CLASE4: PWM Y ADC

1. DESCRIPCION Y MANIPULACION DE FUNCIONES PARA PWM
   1.1. DEFINICION

MODULO CPP

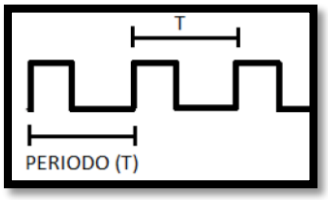
El microcontrolador PIC16F877A posee 2 módulos CCP (comparador, captura y PWM), estos módulos permiten realizar tres funciones basadas en el manejo de temporizadores.

- Comparador: Compara el valor del temporizador con el valor de un registro
- Captura: Obtiene el valor del temporizador en un momento dado, fijado por la acción de un terminal del PIC.
- PWM: Genera una señal modulada en ancho de pulso.

PWM

La modulación del ancho de pulso (PWM) de una señal es una técnica en la que se modifica el ciclo de trabajo de una señal periódica por ejemplo una senoidal o una cuadrada.

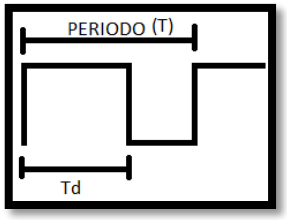
El periodo es un espacio de tiempo durante el cual se realiza una acción, en esta sección trataremos señales digitales ("1"s y "0"s); en una señal digital podemos represar el periodo como en la siguiente imagen:



$$\text{Frecuencia} = \frac{1}{\text{Periodo}}$$

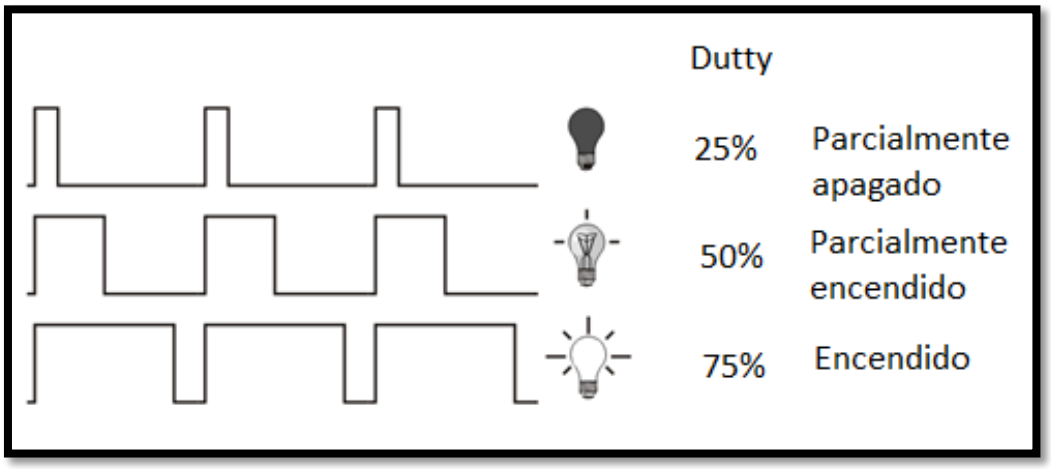
La frecuencia determina que tan rápido se completa un ciclo (por ejemplo: 20 Hz corresponde a 20 ciclos en un segundo), y por consiguiente que tan rápido se cambia entre los estados lógicos alto y bajo.

El ciclo de trabajo (duty cycle) de una señal describe la cantidad de tiempo que la señal está en un estado lógico alto, como un porcentaje de tiempo total que esta toma para completar un ciclo completo.



$$\text{Ciclo de trabajo} = \frac{\text{Td}}{\text{T}} \text{x} 100\%$$

Las aplicaciones de la técnica PWM son variadas y podemos citar algunas como controlar la intensidad de luz en un led, determinar la posición un servomotor, controlar la velocidad de un motor, etc.



ECUACION PARA PWM

$$PWM\ PERIOD = (PR2 + 1) * 4 * TOSC * (TMR2\ PRESCALER\ VALUE)$$

ECUACION PARA EL CICLO DE TRABAJO

$$PWM\ DUTY\ CYCLE = (CCPR1L{:}CCP1CON < 5{:}4 >)) * TOSC * (TMR2\ PRESCALER\ VALUE)$$

1.2. MANIPULACION DE DIRECTIVAS PARA PWM

Para utilizar en módulo CCP se usa el comando setup_ccpX(modo), donde X hace referencia al módulo, para el PIC16F877A será SETUP_CCP1(MODO) o SETUP_CCP2(MODO).
Modo se explicara en la siguiente tabla:

SETUP_CCPX(***MODO***)	**Descripción**
CCP_OFF	Deshabilita el módulo CCP
CCP_CAPTURE_FE	Captura por flanco de bajada
CCP_CAPTURE_RE	Captura por flanco de subida
CCP_CAPTURE_DIV_4	Captura tras 4 pulsos
CCP_CAPTURE_DIV_16	Captura tras 16 pulsos
CCP_COMPARE_SET_ON_MATCH	Salida a 1 en comparación
CCP_COMPARE_CLR_ON_MATC	Salida a 0 en comparación
CCP_COMPARE_INT	Interrupción en comparación
CCP_COMPARE_RESET_TIMER	Reset TMR en comparación
PWM	Habilitación PWM

1.3. MANIPULACION DE FUNCIONES

El PWM en el microcontrolador PIC16F877A tiene un registro de 10 bits dedicado a esta acción. Esto quiere decir que posee una resolución de 10 bits y no se debe exceder a este. La resolución se expresa con la ecuación y no debe exceder a 10 por que no funcionaría:

$$Resolución = \frac{log\left(\frac{Fosc}{Fpwm * prescaler}\right)}{log(2)}$$

El periodo de la señal PWM se obtiene al configurar el timer2, por lo que tendremos la siguiente ecuación:

$$PWMT = (PR2 + 1) * 4 * Tosc * (prescaler\ del\ timer2)$$

Donde PR2 es el periodo que se colocara en el comando del timer2, **setup_timer_2 (modo, periodo, postscaler)**, además el postscaler siempre debe tomar el valor de 1 y modo depende del prescalador.

Así obtendremos el comando final para configurar el timer2 y hacer uso del PWM:

setup_timer_2 (modo, PR2, 1)

Ahora para habilitar los módulos PWM tendremos el siguiente comando:

SETUP_CCP1(CCP_PWM)
SETUP_CCP2(CCP_PWM)

Para habilitar y modificar el ciclo de trabajo (Duty) usaremos los comandos:

SET_PWM1_DUTY (VALOR)
SET_PWM2_DUTY (VALOR)

Dónde: Valor = PR2*X%

1.4. CONFIGURACION PWM1 Y PWM2

Los módulos PWM del microcontrolador PIC16F877A pueden ser usados por separado y al mismo tiempo pero con algunas restricciones:
Si son usados al mismo tiempo las restricciones son:

- Los PWM deberán tener la misma frecuencia de funcionamiento.
- Deben tener la misma tasa de actualización de timer2.
- Lo único que varía es el ciclo de trabajo es decir el PWM1 puede tener un ciclo de trabajo del 20% y el PWM2 puede estar funcionando a un ciclo de trabajo de 65%.

2. DESCRIPCION Y MANIPULACION DE FUNCIONES PARA ADC

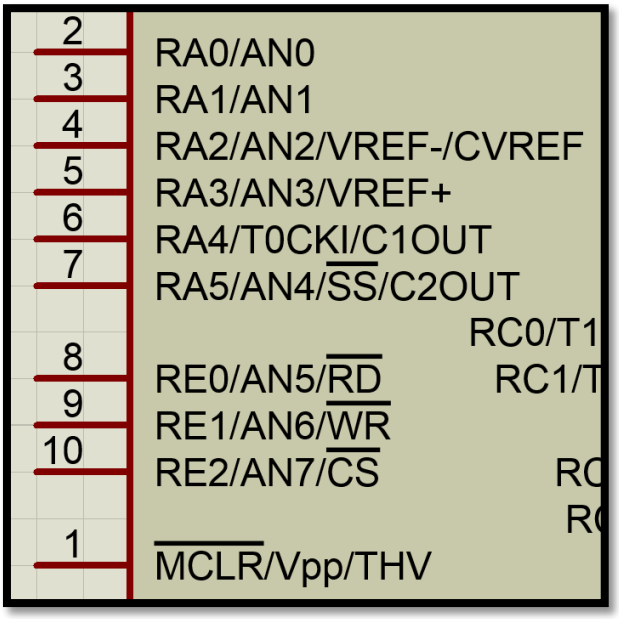
2.1. DEFINICION

El conversor análogo a digital es un dispositivo electrónico capaz de convertir una señal analógica, ya sea tensión o corriente, en una señal digital mediante un cuantificador y codificándose en muchos casos en código binario. Donde un código es la representación unívoca de los elementos, en este caso, cada valor numérico binario hace corresponder a un solo valor de tensión o corriente.

En la cuantificación de la señal se produce pérdida de la información que no puede ser recuperada en el proceso inverso, es decir, en la conversión de señal digital a analógica y esto es debido a que se truncan los valores entre 2 niveles de cuantificación, mientras mayor cantidad de bits mayor resolución y por lo tanto menor información pérdida.

Los microcontroladores PIC pueden incorporar un módulo de conversión de señal análoga a señal digital. Los módulos AD que utiliza Microchip hacen muestreo y retención con un condensador y después utiliza el módulo de conversión. El módulo de conversión A/D es de tipo de aproximaciones sucesivas

Los microcontroladores PIC16F877A dispone del módulo A/D y tiene a disposición 8 pines dedicados a dicha conversión, estos pines están ubicados en los puertos A y E.



Especificaciones del ADC

- Resolución: Especifica la precisión del ADC al medir señales de entrada analógica.

$$Resolución\ del\ ADC = \frac{(+Vref) - (-Vref)}{2^n - 1}$$

Donde n es el número de bits del conversor.

El PIC16F877A tiene 10 bits de conversión y el voltaje máximo para la conversión es de 5V.

$$Resolución\ del\ ADC = \frac{(5) - (0)}{2^{10} - 1}$$

$$Resolución\ del\ ADC = 4.8mV$$

- Voltaje de referencia
  La tensión de referencia específica el rango mínimo y máximo de tensión de entrada analógica. En PIC 18 hay dos tensiones de referencia, uno es el Vref- y uno es Vref +. El Vref- especifica el voltaje de entrada mínimo de entrada analógica, mientras que el Vref + especifica el máximo. Por ejemplo, si la señal de entrada se aplica a Vref- canal de entrada analógica a continuación, el resultado de la conversión será 0 y si el voltaje igual a Vref + se aplica al canal de entrada el resultado será 1023 (valor máximo para ADC de 10 bits).

- Canales de ADC
  El módulo ADC está conectado a varios canales a través de un multiplexor. El multiplexor puede conectar la entrada del ADC a cualquiera de los canales disponibles. Esto le permite conectar muchas señales analógicas a la MCU (digamos 3 sensores de temperatura.

- Tiempo de adquisición
  Cuando se selecciona un canal específico de la tensión de ese canal de entrada se almacena en un condensador de retención interna. Se necesita algún tiempo para el condensador para conseguir completamente cargada y convertirse igual a la tensión aplicada. Esta vez se llama tiempo de adquisición

- ADC Reloj (tiempo de conversión)
  ADC Requiere una fuente de reloj para hacer su conversión, esto se llama ADC Reloj. El período de tiempo del reloj ADC se llama TAD. También es el tiempo requerido para generar 1 bit de la conversión. El ADC requiere 12 TAD que hacer una conversión de 10 bits. Puede ser derivado desde el reloj de la CPU (llamado TOSC) dividiéndolo por un factor de división adecuado. Hay siete opciones posibles.

  - 2 x TOSC
  - 4 x TOSC
  - 8 x TOSC
  - 16 x TOSC
  - 32 x TOSC
  - 64 x TOSC
  - Internal RC

2.2. MANIPULACION DE FUNCIONES PARA ADC

El software ccs c compiler dispone de una librería dedicada a la conversión A/D, dicha librería se encuentra en <16f877a.h>, en esta librería encontramos todos los las configuraciones para el microcontrolador.
Para la construcción del programa deberemos seguir los siguientes pasos:
Primero debemos declarar el dispositivo ADC e indicar la cantidad de bits que posee con el comando **#device adc=10** en el caso del PIC16F877A. Este comando tiene que ubicarse inmediatamente después de la librería <16f877a.h> como se ve en la imagen de no ser así el compilador lo tomara como error.

```
#device  adc=10
```

Para configurar el ADC se hace uso del comando **SETUP_ADC(MODO)**, modo se describe en el siguiente cuadro:

SETUP_ADC(**MODO**)	**Descripción**
ADC_OFF	ADC deshabilitado
ADC_CLOCK_INTERNAL	Uso del reloj interno
ADC_CLOCK_DIV_2	Uso de reloj dividido entre 2
ADC_CLOCK_DIV_8	Uso de reloj dividido entre 8
ADC_CLOCK_DIV_32	Uso de reloj dividido entre 32

Para habilitación de los puertos ADC se hace uso del comando **SETUP_ADC_PORTS(VALOR)**, valor se describe en el siguiente cuadro:

SETUP_ADC_PORTS(VALOR)
NO_ANALOGS
ALL_ANALOG
AN0_AN1_AN2_AN4_AN5_AN6_AN7_VSS_VREF
AN0_AN1_AN2_AN3_AN4
AN0_AN1_AN2_AN4_VSS_VREF
AN0_AN1_AN3
AN0_AN1_VSS_VREF
AN0_AN1_AN4_AN5_AN6_AN7_VREF_VREF
AN0_AN1_AN2_AN3_AN4_AN5
AN0_AN1_AN2_AN4_AN5_VSS_VREF
AN0_AN1_AN4_AN5_VREF_VREF
AN0_AN1_AN4_VREF_VREF
AN0_AN1_VREF_VREF
AN0
AN0_VREF_VREF

Los comandos mencionados hasta el momento tendrá que ubicarse dentro de la función principal, en la imagen se muestra un ejemplo:

```
void main()
{
   //CONFIGURA RELOJ INTERNO
   SETUP_ADC(ADC_CLOCK_INTERNAL);
   //CONFIGURA TODOS LOS PINES A/D
   SETUP_ADC_PORTS(ALL_ANALOG);
}
```

Para finalizar usaremos 2 comandos un es **SETUP_ADC_CHANNEL(CANAL)** y el otro será **VALOR = READ_ADC().**

SETUP_ADC_CHANNEL(CANAL) activa el canal de donde se toma el dato, cabe resaltar tiempo que le toma en adquirir y convertir una muestra es de 20us por ello siempre tendrá que ser acompañado por un retardo, delay_us(20).
Canal es un valor referencial y se describe en el siguiente cuadro:

PIN	CANAL
AN0	0
AN1	1
AN2	2
AN3	3
AN4	4
AN5	5
AN6	6
AN7	7

VALOR = READ_ADC() en esta función se almacena el valor adquirido del canal previamente elegido, valor se declara como número entero de 16 bits porque hay que recordar que son 10 bits de adquisición, además valor toma valores de 0 a 1023.
Los comandos mencionados se usan generalmente dentro de una estructura de control, un ejemplo de ello en la siguiente imagen:

```
while(true)
{
   set_adc_channel(0);
   delay_us(20);
   valor=read_adc();
   num= 5.0*valor/1023.0;
```

3. EJEMPLOS

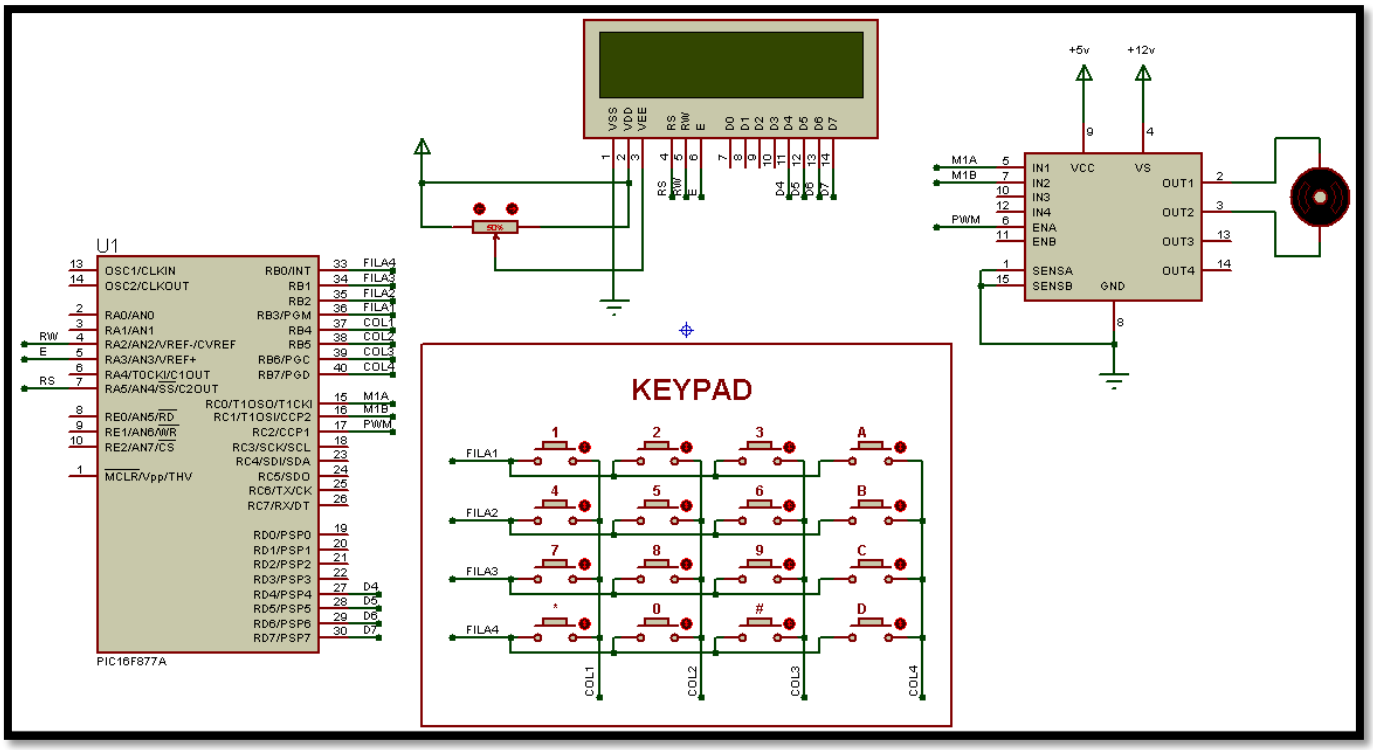
3.1. Ejemplo1

Control de velocidad y de giro de un motor DC

3.1.1.Código

```
for(;;){
   tecla = kbd_getc();
   if(tecla != 0){
      if(tecla == 'A'){
         lcd_gotoxy(1,1);
         printf(lcd_putc,"GiroDerecho");
         output_high( MOTOR1A ); output_low( MOTOR1B );
      }
      if(tecla == 'B'){
         lcd_gotoxy(1,1);
         printf(lcd_putc,"GiroDerecho");
         output_low( MOTOR1A ); output_high( MOTOR1B );
      }
      if(tecla == 'C'){
         duty++;
         SetPWM1( duty );
         lcd_gotoxy(1,2);
         printf(lcd_putc,"duty:%u",duty);
      }
      if(tecla == 'D'){
         duty--;
         SetPWM1( duty );
         lcd_gotoxy(1,2);
         printf(lcd_putc,"duty:%u",duty);
      }
   }
}
```

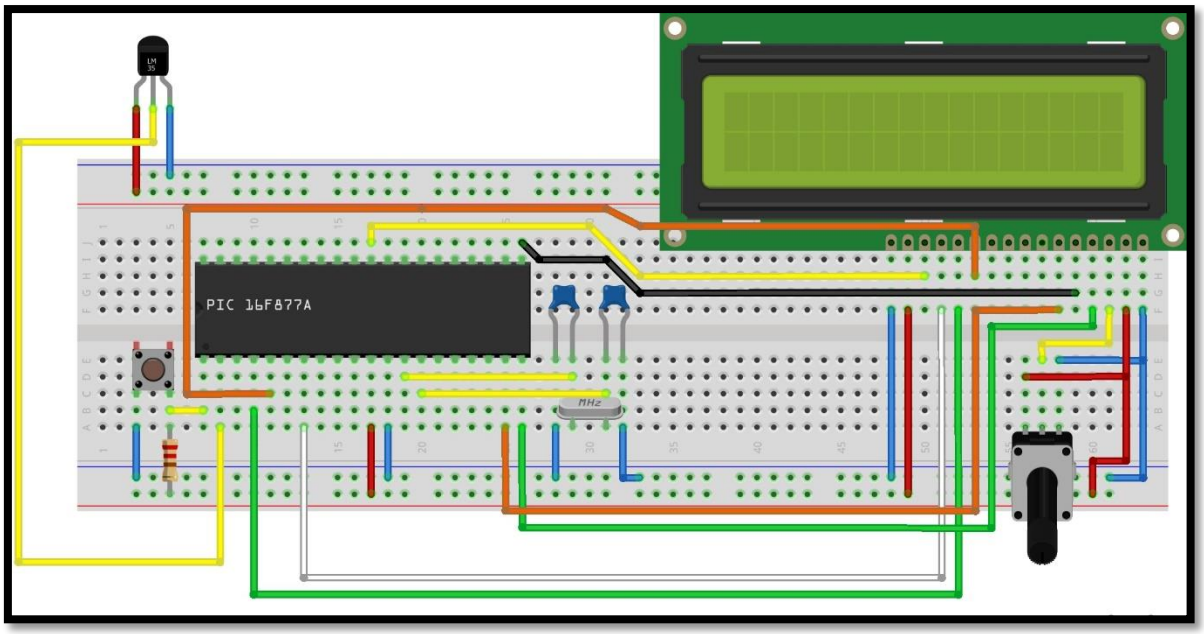
3.1.2.Simulación



3.2. Ejemplo2

Mostrar en la Lcd la Lectura del sensor lm35

3.2.1.Conexión



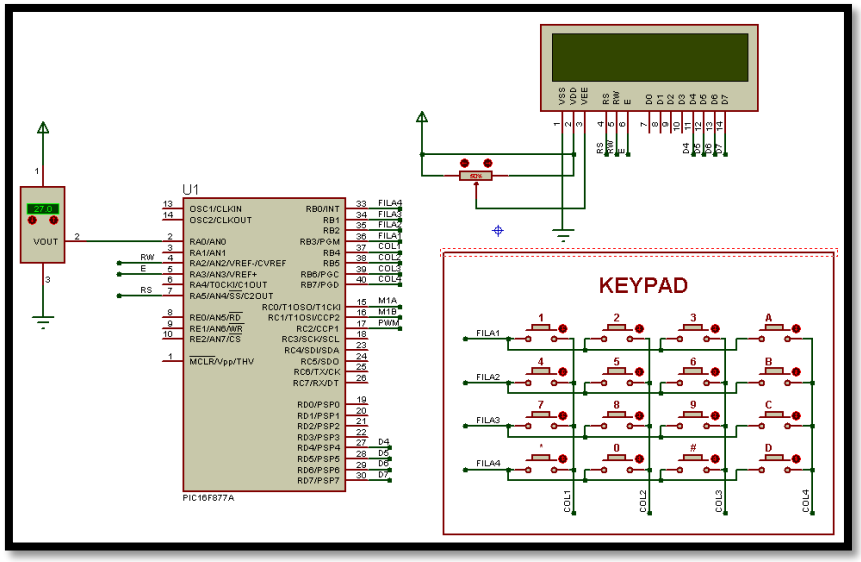
3.2.2.Código

```
void main(){
   int16 adc1;
   float sens;

   setup_adc(  ADC_CLOCK_INTERNAL );
   setup_adc_ports(  AN0 );
   lcd_init();

   for(;;){
      set_adc_channel( 0 );
      delay_us(20);
      adc1 = read_adc();
      sens = ((adc1*5)/1023.0)*100;
      lcd_gotoxy(1,1);
      printf(lcd_putc,"CanalAN0 = %Lu",adc1);
      lcd_gotoxy(1,2);
      printf(lcd_putc,"Temp = %f",sens);
      delay_ms(100);
      lcd_putc('\f');
   }
}
```

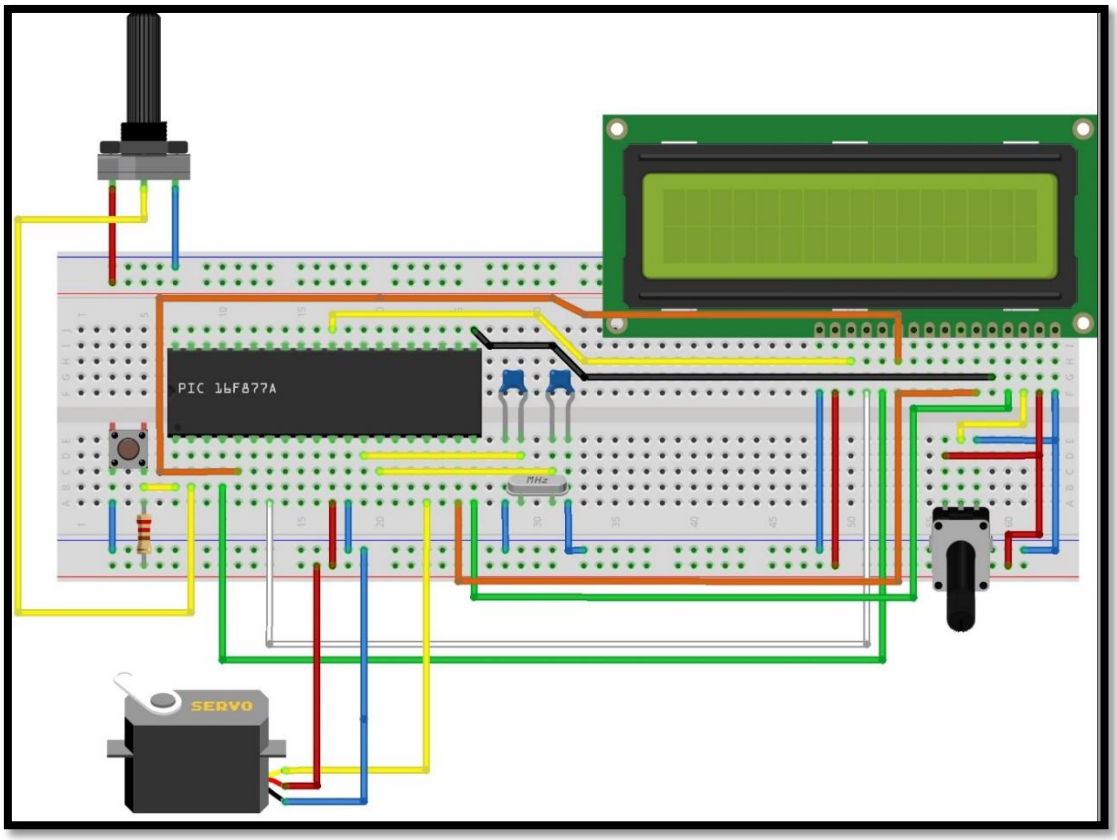
3.2.3.Simulación



3.3. Ejemplo3

Controlar el Angulo de giro de un servomotor mediante un potenciómetro conectado al canal AN0

3.3.1.Conexión



3.3.2.Código

```
void main(){
   int16 adc1;
   float sens;
   unsigned int16 i;

   setup_adc(  ADC_CLOCK_INTERNAL );
   setup_adc_ports(  AN0 );
   lcd_init();
   servomotor_init();

   for(;;){
      set_adc_channel( 0 );
      delay_us(20);
      adc1 = read_adc();
      i = ServoWriteAdc_To_Angle( adc1 );
      ServoWriteAngle(i);
      lcd_gotoxy(1,1);
      printf(lcd_putc,"CanalAN0 = %Lu",adc1);
      lcd_gotoxy(1,2);
      printf(lcd_putc,"Angulo = %Lu",i);
      delay_ms(100);
      lcd_putc('\f');
   }
}
```