がりエッジとは INT0 端子が High レベルから Low レベルに変化することをいう）

(4) 外部割り込みマスク（外部割り込みマスクレジスタ EIMSK）

割り込みを使用するには、割り込み機能全般の許可ビット（SREG の I ビット）の他に、割り込みデバイス毎にも割り込みを有効にする必要がある。INT0 外部割り込みでは INT0 ビット（ 1 ビット目）を1にする。検出方法は EICRA のISC01, ISC00ビットで設定する。

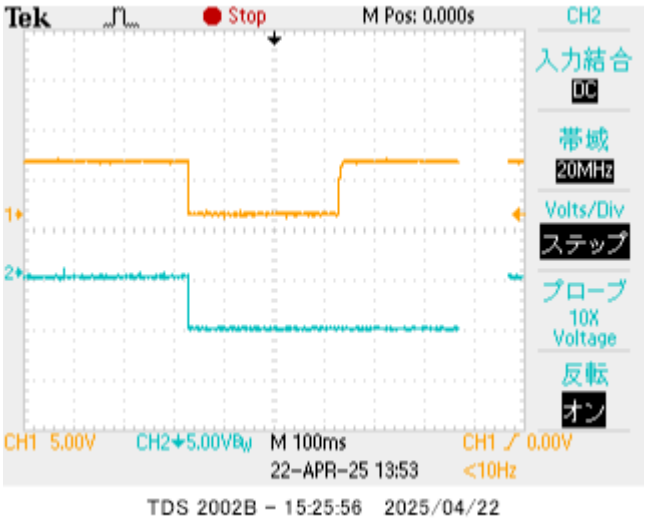
3.4 サンプルプログラムの実行

プログラム

```
#include <asf.h> //int.c
void interrupt_init(void);
void io_init(void);
int main (void)
{
    io_init(); //IO ポート設定
    interrupt_init(); //外部割り込み設定
    sei(); //割り込み機能全体を許可
    while(1); //通常処理
```

```
    return 0;
}
ISR(INT0_vect) //INT0 割り込みサービスルーチン
{ //INT0_vect は割り込みベクタ番号
    cli(); //割り込み機能全体を禁止
    PORTB ^= 0b00100000;
    // +---------- PB5 の出力を反転
    sei(); //割り込み機能全体許可
    return;
}
void interrupt_init(void) //外部割り込み設定
{
    EICRA = 0b00000010; //割り込み制御レジスタ
    // ++----- ISC01,ISC00 立ち下がりエッジ検出
    EIMSK = 0b00000001; //割り込みマスクレジスタ
    // +----- INT0 割り込み有効
    return;
}
void io_init(void) //IO ポート設定
{
    DDRB = 0b00100000;
    // +---------- PB5 を出力に設定
    DDRD = 0b00000000;
    // +------- PD2 を入力に設定（INT0,PD2 共用端子）
    PORTD = 0b00000100;
    // +------- PD2 をプルアップ（INT0,PD2 共用端子）
    return;
}
```

波形



- CH1・・・INT0
- CH2・・・PB5

サンプルプログラムを実行したところボタンを押すことでLEDの状態が反転することが確認できた。 また上記の波形からボタンを押し始めると同時に、LEDへの出力が変わっていることがわかる。

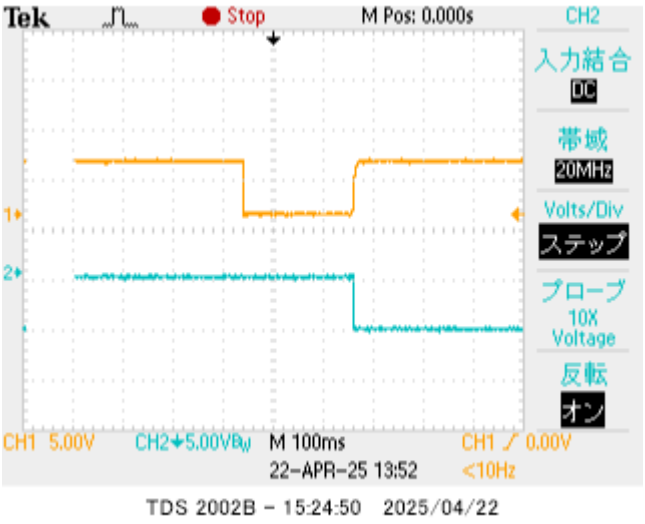
3.5 検出エッジの変更

プログラム

```
void interrupt_init(void)
{
    EICRA = 0b00000011; //割り込み制御レジスタ ここを変更
    // ++------ ISC01,ISC00 立ち上がりエッジ検出
    EIMSK = 0b00000001;
    // +------ INT0 割り込み有効
    return;
}
```

Table12-2より立ち上がり(rising edge)の時は、ISC01, ISC00を 1, 1にすればよいとわかる

波形



実行したところボタンを押離すときLEDの状態が反転することが確認できた。 また上記の波形からボタンを離し始めると同時に、LEDへの出力が変わっていることがわかる。

3.6 スイッチの追加

プログラム

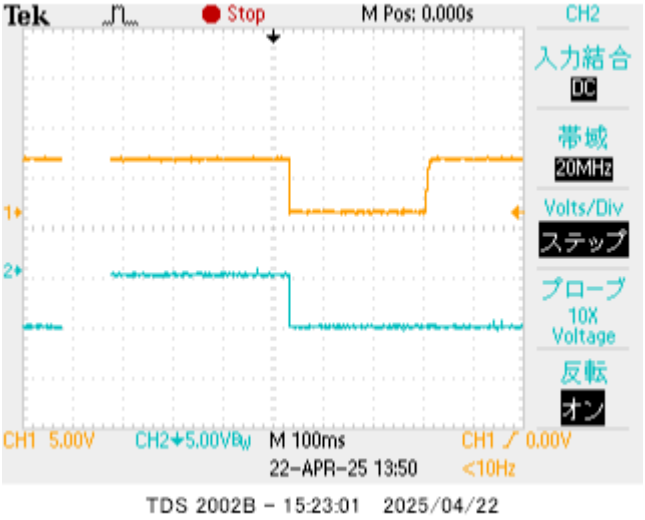
```
#include <asf.h> //int.c
void interrupt_init(void);
void io_init(void);
int main (void)
{
```

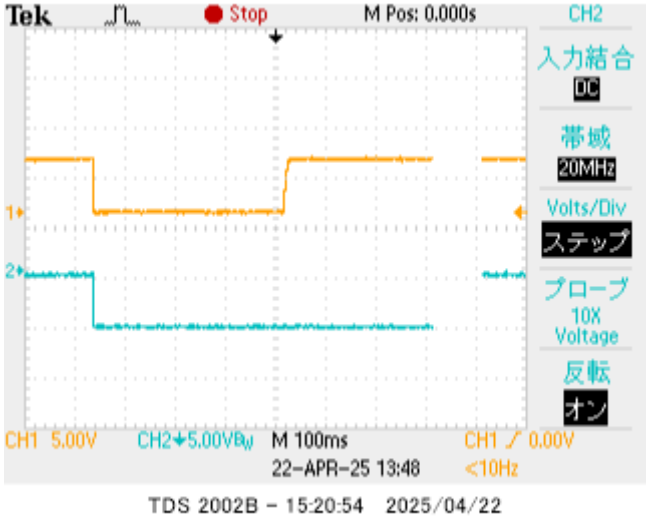
```
    io_init(); //IO ポート設定
    interrupt_init(); //外部割り込み設定
    sei(); //割り込み機能全体を許可
    while(1); //通常処理
    return 0;
}
ISR(INT0_vect) //INT0 割り込みサービスルーチン
{ //INT0_vect は割り込みベクタ番号
    cli(); //割り込み機能全体を禁止
    PORTB ^= 0b00100000;
    // +---------- PB5 の出力を反転
    sei(); //割り込み機能全体許可
    return;
}
ISR(INT1_vect)
{
    cli(); //割り込み機能全体を禁止
    PORTB ^= 0b00100000;
    // +---------- PB5 の出力を反転
    sei(); //割り込み機能全体許可
    return;
}

void interrupt_init(void) //外部割り込み設定
{
    EICRA = 0b00001010; //割り込み制御レジスタ
    // ++----- ISC11, ISC10 ISC01,ISC00 立ち下がりエッジ検出
    EIMSK = 0b00000011; //割り込みマスクレジスタ
    // +----- INT0 割り込み有効
    return;
}
void io_init(void) //IO ポート設定
{
    DDRB = 0b00100000;
    // +---------- PB5 を出力に設定
    DDRD = 0b00000000;
    // +------- PD2 を入力に設定（INT0,PD2 共用端子）
    PORTD = 0b00001100;
    // +------- PD2 をプルアップ（INT0,PD2 共用端子）
    return;
}
```

Table13-1よりINT1を使用するし、立ち下り(The falling edge)を検出したい時は、ISC11, ISC10をそれぞれ1, 0にすれ場よいとわかる。

波形





- 左 INT0の時
- 右 INT1の時

波形を見るとどちらもボタンを押した時に、LEDの出力が反転していることがわかる。

4.タイマ／カウンタモジュールを用いた定間隔割り込み

4.1データシートの読み取り

(1) 概要

タイマ／カウンタ 0 は8ビットのモジュールである。

(2) クロックソース

動作クロックは TCCR0B の CS02~CS00 ビットで設定された比で分周された後（プリスケーラ）、タイマ／カウンタ 0 に入力される。たとえば、この値を 011 とすると動作クロック（clkI/O）が 20MHz（周期 0.05us）のとき、3.2us (0.05 × 64) ごとにカウントアップする。

(3) CTCモード

WGM02 WGM01 WGM00 ビットが0, 1, 0のとき CTC（コンペアマッチ）モードとなる。OCR0A はタイマ／カウンタ 0 (TCNT0)の 値を示す。カウンタの値は OCR0A と一致したら、0 に戻る。また、このとき OC0A 割り込みフラグ（OCF0A）がセットされ、割り込みが発生する。（２）の例ではOCR0A 32 にをセットすると 32us 毎に割り込みが発生することになる。 注、コンペアマッチとは設定した値とカウンタの値とが一致する条件をいう。

(4) 割り込みマスク

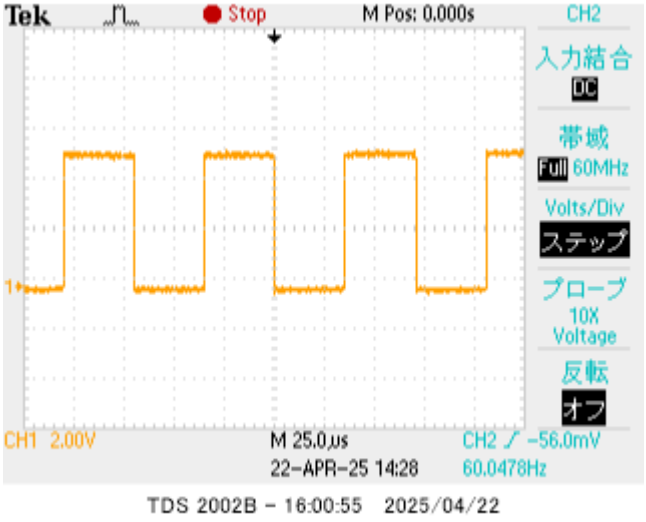
OCIE0Aビット（1ビット目）が 1 のとき、タイマ／カウンタ 0 コンペアマッチ A の割り込みが有効となる。

4.2 サンプルプログラムの実行

プログラム

```c
#include <asf.h> //ctc.c
void io_init(void);
void timer0_ctcmode_init(uint8_t top);
int main (void)
{
    io_init(); //IO ポート設定
    timer0_ctcmode_init( 10 ); //タイマ／カウンタ ０ＣＴＣモード割り込み設定
    sei(); //割り込み許可
    while(1); //通常処理
    return 0;
}
ISR(TIMER0_COMPA_vect) //タイマ／カウンタ 0 割り込みサービスルーチン
{ //TIMER0_COMPA_vect は割り込みベクタ
    cli(); //多重割り込み禁止
    PORTD ^= 0b00000001;
    // +---- PD0 の出力を反転
    sei(); //割り込み許可
}
void timer0_ctcmode_init(uint8_t top) //タイマ／カウンタ 0 ＣＴＣモード割り込み設定
{
    // タイマ／カウンタ 0 比較レジスタ A
    OCR0A = top; //タイマ／カウンタ 0 最大値
    // タイマ／カウンタ 0 制御レジスタ A
    TCCR0A = 0b00000010;
    // || ++ WGM01 WGM00 CTC モード
    // ++--------COM0A1 COM0A0 OC0A 端子出力 OFF
    // タイマ／カウンタ 0 制御レジスタ B
    TCCR0B = 0b00000011;
    // | |+++ CS02 CS01 CS00 プリスケーラ 64 分周
    // | +--- WGM02 CTC モード
    // +-------- FOC0A 0(CTC モード)
    // タイマ／カウンタ 0 割り込みマスクレジスタ
    TIMSK0 = 0b00000010;
    // +- OCIE0A コンペアマッチ A 割り込み有効
}
void io_init(void) //IO ポート設定
{
    DDRD = 0b00000001;
    // +----- PD0 を出力に設定
    PORTD = 0b00000001;
    // +----- PD0 に 1 を出力（動作テスト）
}
```

波形



4.1のデータシートの読み取り(2)より3.2 µ秒ごとにカウントアップする。 32µ秒を計測したいので`32 ÷ 3.2 = 10`よりカウントトップを10に設定する。

4.3処理間隔の変更

1. 20us 毎に H,L を繰り返すように変更しなさい。

PD0 から 20us 毎に H,L を繰り返す信号を出力するプログラムに変し、オシロスコープで動作を確認しなさい。（パラメータ OCR0A,CS02,CS01CS00 を変更する）。また High レベル幅、LOW レベル幅を確認しなさい。

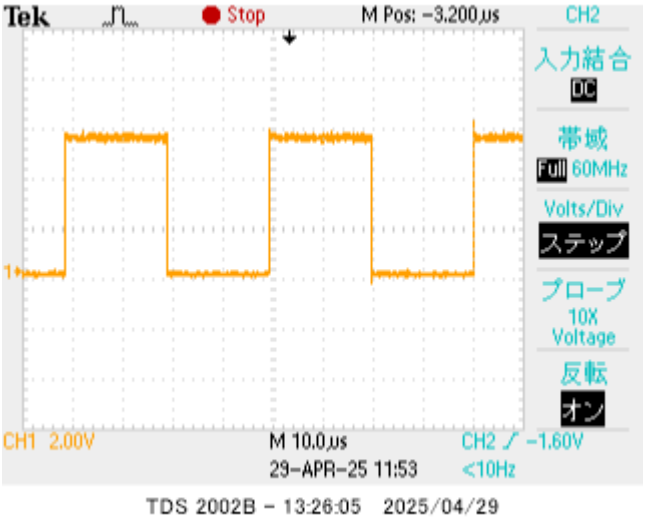
プログラム(4.2より変更した部分を抜粋)

```
#include <asf.h> //ctc.c
void io_init(void);
void timer0_ctcmode_init(uint8_t top);
int main (void)
{
    io_init(); //IO ポート設定
    timer0_ctcmode_init( 50 ); //ＣＴＣモード割り込み設定 <== ここを変更 #1
    //･･･
}
//･･･
void timer0_ctcmode_init(uint8_t top) //タイマ／カウンタ 0 ＣＴＣモード割り込み設定
{
    // ･･･
    // タイマ／カウンタ 0 制御レジスタ B
    TCCR0B = 0b00000010;  //<== ここを変更 #2
    // | |+++ CS02 CS01 CS00 プリスケーラ 8 分周
    // | +--- WGM02 CTC モード
    // +------- FOC0A 0(CTC モード)
    // タイマ／カウンタ 0 割り込みマスクレジスタ
    TIMSK0 = 0b00000010;
    // +- OCIE0A コンペアマッチ A 割り込み有効
}
```

20μ秒を計測したいときは分周を変える必要がある。8分周の時、`8 × 0.05 = 0.4`より0.4μ秒ごとにカウントアップする。よって、`20 ÷ 0.4 = 50`からカウンタトップは50に設定すればよい。(#1)

また、8分周を設定する際には、データシートのTable 15-9より、CS02,CS01,CS00を 0, 1, 0に設定する必要があることがわかる。

波形

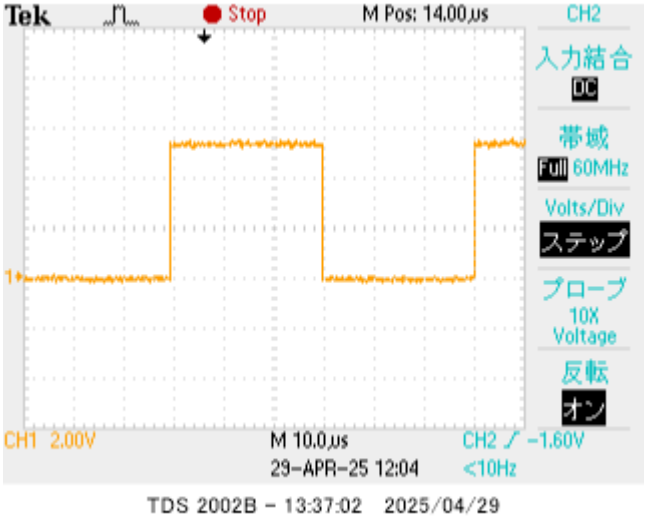


2. 同様に 30us 毎に H,L を繰り返すように変更しなさい。

プログラム(1より変更した部分のみ一部抜粋)

```
#include <asf.h> //ctc.c
void io_init(void);
void timer0_ctcmode_init(uint8_t top);
int main (void)
{
    io_init(); //IO ポート設定
    timer0_ctcmode_init( 75 ); // <==  ここを変更
    //・・・
}
```

波形



4.4並列処理

PD0 から 20us 毎、PD1 から 30us 毎に H,L を繰り返す信号を出力するプログラムを作成しなさい。PD0 にはタイマ／カウンタ 0 を、PD1 にはタイマ／カウンタ 1 （資料）を用いること。

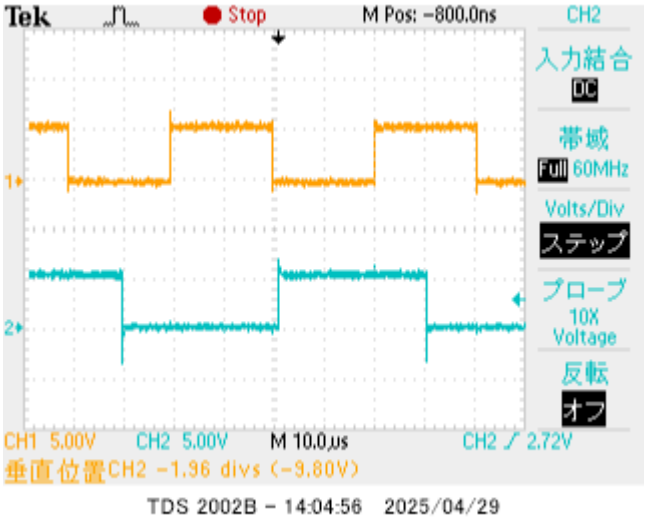
```
#include <asf.h>
void io_init(void);
void timer0_ctcmode_init(uint8_t top);
void timer1_ctcmode_init(uint16_t top);
int main (void)
{
    io_init(); //IO ポート設定
    timer0_ctcmode_init( 50 ); //タイマ／カウンタ 0ＣＴＣモード割り込み設定
    timer1_ctcmode_init( 75 ); //タイマ／カウンタ 1ＣＴＣモード割り込み設定
    sei(); //割り込み許可
    while(1); //通常処理
    return 0;
}
ISR(TIMER0_COMPA_vect) // タイマ／カウンタ 0 割り込みサービスルーチン
{ //TIMER0_COMPA_vect は割り込みベクタ
    cli();
    PORTD ^= 0b00000001;
    sei();
}
ISR(TIMER1_COMPA_vect) // タイマ／カウンタ 1 割り込みサービスルーチン
{ //TIMER1_COMPA_vect は割り込みベクタ
    cli();
    PORTD ^= 0b00000010;
    sei();
}
void timer0_ctcmode_init(uint8_t top) // タイマ／カウンタ 0 ＣＴＣモード割り込み設定
{
    OCR0A = top;
    TCCR0A = 0b00000010;
    TCCR0B = 0b00000010;
```

```
    TIMSK0 = 0b00000010;
}
void timer1_ctcmode_init(uint16_t top) // タイマ／カウンタ 1 ＣＴＣモード割り込み設定
{
    OCR1A = top;
    TCCR1A = 0b00000000;
    TCCR1B = 0b00001010;
    TIMSK1 = 0b00000010;
}
void io_init(void) // IO ポート設定
{
    DDRD = 0b00000011;
    // ++------ PD0,PD1 を出力に設定
    PORTD = 0b00000011;
    // ++------ PD0,PD1 に 1 を出力（動作テスト）
}
```

INT0については先ほどのプログラムと同様に設定した。 INT1については以下のように設定した。

- データシートの記述には`WGM13:0 = 4 or 12`とあったので、4にするためには`WGM13, WGM12, WGM11, WGM10 = 0, 1, 0, 0`と設定した。
- 4に設定したときはOCR1Aをtopのレジスタに、12のときICR1をtopのレジスタにする必要があることがデータシートから読み取れた。
- 8分周にしたいのでTable16-5より`CS12, CS11, CS10は 0 1 0`に設定した。
- マスクはOCIR1A(TIMSK1の1ビット目)を1にすることで割り込みを有効かできることが読み取れた。

波形



- CH1 PB0
- CH2 PB1

この波形を見るとPB0は20µs、PB1は30µsぴったりの周期になっていることがわかる。

4.5 delayルーチンとの比較

4.4 のプログラムを_delay_us()を用いて作成してみよ。（割り込みは用いない。util/delay.hをインクルードせよ。）

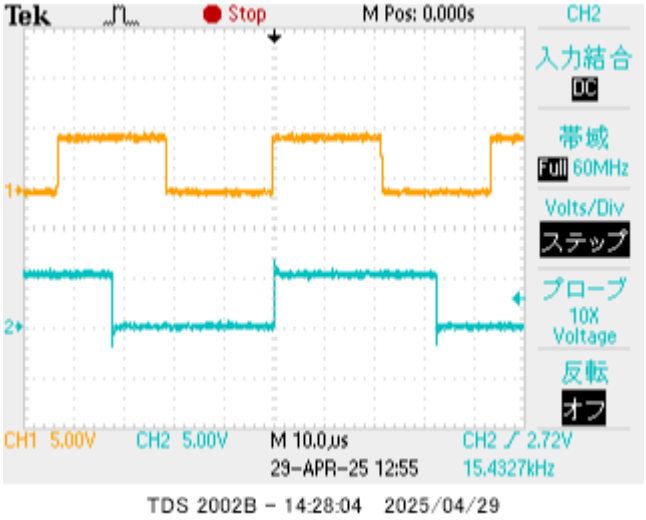
プログラム

```
#include <asf.h>
#define F_CPU 20000000UL
#include <util/delay.h>

void io_init(void);
int main (void)
{
    int counter_pd0 = 0;
    int counter_pd1 = 0;
    io_init(); //IO ポート設定
    while(1){
        _delay_us(10);
        counter_pd0++;
        counter_pd1++;
        if(counter_pd0 >= 2){
            PORTD ^= 0b00000001;
            counter_pd0 = 0;
        }
        if(counter_pd1 >= 3){
            PORTD ^= 0b00000010;
            counter_pd1 = 0;
        }
    }
    return 0;
}

void io_init(void) // IO ポート設定
{
    DDRD = 0b00000011;
    // ++------ PD0,PD1 を出力に設定
    PORTD = 0b00000011;
    // ++------ PD0,PD1 に 1 を出力（動作テスト）
}
```

波形



上記のようなプログラムで行った場合上のような波形となった。 この波形と4.4の波形を比較すると20µs, 30µsぴったりではなく遅れていることがわかる。

考察

1. タイマ／カウンタ 0 およびタイマ／カウンタ 1 ではそれぞれ最大何秒間隔で割り込みが可能か計算してみよ。

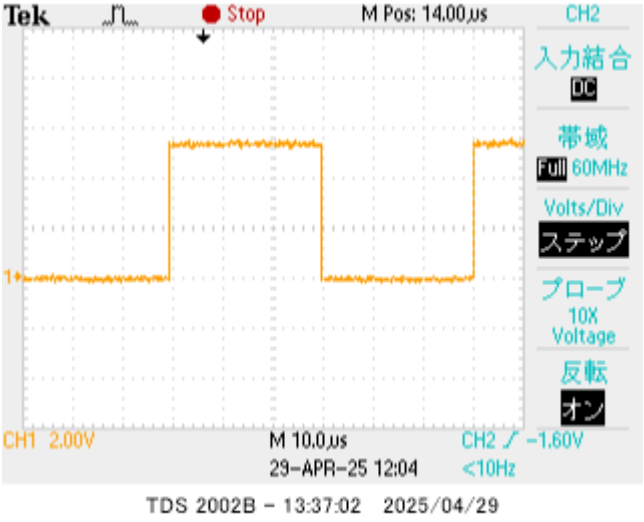
   ### タイマ/カウンタ0の場合

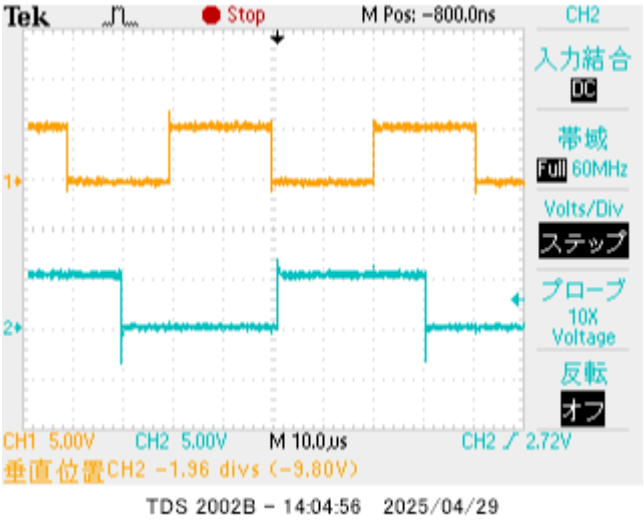
   - 最大の分周は1024なので`1024×0.05=51.2`より、51.2µsごとにカウントアップすることができる。
   - タイマ/カウンタ0は8ビットのカウンタなので、`51.2×255=13,056`より最大13.056ミリ秒の間隔で割り込みができる。

   ### タイマ/カウンタ1の場合

   - 最大の分周は1024なのでカウンタ0と同様に51.2µsごとにカウントアップすることができる。
   - 16ビットのカウンタなので、`51.2×65,535=3,355,392`より最大3.355392秒の間隔で割り込みができる。

2. 単一処理（4.3）を行った場合と並列処理（4.4）を行った場合において、間隔（HIGH レ ベル幅、LOW レベル幅）には違いがあるだろうか。





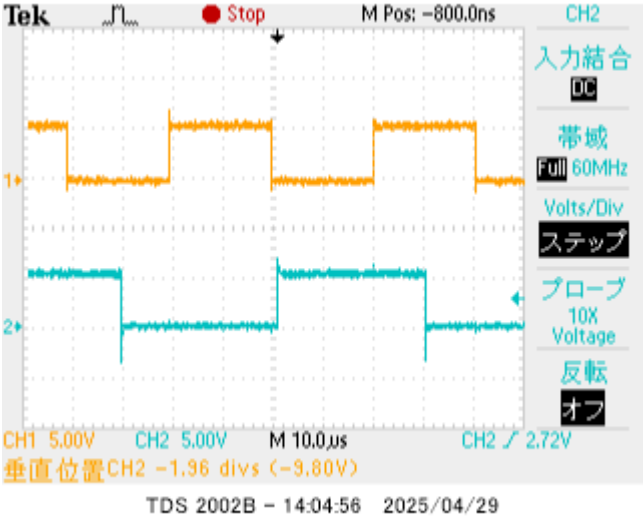
- 左 4.3の単一処理時
- 右 4.4の並列処理時

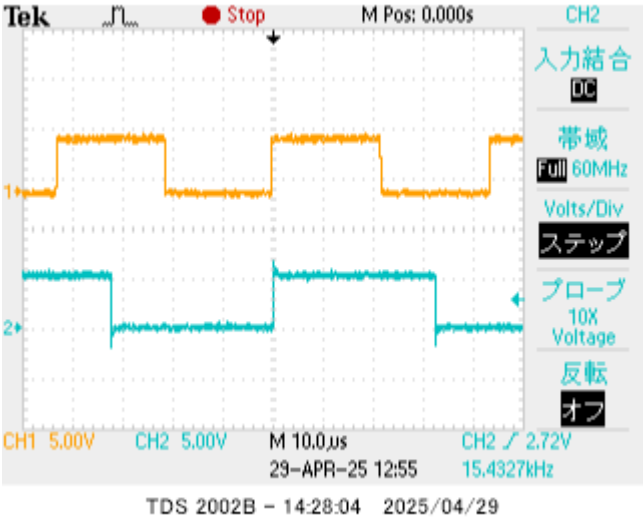
単一処理時は完璧にHIGHレベルでも、LOWレベルでも30µsで動いている。 並列処理時は左上のCH1ではLOWが少しだけ幅が長くなっていたりしてずれやすくなってしまっている。

3. 1 つのタイマ／カウンタを用いて、LED の点滅間隔が SW0 を押すと約 0.8s、SW1 を押 すと約 0.4s と切り替えるにはどうしたらよいか。（使用タイマ／カウンタ、パラメータ設 定などを検討）

    - 1つのカウンターで2つの設定を使う。
    - 0.4sと0.8s間隔でタイマ/カウンタを使用したいので、考察の1より16ビットのタイマ/カウンタ1を使用する。
    - 1024分周の時、`1024 × 0.05 = 51.2`より、51.2µsごとにカウントアップすることがわかる。
    - `400,000µs (0.4s) ÷ 51.2µs = 7,812.5`よりトップを7813に設定すると、約0.4sになる。
    - `800,000µs (0.4s) ÷ 51.2µs = 15,625`よりトップを15,625に設定すると、約0.8sになる。
    - SW0を押したときにカウンタトップを7813に設定する。
    - SW1を押したときにカウンタトップを15,625に設定する。

4. 定間隔で処理を行う（開始する）とき、delay ルーチンを用いた場合とタイマ／カウンタ を用いた場合の違いを説明しなさい。



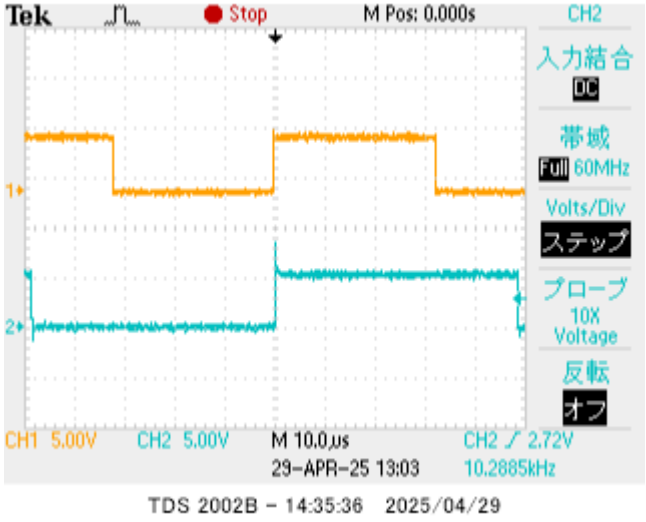


- 左 4.4でタイマ/カウンタを使用した時
- 右 4.5でdelayルーチンを使用した時

delayルーチンを用いたプログラムはタイマ/カウンタを使った場合に比べて遅れてしまう。

4.5のプログラムを以下のように変更して、波形を見ると以下のようになった。

```
// 省略
_delay_us(1);
// ・・・
if(counter_pd0 >= 20){
// ・・・
}
if(counter_pd1 >= 30){
// ・・・
}
```



プログラムの内容的には同じように20µsと30µs周期で動くようにしているはず、波形を見ると、10µsほど遅れていることがわかる。このことから、_delay_us()以外の部分での処理にかかる分だけ、遅れてしまうことがわかる。

＜考察追加（外部割り込み）＞

5. EIFR レジスタの INTF0 の記述を訳し、INTF0 の役割を説明しなさい。

> INT0ピンの割り込みがあったときに、INTF0の値が1になる。もし、SREGのI-bitとEIMSKのINT0 bitが1になっているとき、MCUは対応する割り込みベクターの処理に移動する。割り込みルーチンが実行されたらフラグがクリアされる。または、1を書き込むことでフラグをクリアすることもできる。

データシートからこのような内容が読み取れたので、INTF0は割り込みがきたことを管理するフラグの役割があると考えられる。

6. スイッチを扱う場合、チャタリングを考慮する必要がある。チャタリングとは何か説明 しなさい。

スイッチなどで、接触状態になる際に短い間でON/OFFを繰り返す現象のこと。