EduCake:　割り込みとイベント

1. 割り込みやイベント紹介

前の章では、86Duino Educake と外部周辺機器（デジタル IO、アナログ入出力、センサ、モータ等）のインターフェイスについて話しました。本章では反応 が良く効率的なアプリケーションの開発に重要な割り込みとイベントについて話します。

以前に digitalRead/Write()、analogRead/Write()そして delay()機能を使用してLEDの明るさを変更したり予め設定 された時間間隔でアナログデータを読み取りましたが、これはポーリングと呼ばれます。ポーリングが働いている状態では、継続的な ループ内で delay()機能を用いて一つまたは複数の入力データを読み取り捕捉する事は最適かつ 効率的ではありません。プログラムループが実行される間 CPU やメモリ消費量が高くなるのでコントローラに不必要な負荷を掛けてしまいます。

複数の入力データを読み取る場合、ポーリングより割り込みとイベントを用いたアプリケーションが効率的です。外部イベントが発生する時にシス テムが CPU の割り込みから仕事 を受けて対応コードを実行します。つまりイベント事に必要なコードだけ実行しますので、CPU やメモリの負荷を減らせます。こう言ったプロセスは PC や CPU（Vortex86EX や 86Duino Educake も含む）の基本デザインであります。

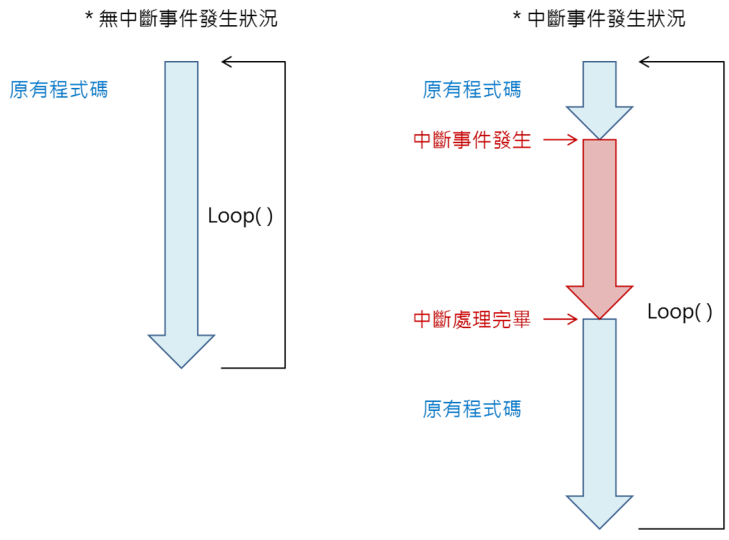


図１： 典型的な CPU の割り込みプロセス

図１で示すように割り込み作業に対応していない間は CPU が他の作業を行えます。割り込みが発生した時に作業中の仕事を一時停止し、割り込み ハンドラに入ってタスクを実行します。割り込み作業が終わった後停止した作業を再開します。割り込み対応コードが停止したプロセスに影響あり ますので、なるべく最短プロセスで終えるように最適化が必要です。

割り込みには内部割り込みと外部割り込みの 2 種類があります。内部割り込みは CPU の組み込み機能（例えばある設定時間間隔でトリガを起こす 内臓タイマー）の一部です。例えば入力 データを 30 秒間隔で読み取るケース、delay()機能を使用する場合には CPU が delay()を待っている間に他の作業を行えませんが、内臓タイ マーを使用したらその 30 秒の間他の作業を行えます。その上内臓タイマーが起動されるイベントの時間間隔は delay()機能より正確です。

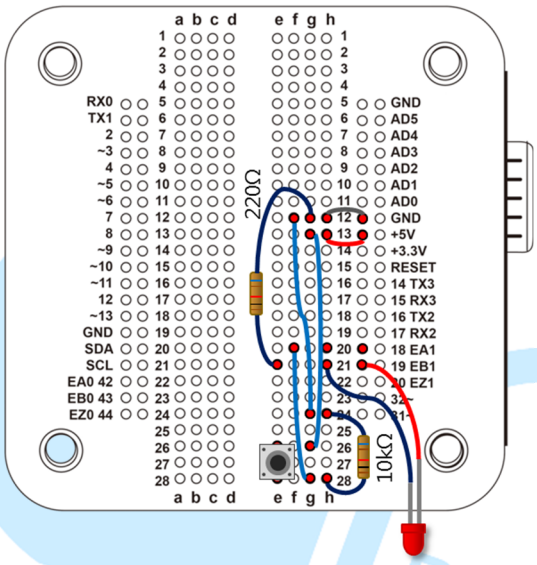
外部割り込みは名前の通り CPU 外のソース（例えば GPIO ピンの電圧変化等）から起動されます。適当な回路と器具があれば、外部機器での割 り込みとイベントの発生ができ、効率的な開発環境を作れます。外部割り込みについては本章の後半でより詳しく触れます。

割り込みの他に 86Duino Educake は pulseIn()機能でも外部機器の変化を検出出来ます。pulseIn()機能は電圧の変化や信号ピンの HIGH/LOW 状態時間間 隔を検出できます。本章の後半で pulseIn()機能の演習を行い割り込みの結果と比べてみましょう。

これから割り込みや pulseIn()機能の使い方を勉強します。

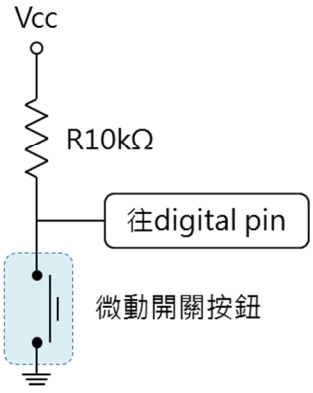
2. 課題 1 ―attachInterrupt()と deattachInterrupt()

始めの課題では 86Duino Educake の割り込み使用方法を勉強します。次の回路図を参照してください。

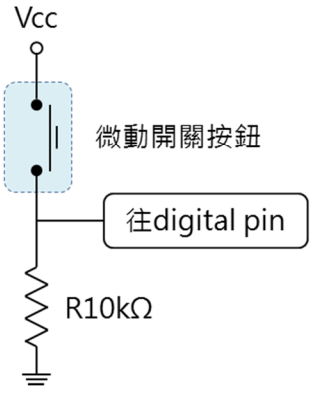


ボタン回路の実現には以下の図の様な二つの異なる方法があります。これらの回路はボタンが押されていない時が電圧 LOW 状態であり、ボタンを押したら電圧が HIGH 状態に変わるように設計されています

按鈕一般未按下時，控制板偵測到HIGH
(Normal HIGH)



按鈕一般未按下時，控制板偵測到LOW
(Normal LOW)

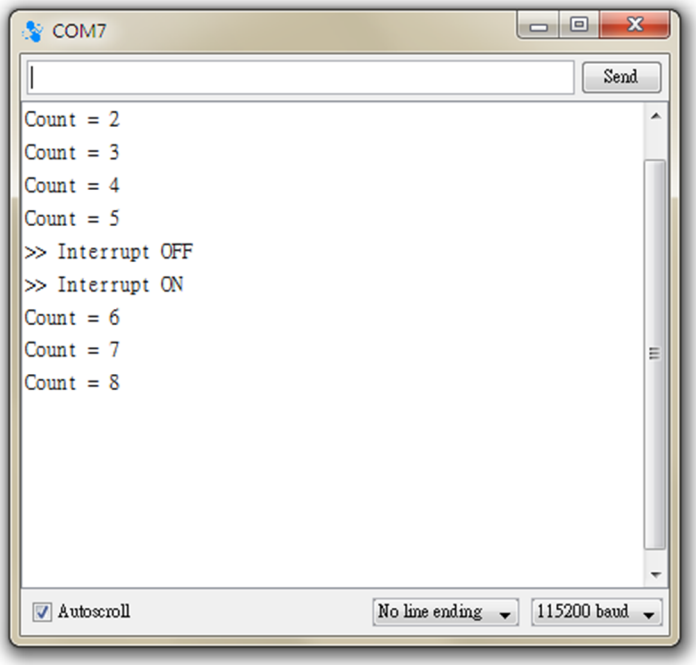


86Duino の IDE を起動して下記のコードを入力してください：

```
nt BTN_pin = 3;// Normal LOW, pin 18 = interrupt 3
int LED_pin = 19;
volatile int state = LOW;
int count = 0;

void setup()
{
  Serial.begin(115200);// Configure Serial port
  pinMode(LED_pin, OUTPUT);// Configure signal pin
  digitalWrite(LED_pin, LOW);// Initialize LED to OFF

  // attach interrupt to signal pin
  attachInterrupt(BTN_pin, InterruptHandler, RISING);
}

void loop()
{
  if(Serial.available()){// check Serial Port for data
    char data = Serial.read();
    if(data == 'A'){// when an "A" is detected

      // attach interrupt for the signal pin
      attachInterrupt(BTN_pin, InterruptHandler, RISING);
      Serial.println(">> Interrupt ON");
    }
    else if(data == 'B'){// when a "B" is detected

      // detach interrupt from signal pin
      detachInterrupt(BTN_pin);
      Serial.println(">> Interrupt OFF");
    }
  }

  digitalWrite(LED_pin, state);// turn LED ON
}

void InterruptHandler()

I{
state = !state;// Change LED state
  count++;// increment counter by 1
  Serial.print("Count = ");
  Serial.println(count);
}
```

上記プログラムについてボタンを一度押したら割り込みが発生します。そして関連機能が LED のステータスを変更してカウンタを一つ増やします。シリアルモニターに結果を表示します。シリアルポートから文字”A”を検出したら割り

込みが ON になり文字”B“を検出したら割り込みが OFF になります。次図を参照してください。



プログラムは変数宣言から始まります。その変数は信号ピンを定め、LED ステータスとカウンタ値を初期化します。次に setup()内の attachInterrupt()機能を呼び出して割り込み番号、関連割り込みハンドラ、割り込み検出モード等のパラメータを設定します。

コードの 1 列にて「BTN_pin = 3」が書いてありますが、ボタンは GPIO の 18 ピンにつながっています。86Duino は複数の GPIO グループがあり、そしてグループによって関連割り込み番号が以下の表の様になります。(詳細はピンや割り込みテーブルを参考してください。

中斷編號	int.0	int.1	int.2	int.3	int.4	int.5
EduCake	42	43	44	18	19	20

続いて割り込みの検出モードを話します。attachInterrupt(pin, ISR, MODE)と deattachInterrupt(pin)内の pin パラメータは内部割り込み番号（物理 GPIO ピン番号ではない）を参照しています。割り込み検出モード(MODE)は下記の三つに設定できます：

- HIGH から LOW 又は LOW から HIGH への電圧変化
- RISING－LOW から HIGH への電圧変化
- FALLING－HIGH から LOW への電圧変化

回路によって上記検出モードから選択出来ます。本プログラムではボタンを押したら LOW から HIGH に変わりますので、「RISING」が使われています。

割り込みハンドラが毎回呼び出される時に LED のステータスを反対に変更してカウンタを一つ増やします。そして Serial.print()機能でシリアルモニターに結果を出力します。

loop()内ではシリアルポートから受信したデータを確認するコードがあります。”A”文字を検出したら attachInterrupt(pin, ISR, MODE)機能を呼び出し、継続して割り込みを有効とします。”B”文字を検出したら deattachInterrupt(pin)を呼び出し、割り込みを無効とします。そして loop()内の digitalWrite()機能は STATE パラメータで LED の明るさを調整します。

86Duino は IDE のアプリケーションコードを実行する間､他の割り込みイベントにも対応します（例えばシリアルポートデータの受信）。

上記課題にて attachIterrupt()と deattachInterrupt()機能は I/O ピンの割り込み検出の開始・停止で使われています。86Duino は他の割り込み対応機能もあります。interrupts()と nointerrupts()機能は割り込みのバックグラウンド対応の開始・停止で使われています。

上記コードを実行する時にボタンを一度押したのに複数のイベントが呼び出される可能性があり、この現象は Bounce と呼ばれます。アプリケーションでの Bounce 対策は一般的に知られておりハードやソフト設計で解決出来ます。

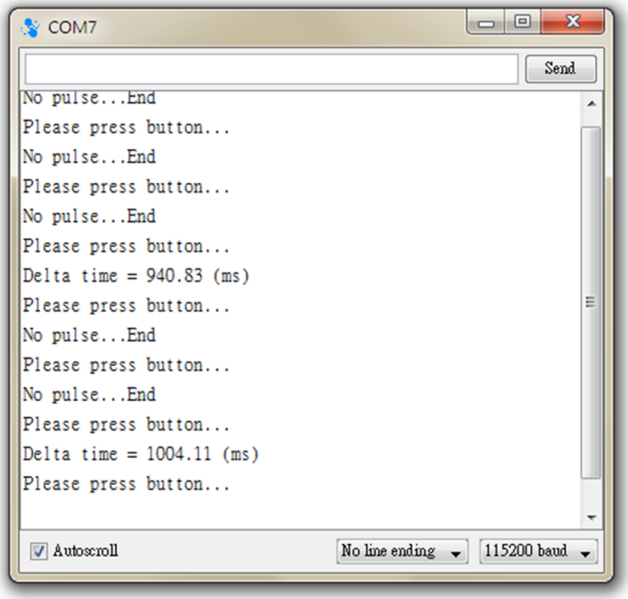
3. 課題２—pulseIn()

本課題は課題１と同じような回路で pulseIn()機能を検証します。下記コードを

```
// pulseIn Timer
// I/O pin connected to button, normal low
//  pin 18 = interrupt 3
int btn_pin = 18;

// pulseIn timeout period, in micro-second
unsigned long max_duration = 2000000;

void setup() {
  Serial.begin(115200);// initialize Serial port
  pinMode(btn_pin, INPUT);// initialize I/O pin
}

void loop() {
  // Output message to serial monitor
  Serial.println("Please press button...");

  unsigned long duration = pulseIn(btn_pin, HIGH, max_duration);

  if(duration>0){
    Serial.print("Delta time = ");
    // output interval to serial monitor, in millisecond
    Serial.print(((float)duration)/1000);
    Serial.println(" (ms)");
  }
  else{
    Serial.println("No pulse...End");
  }

  delay(2000);
}
```

86Duino IDE に入力してください。

今回のアプリケーションはボタンが押される期間を検出します。アプリケーションコードを実行する前にシリアルモニターを開いてください。「Please press button...」メッセージが表示されたら、ボタンを押して結果を見てください。2 秒の間コードがボタンが押されたかどうか検出します。2 秒後ボタンイベントが検出されない場合「No Pulse...End」メッセージをシリアルモニターに表示して、検出プロセスを繰り返します。以下の図を参照してください。



本課題では pulseIn()機能が割り込みを利用しないでボタンを押すイベントを検出します。ボタンを押すイベントは pulseIn()機能が動作している時検出されるので、コードはプログラムのメイン loop()内で連続的に動作しなければなりません。pulseIn()の呼び出し方法は二通りあります。

- pulseIn(pin, value)
- pulseIn(pin, value, timeout)

pulseIn()機能は以下の図のように動作します。

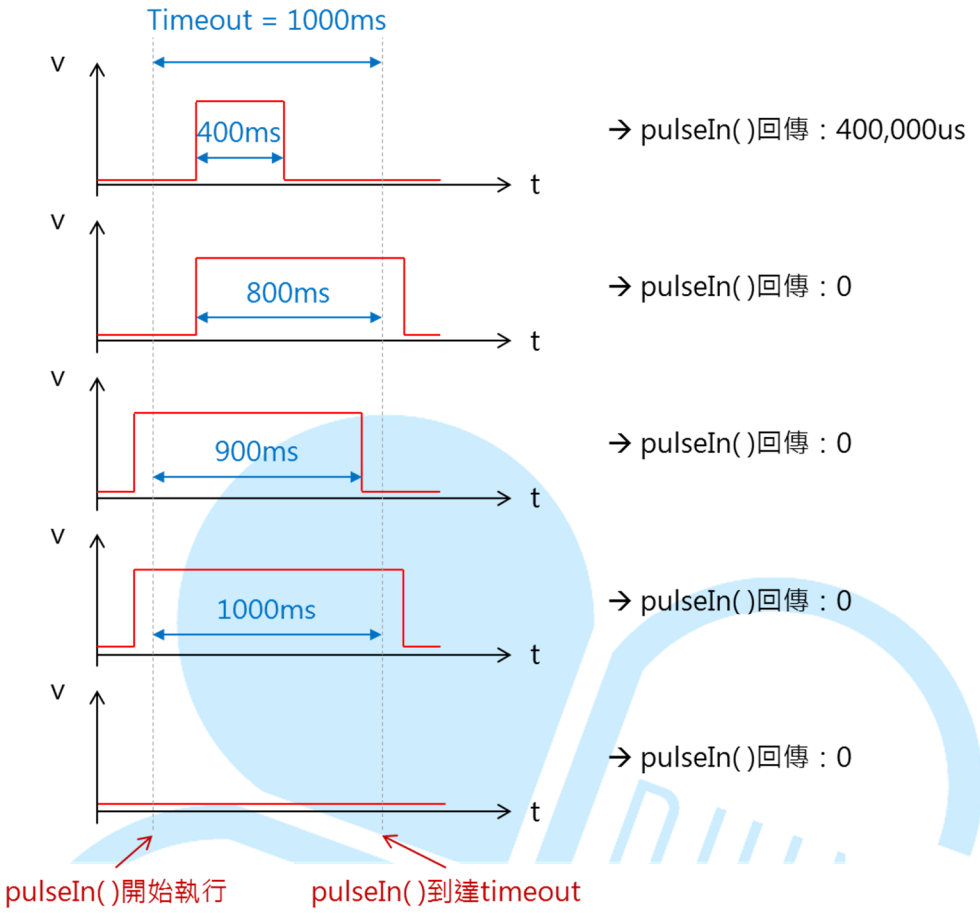


図２：pulseIn()機能リターン値

コードが電圧 HIGH 状態を検出するような例では、設定したタイムアウト期間以内に全波（波の上下がタイムアウト期間内に起こる）を検出したら、検出した値をリターンします。それ以外は０をリターンします。

上記例のタイムアウト期間は「unsigned long max_duration= 2000000」変数で設定しています。この値がマイクロ秒です。pulseIn()はこの変数を利用して 2 秒以内に信号ピンの変化を検出します。pulseIn(pin value)機能が使われる時にタイムアウト期間は 1 秒に設定されます。setup()内に pinMode(btn_pin, INPUT)機能が起動されてボタンに繋がっている信号ピンを入力信号として初期化します。プログラムのメイン loop()内の Serial.print()機能は「Please press button...」メッセージをユーザに表示します。そして下記のコードでボタンを押すイベントを検出します。

- unsigned long duration = pulseIn(btn_pin, HIGH, max_duration)

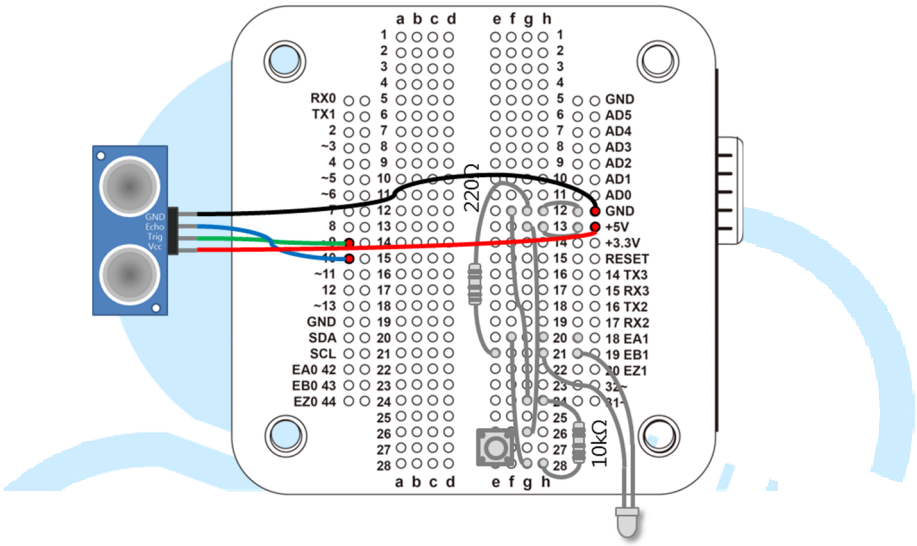
ボタンの信号ピンが通常 LOW 状態に設定されていますので、pulseIn()機能は HIGH 状態の検出に設定されます。ボタンの信号ピンが通常 HIGH 状態であれば、pulseIn()は LOW 状態の検出に設定します。pulseIn()のリターン値はマイクロ秒ですので、1000 で割ればミリ秒になります。タイムアウト値を 5 秒等にしたら、結果も変わります。

本課題で pulseIn()機能と割り込みの違いが分かります。割り込みは外部イベントにすぐに反応できますが、pulseIn()はイベント検出のために別のコードを実行しなければなりません。

上記コードに基づいて、LED を点灯してからボタンを押すまでの期間を検出するためにはどのようにコードを変更したら良いでしょうか？

4. 課題３

本課題では pulseIn()機能を使用して面白いアプリケーションを作成します。今回は以前使った回路に HC-SR04 超音波センサを接続します。いろいろな超音波ソニックセンサがありますが、ほとんどのセンサは３つ又は４つの電線が付いてます。電線の二つは 5V と GND で、残りはセンサ値のためです。下記の図を参照してください。



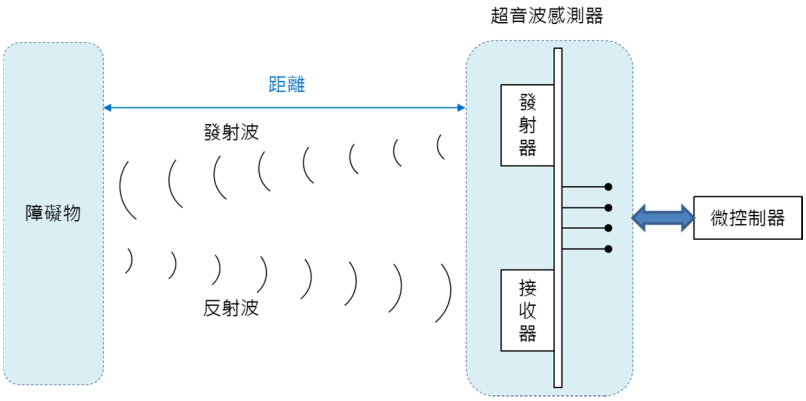
下記コードを 86Duino IDE に入力してください。

```
#define trigPin_1 9 // set pin number for trigPin
#define echoPin_1 10 // set pin number for echoPin
#define intervaltime 100 // set measurement interval in ms
#define LED_Pin 19// set output pin for LED
boolean LED_ON = false;// LED status
unsigned int LED_ON_count = 0;// LED on counter
```

```
unsigned int LED_ON_count_max = 1;// LED on max duration
unsigned int LED_OFF_count = 0;// LED off duration
int timeout = 12000; // set timeout period for pulseIn

// speed of sound in cm/micro second
float Sound_speed = 343.0f * 100 / 1000000;

void setup()
{
  Serial.begin(115200);
  pinMode(trigPin_1, OUTPUT);// set trigPin to output mode
  pinMode(echoPin_1, INPUT);// set echoPin to input mode
  pinMode(LED_Pin, OUTPUT);// set LED_pin to output mode
  digitalWrite(trigPin_1, LOW);// initialize trigpin to LOW
  delay(1);
}

void loop() {
  // Read sensor value from ultrasonic sensor
  // Convert sensor value to float
  float distance = Get_US();
  if(distance>0){
    Serial.print(", Dis= ");
    Serial.print(distance);

    if(LED_ON) {    // LED On
      LED_ON_count++;
      if(LED_ON_count >= LED_ON_count_max) {
        LED_ON_count = 0;
        LED_ON = false;
      }
      digitalWrite(LED_Pin,HIGH);
      Serial.print(", LED ON");
    }
    else {        // LED Off
```

```
    LED_OFF_count++;
        if(LED_OFF_count >= int(distance/10)) {
          LED_OFF_count = 0;
          LED_ON = true;
        }
        digitalWrite(LED_Pin,LOW);
      }
      Serial.println();
    }
    else {
      Serial.println("Out of range !");
    }
    delay(intervaltime);
}

// function to read sensor value from ultrasonic sensor
float Get_US() {
  // Trigger
  digitalWrite(trigPin_1, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin_1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin_1, LOW);

  // Read
  long duration = pulseIn(echoPin_1, HIGH, timeout);// timeout in us
  Serial.print("Dur= ");
  Serial.print(duration);

  // convert sensor value to distance
  float distance = (float)(duration) / 2 * Sound_speed;
  return distance;
}
```

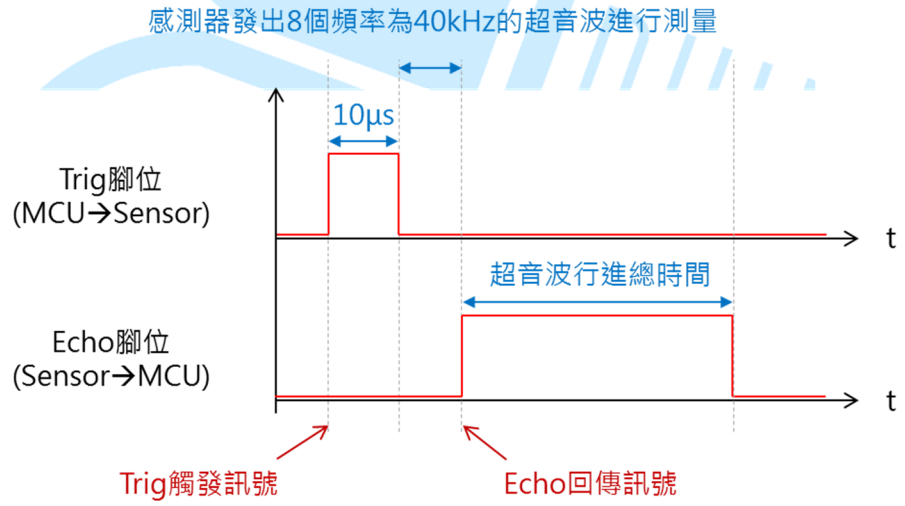
今回のアプリケーションは自動車のバックアップ警報センサと似ています。ものが近づいているとLEDの電灯周波数が増加します。下記の図を参照してください。



センサはコントローラから信号を受信する時に超音波を放ちます。そして反射された反射波の遅延時間を算出して値をコントローラに送信します。

HC-SR04 センサには「Trig」と「Echo」という 2 つの信号ピンが付いてます。「Trig」ピンはコントローラからセンサ検出プロセスを開始する際のトリガとして使われます。「Echo」ピンはコントローラが検出したセンサ値の送信に使われます。「Trig」と「Echo」共に通常は LOW 状態です。コントローラがトリガ信号を送信する時「Trig」ピンに 10μs の HIGH 状態を発生させ、その後 LOW 状態に戻りセンサ検出プロセスを開始します。

センサが反射波を検出すると「Echo」ピンに HIGH 状態を発生させます。この HIGH 状態になった時が即ち送出した超音波が反射波として返ってきてセンサに検出された期間となり、その値がμ秒で検出されます。この場合は pulseIn()機能が役に立ちます。下記の図にセンサの検出プロセスを表示します。



このアプリケーションのコードは初めに測定間隔、pulseIn()のタイムアウト値、LED の点灯パラメータ、音波のスピード等の変数を定めて初期化します。センサに接続されている信号ピンは setup()内で初期化し、適当なモードに設定されます。ご覧の通りメインプログラム loop()内は非常に少ないコードです。1 つのプログラム内の多くの場所で同じコードを繰り返し記述する事を避ける為に、コードを関数に集約して必要な時のみ関数を呼び出します。これによりコードのメンテナンスも簡潔になります。

本アプリケーションの超音波センサ対応コードは Get_US()関数に集約されています。Get_US()機能は呼び出される時にセンサ値を読み込み、メインプログラムでの LED コントロール、シリアルモニタへの出力に使われる値の返信処理を行います。

Get_US()機能が実行される時に「Trig」ピンに関連ある Educake の I/O ピンの変更を避けるために digitalWrite(trigPin_1, LOW)機能が呼び出されます。delayMicroseconds(2)は信号ピンが安定になるまで十分な時間を与えます。そして digitalWrite(trigPin_1, HIGH)が呼び出されて、センサが電圧 HIGH 状態になって超音波を放出します。10 マイクロ秒後電圧が LOW 状態に戻って検出プロセスが開始されます。そのあと pulseIn(echoPin_1, HIGH, timeout)機能が呼び出されてセンサ値を獲得します。センサが音波を検出した時間は音波が放出されて、対象物から跳ね返ってきた時間であり 2 で割る事でセンサから対象物迄の時間となります。

距離算出方式は「距離＝時間＊音波スピード」となり、音波スピードの算出方式は「音波スピード＝331＋（0.6＊摂氏度）です。本課題では温度が 20 度で音波スピードが 343m/s で算出します。距離を CM に変更するのは下記の方式を使います。

- 音波スピード（cm/ μs）= 343（m/s）＊ 100（cm/m）/ 1000000

次のコードでは：

- Float distance = float(duration)/2 * Sound_Speed

対象物からの距離を算出する際、浮動小数点数演算を用いセンサから対象物への往復の距離を２で割って音波スピードに掛けます。

センサーから検出した値を用い距離を算出したらメインプログラム loop()内で LED の点灯周波数に反映させ、Serial.print()機能でシリアルモニタへ結果を出力します。

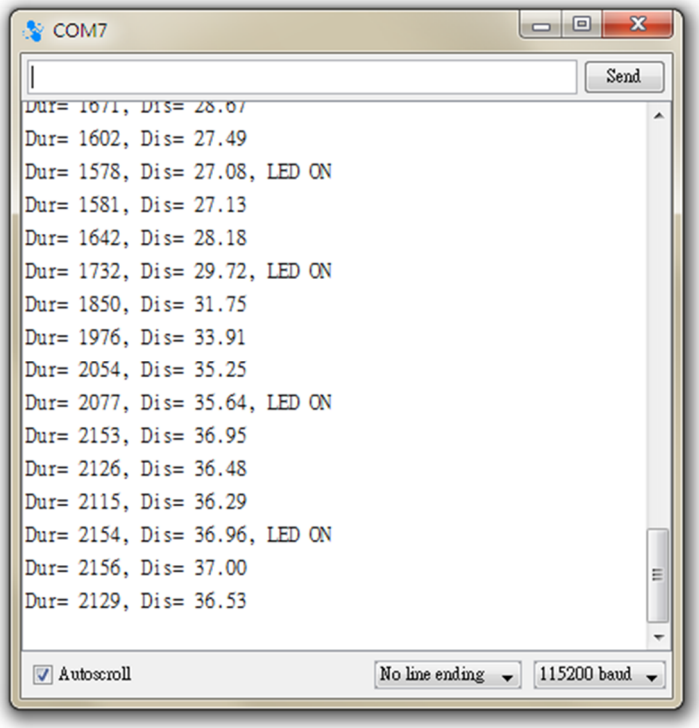
メインプログラム loop()内で delay(intervaltime)機能は固定値で呼び出されますが、センサのデータの獲得と処理は可変的な時間が必要ですので、変動している時間ループが出来てしまいます。LED はカウンターを使用して点灯周波数を調整します。

LED_ON_count、LED_OFF_count と LED_ON 等の LED 機能に関するグローバル変数はプログラムの先頭で定義しています。LED_ON_count の上限は１に設定されて

おり（上限を変更したら結果が変わります）、メインプログラム loop()が実行される時に LED_ON_count 値を一つ増やします。LED_ON_count が上限になったら LED_ON 変数を false に設定し、LED_ON_count を 0 に設定して LED が消えます。

LED_ON が false になる時に LED_OFF_count を一つ増やします。LED_OFF_count の上限は距離値に設定されており、int(distance/10)の値となります。メインプログラム loop()の遅延時間が 100ms に設定されているので、距離が 200cm になったら LED が約 2 秒間消灯します。

コードが実行されている間にシリアルモニタに出力した情報や LED の点灯周波数で距離が分かります。下記の図を参照してください。



本課題で超音波センサがどのように働くかを勉強しました。異なる発想で割り込みや digitalRead()機能による Echo 信号ピンのデータ読み取り方法を考えてください。LED の代わりにブザーを使用して自動車のような障害物警告システム等にも応用可能です。