Microcontroladores PIC

Autor: Rafael Ferrari Galindo

OJO: Este documento fue hecho en 2 días y es su primera versión y probablemente su última dado fue creado para ayudar con la prueba de DA por lo tanto pueden haber errores de programación y o contenido así que para más información puede consultar las presentaciones o los videos de https://www.youtube.com/watch?v=SXl-VD5WU-s&list=PLONPO-iVba9nbY_KTCHt9GGj9dGSj1qYo

Todos los códigos usados están en mi github
https://github.com/rafaferrarigl?tab=repositories

Capítulo I:

Interrupción externa.

Introducción:

Las interrupciones externas como su nombre lo indica su uso es interrumpir el normal funcionamiento del código (Función main) a través de un botón o sensor externos al micro controlador, un ejemplo de esto será el de interrumpir el funcionamiento de un bucle infinito (While(TRUE))

Declaración:

Para la habilitación de la interrupción externa en el main usaremos las líneas 4 y 6 habilitaremos las interrupciones externas y las globales respectivamente mientras que en la línea 5 le diremos que se active la interrupción con RB0.

```
1  void main()
2  {
3
4     enable_interrupts(int_ext);
5     ext_int_edge(H_to_L);
6     enable_interrupts(GLOBAL);
7
8     while(TRUE)
9     {
10
11       // empieza la estera en bucle
12    }
13
14 }
```

La función de la interrupción en si se declarara de la siguiente manera

```
1 #INT_EXT
2
3 void ext_isr() {
4
5 }
```

Siempre deberá comenzar con la línea 1 que varía según el tipo de interrupción es lo que le dice al programa que a continuación está declarada la función de interrupción. A continuación se muestran algunas de las interrupciones en el micro controlador

#INT_EXT INTERRUPCIÓN EXTERNA

#INT_RTCC DESBORDAMIENTO DEL TIMER0(RTCC)

#INT_RB CAMBIO EN UNO DE LOS PINES B4,B5,B6,B7

#INT_AD CONVERSOR A/D

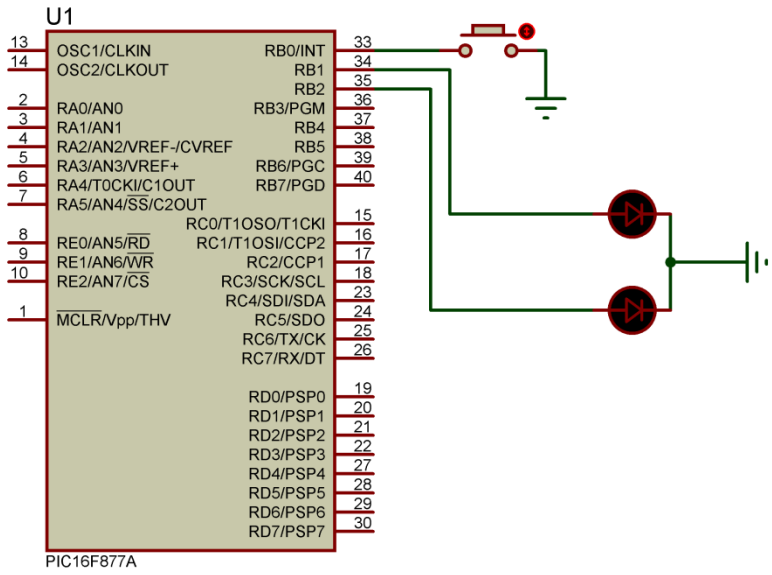
#INT_EEPROM ESCRITURA EN LA EEPROM COMPLETADA

#INT_TIMER1 DESBORDAMIENTO DEL TIMER1

#INT_TIMER2 DESBORDAMIENTO DEL TIMER2

Por ultimo tenemos que dentro de los corchetes de las líneas 3 y 4 va el código a ejecutarse en la interrupción.

Ejemplo:



Para el ejemplo se utiliza un pic 16f877A, un botón para que funciones como interrupcion externa y 2 leds para ver el funcionamiento de esto.

```
1  #include <16f877.h>
2
3  #FUSES XT, NOWDT
4  #use delay(clock=2M)
5  #use standard_io(D)
6  #BYTE OPTION_REG = 0x81
7
8
9
10 #INT_EXT
11
12 void ext_isr() {
13         while(TRUE){
14                 output_low(PIN_B1);
15                 output_high(PIN_B2);
16         }
17 }
18
19 void main()
20 {
21         bit_clear(OPTION_REG, 7);
22         enable_interrupts(int_ext);
23         ext_int_edge(H_to_L);
```

```
24          enable_interrupts(GLOBAL);
25
26          output_low(PIN_B2);
27
28          while(TRUE)
29          {
30
31                  output_high(PIN_B1);
32          }
33
34 }
```

En la línea uno declaramos la librería del micro que utilizaremos en nuestro caso el 16f877, en la línea 4 la frecuencia del microprocesador (4MHz), en la 5 que usaremos los puertos D ya sea como entrada o salida y en la 6 declaramos el Pull-up para habilitarlo mas adelante en la línea 21 en el main.

Para la parte de la interrupción declaramos que será una interrupción externa y en la función hacemos un bucle infinito con while(true) apangado el led en el puerto B1 y prendiendo el del puerto B2

En la función principal habilitamos el pull-up y las interrupciones externas y globales, luego declaramos que el pin B2 estará en low(el del apagado) y en el bucle while(true) que el pin B1 estará en high ósea el led estará prendido. El led conectado al puerto B1 permanecería prendido mientras que el microprocesador esté conectado ya que el orden de output_high() se encuentra en un bucle infinito pero como tenemos nuestra interrupción en su lugar una ves presionado el botón en B0 entrara está en juego apagando el led en B1 y prendiendo el de B2. Esto es un ejemplo sencillo del funcionamiento de lo que es una interrupción externa pero será necesario tener este conocimiento para realizar ejemplos de mayor complejidad.

Capítulo II:

Interrupción TIMERS.

Introducción:

Estas interrupciones pueden ser usadas de varias maneras entre ellas la de contador y temporizador. Los timer del PIC 16f877 son el TIMER0, TIMER1 y TIMER2. Cabe decir que estas interrupciones ocurren sin la necesidad de ningún factor externo como botones o sensores aunque pueden utilizarse a través de las interrupciones externas (en los casos de estudio hay un ejemplo de esto).

Timer0:

El Timer 0 también llamado RTCC se puede cargar con un valor cualquiera entre 0 y 255 debido a su naturaleza de 8 bits y puede ser incrementado a través del reloj interno y dividido por un valor que se escoge. Esto se conoce como el valor del pre-escalador (preescaler).

Modos y preescalers:

Modos:

RTCC_INTERNAL (Modo de temporizador)

RTCC_L_TO_H (Modo de contador)

RTCC_H_TO_L (Modo de contador)

RTCC_8_BIT (Se puede colocar de manera opcional)

RTCC_16_BIT (Se puede colocar de manera opcional)

Preescalers:

RTCC_DIV_2

RTCC_DIV_4

RTCC_DIV_8

RTCC_DIV_16

RTCC_DIV_32

RTCC_DIV_64

RTCC_DIV_128

RTCC_DIV_256

Cálculos previos:

Para poder determinar algunos valores como el tiempo máximo que el timer0 soporta y cuál será el valor con el que se configurara se necesitan realizar algunos cálculos.

$$T_{cm} = \frac{4}{F_{osc}} = \frac{4}{2 \cdot 10^6} = 0.000002seg$$

$$T_{max} = \frac{4}{F_{osc}} \cdot preescaler \cdot (preescaler - TMR0)$$

$$T_{max} = \frac{4}{2 \cdot 10^6} \cdot 256 \cdot (256 - 0) = 0.131seg$$

$$TMR0 = 256 - \left(\frac{Tiempo \cdot F_{osc}}{4 \cdot preescaler}\right) = 256 - \frac{0.001 \cdot 2 \cdot 10^6}{4 \cdot 256} = 254$$

Donde:

T_{cm} *Ciclos maquina*

F_{osc} *Frecuencia del oscilador*

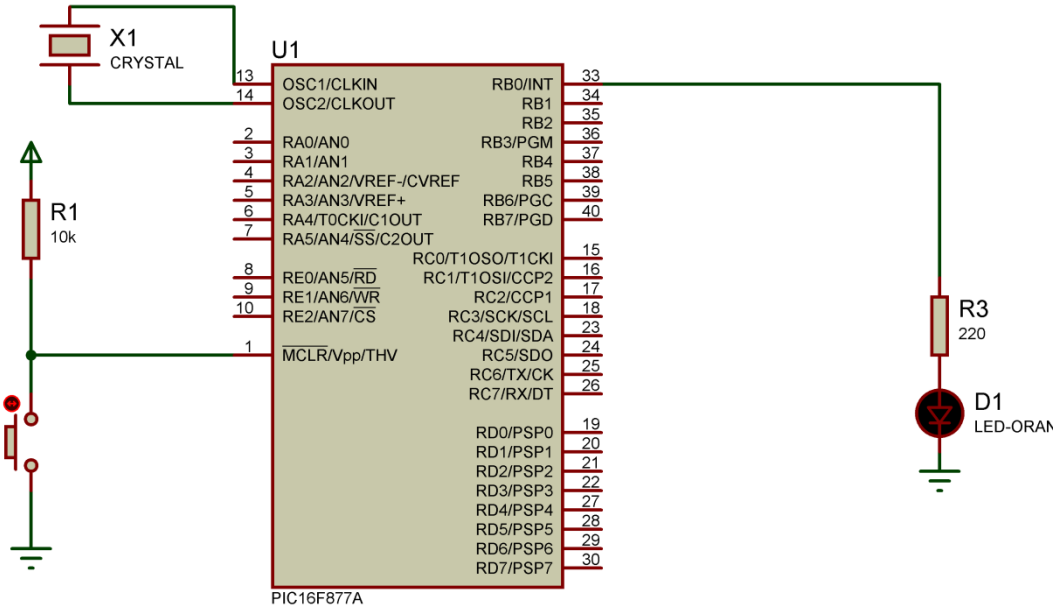
T_{max} *Tiempo máximo que podrá contar el Timer0 con esa frecuencia del oscilador del microprocesador y el preescaler.*

$TMR0$ *Sera el valor que se utilizara para configurar el timer0*

Ejemplo:

```
1     #include <16f877a.h>
2
```

```
3     #fuses HS,NOWDT
4     #use delay(clock=2M)
5     #use standard_io(B)
6
7     long contador = 0;
8
9     #INT_TIMER0
10    void timer0_interrupcion()
11    {
12       contador++;
13       if(contador == 1000)        // Contador para 1 segundo
14       {
15          output_toggle(PIN_B0);
16          contador = 0;
17       }
18       set_timer0(254);
19    }
20
21    void main()
22    {
23       setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256);
24       enable_interrupts(INT_TIMER0);
25       enable_interrupts(GLOBAL);
26       set_timer0(254);
27       output_high(PIN_B0)
28 }
```



Con este ejemplo configuramos una luz led para que se encienda y apague cada 1 segundo a través del timer 0.

Con las líneas 24 y 25 habilitamos el timer0 y las interrupciones globales.

```
enable_interrupts(INT_TIMER0);
enable_interrupts(GLOBAL);
```

En la línea 26 configuramos el timer0 para 0.001segundo que fue lo que calculamos como $TMR0$ y nos da 254.

```
set_timer0(254);
```

Para la función del timer en si lo que hacemos es declaramos que acontinuacion estara la función del timer con #INT_TIMER0 *luego haremos que cada ves que se ejecute incremente 1 a contador el cual es una variable flotante y que cuando llegue a 1000 entre en el estado contrario al que se encuentra perdiéndose si está apagado o apagándose si esta prendido. El funcionamiento de esto es simple* ***si contamos 1000 veces 0.001 segundo tendremos como resultado 1 segundo.***

Timer1:

Modos y prescalers:

Modos:

T1_DISABLE (Para deshabilitarlo)

T1_INTERNAL (Como temporisador)

T1_EXTERNAL (Como contador)

T1_EXTERNAL_SYNC (Como contador)

T1_CLK_OUT

Preescalers:

T1_DIV_BY_1

T1_DIV_BY_2

T1_DIV_BY_4

T1_DIV_BY_8

Cálculos previos:

Los cálculos a realizar con el timer1 son los mismos que con el timer0 solo cabiando algunos valores debido a que este es de 16 bits mientras que el timer0 es de 8 bits admitiendo de esta forma un tiempo máximo mayo. Además de que los peescarles del timer1 son diferentes a los del timer0

$$T_{cm} = \frac{4}{F_{osc}} = \frac{4}{2 \cdot 10^6} = 0.000002seg$$

$$T_{max} = \frac{4}{F_{osc}} \cdot preescaler \cdot (65536 - TMR0)$$

$$T_{max} = \frac{4}{2 \cdot 10^6} \cdot 8 \cdot (65536 - 0) = 1.048576seg$$

$$TMR1 = 65536 - \left(\frac{Tiempo \cdot F_{osc}}{4 \cdot preescaler}\right) = 65536 - \frac{0.1 \cdot 2 \cdot 10^6}{4 \cdot 8} = 59286$$

Donde:

T_{cm} *Ciclos maquina*

F_{osc} *Frecuencia del oscilador*

T_{max} *Tiempo máximo que podrá contar el Timer0 con esa frecuencia del oscilador del microprocesador y el preescaler.*

$TMR0$ *Sera el valor que se utilizara para configurar el timer0*

Ejemplo:

Si queremos generar un tiempo de 100ms tenemos un oscilador de 2 MHz y usando el preescaler de 8 tendremos

$$TMR1 = 65536 - \left(\frac{Tiempo \cdot F_{osc}}{4 \cdot preescaler}\right) = 65536 - \frac{0.1 \cdot 2 \cdot 10^6}{4 \cdot 8} = 59286$$

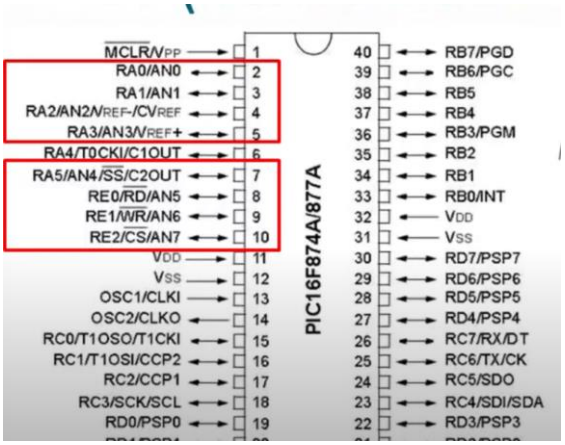
```
1  #INT_TIMER1
2  void timer1_interrupcion(){
3      //codigo del timer1
4  }
5
6  void main(){
7      Setup_timer_1 (T1_INTERNAL | T1_DIV_BY_8);
8      enable_interrupts(INT_TIMER1);
9      enable_interrupts(GLOBAL);
10     set_timer1 (59286);
11
12 }
```

Capítulo III:

ADC Conversor análogo-digital.

Introducción.

El conversor análogo-digital conocido comúnmente como adc nos permitirá realizar lecturas de ciertos sensores que nos entreguen señales analógicas y convertirlas en digital. En el PIC 16f877A tiene 8 pines que pueden ser configurados como analógicos.



Instrucciones para el ADC.

setup_adc_ports(); - Configurar los puertos como analógicos.

setup_adc(); - Seleccionar el modo para realizar la conversión.

setup_veref(); - Habilitar los pines para tensión de referencia.

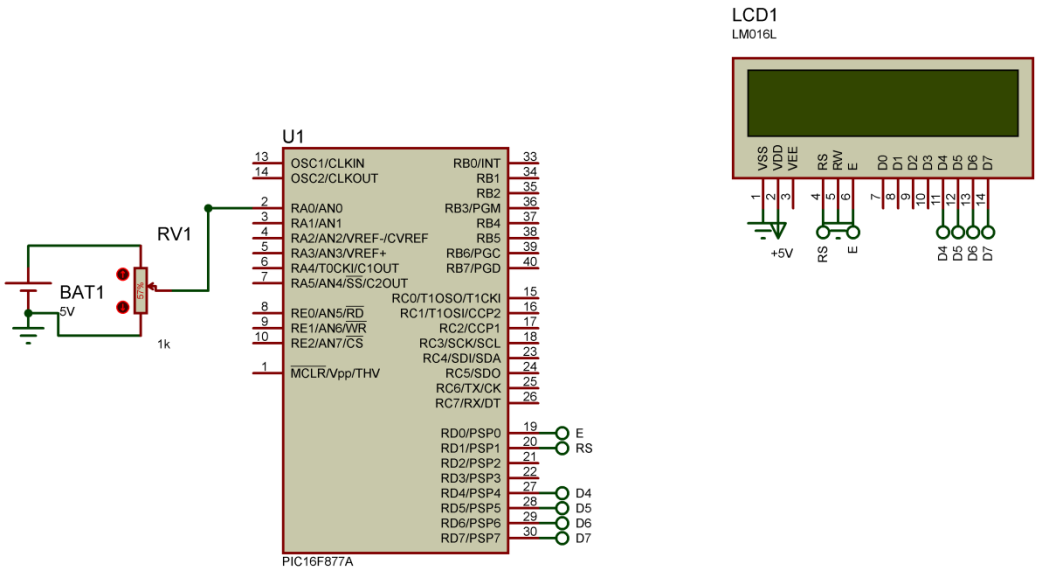
set_adc_channel(); - Seleccionar el canal para hacer la lectura.

read_adc(); - Retornar el valor obtenido de la lectura del adc

Ejemplo.

```
1
 2 #include <16f877a.h>
 3 #device ADC = 10
 4 #FUSES XT, NOWDT
 5 #use delay(clock=2M)
 6 #use standard_io(B)
 7
 8 #define LCD_DB4   PIN_D4         // Pines de la pantalla LCD
 9 #define LCD_DB5   PIN_D5
10 #define LCD_DB6   PIN_D6
11 #define LCD_DB7   PIN_D7
12 #define LCD_RS    PIN_D2
13 #define LCD_E     PIN_D3
14
15 #include <lcd.c>            // Libreria para el manejo de la
16
17
18 float valor_adc;
19 float valor_res;
20
21 void main()
22 {
23    lcd_init();
24    setup_adc_ports(AN0); //Declaramos adc en el puerto AN0
25    setup_adc(adc_clock_internal);
26    while(TRUE)
27    {
28       set_adc_channel(0);
29
30       valor_adc = read_adc();
31       valor_res = 5* valor_adc/1024;
32
33       lcd_gotoxy(1,1);
34       printf(lcd_putc,"ADC: %f ",valor_adc);
35       lcd_gotoxy(1,2);
36       printf(lcd_putc,"Tension: %01.2fV ",valor_res);
37
38    }
39
40 }
41
```

De



,

De las cosas nuevas no relacionadas con el ADC son las líneas de la 7 a la 12 donde se declaran los pines que se usaran para la lcd de 16x2.

Entrando en términos del adc lo primero será declarar su resolución que es este caso será de 10 para este microprocesador, aunque se puede bajar esta resolución no es recomendable.

```
#device ADC = 10
```

Se crean 2 variables para almacenar el valor de la lectura del acd y el valor real de la medición.

```
float valor_adc;
float valor_res;
```

se selecciona los puertos analógicos que se trabajaran en este caso AN0 y se configura el modo para el adc el cual en la mayoría de los casos será adc_clock_internal, el cual le dirá al microprocesador que el adc trabajara la lectura con el reloj interno.

```
setup_adc_ports(AN0);
setup_adc(adc_clock_internal);
```

Por ultimo seleccionamos el canal para hacer la lectura (`set_adc_channel(0)`*) , con* `valor_adc = read_adc()` *almacenare el valor de la lectura del adc en la variable valor_adc el cual tendrá un valor entre 0 y 1025 debido a la resolución de 10 bits como es lógico este valor no es de tensión por lo que necesitaremos convertirlo a uno para poder saber el valor real que tenemos en ese puerto y para esto tenemos la siguiente ecuación* $V_{real} = \frac{V_{max} \cdot ValorADC}{Resolucion(1025)} = \frac{5 * ValorADC}{1025}$.

V_{max} *esta variable tomara el valor de 5 en este caso debido a que el valor de la fuente conectada al adc además de ser el máximo valor que este puerto soportara.*

Aclaración: esta fórmula podrá ser utilizada para otro tipo de conversiones donde se necesite llevar de un valor en un rango de 0-1025 a otro un ejemplo de esto podría ser de 0-100 para porcentajes:

$$\% = \frac{100 \cdot ValorADC}{Resolucion(1025)}$$

O lo mismo aplicado a una escala de 0-256 para PWM que se verán en el siguiente capítulo los cuales constan con una resolución de 8 bits

$$PWM = \frac{256 \cdot ValorADC}{Resolucion(1025)}$$

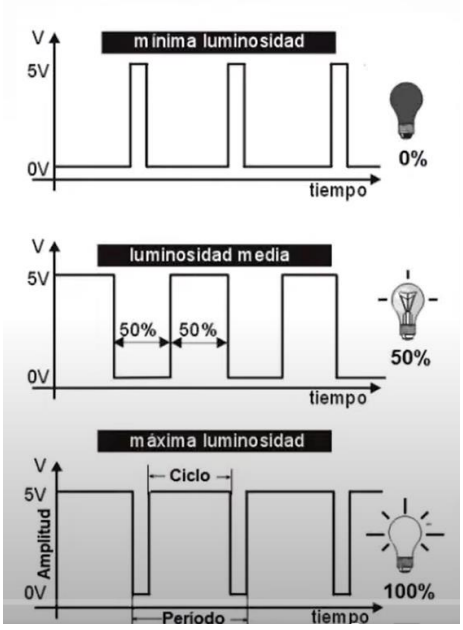
Recuadro 1.1: Cambio de rango

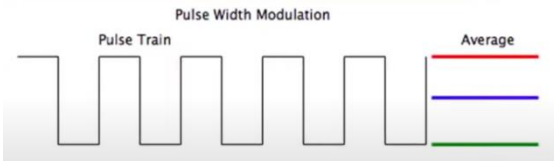
Capítulo IV:

PWM Modulación por ancho de pulso.

Introducción.

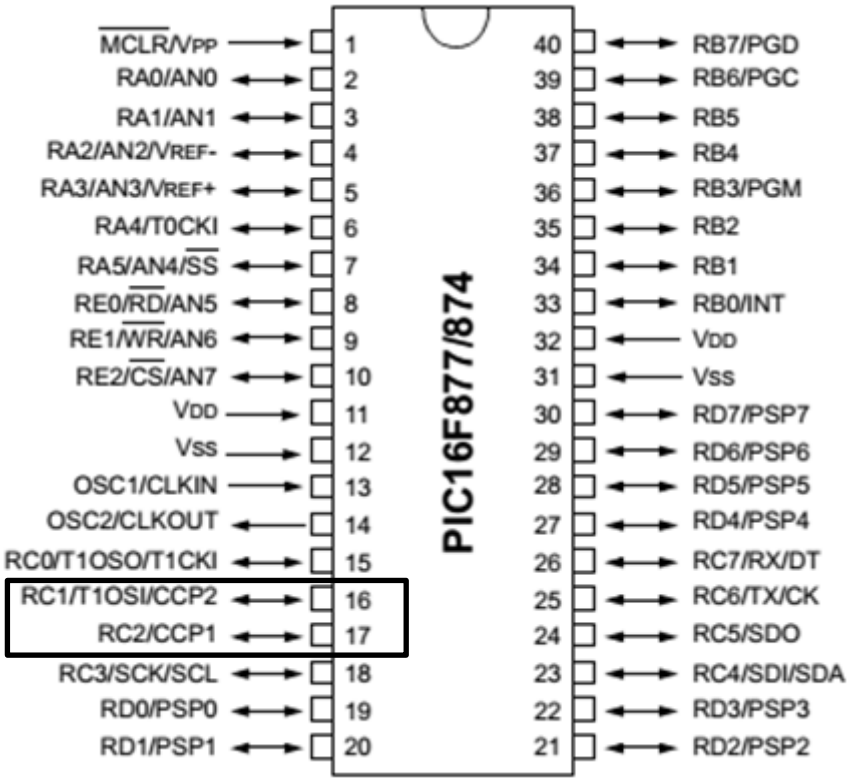
La aplicación del pwm es la variación del ancho de pulso con el cual se puede controlar motores luces o incluso el disparo de los transistores y tiristores. Lo que se varia en el pwm lo que se varia es el ciclo de trabajo o también conocido como duty.





Módulos CCP.

Para poder generar estas señales pwm con nuestro PIC, se hace uso de los módulos CCP (Comparador, Captura y PWM). Dicho modulo permite realizar estas tres funciones básicas



Configuración.

setup_timer_2(ts_div_by_16, 255,1); Utilizamos el timer 2 ya que es usado para generar frecuencias, preescaler de 16 (en este ejemplo), 255 que será el periodo máximo del pwm y un poscaler de 1

Un ejemplo de como configurar esto seria

Se desea generar un PWM que trabaje a 5 kHZ con diferentes ciclos útiles

Si se trabaja con un Pre-escalador del temporizador 2 unitario, entonces

$$PR2 = \frac{f_{osc}}{4 \cdot f_{PWM}} - 1 = \frac{4000000}{4 \cdot 5000} - 1 = 199$$

setup_timer_2(t2_div_by_1, 199,1)

setup_ccpx(modo);

*set_pwmx_duty(valor); (**valor = (0 a 255)**)*

SETUP_CCPx(MODO);	MODO	Registro CCPxCON (17h/D1h)
CCP_OFF	Deshabilitación	00000000 (00h)
CCP_CAPTURE_FE	Captura por flanco de bajada	00000100 (04h)
CCP_CAPTURE_RE	Captura por flanco de subida	00000101 (05h)
CCP_CAPTURE_DIV_4	Captura tras 4 pulsos	00000110 (06h)
CCP_CAPTURE_DIV_16	Captura tras 16 pulsos	00000111 (07h)
CCP_COMPARE_SET_ON_MATCH	Salida a 1 en comparación	00001000 (08h)
CCP_COMPARE_CLR_ON_MATCH	Salida a 0 en comparación	00001001 (0A9)
CCP_COMPARE_INT	Interrupción en comparación	00001010 (0Ah)
CCP_COMPARE_RESET_TIMER	Reset TMR en comparación	00001011 (0Bh)
CCP_PWM	Habilitación PWM	00001100 (0Ch)

*Como se ve en esta tabla para configurar el módulo CCP existen varios modos, pero solo usaremos el ultimo ya que es el del PWM y es lo que queremos conseguir con este modulo. Esto quedaría **setup_ccp1(CCP_PWM**)*

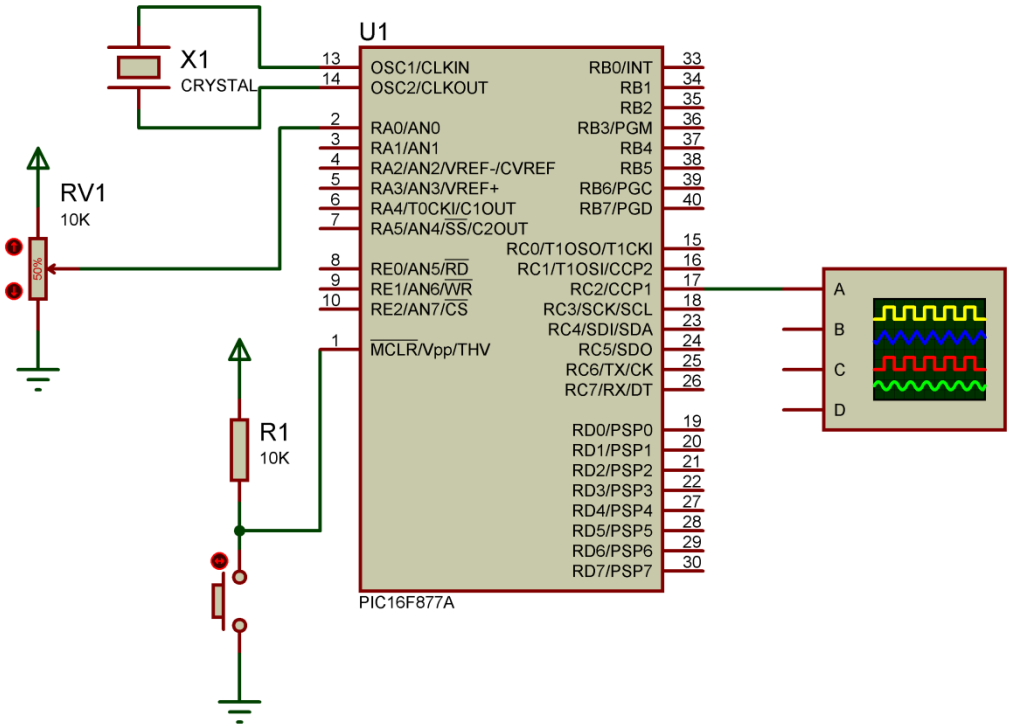
Ejemplos.

Ejemplo 1:

Este ejemplo es el más sencillo posible donde con un potenciómetro variamos el ciclo útil del pwm.

```
1  #include <16f877a.h>
2  #device ADC = 10
3  #fuses HS,NOWDT,NOPROTECT,NOPUT,NOLVP,BROWNOUT
4  #use delay(clock=20M)
5
6
7  long valor_adc;
8  int valor_pwm;
9
10 void main()
11 {
12    setup_timer_2(t2_div_by_16, 255, 1);          // Configura timer 2
13    setup_ccp1(ccp_pwm);                          // Configura modulo
14    setup_adc_ports(AN0);                         // ADC canal 0
15    setup_adc(adc_clock_internal);                // Reloj interno
```

```
16    set_pwm1_duty(0);                              // Inicia el pwm 1
17
18    while(true)
19    {
20       set_adc_channel(0);
21       delay_us(2);
22       valor_adc = read_adc();
23       valor_pwm = 255*valor_adc/1025;
24       set_pwm1_duty(valor_pwm);
25    }
26 }
```



Entonces en estas líneas

```
setup_timer_2(t2_div_by_16, 255, 1);          // Configura timer 2
setup_ccp1(ccp_pwm);                          // Configura modulo
setup_adc_ports(AN0);                         // ADC canal 0
setup_adc(adc_clock_internal);                // Reloj interno
set_pwm1_duty(0);                             // Inicia el pwm 1
```

Como sus comentarios indican primero configuramos el timer2 como explicamos previamente al igual que el módulo ccp1. Luego configuramos el canal del ad casi como el modo como se vio en el capítulo anterior.Por último se inicia el pwm en 0.

Para la parte del bucle tenemos que

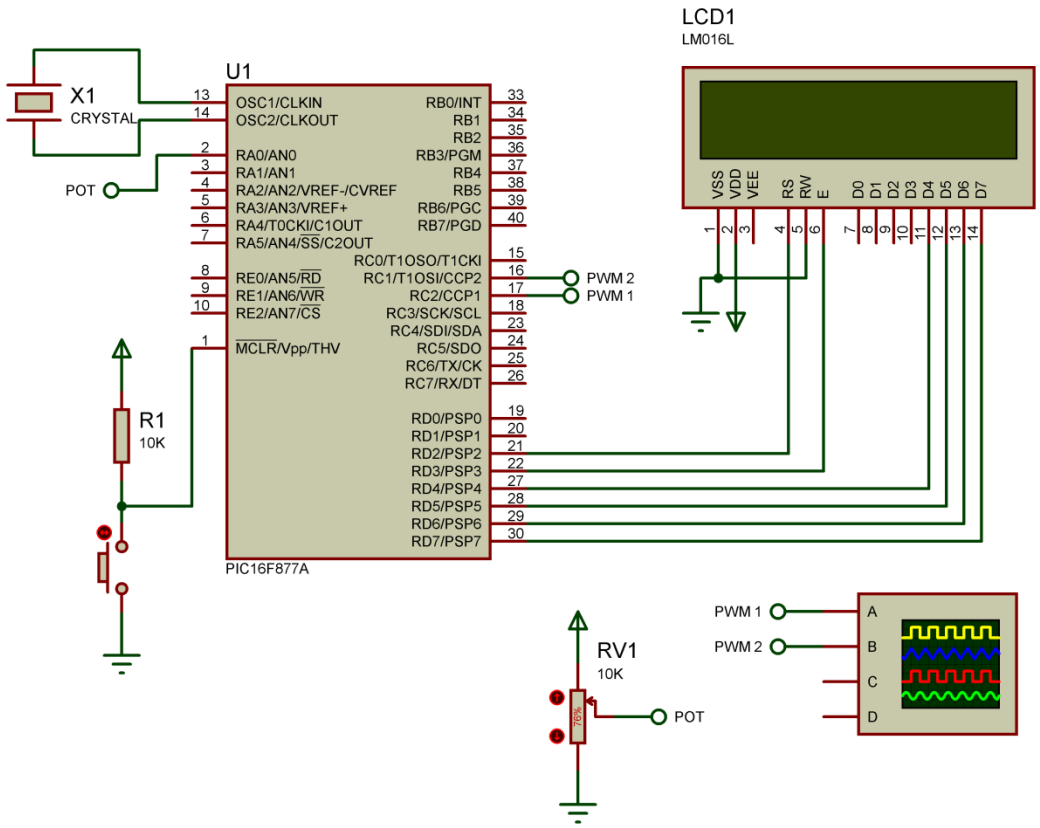
```
set_adc_channel(0);
valor_adc = read_adc();
valor_pwm = 255*valor_adc/1025;
set_pwm1_duty(valor_pwm);
```

Lo primero es la configuración del adc para poder leer la señal analógica que nos dará el potenciómetro asi como almacener la lectura de este en la variable valor_adc. Para finalizar convertimos el valor de la lectura del adc de 0-1025 a 0-255 (Recuadro 1.1) que será la capacidad del pwm y se lo pasamos a la función set_pwm1_duty() de esta forma estableciendo el ciclo útil en proporción al el potenciómetro.

Ejemplo 2:

```
1  #include <16f877a.h>
 2 #device ADC = 10
 3 #fuses HS,NOWDT,NOPROTECT,NOPUT,NOLVP,BROWNOUT
 4 #use delay(clock=20M)
 5 #use standard_io(D)
 6
 7 #define LCD_DB4   PIN_D4         // Pines de la pantalla LCD
 8 #define LCD_DB5   PIN_D5
 9 #define LCD_DB6   PIN_D6
10 #define LCD_DB7   PIN_D7
11 #define LCD_RS    PIN_D2
12 #define LCD_E     PIN_D3
13 #include <LCD_16X2.c>
14
15
16 long valor_adc;
17 int valor_pwm;
18 int porcentaje;
19
20 void main()
21 {
22    lcd_init();
23    setup_timer_2(t2_div_by_16, 255, 1);
24    setup_ccp1(ccp_pwm);
25    setup_ccp2(ccp_pwm);
26    setup_adc_ports(AN0);
27    setup_adc(adc_clock_internal);
28    set_pwm1_duty(0);                // Motor PIN_C2
29    set_pwm2_duty(0);                // Led PIN_C1
30
31    while(true)
```

```
32    {
33       set_adc_channel(0);
34       delay_us(2);
35       valor_adc = read_adc();
36       valor_pwm = 255*valor_adc/1025;
37       porcentaje = 100*valor_pwm/255
38       set_pwm1_duty(valor_pwm);
39       set_pwm2_duty(valor_pwm);
40
41       lcd_gotoxy(1,1);
42       printf(lcd_putc,"PWM: %u ",valor_pwm);
43       lcd_gotoxy(1,2);
44       printf(lcd_putc,"Porcentaje: %u%% "porcentaje);
45       delay_ms(50);
46    }
47 }
```



En este ejemplo es mas de lo mismo de las pocas cosas que cambian es q se usaran 2 salidas de pwm configuradas con el mismo valor y se usara una conversión para dar porcentaje.

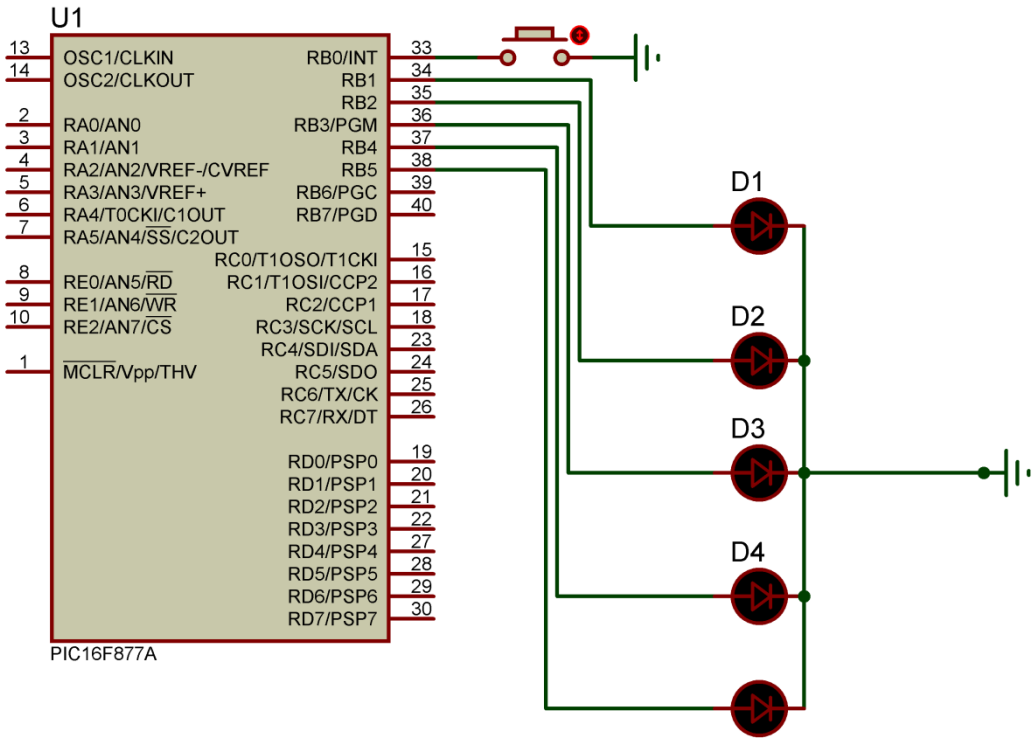
Capítulo V:

Problemas y como abordarlos.

Interrupciones externas.

Diseñe un sistema que controla el encendido de dos leds de diferentes colores. Una interrupción externa presupone encender un led diferente cada vez

Cabe recalcar que para un problema tan sencillo como este no es necesario el uso de interrupciones externas pero como es parte de la orden se hará. Para el hardware se usara un micro controlador PIC16f877A ya que es el que hemos venido usando a través de los ejemplos pero podría usarcé uno más pequeño dado que no será necesario tantos pines. Además se usara un botón para la interrupción conectado en RB0 y bueno claro algunos leds quedando.



Para realizar la programación de esto empezaremos con lo normal

```
1 #include <16f877a.h>
2 #fuses HS,NOWDT
3 #use delay(clock=2M)
4 #use standard_io(B)
5 #BYTE OPTION_REG = 0x81
```

Declarando en la librería del pic la frecuencia del reloj, que utilizaremos los puertos B y declaramos el pull-up

```
1  void main()
2  {
3          bit_clear(OPTION_REG, 7);
4          enable_interrupts(int_ext);
5          ext_int_edge(H_to_L);
6           enable_interrupts(GLOBAL);
7
8          while(TRUE)
9          {
10                 //TODO: User Code
11         }
12
13 }
```

Para la función de la interrupción tenemos que crear una variable que llevara el conteo de cuantas veces se ha entrado en la interrupción.
int Veces;

```
1  #INT_EXT
2
3  void ext_isr() {
4     Veces++;
5
6
7     if (Veces == 1){
8        output_high(PIN_B1);
9        output_low(PIN_B5);
10    }
11    else if(Veces == 2){
12       output_high(PIN_B2);
13       output_low(PIN_B1);
14    }
15    else if(Veces == 3){
16       output_high(PIN_B3);
17       output_low(PIN_B2);
18    }
19    else if(Veces == 4){
20       output_high(PIN_B4);
21       output_low(PIN_B3);
22    }
23    else if(Veces == 5){
24       output_high(PIN_B5);
25       output_low(PIN_B4);
26       Veces = 0;
27    }
28
29
30
31 }
```

Las líneas 1 y 3 como vimos en el ejemplo se mantiene igual y siempre será igual. Para la línea 4 lo que se hace es incrementar el contador previamente creado y asi llevaremos la cuenta.

Los if y else if en su condición esta

Veces == #

Veces es el contador previamente creado

El == se usa dado que se hará una comparación ***importante nunca en un if o while poner un solo igual dado que en su argumento (lo que se encuentra entre el paréntesis) es una comparación.*** *Para más información de cuando*

se usa un igual o dos ver una bibliografía más básica sobre programación C o C++

Y # será la cantidad de veces presionada la interrupción y corresponderá a un led en específico y en el último else if volverá a 0 para que el ciclo pueda ser utilizado nuevamente.

Dentro de los corchetes de las condiciones lo que se hace es que se prende el led correspondiente y se apaga el anterior ya q estos se prenderán en orden.

El código deberá quédate algo así.

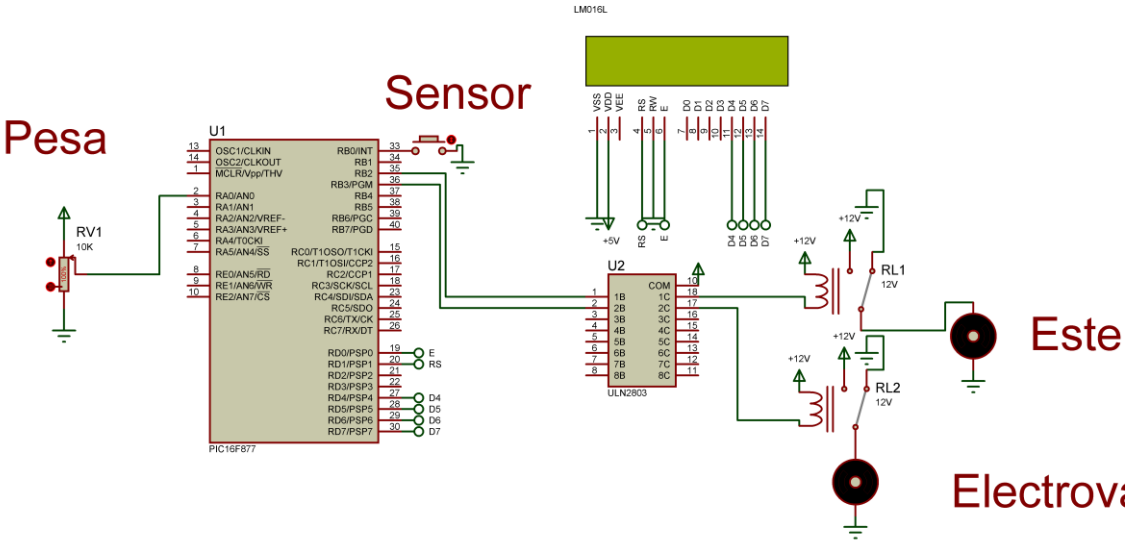
```
#include <16f877a.h>
#fuses HS,NOWDT
#use delay(clock=2M)
#use standard_io(D)
#BYTE OPTION_REG = 0x81

int Veces;


#INT_EXT

void ext_isr() {
   Veces++;


   if (Veces == 1){
      output_high(PIN_B1);
      output_low(PIN_B5);
   }
   else if(Veces == 2){
      output_high(PIN_B2);
      output_low(PIN_B1);
   }
   else if(Veces == 3){
      output_high(PIN_B3);
      output_low(PIN_B2);
   }
   else if(Veces == 4){
      output_high(PIN_B4);
      output_low(PIN_B3);
   }
   else if(Veces == 5){
      output_high(PIN_B5);
      output_low(PIN_B4);
      Veces = 0;
   }


}
```

```
41
42
43 void main()
44 {
45    bit_clear(OPTION_REG, 7);  //Habilitacion PULL-UP
46    enable_interrupts(int_ext);  //Habilitacio interrupciones
47    ext_int_edge(H_to_L);       //Interrupcion externa en RB0
48    enable_interrupts(GLOBAL); //Habilitacio interrupciones Globales
49
50    while(TRUE)
51    {
52
53    }
54
55 }
```

Frascos de chocolate.

En una fábrica de procesamiento de chocolate en polvo se desea automatizar el proceso final consistente en el llenado de frascos de 1 kg. Se dispone de un PIC16F877 y una pesa digital. Los requerimientos a cumplir son los siguientes.

- ***Una vez energizado el sistema de control una estera se pone en movimiento desplazando el frasco a llenar.***
- ***La llegada a su posición de llenado se detecta por una IT externa en cuyo caso se abre una electroválvula.***
- ***La pesa digital permite conocer cuándo se ha depositado 1 kg de chocolate en el envase siendo su escala máxima 2 kg***
- ***Se considera válido chequear el estado de llenado cada 400 ms.***
- ***Una vez alcanzado el peso requerido se cierra la electroválvula y se pone en movimiento nuevamente la estera hasta la llegada de un nuevo frasco a la posición de llenado en cuyo caso se repite el proceso.***
- ***El manejo del frasco lleno se realiza manualmente***




```c
#include <16f877.h>
#device ADC = 10
#FUSES XT, NOWDT
#use delay(clock=2M)
#use standard_io(D)
#BYTE OPTION_REG = 0x81
#define LCD_DB4   PIN_D4
#define LCD_DB5   PIN_D5
#define LCD_DB6   PIN_D6
#define LCD_DB7   PIN_D7
#define LCD_RS    PIN_D1
#define LCD_E     PIN_D0


#include <lcd.c>
int i=0;
float valor_adc;


#INT_EXT

void ext_isr() {
   lcd_gotoxy(1,1);
   printf(lcd_putc,"Llenando ");
   lcd_gotoxy(1,2);
   printf(lcd_putc,"PWM: 0.00                ",);
   output_low(PIN_B2);
   output_high(PIN_B3);
```

```
29    Setup_timer_1 (T1_INTERNAL | T1_DIV_BY_8);
30    set_timer1 (59286);
31    enable_interrupts(INT_TIMER1);
32
33
34
35
36 }
37
38 #int_TIMER1
39 void TIMER1_isr(void) {
40
41    set_timer1(59286); // Reset timer to get 400ms delay
42
43    if (i<3)  i++;
44    else {
45       i=0;
46       valor_adc = read_adc ();
47       lcd_gotoxy(1,1);
48       printf(lcd_putc,"Llenando ");
49       lcd_gotoxy(1,2);
50       printf(lcd_putc,"PWM: %f ",valor_adc);
51       if(valor_adc > 1000) {
52
53          output_low(PIN_B3);
54          output_high(PIN_B2);
55          Setup_timer_1 (T1_DISABLED);
56          lcd_gotoxy(1,1);
57          printf(lcd_putc,"Lleno           ");
58
59
60 aa
61          lcd_gotoxy(1,2);
62          printf(lcd_putc,"PWM: %f ",valor_adc);
63          delay_ms(1000);
64          lcd_gotoxy(1,1);
65          printf(lcd_putc," Moviendo");
66          lcd_gotoxy(1,2);
67          printf(lcd_putc,"PWM: 0.00             ",);
68
69       }
70
71    }
72
73
74
75
76
77
78 }
79
80
81
82
83 void main()
84 {
85    lcd_init();
```

```
 86
 87    bit_clear(OPTION_REG, 7);
 88    enable_interrupts(int_ext);
 89    ext_int_edge(H_to_L);
 90    enable_interrupts(GLOBAL);
 91
 92    setup_adc_ports(ALL_ANALOG);
 93    setup_adc(ADC_CLOCK_INTERNAL);
 94    set_adc_channel(0);
 95
 96    lcd_gotoxy(1,1);
 97    printf(lcd_putc," Moviendo");
 98    lcd_gotoxy(1,2);
 99    printf(lcd_putc,"PWM: %f ",valor_adc);
100    output_low(PIN_B3);
101    output_high(PIN_B2);
102
103
104
105
106
107    while(TRUE)
108    {
109
110       // empieza la estera en bucle
       }

    }
```

La forma para abordar esto es simple se nos pide que una vez que el frasco llegue a la interrupción externa abres la electroválvula y paras la estará

```
output_low(PIN_B2);
output_high(PIN_B3);
```

Se configura y habilita el timer 1 para que empiece a contar

```
Setup_timer_1 (T1_INTERNAL | T1_DIV_BY_8);
set_timer1 (59286);
enable_interrupts(INT_TIMER1);
```

El valor de 59286 viene dado por la formula

$$TMR1 = 65536 - \left(\frac{Tiempo \cdot F_{osc}}{4 \cdot preescaler}\right) = 65536 - \frac{0.1 \cdot 2 \cdot 10^6}{4 \cdot 8} = 59286$$

El cual contara 0.1 seg o 100ms y como lo que queremos es que cheque cada 400ms hacemos q cuente 4 veces y en la cuarto no se cumplirá esta condición y entrara en

```
if (i<3)  i++;


else {
      i=0;
      valor_adc = read_adc ();
```

Con el cual iremos leyendo el valor del ADC hasta un punto en nuestro caso pusimos un valor simbólico para representar los 2kg del peso del frasco

```
if(valor_adc > 1000) {

         output_low(PIN_B3);
         output_high(PIN_B2);
         Setup_timer_1 (T1_DISABLED);
```

Que una vez que se cumpla apagaremos la electroválvula encenderemos la estera y deshabilitamos el timer1 para que este no siga contando.