INTERRUPCIONES INTERNAS Y EXTERNAS, CONTROL PID

Lenguajes Técnicos de programación

Por: Hugo Murillo

UNIVERSIDAD EAFIT®

INTERRUPCIONES

Las llamadas funciones desde el programa principal hacen que el programa ejecute un subprograma y luego regrese al programa principal, sin embargo el programador está definiendo en qué momento debe saltar a ejecutar la función mediante las instrucciones, por esta razón se considera síncronas.

Las interrupciones son desviaciones de flujo de control del programa originadas asincrónicamente por diversos sucesos que no dependen del programador, es decir, ocurren en cualquier momento.

Las interrupciones ocurren por sucesos externos como la generación de un flanco o nivel en una terminal del microcontrolador Arduino o eventos internos tales como el desbordamiento de un contador, terminación del conversor análogo a digital, entre otras.

INTERRUPCIONES

El comportamiento del microcontrolador ante la interrupción es similar al procedimiento que se sigue al llamar una función desde el programa principal. En ambos casos se detiene la ejecución del programa en curso, se guarda la dirección a donde debe retornar cuando termine de ejecutar la interrupción, atiende o ejecuta el programa correspondiente a la interrupción y luego continua ejecutando el programa principal, desde donde lo dejo cuando fue interrumpido.

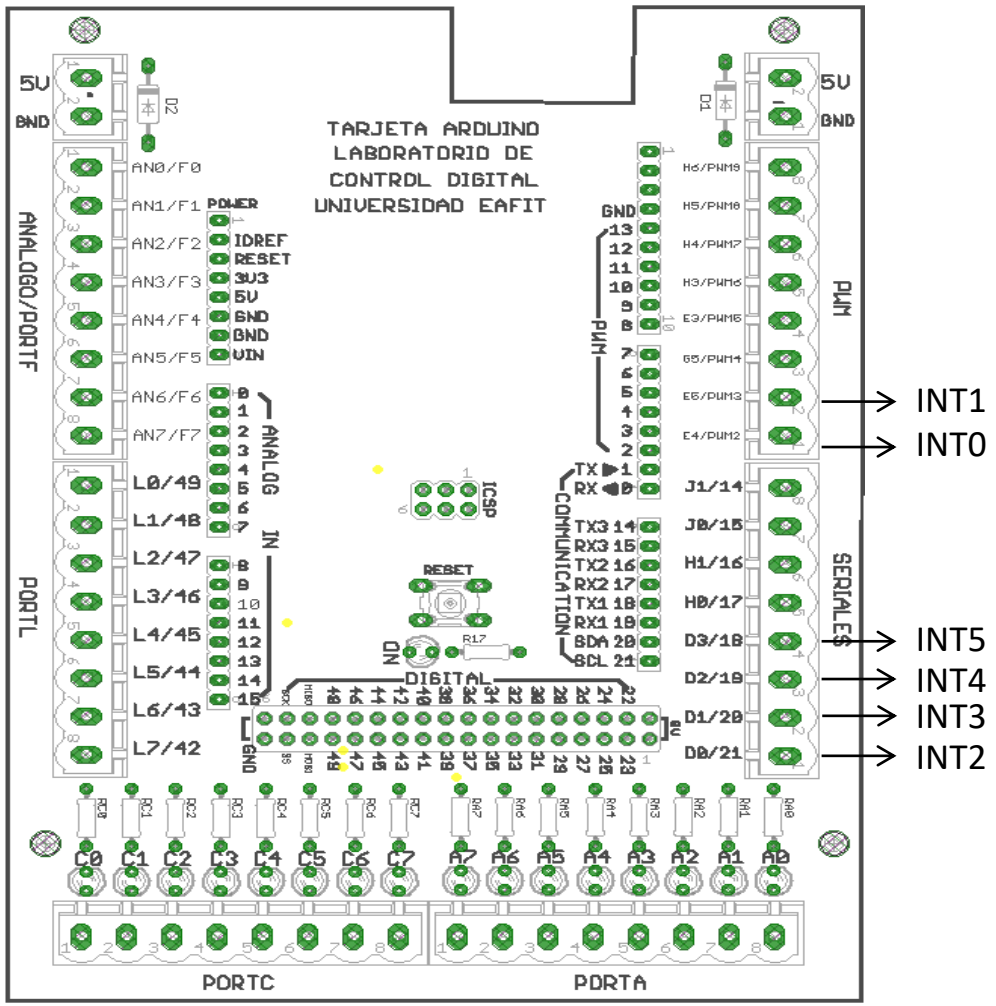
Existen 4 diferentes causas **“LOW,CHANGE,RISING,FALLING”** que producen una interrupción externa, por lo tanto el primer paso de la rutina de interrupción será identificar la causa de la interrupción.

AttachInterrupt ()

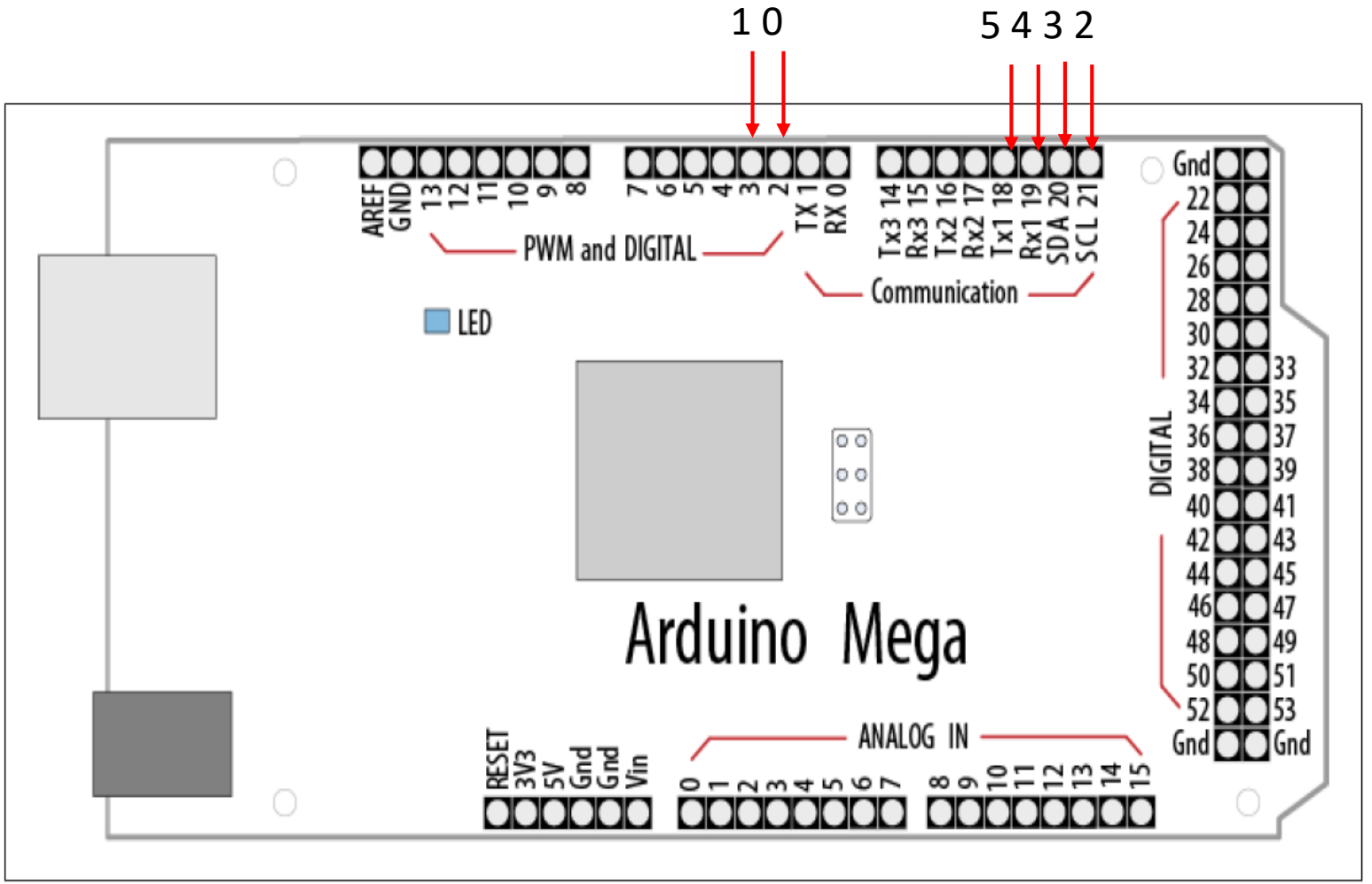
Especifica una función a llamar cuando se produce una interrupción externa. La placa Arduino Mega tiene seis pines que se pueden utilizar como interrupciones externas:

NOMBRE DE LA INTERRUPCIÓN	PIN DIGITAL	PIN FÍSICO
0	2	6
1	3	7
2	21	43
3	20	44
4	19	45
5	18	46

IDENTIFICACIÓN DE LOS PINES DE LAS INTERRUPCIONES



IDENTIFICACIÓN DE LOS PINES DE LAS INTERRUPCIONES



HABILITAR LAS INTERRUPCIONES

SINTAXIS

attachInterrupt (interrupción, la función, el modo)

PARÁMETROS

- **Interrupción**: el número de la interrupción (int)
- **Función**: nombre de la función a llamar cuando se produce una interrupción, esta función no permite ni ingreso ni retorno de parámetro. Esta función se denomina una rutina de servicio de interrupción.
- **El modo** es para definir la forma como se activara la interrupción por flanco o por estado. Cuatro constantes están predefinidas como valores válidos:
  - LOW: Genera la interrupción cada vez que el pin pasa a bajo.
  - CHANGE: Genera la interrupción cada vez que hay altos o bajos en el pin.
  - RISING: Genera la interrupción cada vez que el pin pasa a alto.
  - FALLING: Genera la interrupción cada vez que hay un cambio de estado en el puerto de alto a bajo.

DESHABILITAR LAS INTERRUPCIONES

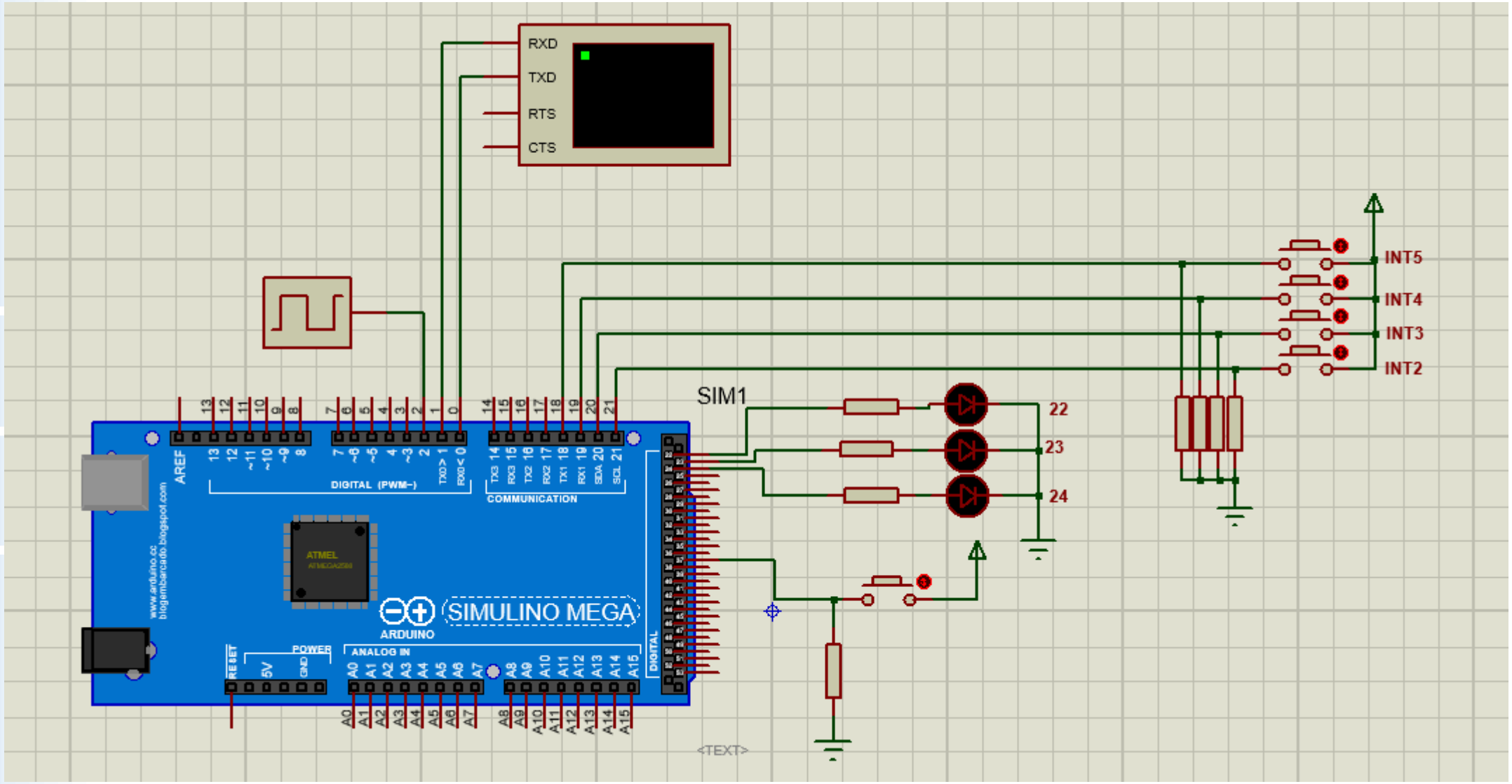
SINTAXIS

dettachInterrupt (numero de la interrupción)

EJEMPLO:

```
detachInterrupt(2);    //Deshabilita la interrupción 2
```

CIRCUITO EN PROTEUS PARA LOS EJEMPLOS



Ejemplo 1

Hacer un programa que incremente un contador cada segundo. Si se activa el pulsador de reset (Interrupción), resetea el contador

```
int cont = 0;                    //Define cont como entero


void setup()
{
  attachInterrupt(2, reset, RISING);
  Serial.begin(9600);
}


void reset()
{
  delay(300);
  cont = 0;
  Serial.println(cont);
}
```

Ejemplo 1

```
void loop()
{
  delay(1000);
  cont++;
  Serial.println(cont);
}
```

Ejemplo 2

Hacer un programa de un contador ascendente o descendente de acuerdo a la selección que haga el usuario. Si se activa el pulsador de reset (Interrupción), resetea el contador. El contador incrementa cada segundo.

```
int cont = 0,ascendente=1;                    //Define cont como entero
void setup()
{
  attachInterrupt(2, reset, RISING);
  attachInterrupt(3, cont_ascendente, RISING);
  attachInterrupt(4, cont_descendente, RISING);
  Serial.begin(9600);
}
void reset()
{
  delay(300);
  cont = 0;
  Serial.println(cont);
}
```

Ejemplo 2

```
void cont_ascendente()
{
  ascendente = 1;
}

void cont_descendente()
{

  ascendente = 0;
}
```

Ejemplo 2

```
void loop()
{
  delay(1000);
  if (ascendente == 1)
  {
    cont++;
  }
  else
  {
    cont--;
  }
  Serial.println(cont);

}
```

TIMER

En general:

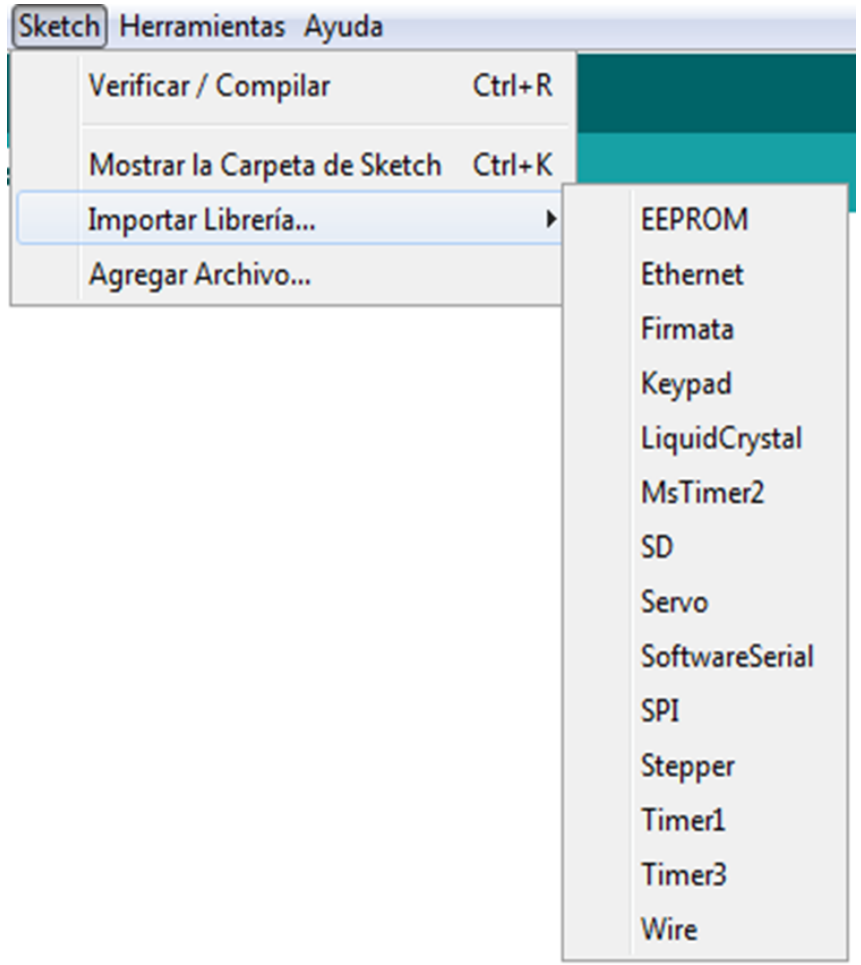
Máximo = Periodo (pre-escala) * (1/Frecuencia) * (2^17)

Tiempo de conteo = (pre-escala) * (1/Frecuencia)

Siempre que se va a trabajar con timers se debe agregar la librería de este que ya viene con todos los parámetros definidos ahorrando tiempo al programar ya que evita el proceso de estar haciendo los cálculos manualmente con esta librería, lo único que hay que hacer es definirle el tiempo de trabajo con el que se desea trabajar el timer este tiempo se le debe dar en microsegundos (uS) y habilitar la interrupción.

TIMER

- Para descargar la librería del timer1 se encuentra en el siguiente enlace el cual lo lleva a la descarga.
- **http://code.google.com/p/arduino-timerone/downloads/list**



- Para instalarlo, simplemente descomprimir y poner los archivos en **Arduino/hardware/libraries/Timer1 /**
- Para agregar una librería al sketch lo único que hay que hacer es ubicarse en la barra de herramientas y seleccionara la opción sketch, importar la librería y agregar la librería deseada.

PASOS PARA TRABAJAR CON EL TIMER 1

1.Inicializar el tiempo del Timer1. El tiempo esta en microsegundos.

```
Timer1.initialize(1000000);
```

2. Habilitar la interrupción por timer1, e indicar que función se debe ejecutar cuando se genere la interrupción.

```
Timer1.attachInterrupt(reloj);
```

3. Deshabilitar la interrupción por timer1, cuando no se requiera.

```
Timer1.detachInterrupt();
```

Ejemplo 3

Incrementar un contador de 0 a 20 cada segundo, usando interrupciones por Timer.

```
#include <TimerOne.h>            //incluye la librería del rimer1
int seg;

void reloj()                     //rutina de interrupción por timer
{
  seg++;
  Serial.println(seg);
}
void setup()
{
  seg = 0;
  Timer1.initialize(1000000);          //define el tiempo de trabajo del timer a 1seg
  Serial.begin(9600);
Timer1.attachInterrupt(reloj);
}
```

Ejemplo 3

```
void loop()
{
  if (seg == 20)
  {
    seg = 0;
    Serial.println(seg);
  }
}
```

Ejemplo 4

Realizar un temporizador de 10 segundos, cuando se oprima el pulsador de inicio, usando interrupciones por Timer.

```
#include <TimerOne.h>          //incluye la libreria del rimer1
int inicio  = 37;       // inicio en el pin 37
int led = 22;
int seg;

void reloj()                    //rutina de interrupcion por timer
{
  seg--;
  Serial.println(seg);
}
```

Ejemplo 4

```
void setup()
{
  pinMode(inicio, INPUT);
  pinMode(led, OUTPUT);
  Timer1.initialize(1000000);       //define el tiempo de trabajo del timer a 1seg
  Timer1.detachInterrupt();      //deshabilita y llama la interrupción por timer
  digitalWrite(led, LOW);
  Serial.begin(9600);
}

void loop()
{
   Serial.println("oprima la tecla inicio para comenzar");
  while (digitalRead(inicio) == LOW)
  {
  }
Timer1.attachInterrupt(reloj);    //habilita y llama la interrupción por timer
  seg = 10;
  Serial.println(seg);
```

Ejemplo 4

```
while (seg > 0)
  {
    digitalWrite(led, HIGH);
  }
  Timer1.detachInterrupt();      //deshabilita y llama la interrupción por timer
  digitalWrite(led, LOW);
}
```

Ejemplo 5

Encender y apagar un led a dos frecuencias diferentes de tal manera que la frecuencia se pueda cambiar al oprimir el pulsador. En el void loop no se debe programar nada. Todo el programa se debe hacer usando las interrupciones

```
#include <TimerOne.h>      //incluye la libreria del timer1
int led=22;
int tiempo=0;

void parpadeo()
{
  if(digitalRead(led)==HIGH)
   {
     digitalWrite(led,LOW);
   }
   else
   {
    digitalWrite(led,HIGH);
   }
}
```

Ejemplo 5

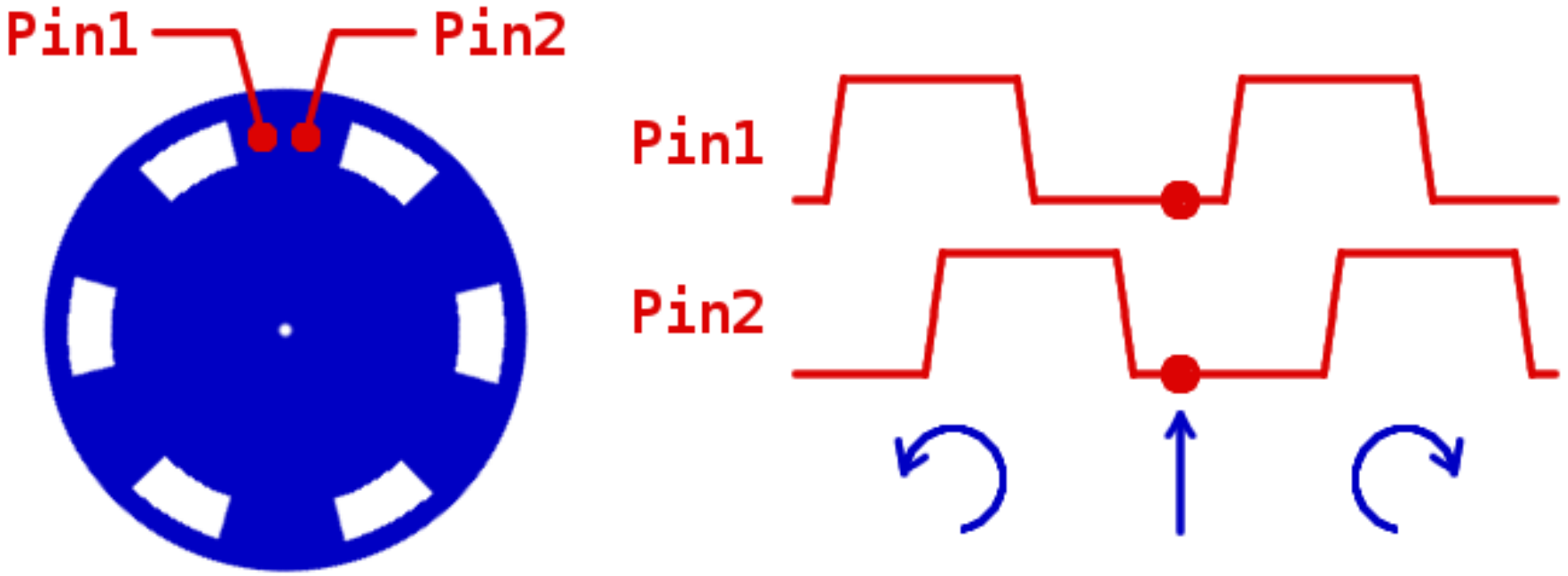
```
void modo()
{
   delay(300);
  if(tiempo==0)
   {
    Timer1.initialize(100000);
    tiempo=1;
   }
   else
   {
    Timer1.initialize(1000000);
    tiempo=0;
    }
}
```

Ejemplo 5

```
void setup()
{
  pinMode(led,OUTPUT);
  Timer1.initialize(1000000);          //define el tiempo de trabajo del timer a 1seg
  Timer1.attachInterrupt(parpadeo);
  attachInterrupt(2,modo,RISING);
}

void loop()
{
}
```

ENCODER



Ejemplo 6

```
#include <TimerOne.h>      //incluye la libreria del Timer1
#include <dis7seg.h>
int inicio  = 37;        // inicio en el pin 37
int led = 28;
int tdec=23;
int tuni=22;
int seg,uni,dec;
Display d1(49,48,47,46,45,44,43,LOW); // Display <nombre>(a,b,c,d,e,f,g,HIGH/LOW); HIGH = Anodo Comun; LOW = Catodo comun

void reloj()               //rutina de interrupcion por timer
{
   seg++;
   Serial.println(seg);
 }
```

Ejemplo 6

```
void mostrar()
 {
        dec = seg / 10;
        uni = seg % 10;
        digitalWrite(tdec, LOW);   // apaga el display de decenas
        digitalWrite(tuni, HIGH);   // enciende el display de unidades
        d1.print(uni);            // Muestra lo que hay en unidades
        delay(1);                 //retardo de 1 milisegundo
        digitalWrite(tuni, LOW);   // apaga el display de unidades
        digitalWrite(tdec, HIGH);   // enciende el display de decenas
        d1.print(dec);                // Muestra lo que hay en decenas
        delay(1);
 }
```

Ejemplo 6

```
void setup()
 {
        pinMode(inicio, INPUT);
        pinMode(led, OUTPUT);
        pinMode(tdec, OUTPUT);
        pinMode(tuni, OUTPUT);
        Timer1.initialize(1000000);       //define el tiempo de trabajo del timer a 1seg
        Timer1.detachInterrupt();     //deshabilita y llama la interrupción por timer
        digitalWrite(led, LOW);
        Serial.begin(9600);
 }
void loop()
{
 seg=0;
 Serial.println("oprima la tecla inicio para comenzar");
while (digitalRead(inicio) == LOW)
 {
```

Ejemplo 6

```
        mostrar();
    }
        Timer1.attachInterrupt(reloj);    //habilita y llama la interrupción por timer
        seg = 0;
        Serial.println(seg);

        while (seg < 10)
        {
                digitalWrite(led, HIGH);
                mostrar();
        }
        Timer1.detachInterrupt();     //deshabilita y llama la interrupción por timer
        digitalWrite(led, LOW);
}
```

Control PID

Lenguajes Técnicos de programación

Por: Hugo Murillo

UNIVERSIDAD EAFIT®

Que es el control PID

El PID es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y el valor que se quiere obtener, para aplicar una acción correctora que ajuste el proceso. El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo.

Valor proporcional: determina la reacción del error actual.

Valor integral: genera una corrección proporcional a la integral del error, esto nos asegura que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero.

Valor derivativo: determina la reacción del tiempo en el que el error se produce

.

La suma de estas tres acciones es usada para ajustar al proceso vía de un elemento de control.

Los controladores PI son particularmente comunes, ya que la acción derivativa es muy sensible al ruido, y la ausencia del proceso integral puede evitar que se alcance al valor deseado debido a la acción de control.

Control PID

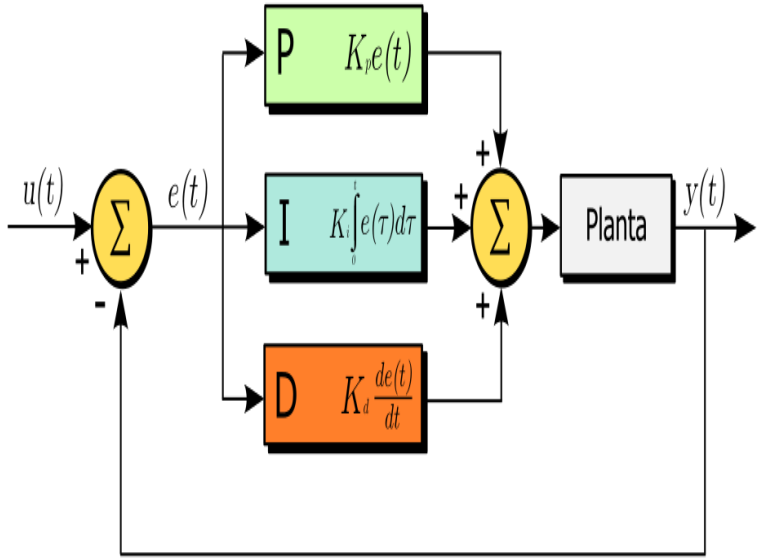
Se requiere de:

Sensor: proporciona una señal analógica la cual representa el punto actual en donde se encuentra el proceso en el sistema (Ejm: Termómetro, manómetro, etc.)

Controlador: lee una señal externa que representa el valor que se desea alcanzar(Punto de consigna)

Actuador: modifica el sistema de manera controlada (Ejm: Resistencia eléctrica, motor, válvula, bomba, etc.)

Diagrama en bloques de un control PID



Tomado de:
http://es.wikipedia.org/wiki/Proporcional_integral_derivativo

Uso de la librería PID en arduino

El uso de la librería tiene 2 grandes ventajas:

- Hay diversas formas de escribir el algoritmo del PID, pero en ello se ha venido trabajando desde hace tiempo, es por ello que las funciones yacreadas son las mejores para trabajar.
- Cuando se usa la librería del PID, todo el código necesario esta contenido en el algoritmo, esto hace que sea mucho más fácil de entender el programa.

Existen diversas funciones que serán explicadas a continuación.

Funciones: PID( )

El uso de la librería tiene 2 grandes ventajas:

- Hay diversas formas de escribir el algoritmo del PID, pero en ello se ha venido trabajando desde hace tiempo, es por ello que las funciones yacreadas son las mejores para trabajar.
- Cuando se usa la librería del PID, todo el código necesario esta contenido en el algoritmo, esto hace que sea mucho más fácil de entender el programa.

Existen diversas funciones que serán explicadas a continuación.

Funciones: PID( )

- **Descripción:** crea un regulador PID donde se vincula la entrada especificada, la salida y el setpoint. El algoritmo PID esta en la forma paralela
- **Sintaxis:** PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction)
- **Parámetros:**
- **Input:** La variable a controlar (double)
- **Output:** La variable que sera ajustada por el PID (double)
- **Setpoint:** El valor de Referencia en la cual se debe mantener (double)
- **Kp, Ki, Kd:** Parámetros de ajuste. Estos parámetros afectan el cambio en la variable de salida del PID (double>=0)
- **Direction:** DIRECT o REVERSE. Determina en que dirección se movera la salida ante un error determinado. La mas común es la DIRECT.
- **Returns:** None

Ejemplo: PID myPID(&Input, &Output, &Setpoint, kp, ki, kd, DIRECT);

Funciones: PID( )

- **Descripción:** Contiene el algoritmo de PID; este debería ser llamado cada vez que se realice un Loop(). La mayoría de veces este retornara sin hacer nada. A la frecuencia especificada por el SetSampleTime este calculara una nueva salida.
- **Sintaxis:** Compute()
- **Parametros:** Ninguno
- **Returns:**

  Verdadero: cuando la salida es computarizada

  Falso: cuando no se ha realizado nada

Ejemplo: myPID.Compute();

Funciones: SetMode( )

- **Descripción:** especifica cuando el control PID debe estar en modo automático o en modo manual.
- **Sintaxis:** SetMode(mode)
- **Parametros:** mode: AUTOMATIC o MANUAL
- **Returns:** Nada

Ejemplo:

```
myPID.SetMode(AUTOMATIC);
```

Funciones: SetOutputLimits( )

- **Descripción:** El controlador PID esta diseñado para variar en un rango determinado, por defecto de 0 a 255 correspondiente al rango de variación del PWM, sin embargo el usuario puede definir otros limites diferentes no mayores a 255.
- **Sintaxis:** SetOutputLimits(min, max)
- **Parametros:**
  - **min:** Valor mínimo del rango. Debe ser < max (double)
  - **max:** Valor maximo del rango. Debe ser > min (double)
- **Returns:** Nada

Ejemplo: myPID.SetOutputLimits(0, 255);

Funciones: SetTunings( )

- **Descripción:** utilizado para organizar parámetros de ajuste que definen el comportamiento dinámico del PID, entre estos se puede preguntar ¿Oscilara o no el PID?, ¿Va a ser rápido o lento?. Este conjunto de decisiones se toman al inicio de la creación del PID; sin embargo estos parámetros pueden ser cambiados en el tiempo, es allí cuando se debe llamar a esta función.
- **Sintaxis:** SetTunings(Kp, Ki, Kd)
- **Parametros:**
  - **Kp:** Determina como reaccionará el PID a la cantidad de errores actuales del programa(Proportional) (double >=0)
  - **Ki:** Determina como reaccionará el PID respecto al tiempo (Integral) (double>=0)
  - **Kd:** Determina como reaccionará el PID a los cambios de los errores actuales del programa(Derivative) (double>=0)
- **Returns:** Nada

Ejemplo: myPID.SetTunings(kp, ki, kd);

Funciones: SetSampleTime( )

- **Descripción:** Determina con qué frecuencia se evalúa el algoritmo PID; el valor por defecto es 200ms. Para las aplicaciones de robótica este debería de ser mucho más rápido, pero para la gran mayoría de programas, cada 200ms puede llegar a ser bastante rápido.
- **Sintaxis:** SetSampleTime(SampleTime)
- **Parametros:**

  SampleTime: Con que frecuencia, en milisegundos, sera evaluado el PID. (int>0)
- **Returns:** Nada

Ejemplo: myPID.SetSampleTime(100);

Funciones: SetControllerDirection( )

- **Descripción:** Esta función define el tipo de acción de la salida Directa o inversa. Es poco probable que el PID cambie de acción directa a inversa, la función SetControllerDirection() no es muy usada.
- **Sintaxis:** SetControllerDirection(Direction);
- **Parametros: Direction:** DIRECT o REVERSE
- **Returns:** Nada

Ejemplo: myPID. SetControllerDirection(DIRECT);

Funciones adicionales del la libreria

Display Functions:

GetKp(); GetKi(); GetKd(); GetMode(); GetDirection()

- **Descripción:** Estas funciones se utilizan para saber los valores actuales del PID. Son útiles para fines de visualización.
- **Sintaxis:**
  - GetKp()
  - GetKi()
  - GetKd()
  - GetMode()
  - GetDirection()
- **Parametros:** Nada
- **Returns:** El correspondiente valor interno

Implementación del PID en Arduino

```
#include <PID_v1.h>           //Incluyo Libreria PID.
double SETPOINT = 30;              //Sera el valor que deseamos mantener en el tiempo.
double INPUT_PID;          //Sera la el valor de entrada a controlar.
double OUTPUT_PID;              //Sera nuestra salida, efectúa la acción de control.
double KP = 2;             //Constante Proporcional, nosotros asignamos el valor.
double KI = 1;                  //Constante Integral, nosotros asignamos el valor.
double KD = 1;             //Constante Derivativa, nosotros asignamos el valor.
#define TIME_PID 100       //Cada 100ms se actualizara el PID.
#define MINIMO 0                //El valor mínimo de OUTPUT será 0.
#define MAXIMO 255         //EL valor máximo de OUTPUT sera 255.
#define RESISTENCIA 3      //EL PIN 3 ESTA ASOCIADO A UNA RESISTENCIA ELECTRICA
PID namePID(&INPUT_PID, &OUTPUT_PID, &SETPOINT , KP, KI, KD, DIRECT);
```

Implementación del PID en Arduino

```
void setup()
{
  namePID.SetMode(AUTOMATIC); //Seleccionamos AUTOMATIC o MANUAL.
  namePID.SetSampleTime(TIME_PID); //Asigno valor de muestreo.
  namePID.SetOutputLimits(MINIMO, MAXIMO); //Declaro rango de salida.
  pinMode(RESISTENCIA, OUTPUT); //Asigno este pin como salida.
  Serial.begin(9600);
}
```

Implementación del PID en Arduino

```
void loop()
{
//Formula para convertir lectura de un LM35 Grados °C:
  INPUT_PID = ((analogRead(A0) * 5000.0) / 1023) / 10;
  namePID.Compute(); //ACTIVO MI PID.
  analogWrite(RESISTENCIA, OUTPUT_PID); // APLICO SALIDA
  Serial.print( "Temp: “ );
  Serial.println(INPUT_PID);
  Serial.print( "PID: “ );
  Serial.println(OUTPUT_PID);
  }
```

Ejemplo Control temperatura

```
#include <PID_v1.h>                    // Incluir Librería PID_v1.
#include <TimerOne.h>                  // Incluir Librería TimerOne.
volatile int i = 0;                    // Variable usada por el contador.
volatile boolean cruce_cero = 3;       // variable que actúa como switch al detectar cruce por cero.
int Triac = 5;                         // Salida conectada al optoacoplador MOC 3021.
int dim;                               // Controla la intensidad de iluminación, 0 = ON ; 83 = OFF
int T_int = 100;                       // Tiempo en el cual se producen la interrupciones en us.
double Setpoint, Input, Output;        // Definir variables a las que nos conectaremos.
double Kp = 0.9354, Ki = 0.10998, Kd = 0.0000;   // Especificar los enlaces y los parámetros de ajuste iniciales.
int lm35 = A7;                                   // Pin del sensor de temperatura LM35.
int Temperatura = 0;
const long  retraso = 500;                       // valor de lectura en tiempo (1/2 segundo).
unsigned long inicio_tiempo;                     // Variable usada para la función millis.

int potPin = A6;                       // El número del pin analógico donde está conectado el potenciómetro.
int pwmPin = 6;                        // El número del pin digital donde está conectado el transistor.

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

Ejemplo Control temperatura

```
void setup() {
  Serial.begin(9600);
  pinMode(Triac, OUTPUT);
  pinMode(pwmPin, OUTPUT);
  pinMode(cruce_cero, INPUT);
  inicio_tiempo = millis();
  attachInterrupt(1, deteccion_Cruce_cero, RISING);  // Realiza una interrupción al detectar el crice
                                                     por cero en el pin 3
  Timer1.initialize(T_int);                          //Inicializa la librería con el tiempo deseado.
  Timer1.attachInterrupt(Dimer, T_int);              // En cada interrupción ejecuta el codigo Dimer.
  Setpoint = 20.0;
  myPID.SetMode(AUTOMATIC);                          //turn the PID on
}
void deteccion_Cruce_cero() {                        // Si existe un cruce por cero entonces la variable
"cruce_cero" cambia a TRUE...
  cruce_cero = true;                                 //...reseteando el valor de "i", y colocando la salida
conectada al Triac en estado...
  i = 0;
  digitalWrite(Triac, LOW);
}
```

Ejemplo Control temperatura

```
 void Dimer() {                          // Funcion para controlar el disparo del
Triac
  if (cruce_cero == true) {
    if (i >= dim) {
      digitalWrite(Triac, HIGH);
      i = 0;
      cruce_cero = false;
    } else {
      i++;
    }
  }
}
```

Ejemplo PID túnel de viento

```
void loop() {
  int potValue = analogRead(potPin);                    // Lee el valor del potenciómetro
  int pwmValue = map(potValue, 0, 1023, 0, 255);        // Convierte el valor del potenciómetro a un valor PWM de 0 a 255
  analogWrite(pwmPin, pwmValue);                        // Controla el transistor con el valor PWM
  myPID.Compute();
  dim = map(Output, 0, 180, 0, 120);                    // Convierte el valor de la intensidad de iluminación a un valor PWM de 0 a 120
  if (millis() - inicio_tiempo >= retraso) {            // Función millis para el leer los datos cada medio segundo.
    Temperatura = 5.0 * 100.0 * analogRead(lm35) / 1024.0;   // Lectura del sensor LM35.
    Input = Temperatura;                                // Variable asignada como entrada al PID.
    Serial.print(Temperatura);
    Serial.print(" ");
    Serial.print(dim);
    Serial.print(" ");
    Serial.print(pwmValue);
    Serial.print(" ");
    Serial.print(Output);
    Serial.println(" ");
    inicio_tiempo = millis();
  }
}
```