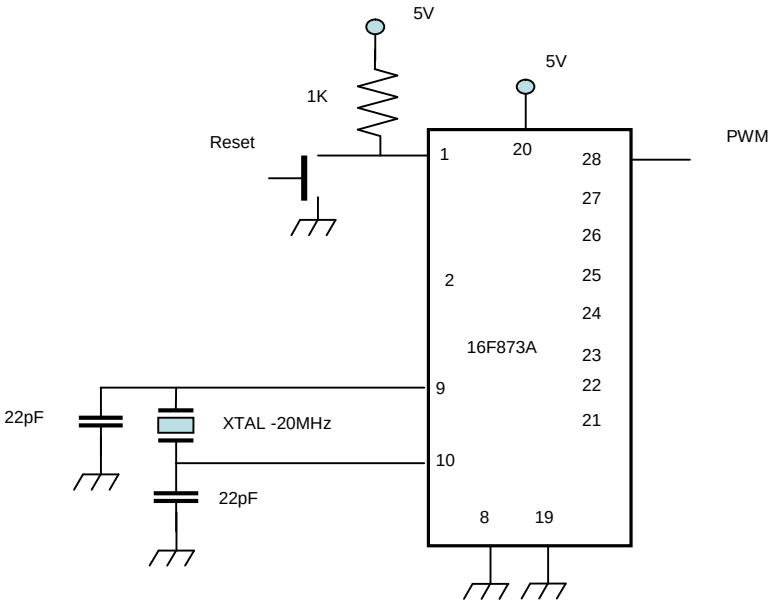
Aplicaciones de up y uC
Laboratorio #5
Generando PWM
Profesores: P. Muñoz, A. Navarro
Preparado Por: Juan Ignacio Huircán

Objetivos

- Armar el circuito básico para verificar el funcionamiento de un PWM
- Generando PWM con TMR0
- Generando PWM con el módulo CCPx

Actividades

1.- Realizar el montaje del circuito siguiente:



2.- Para generar un PWM con el TMR0, esto se hace vía interrupción.

Primero se debe implementar un clock con la frecuencia más grande posible, lo que implica

- PS2,PS1,PS0 con el divisor 1:2
- TMR0=254

Esto generará una base de tiempo muy pequeña. Este será el reloj principal para el PWM. El Duty Cycle más pequeño del PWM generado será periodo de esta señal.
Se definen tres variables, el periodo del PWM, llamado `TPWM`, el Duty Cycle llamado `DUTY` y un `contador` que establece cuando se cumple el `DUTY` y cuando se cumple el periodo `TPWM`.

Filosofía

- Se configura la interrupción TMR0
- Se inicializa `contador=0`
- Se inicializa `TPWM y DUTY`
- Se inicializa `TMR0=254`
- Se define un bit de salida con el valor 1, por ejemplo `PORTB.7=1`
- Cada vez que interrumpe el TMR0, se ejecuta la RSI
- Cuando entra a la RSI y verifica el TM0F y se realizan las siguientes tareas
    - Se incrementa el `contador`
    - Si el `contador =DUTY`, se hace el `PORTB.7=0`
    - Si el `contador=TPWM`, se hace el `PORTB.7=1` y `contador=0`
    - Se resetea `T0IF=0`

- Se inicializa TMR0=254

Por ejemplo si se define TPWM=100 y DUTY=20, durante las primeras 20 interrupciones del TMR0 PORTB.7 será 1 y luego se hará 0 durante las 80 interrupciones siguientes. Al llegar el contador a 100 se resetea el contador y parte todo de nuevo. Para modificar el ancho del pulso se debe cambiar el valor del DUTY.

```
#include "int16CXX.H"
#pragma origin 4

char contador, TPWM,DUTY;

interrupt servidor_de_int( void)
{
    int_save_registers    // W, STATUS (and PCLATH)
    if(T0IF)
     {
       contador++;
       if(contador==DUTY) PORTB.7=0;
       if(contador==TPWM)
        {
          PORTB.7=1;contador=0;
        }
       TMR0=254;
       T0IF=0; // reset flag de int TMR0
     }
    int_restore_registers // W, STATUS (and PCLATH)
}

void main()
{
  TPWM=100;
  DUTY=20;
  TRISB.7=0; // PORTB.0 de salida
  PORTB.7=1; // Manda un 1

 //Programación del TMR0

  OPTION.5=0; // T0CK=0, fuente clock interno
  OPTION.4=0; // T0SE,  Incremento con Canto de subida
  OPTION.3=0; // PSA=0  Pre escalar asignado al TMR0

  OPTION.2=0; // PS2=0 // Pre escalar 1:2
  OPTION.1=0; // PS1=0
  OPTION.0=0; // PS0=0
  TMR0=254;    // Carga con el valor inicial 254
  INTCON.5=1;  // habilita la int por TMR0  TMR0IE=1
  GIE=1;       // Habilita todas las int
  while(1) // Loop forever
  {
  }
}
```

Tareas a realizar

a.- Genere un proyecto con el programa indicado.

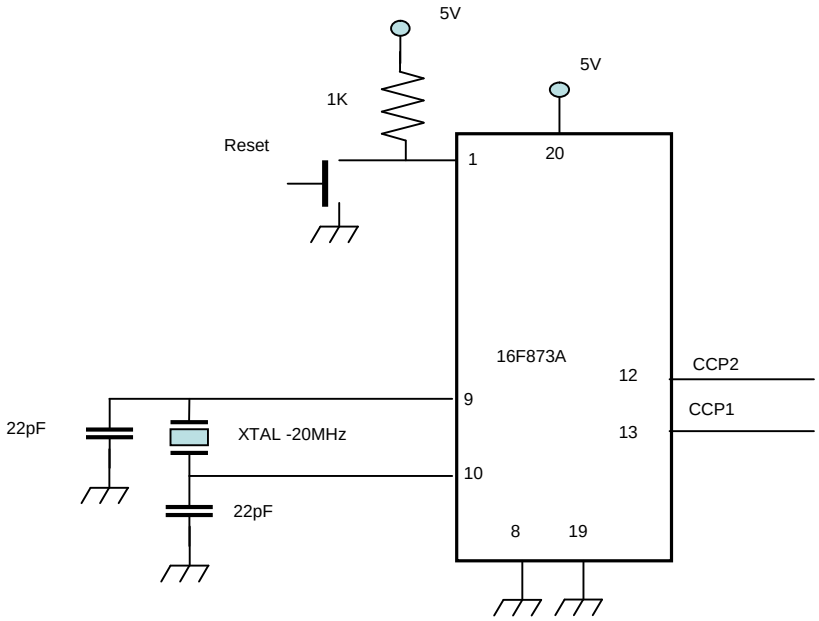
b.- Una vez grabado el programa en el uC, colocar el osciloscopio en el terminal PORTB.7 (28) y visualizar el PWM.

b.- Modificar cambiando el DUTY, ya sea a 25 a 30, a 75, etc, compilar, grabar y observar el cambio de duty cycle del PWM.

c.- Modificar el TPWM a 200 y probar con DUTY 25, 50 y 75.

3.- Generando PWM programando el módulo CCPx.

Montaje de trabajo



El sistema tiene dos módulos, el CCP1 o CCP2 cuya salida son la 13 y 12 respectivamente. El programa siguiente muestra la configuración de los registros de control accesando en forma directa los bits respectivos.

```
// Programación de acuerdo a los bits
// Prescalar de 16 PWM Freq = 1.22kHz - Clock 20MHz

void main()
{
   TRISC.2 = 0;

 // CCP1CON=0x00001100;

   CCP1M0=0; // Modo PWM
   CCP1M1=0;
   CCP1M2=1;
   CCP1M3=1;

// Los 2 bit siguientes forma los 10bit del duty con CCPR1L
// es decir:   CCPR1L CCP1X CCP1Y
// De acuerdo a la codificación sera
//             1000 0000 0   0

   CCP1X=0;  // Forman 10 bit  1000 0000 00=512
   CCP1Y=0;

 // CCPR1L=0b10000000;

   CCPR1L=0x80;

  //T2CON = 0b00000110;

  T2CKPS0=0;  //Presecalar 16
  T2CKPS1=1;
  TMR2ON=1;  // Timer2 ON

  TOUTPS3=0; // Post escalares 1:1
  TOUTPS2=0;
  TOUTPS1=0;
  TOUTPS0=0;
  PR2 = 0xFF; // PERIODO Seteado a 0xFF
```

```
   while(1)
   {
     CCPR1L=0x80;
   }
 }
```

Actividad

- ☐ Programe la aplicación
- ☐ Mida en el Terminal 13

Programación Simplificada para probar los dos PWM. La programación puede hacerse más rápida si se trabaja con el registro completo. En el ejemplo siguiente se accede directamente a los registros de control. La siguiente rutina prueba los 2 CCPx generando 2 pwm.

```
void main()
{
 TRISC = 0b10111001;
 PR2 = 0xFF; // PERIODO Seteado a 0xFF
 T2CON = 0b00000110; // Prescalar de 16 PWM Freq = 1.22kHz -- Clock 20MHz
 CCP1CON=0b00001100; //
 CCP2CON=0b00001100;
 while(1)
 {
  CCPR1L=0x40;
  CCPR2L=0x80;
 }
}
```

- ☐ Programar el uC y medir en los terminales 12 y 13

4.- Funciones de configuración y seteo.

Una vez probadas las modalidades del CCPx construir, implementar y probar las funciones:

```
void init_pwm(char numpwm)
{
}

void set_pwm(char numpwm, uns16 duty)
{

}
```

Ninguna de ellas retorna valor. Las variables `numpwm` hace referencia al módulo 1 o 2 (CCP1 o CCP2), la variable `duty` debe recibir un numero de 10bit para setear el duty cycle del pwm, esto es entre 0 y 1023.
Obs. Debe usar un if para discriminar el `numpwm` y luego de acuerdo a la condición (1 o 2) programar los registros de control correspondiente, `CCPCON1` o `CCPCON2`. Idem para la segunda función, pero además debe el valor de la variable `duty` debe desglosarla en dos partes, los 2 bit menos significativos van a los 2 bit del CCP1CON o CCP2CON, y los 8 bit más significativos al CCPR1L o CCPR2L dependiendo del `numpwm`.