

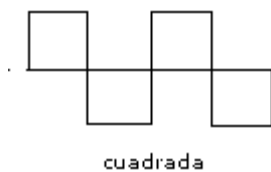
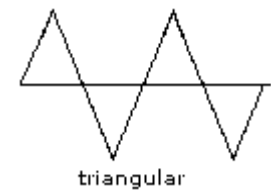
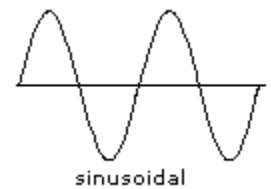
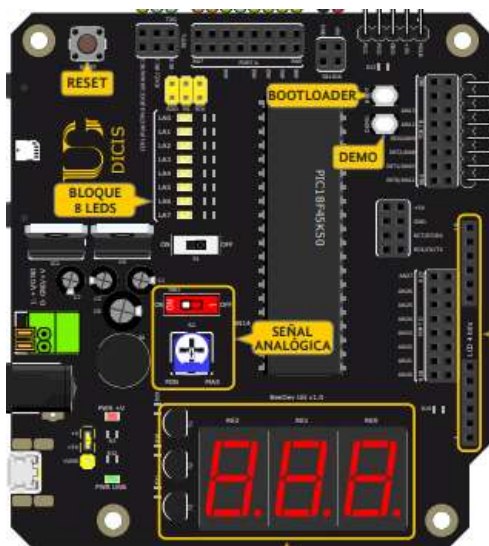


MICROCHIP

30-5-2019

Generador de Señales (PWM, SENOIDAL, TRIANGULAR)

Microprocesadores y Micro controladores



Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar

Dr. Gustavo Cerda Villafaña
PROYECTO FINAL



Contenido

INTRODUCCION	2
OBJETIVO	3
MICROCHIP	3
MICROCONTROLADOR 18F45K50 Y PLACA DE DESARROLLO	4
.....	7
PROCEDIMIENTO	7
MATERIALES	7
SOLDAR PLACA	8
SIMULACIONES	11
DESARROLLO DE CODIGO	15
RESULTADOS	27
CONCLUSIONES	27
BIBLIOGRAFIA	28



Universidad de Guanajuato

División de Ingenierías
Campus Irapuato – Salamanca

Docente: **Dr. Gustavo Cerda Villafaña** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

INTRODUCCION

Un microcontrolador es un circuito integrado digital que puede ser usado para muy diversos propósitos debido a que es *programable*. Está compuesto por una unidad central de proceso (CPU), memorias (ROM y RAM) y líneas de entrada y salida (periféricos).

Como el hardware ya viene integrado en un solo chip, para usar un microcontrolador se debe especificar su funcionamiento por software a través de programas que indiquen las instrucciones que el microcontrolador debe realizar.

En una memoria se guardan los programas y un elemento llamado CPU se encarga de procesar paso por paso las instrucciones del programa. Los lenguajes de programación típicos que se usan para este fin son *ensamblador* y *C*, pero antes de grabar un programa al microcontrolador hay que compilarlo a hexadecimal que es el formato con el que funciona el microcontrolador.

Microchip Technology, denominada comúnmente *Microchip* es una de las empresas líderes en la fabricación de microcontroladores. Para esta empresa, los microcontroladores se conocen con el apodo «PIC». Debido a sus bajos costos, desempeño eficiente, gran documentación y fácil adquisición de los kit de desarrollo, los microcontroladores de Microchip, conocidos simplemente como PIC, serán los que utilizaré a lo largo del tutorial. Los microcontroladores están diseñados para reducir el costo económico y el consumo de energía de un sistema en particular. Por eso el tamaño de la unidad central de procesamiento, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación.

Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar



Los microcontroladores representan la inmensa mayoría de los chips de computadoras vendidos, sobre un 50% son controladores "simples" y el restante corresponde a DSP más especializados. Mientras se pueden tener uno o dos microprocesadores de propósito general en casa (Ud. está usando uno para esto), usted tiene distribuidos seguramente entre los electrodomésticos de su hogar una o dos docenas de microcontroladores.

Un microcontrolador difiere de una unidad central de procesamiento normal, debido a que es más fácil convertirla en una computadora en funcionamiento, con un mínimo de circuitos integrados externos de apoyo. La idea es que el circuito integrado se coloque en el dispositivo, enganchado a la fuente de energía y de información que necesite, y eso es todo. Un microprocesador tradicional no le permitirá hacer esto, ya que espera que todas estas tareas sean manejadas por otros chips. Hay que agregarle los módulos de entrada y salida (puertos) y la memoria para almacenamiento de información.

OBJETIVO

Generador de funciones (senoidal, triangular) y PWM con ayudada de una pantalla LCD y Push Buttons. El microcontrolador tendrá un menú donde se seleccionarán cuatro opciones (apagado, señal senoidal, señal triangular, PWM).

MICROCHIP

El proyecto final se desarrolló con la ayuda de componentes que la empresa Microchip desarrolla, en especial con el PIC 18F45K50, para continuar con el proyecto se pusieron en práctica los conocimientos aprendidos a lo largo del curso, microchip como empresa nos proporcionó este material, para poder hacer que se generaran las funciones pedidas en el proyecto.

(MICROCHIP, s.f.)



Inicialmente la empresa GI (*General Instruments*) creó una subdivisión para fabricar dispositivos microelectrónicos. Más tarde esta subdivisión fue vendida a Venture Capital Investors que decidió crear una nueva empresa llamada Arizona Microchip Technology.

MICROCONTROLADOR 18F45K50 Y PLACA DE DESARROLLO



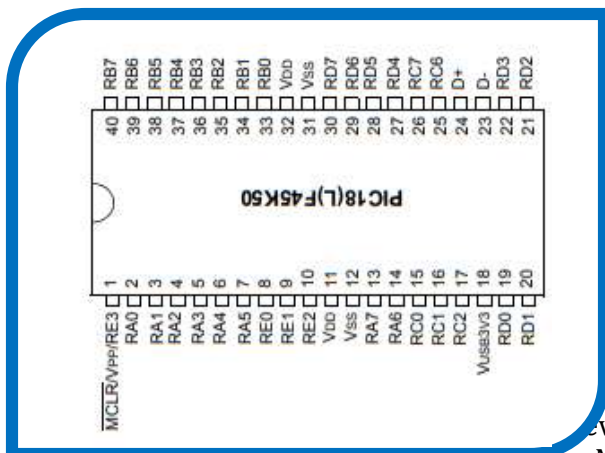
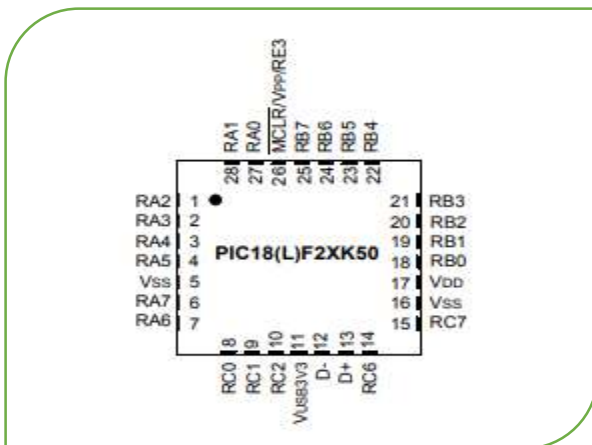
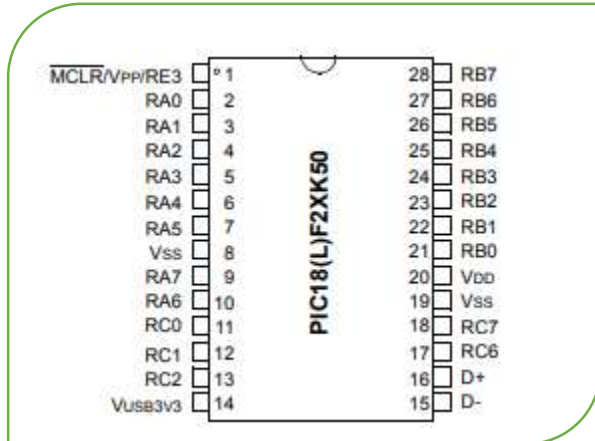
CARACTERISTICAS:

- Núcleo PIC18 de alto rendimiento con 8x8 de hardware multiplicado
- Memoria de programa flash con capacidad de auto lectura / escritura
- 256 bytes de EEPROM integrada
- Oscilador interno de 48MHz con precisión USB - Ajuste activo de reloj desde host USB
- Módulo bus serie universal 2.0
- Módulo PWM (ECCP) de captura mejorada con hasta 4 salidas
- Hasta 25 canales ADC de 10 bits con referencia de voltaje
- 2 comparadores analógicos
- Convertidor digital a analógico de 5 bits (DAC)
- Módulo MI2C / SPI (MSSP)
- Módulo USART direccionable mejorado
- Unidad de medición del tiempo de carga (CTMU) para aplicaciones de medición
- 25mA Fuente / E / S de corriente de fregadero
- Temporizadores 2x de 8 bits
- Temporizadores 2x de 16 bits

Este microcontrolador cuenta con cuarenta pines de los cuales tiene salidas y entradas digitales, para poder utilizarlas, su estructura se mostrara en la siguiente imagen con más detalle del cómo se compone y de los pines que utiliza para hacer la conexión de datos con los componentes que estén en interacción con el como una pantalla LCD o unos Push Buttons. Es importante hablar de este PIC ya que con este se desarrolló el proyecto, siendo así un proyecto funcional y de cierta manera sencilla en su forma de programar, como ya se verá más adelante tiene dos opciones de programación como la mayoría de los microcontroladores, en ensamblador y en c.



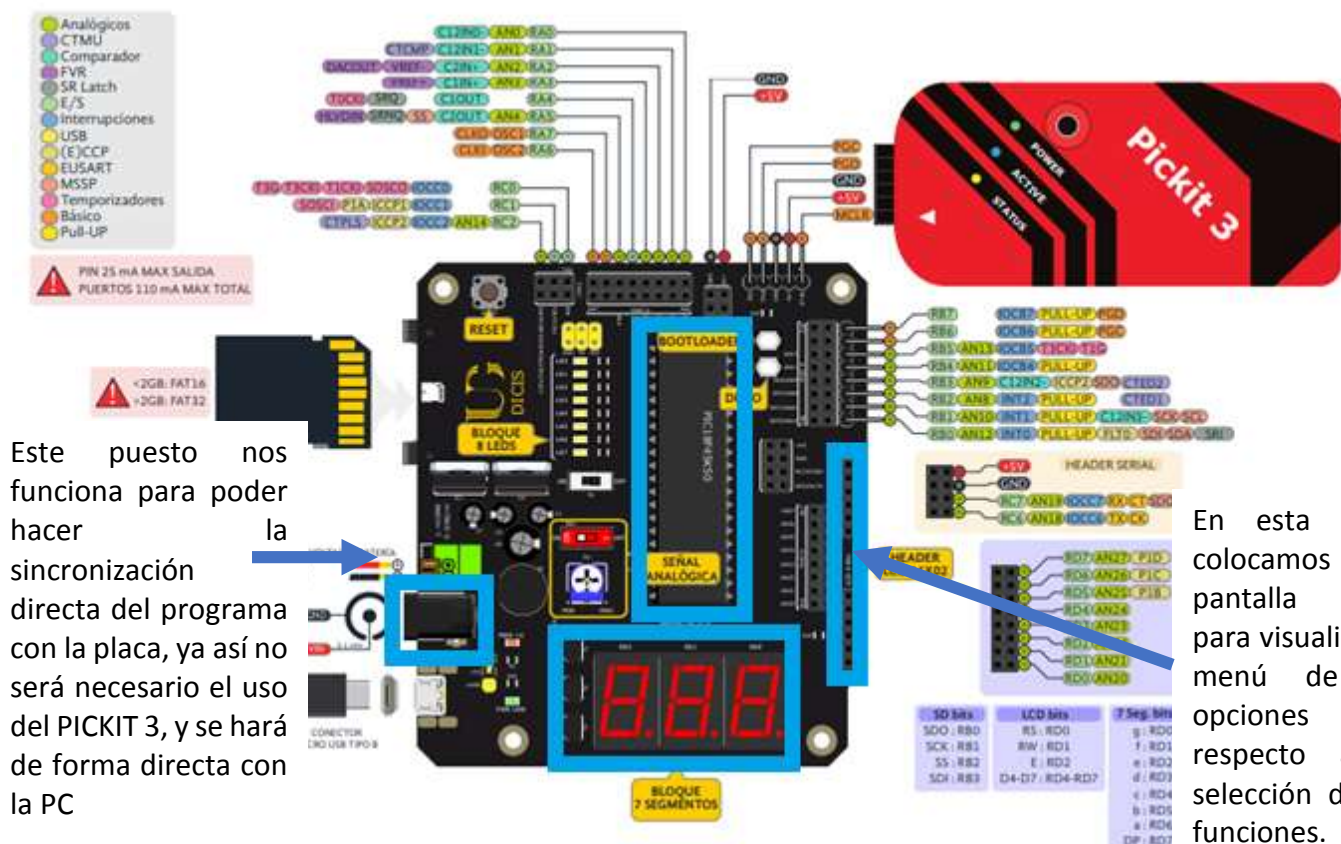
Este microcontrolador cuenta con tres estructuras que nos permiten trabajar para este proyecto se utilizó solo una de estas por beneficios prácticos del curso y como fue el que se estudió en el curso, se tomó la decisión de utilizarlo, estos son los modelos que se manejan de esta familia:





Esta es la que se utilizó para el desarrollo del proyecto, a la par que la placa que se nos proporcionó por un proveedor encargado de fabricar este tipo de placas, en específico se consiguió la estructura especificada de acuerdo al diseño de proyecto, a continuación se muestra el diseño de la placa que nos proporcionaron, así como la conexiones que se deberían de realizar, en este sentido se comenzaron a analizar en donde irían conectados los componentes que se mostrarían según la lista que se nos proporcionó.

En este campo colocamos el PIC de forma como se indica con los pines de forma correcta.



jueves, 30 de mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar



Docente: **Dr. Gustavo Cerda Villafaña** //gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

PROCEDIMIENTO MATERIALES



PARTIDA	DESCRIPCION	CANTIDAD	UNIDAD	P. UNITARIO	IMPORTE
1	Microcontrolador PIC18F45K50-I/P 45K50 Microchip	1	PZ	\$ 150.00	\$ 150.00
2	Base Zócalo Para CI De 40 Pines	1	PZ	\$ 10.00	\$ 10.00
3	Trimpot De 10k OHM, 10000 OHM	1	PZ	\$ 3.00	\$ 3.00
4	Dip Switch 1 Posición Bit Vía Canal Deslizable	1	PZ	\$ 3.00	\$ 3.00
5	Conector Alimentación Jack Hembra 3 Pines 5.5X2.1mm	1	PZ	\$ 5.00	\$ 5.00
6	Bornera Clema Para Circuito Impreso Dos Terminales De Presión	1	PZ	\$ 5.00	\$ 5.00
7	Puente Rectificador De Diodo 2w10 2A 1000V	1	PZ	\$ 10.00	\$ 10.00
8	Regulador De Voltaje Salida De 800MA 3.3V LM1117T LM1117 1117	1	PZ	\$ 25.00	\$ 25.00
9	Mini Interruptor Simple De Dos Posiciones Pines Rectos	1	PZ	\$ 2.00	\$ 2.00
10	Push button Switch SMD 2 pines	2	PZ	\$ 1.50	\$ 3.00
11	Botón Pulsador Push Button De Presión Toque 4 Pines NA	1	PZ	\$ 1.00	\$ 1.00
12	Adaptador Conector/Zocalo Para SD	1	PZ	\$ 16.00	\$ 16.00
13	Display Siete Segmentos de Cátodo Común LED Rojo	3	PZ	\$ 8.00	\$ 24.00
14	Transistor 2N2222A	3	PZ	\$ 4.00	\$ 12.00
15	Resistencias SMD 0805 330 OHM	9	PZ	\$ 1.50	\$ 13.50
16	Resistencias SMD 0805 470 OHM	10	PZ	\$ 1.50	\$ 15.00
17	Resistencias SMD 0805 1k OHM	6	PZ	\$ 1.50	\$ 9.00
18	Resistencias SMD 0805 2k OHM	3	PZ	\$ 1.50	\$ 4.50
19	Resistencias SMD 0805 3.3k OHM	3	PZ	\$ 1.50	\$ 4.50
20	Resistencias SMD 0805 4.7k OHM	1	PZ	\$ 1.50	\$ 1.50
21	Resistencias SMD 0805 10k OHM	3	PZ	\$ 1.50	\$ 4.50
22	Capacitores Cerámicos SMD 100nF 50V	6	PZ	\$ 1.50	\$ 9.00
23	Capacitores Cerámicos SMD 470nF 50V	1	PZ	\$ 1.50	\$ 1.50
24	LED SMD 0603 AMARILLO	8	PZ	\$ 1.50	\$ 12.00
25	LED SMD 0603 VERDE	1	PZ	\$ 1.50	\$ 1.50
26	LED SMD 0603 ROJO	1	PZ	\$ 1.50	\$ 1.50
27	Capacitores Electrolíticos 10UF A 16V	2	PZ	\$ 2.50	\$ 5.00
28	Capacitores Electrolíticos 22UF A 16V	1	PZ	\$ 2.50	\$ 2.50
29	Capacitores Electrolíticos 100uF A 16V	1	PZ	\$ 2.50	\$ 2.50
30	Tira De 40 Pines Header Macho	1	PZ	\$ 5.00	\$ 5.00
31	Tira De 40 Pines Header Hembra	1	PZ	\$ 5.00	\$ 5.00
32	Tira De 40 Pines Header Macho Ángulo Recto 90°	1	PZ	\$ 5.00	\$ 5.00
33	Mini Jumper Puente Corto Circuito Para Tira De Pines 2.54mm	4	PZ	\$ 1.00	\$ 4.00
SUB TOTAL					\$ 376.00
IVA					INCLUIDO

Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar

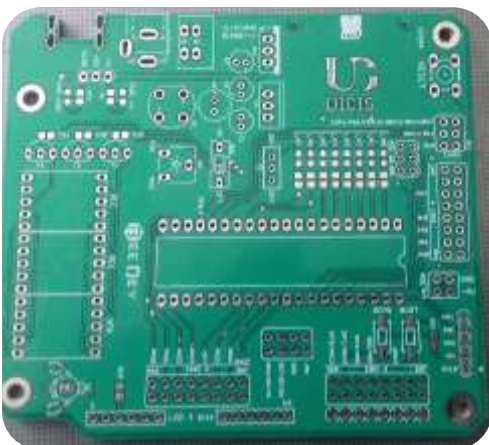


Docente: **Dr. Gustavo Cerda Villafaña** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

SOLDAR PLACA

Para poder continuar con el proyecto en cuanto recibimos el material para utilizar nos pusimos en práctica con la soldadura de los componentes, lo más complicado fue el poder soldar elementos demasiado pequeños como por ejemplo los capacitores y las resistencias y los leds, ya que se tenía que usar un kid especial para poder tomar cada elemento, de esta forma se podría safar o quedar bien, aun así se batallo pero se logró llegar al objetivo inmediato que era el soldar la placa.

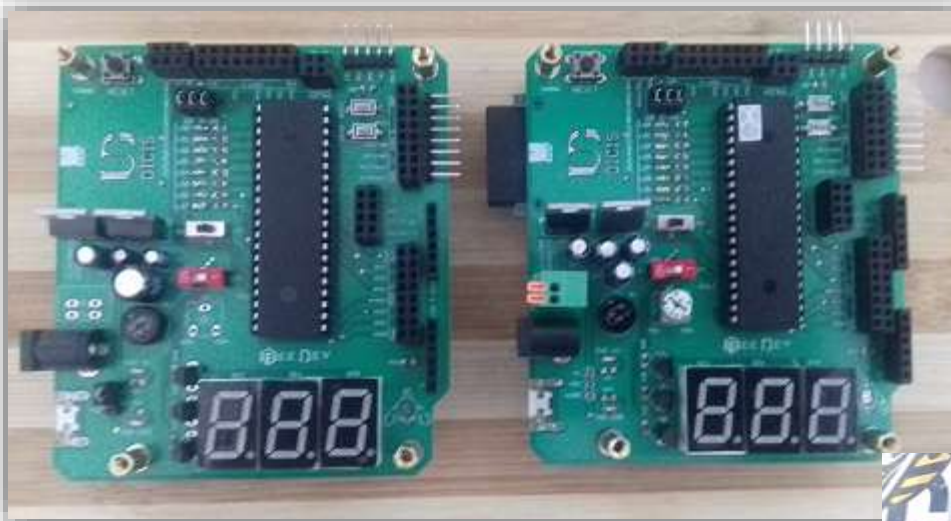
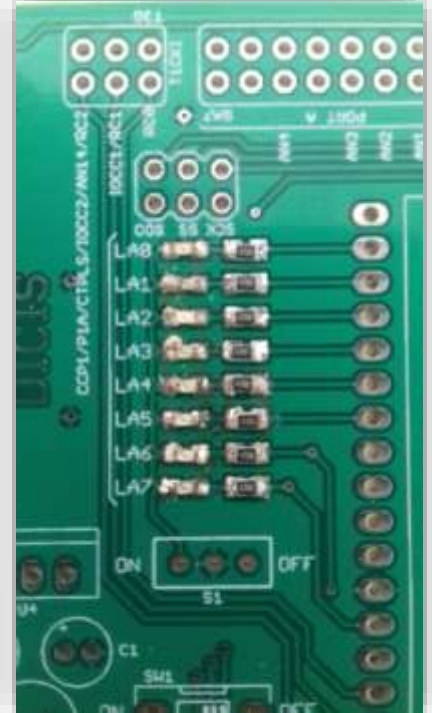
Anexo a este material que se mostró se utilizaron soldadura, así como estaño y cautín y un kid especial para tomar los componentes.



Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar



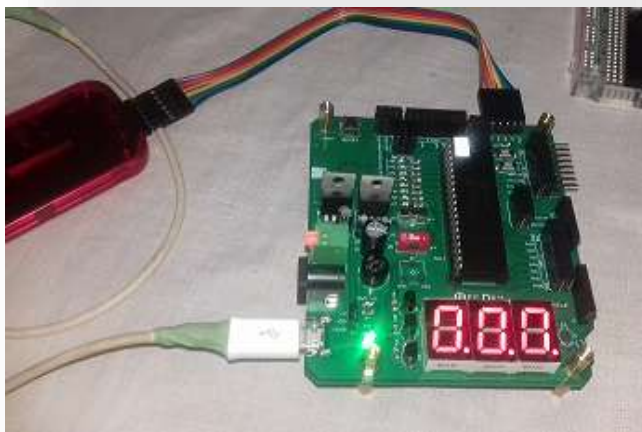
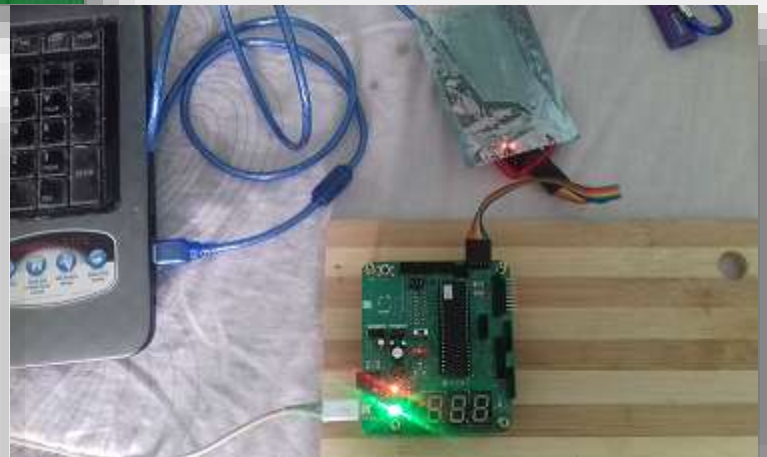
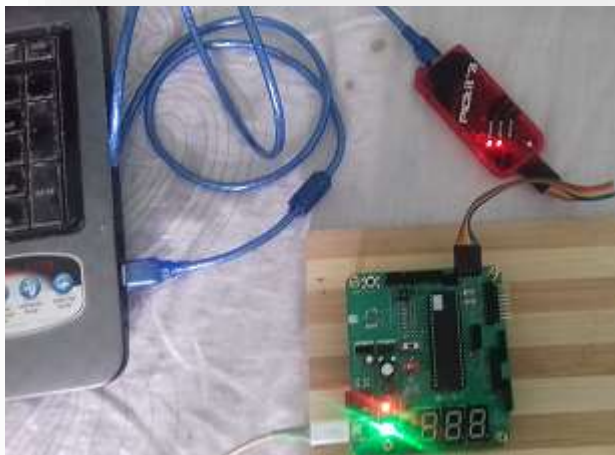
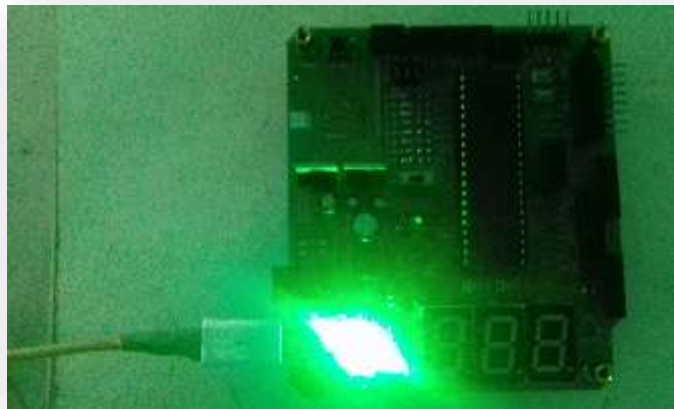
Docente: **Dr. Gustavo Cerda Villafaña** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**



Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar



Docente: **Dr. Gustavo Cerda Villafaña** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**



Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar

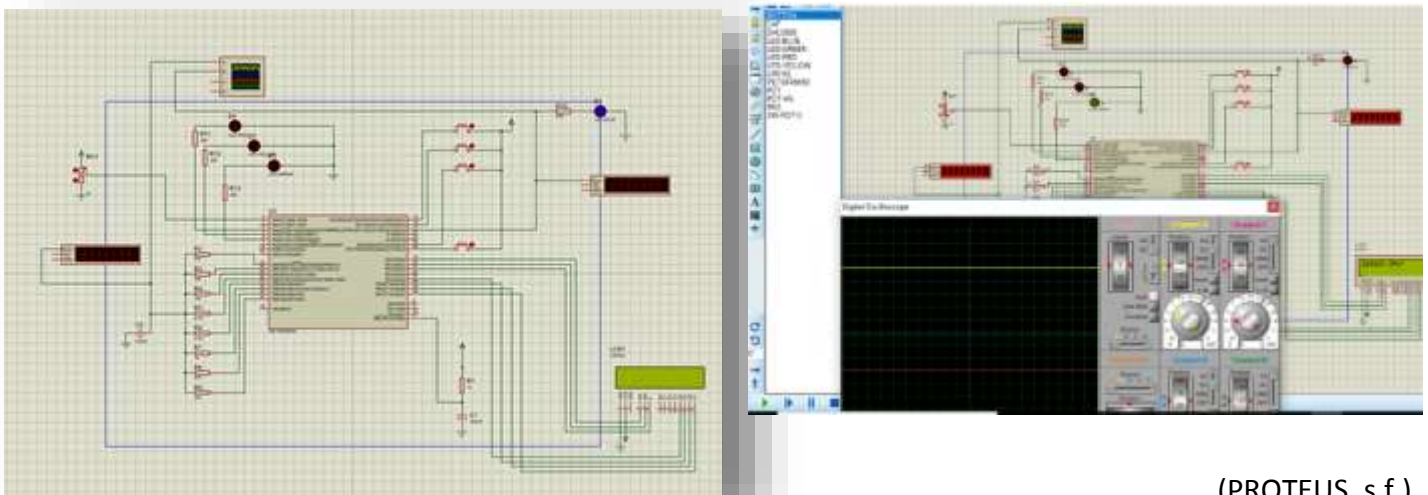


SIMULACIONES

Para continuar con el proyecto final, se comenzaron a hacer simulaciones en Proteus, que es un software que se especializa en este sentido, diseñar circuitos con componentes de cada forma y siendo así o dando como resultado gracias al código una simulación de cómo se vería en tiempo y forma, y tiempo real, así mismo mostramos el proceso de cómo se fueron creando hasta llegar a desarrollarlo de forma que fuera funcional gracias a la ayuda de Microchip.



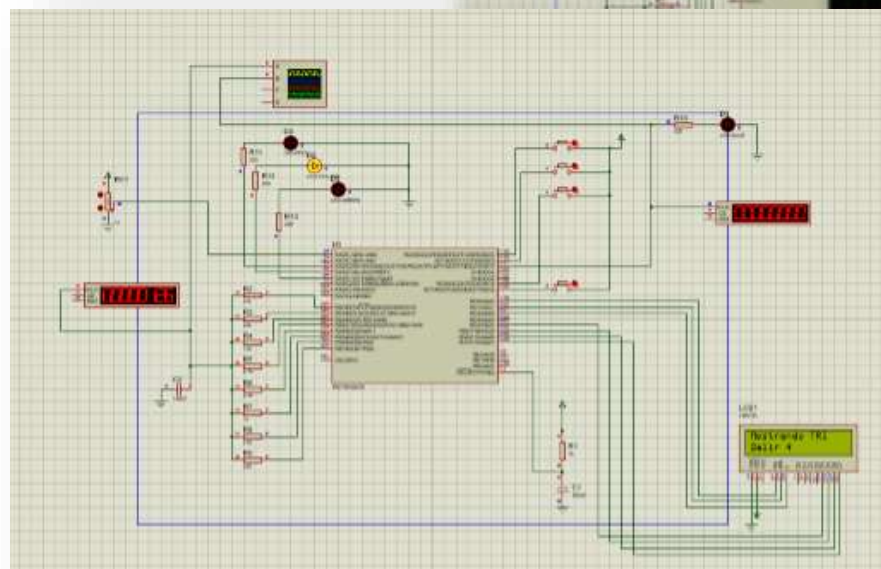
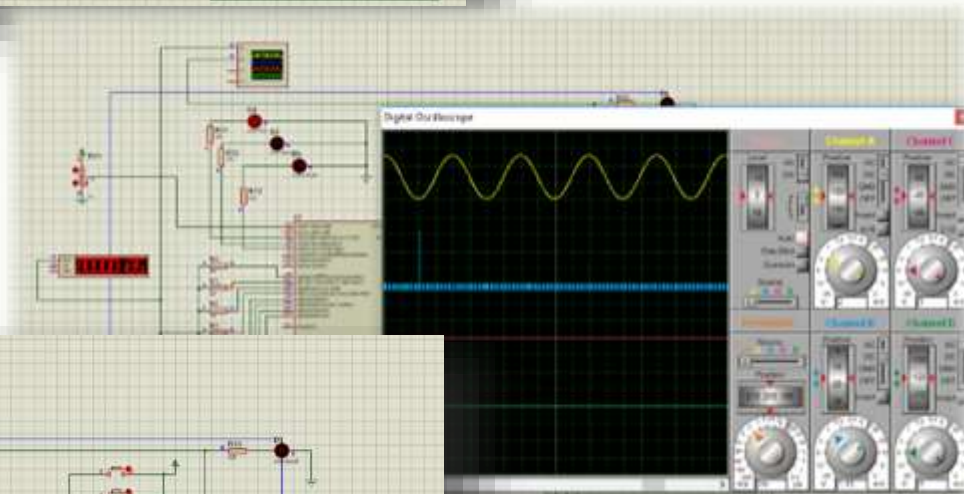
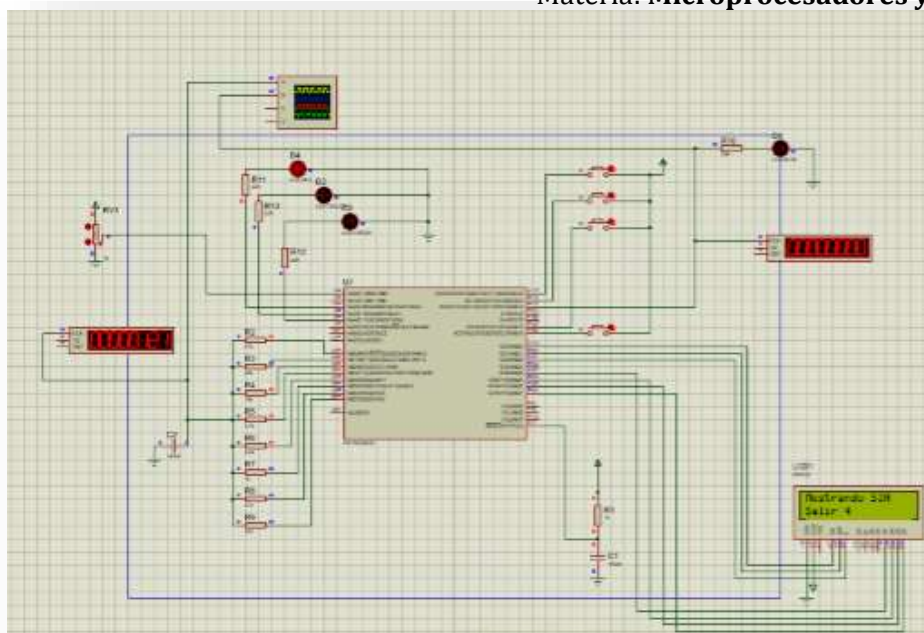
Gracias a la sincronía de estos dos programas se puede llegar a programar el PIC que se estaba utilizando, genera un archivo .h que se sincroniza al mismo tiempo y corre directamente en Proteus para lograr la simulación de lo que se estuviera programando.



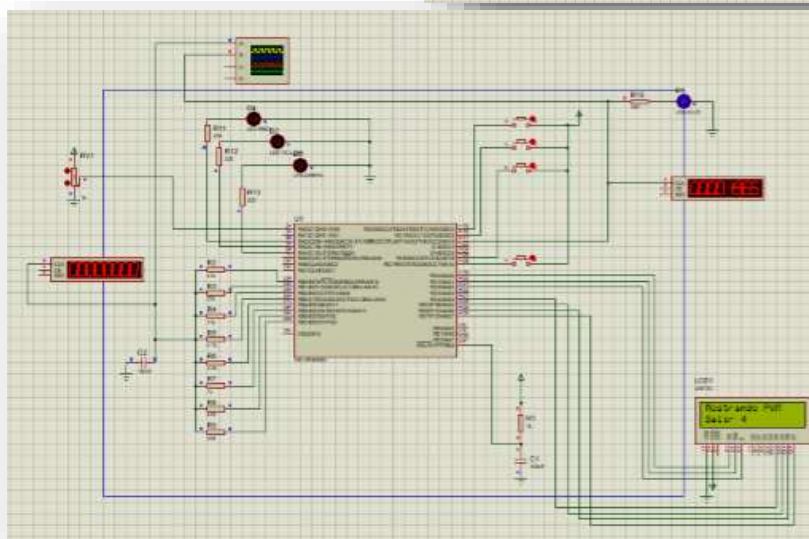
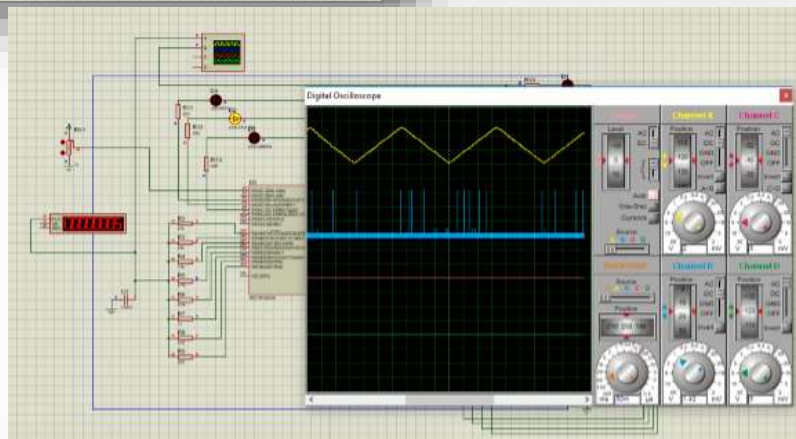
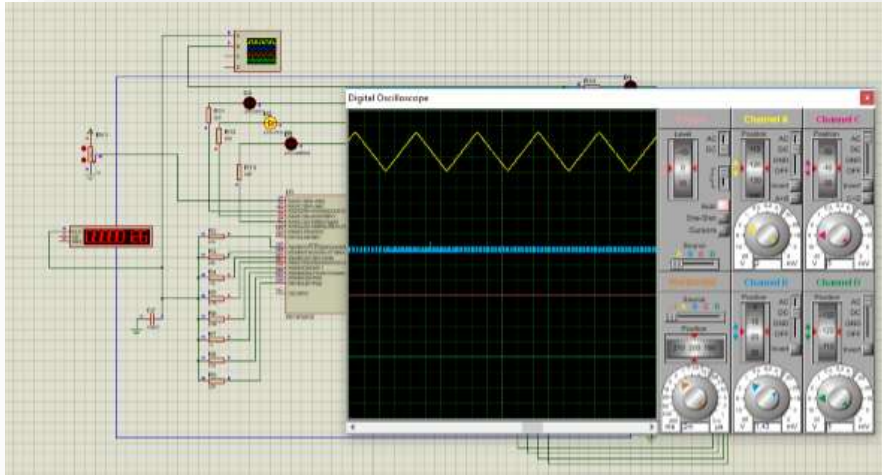
(PROTEUS, s.f.)

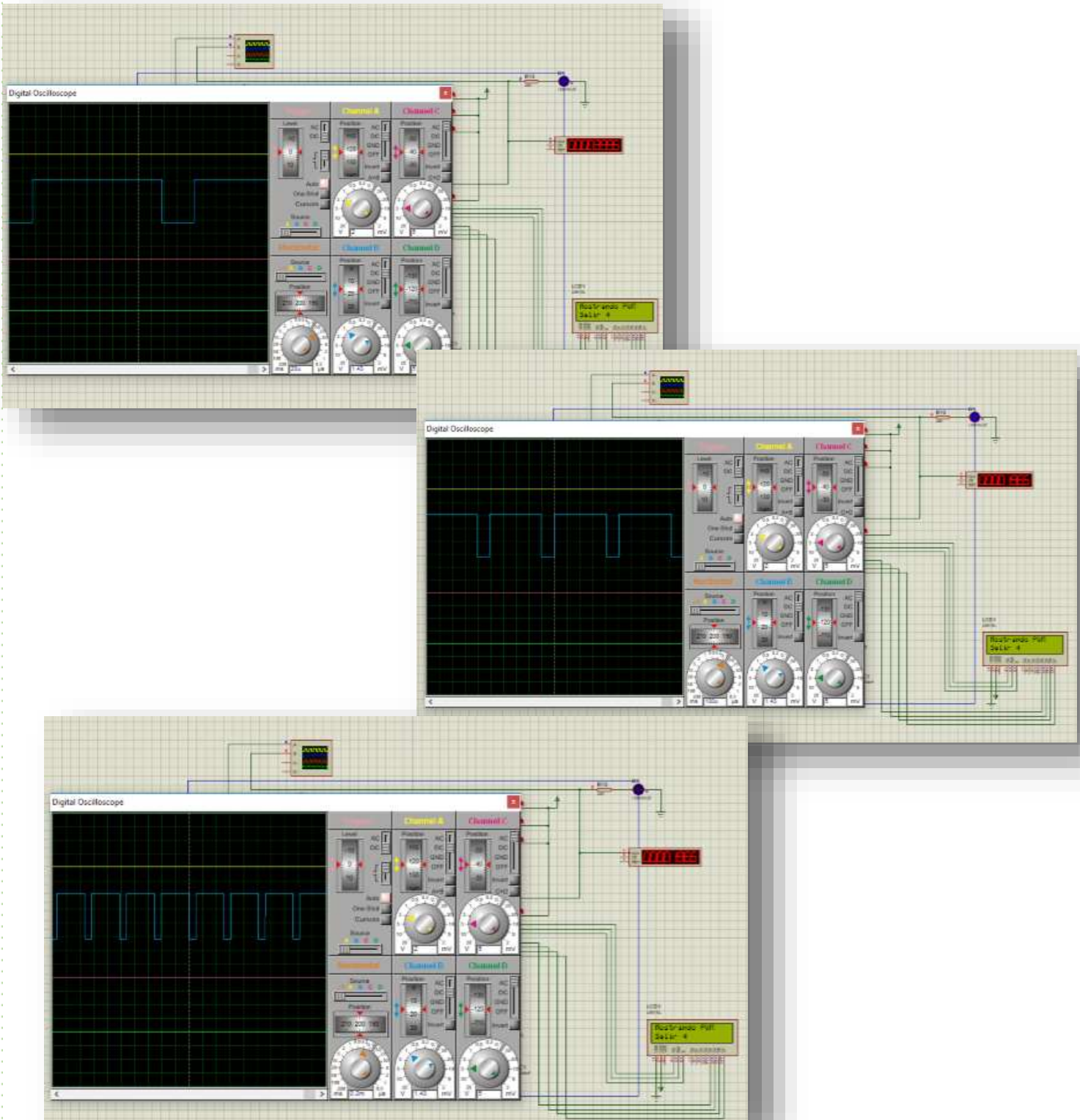


Docente: **Dr. Gustavo Cerda Villafañá** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**



Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar





jueves, 30 de mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar

**DESARROLLO DE CODIGO**

Para desarrollar el código se comenzó como ya se dijo, con la simulación del circuito, colocando cada componente y haciendo pruebas a cada momento, de esta forma nos dábamos cuenta de cómo funcionaba o en que fallaba y podríamos así corregir el error con su debido tiempo.

Para fines prácticos se programó en XC-8 que es un lenguaje de programación para microcontroladores en este caso para el PIC 15F45K50, al comenzar declaramos los bits de configuración, lo que se hace en ensamblador pero desde un menú de selección, de esta manera generaba los mismos bits pero de forma más visual, posterior a esto seguíamos con la declaración de la librería `"#include <xc.h>"` la cual accede a las características del microcontrolador, dentro de los bits de configuración están los de:

- ✓ Selección del oscilador
- ✓ Temporizador (watchdog)
- ✓ Protección del código
- ✓ Set de instrucciones extendido

Y continuaríamos con la declaración de `"#define __XTAL_FREQ 4000000"`

Para la utilización de registros y de variables así como de sentencias de funciones que se realizaron se tuvo que consultar el Data Sheet del PIC de esta manera se nos facilitaría la programación, dentro de los comandos que se utilizaron serían el `LATA=0x00;` que escribe un valor en el puerto A y así como `TRISA` que establece todos los bits del puerto A como salidas y al mismo tiempo `ANSELA` que define los pines del puerto como digitales, y el oscilador `OSCCON` que configura el oscilador a 4 MHz. Estos y más registros se utilizaron.



Una de las partes importantes es que se utilizó el código de la practica 2, que era el convertidor analógico digital, para poder sacar la señal PWM por el puerto 17, la explicación del código se hará de forma más contundente en cada parte de las imágenes que se muestran.

main_project.c

```
#include <xc.h>
#include "config_proyect.h"
#include "lcd_lib.h"
//Frecuencia de 4MHz
#define _XTAL_FREQ 4000000
//Función delay que acepta valores const.
void delay_us(unsigned int milliseconds)
{
    while(milliseconds > 0)
    {
        __delay_us(1);
        milliseconds--;
    }
}

int main(void) {

    // Arreglo sine para señal senoidal -> 128 points.
    const int sine[]={ 0x80,0x86,0x8c,0x92,0x98,0x9e,0xa5,0xaa,
                       0xb0,0xb6,0xbc,0xc1,0xc6,0xcb,0xd0,0xd5,
                       0xda,0xde,0xe2,0xe6,0xea,0xed,0xf0,0xf3,
                       0xf5,0xf8,0xfa,0xfb,0xfd,0xfe,0xfe,0xff,
                       0xff,0xff,0xfe,0xfe,0xfd,0xfb,0xfa,0xf8,
                       0xf5,0xf3,0xf0,0xed,0xea,0xe6,0xe2,0xde,
                       0xda,0xd5,0xd0,0xcb,0xc6,0xc1,0xbc,0xb6,
                       0xb0,0xaa,0xa5,0x9e,0x98,0x92,0x8c,0x86,
                       0x80,0x79,0x73,0x6d,0x67,0x61,0x5a,0x55,
                       0x4f,0x49,0x43,0x3e,0x39,0x34,0x2f,0x2a,
                       0x25,0x21,0x1d,0x19,0x15,0x12,0xf,0xc,
                       0xa,0x7,0x5,0x4,0x2,0x1,0x1,0x0,
                       0x0,0x0,0x1,0x1,0x2,0x4,0x5,0x7,
                       0xa,0xc,0xf,0x12,0x15,0x19,0x1d,0x21,
                       0x25,0x2a,0x2f,0x34,0x39,0x3e,0x43,0x49,
                       0x4f,0x55,0x5a,0x61,0x67,0x6d,0x73,0x79};
```

Sucesión de puntos para generar la señal senoidal a 128 puntos.



```
// Arreglo triangular para señal triangular -> 128 points.
const int triangular[]={0x4,0x8,0xc,0x10,0x14,0x18,0x1c,0x20,
                        0x24,0x28,0x2c,0x30,0x34,0x38,0x3c,0x40,
                        0x44,0x48,0x4c,0x50,0x54,0x58,0x5c,0x60,
                        0x64,0x68,0x6c,0x70,0x74,0x78,0x7c,0x80,
                        0x83,0x87,0x8b,0x8f,0x93,0x97,0x9b,0x9f,
                        0xa3,0xa7,0xab,0xaf,0xb3,0xb7,0xbb,0xbf,
                        0xc3,0xc7,0xcb,0xcf,0xd3,0xd7,0xdb,0xdf,
                        0xe3,0xe7,0xeb,0xef,0xf3,0xf7,0xfb,0xff,
                        0xfb,0xf7,0xf3,0xef,0xeb,0xe7,0xe3,0xdf,
                        0xdb,0xd7,0xd3,0xcf,0xcb,0xc7,0xc3,0xbf,
                        0xbb,0xb7,0xb3,0xaf,0xab,0xa7,0xa3,0x9f,
                        0x9b,0x97,0x93,0x8f,0x8b,0x87,0x83,0x80,
                        0x7c,0x78,0x74,0x70,0x6c,0x68,0x64,0x60,
                        0x5c,0x58,0x54,0x50,0x4c,0x48,0x44,0x40,
                        0x3c,0x38,0x34,0x30,0x2c,0x28,0x24,0x20,
                        0x1c,0x18,0x14,0x10,0xc,0x8,0x4,0x0};
```

Sucesión de
puntos para
generar la
señal
triangular a
128 puntos.

```
/* ---- Configuración de frecuencia del oscilador. ---- */
OSCCON = 0x53; //Oscilador interno 4 MHz
```

```
/* ---- Configuración de puertos. ---- */
```

```
//Puerto B
ANSELB = 0x00; //Declaramos el puerto B como digital.
TRISB = 0x00; //Declaramos el puerto B como salida.
```

```
//Puerto C
ANSELC = 0x00; //Declaramos el puerto C como digital.
```




Docente: **Dr. Gustavo Cerda Villafaña** //gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

//Salidas para Los push botton

TRISCbits.RC0 = 1; //RC0 como entrada -> Push botton -> Señal Senoidal.

TRISCbits.RC1 = 1; //RC1 como entrada -> Push botton -> Señal Triangular.

TRISCbits.RC6 = 1; //RC6 como entrada -> Push botton -> Señal PWM.

TRISCbits.RC7 = 1; //RC7 como entrada -> Push botton -> Salir de Las opciones.

// Registros de configuración del Convertidor analogico digital.

ADCON0 = 0x01; //Canal analógico 14, Habilita ADC

ADCON1 = 0x00; //Referencia: GND, V5+

ADCON2 = 0x24; //FOSC/4, 4TD, justificado a la izquierda // 0010|0100 -> 36

//Puerto A

PORTA = 0x00; //Puerto A como lectura o escritura.

ANSELA = 0x00; //Declaramos el puerto A como digital.

TRISAbits.RA1 = 1; //RA1 como entrada -> Para el potenciómetro.

TRISAbits.RA2 = 0; //RA2 como salida -> Led rojo para señal seno.

TRISAbits.RA3 = 0; //RA3 como salida -> Led amarillo para señal triangular.

TRISAbits.RA4 = 0; //RA4 como salida -> Led verde para señal seleccionar una opción.

//Puerto D

LATD = 0x00; //Declaramos el puerto D como escritura.

ANSELD = 0x00; //Declaramos el puerto D como digital.

TRISD = 0x00; //Declaramos el puerto D como salida.



```
/*----Menú ----*/
```

```
Lcd_Init(); //Función para inicializar LCD
```

```
Lcd_Clear(); //Función para limpiar LCD
```

```
Lcd_Set_Cursor(1,1); //Coloca el cursor en la posición 1,1
```

```
Lcd_Write_String("Generador De"); //Escritura de cadena
```

```
Lcd_Set_Cursor(2,1);
```

```
Lcd_Write_String("Funciones");
```

```
//__delay_ms(500);
```

```
__delay_ms(1500); //Retardo para el mostrar el mensaje.
```

```
Lcd_Clear();
```

```
Lcd_Set_Cursor(1,1);
```

```
Lcd_Write_String("Opcion: 1");
```

```
Lcd_Set_Cursor(2,1);
```

```
Lcd_Write_String("Seinal SIN");
```

```
//__delay_ms(500);
```

```
__delay_ms(1500);
```

```
Lcd_Clear();
```

```
Lcd_Set_Cursor(1,1);
```

```
Lcd_Write_String("Opcion: 2");
```

```
Lcd_Set_Cursor(2,1);
```

```
Lcd_Write_String("Seinal TRI");
```

```
//__delay_ms(500);
```

```
__delay_ms(1500);
```

```
Lcd_Clear();
```

```
Lcd_Set_Cursor(1,1);
```

```
Lcd_Write_String("Opcion: 3");
```

```
Lcd_Set_Cursor(2,1);
```

```
Lcd_Write_String("Seinal PWM");
```

```
//__delay_ms(500);
```

```
__delay_ms(1500);
```

Inicialización
del menú, en
la pantalla
LCD
mostrando
las 4
opciones.

Docente: **Dr. Gustavo Cerda Villafaña** //gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

```
Lcd_Clear();  
Lcd_Set_Cursor(1,1);  
Lcd_Write_String("Select Opc: ");  
__delay_ms(3000);  
//__delay_ms(500);
```

```
/* ---- Configuración del PWM ----*/
```

```
PR2 = 124; //Asignación de valor al PR2  
CCPR1L = 0; // Porcentaje del ciclo de trabajo.
```

```
ANSEL = 0; // Declaramos el puerto C como digital.  
TRISCbits.RC2 = 0; //Declaramos el pin 17 como salida.
```

```
T2CON = 0x00; // Asignacion del valor Prescaler 1 al Timer 2 OFF  
CCP1CON = 0x0C; // Controlador de los dos bits del registro CCPRXL
```

```
TMR2 = 0; // Declaracion de Timer 2  
T2CONbits.TMR2ON = 1; //Timer 2 ON
```

```
//Variable time para aguardar el resultado de la conversión  
unsigned int time;
```

```
int aux = 0;
```



```
while(1){

    //Iniciación de leds
    PORTAbits.RA4 =1; //Led verde encendido.
    PORTAbits.RA3 =0; //Led rojo apagado.
    PORTAbits.RA2 =0; //Led amarillo apagado.

    ADCON0bits.GO=1; //Inicio conversión
    while(ADCON0bits.GO==1);
    time = ADRESH;

    //Caso para la señal senoidal
    if ( PORTCbits.RC0 == 1 && PORTCbits.RC1 == 0 && PORTCbits.RC6 == 0 && PORTCbits.RC7 == 0 ) {

        /* ---- Menú ----*/
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Mostrando SIN ");
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String("Salir 4 ");

        //Iniciación de leds
        PORTAbits.RA4 =0; //Led verde apagado
        PORTAbits.RA2 =1; //Led rojo encendido
        PORTAbits.RA3 =0; //Led amarillo apagado

        CCPRL1 = 0; // Señal PWM apagada.
        aux = 0;

        //Si RC7 es indentico a cero (no se presiona el boton de salir) se realiza la operación interna del
        while(PORTCbits.RC7 == 0){

            LATB = sine[aux];
            delay_us(time);
            //Operador ternario -> (si aux es identico a 127) es verdadero entonces es cero y si es falso al
            aux=(aux==127)? 0 : aux++;

            ADCON0bits.GO=1; //Inicio conversión
            while(ADCON0bits.GO==1);
            time = ADRESH; //Guarda el resultado de la conversión.

        }

    }

    //Caso para la señal triangular.
    else if ( PORTCbits.RC1 == 1 && PORTCbits.RC0 == 0 && PORTCbits.RC6 == 0 && PORTCbits.RC7 == 0 ) {

        /* ---- Menú ----*/
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Mostrando TRI ");
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String("Salir 4 ");

        //Iniciación de leds
        PORTAbits.RA4 =0; //Led verde apagado
        PORTAbits.RA2 =0; //Led rojo apagado
        PORTAbits.RA3 =1; //Led amarillo encendido

        CCPRL1 = 0; // Señal PWM apagada.

        aux = 0;
        while(PORTCbits.RC7 == 0){
```

Aquí comenzamos a entrar en el while, comparando si algún botón está presionado entra al caso y manda llamar la función de sin o triangular, para que se generen los puntos.



```

LATB = triangular[aux];
delay_us(time);
//Operador ternario -> (si aux es identico a 127) es verdadero entonces es cero y si es falso al
aux=(aux==127) ? 0 : aux++;

ADCON0bits.GO=1; //Inicio conversión
while(ADCON0bits.GO==1);
time = ADRESH; //Guarda el resultado de la conversión.

}

}

//Caso para la señal PWM.
else if ( PORTCbits.RC6 == 1 && PORTCbits.RC0 == 0 && PORTCbits.RC1 == 0 && PORTCbits.RC7 == 0 ) {

    /* ---- Menú ---- */
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("Mostrando PWM ");
    Lcd_Set_Cursor(2,1);
    Lcd_Write_String("Salir 4 ");

    //Puerto en cero por lo tanto señales apagadas(senoidal y triangular).
    LATB = 0;
    while (PORTCbits.RC7 == 0) {

        //Iniciación de leds
        PORTAbits.RA4 =0; //Led verde apagado

        ADCON0bits.GO=1;
        while(ADCON0bits.GO==1);
        time = ADRESH; //Guarda el resultado de la conversión.

        //Frecuencia de 2Khz
        if ( time <= 80 ) {
            T2CONbits.T2CKPS = 1;
            PR2 = 124;
            CCPR1L = 100;

        }

        //Frecuencia de 4Khz
        if ( time > 80 && time < 160 ) {
            T2CONbits.T2CKPS = 0;
            PR2 = 249;
            CCPR1L = 200;

        }

        //Frecuencia de 8Khz
        if ( time >= 160 ) {
            T2CONbits.T2CKPS = 0;
            PR2 = 124;
            CCPR1L = 100;

        }

    }

}

return 0;
}

```




Lcd_lib.h

PARTE 1

```
#ifndef LCD_LIB_H
#define LCD_LIB_H

#define LCD_RS PORTDbits.RD0
#define LCD_RW PORTDbits.RD1
#define LCD_EN PORTDbits.RD2
#define LCD_D4 PORTDbits.RD4
#define LCD_D5 PORTDbits.RD5
#define LCD_D6 PORTDbits.RD6
#define LCD_D7 PORTDbits.RD7

#define _XTAL_FREQ 4000000
```

```
void Lcd_Port(char a)
{
    if(a & 1)
        LCD_D4 = 1;
    else
        LCD_D4 = 0;

    if(a & 2)
        LCD_D5 = 1;
    else
        LCD_D5 = 0;

    if(a & 4)
        LCD_D6 = 1;
    else
        LCD_D6 = 0;
```

PARTE 2

```
    if(a & 8)
        LCD_D7 = 1;
    else
        LCD_D7 = 0;
}

void Lcd_Cmd(char a)
{
    LCD_RS = 0;           // => RS = 0
    LCD_RW = 0;           // => RW = 0
    Lcd_Port(a);
    LCD_EN = 1;           // => E = 1
    __delay_ms(4);
    LCD_EN = 0;           // => E = 0
}

void Lcd_Clear(void)
{
    Lcd_Cmd(0);
    Lcd_Cmd(1);
}

void Lcd_Set_Cursor(char a, char b)
{
    char temp,z,y;
    if(a == 1)
    {
        temp = 0x80 + b - 1;
        z = temp >> 4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
```



PARTE 3

```

else if(a == 2)
{
    temp = 0xC0 + b - 1;
    z = temp >> 4;
    y = temp & 0x0F;
    Lcd_Cmd(z);
    Lcd_Cmd(y);
}

void Lcd_Init()
{
    Lcd_Port(0x00);
    __delay_ms(20);
    Lcd_Cmd(0x03);
    __delay_ms(5);
    Lcd_Cmd(0x03);
    __delay_ms(11);
    Lcd_Cmd(0x03);
    //////////////////////////////////////
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x08);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x0C);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x06);
}

```

PARTE 4

```

void Lcd_Write_Char(char a)
{
    char temp,y;
    temp = a&0x0F;
    y = a&0xF0;
    LCD_RS = 1;           // => RS = 1
    Lcd_Port(y>>4);       // Transferencia de dato.
    LCD_EN = 1;
    __delay_us(40);
    LCD_EN = 0;
    Lcd_Port(temp);
    LCD_EN = 1;
    __delay_us(40);
    LCD_EN = 0;
}

void Lcd_Write_String(const char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
        Lcd_Write_Char(a[i]);
}

void Lcd_Shift_Right()
{
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x0C);
}

void Lcd_Shift_Left()
{
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x08);
}

#endif

```



Config_project.h

```
#ifndef CONFIG_MAIN_H
#define CONFIG_MAIN_H

#define _XTAL_FREQ 4000000

#include <xc.h>
#include "lcd_lib.h"

// PIC18F45K50 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1L
#pragma config PLLSEL = PLL4X // PLL Selection (4x clock multiplier)
#pragma config CFGPLEN = OFF // PLL Enable Configuration bit (PLL Disabled (firmware controlled))
#pragma config CPUDIV = NOCLKDIV // CPU System Clock Postscaler (CPU uses system clock (no divide))
#pragma config LS48MHZ = SYS24X4 // Low Speed USB mode with 48 MHz system clock (System clock at 24 MHz, USB clock

// CONFIG1H
#pragma config FOSC = INTOSCIO // Oscillator Selection (Internal oscillator)
#pragma config PCLKEN = ON // Primary Oscillator Shutdown (Primary oscillator shutdown firmware controlled)
#pragma config FCMEN = OFF // Fail-Safe Clock Monitor (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF // Internal/External Oscillator Switchover (Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config nPWRTEN = OFF // Power-up Timer Enable (Power up timer disabled)
#pragma config BOREN = SBORDIS // Brown-out Reset Enable (BOR enabled in hardware (SBOREN is ignored))
#pragma config BORV = 190 // Brown-out Reset Voltage (BOR set to 1.9V nominal)
#pragma config nLPBOR = OFF // Low-Power Brown-out Reset (Low-Power Brown-out Reset disabled)

// CONFIG2H
#pragma config WDTEN = OFF // Watchdog Timer Enable bits (WDT disabled in hardware (SWDTEN ignored))
#pragma config WDTPS = 32768 // Watchdog Timer Postscaler (1:32768)
```



Docente: **Dr. Gustavo Cerda Villafaña** //gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

```
// CONFIG3H
#pragma config CCP2MX = RC1 // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBAEN = ON // PORTB A/D Enable bit (PORTB<5:0> pins are configured as analog input channels)
#pragma config T3CMX = RC0 // Timer3 Clock Input MUX bit (T3CKI function is on RC0)
#pragma config SDO MX = RB3 // SDO Output MUX bit (SDO function is on RB3)
#pragma config MCLRE = ON // Master Clear Reset Pin Enable (MCLR pin enabled; RE3 input disabled)

// CONFIG4L
#pragma config STVREN = ON // Stack Full/Underflow Reset (Stack full/underflow will cause Reset)
#pragma config LVP = OFF // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if MCLRE is also 1)
#pragma config ICPR = OFF // Dedicated In-Circuit Debug/Programming Port Enable (ICPORT disabled)
#pragma config XINST = OFF // Extended Instruction Set Enable bit (Instruction set extension and Indexed Add

// CONFIG5L
#pragma config CP0 = OFF // Block 0 Code Protect (Block 0 is not code-protected)
#pragma config CP1 = OFF // Block 1 Code Protect (Block 1 is not code-protected)
#pragma config CP2 = OFF // Block 2 Code Protect (Block 2 is not code-protected)
#pragma config CP3 = OFF // Block 3 Code Protect (Block 3 is not code-protected)

// CONFIG5H
#pragma config CPB = OFF // Boot Block Code Protect (Boot block is not code-protected)
#pragma config CPD = OFF // Data EEPROM Code Protect (Data EEPROM is not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF // Block 0 Write Protect (Block 0 (0800-1FFFh) is not write-protected)
#pragma config WRT1 = OFF // Block 1 Write Protect (Block 1 (2000-3FFFh) is not write-protected)
#pragma config WRT2 = OFF // Block 2 Write Protect (Block 2 (04000-5FFFh) is not write-protected)
#pragma config WRT3 = OFF // Block 3 Write Protect (Block 3 (06000-7FFFh) is not write-protected)

// CONFIG6H
#pragma config WRTC = OFF // Configuration Registers Write Protect (Configuration registers (300000-3000FFh)
#pragma config WRTB = OFF // Boot Block Write Protect (Boot block (0000-7FFFh) is not write-protected)
#pragma config WRTD = OFF // Data EEPROM Write Protect (Data EEPROM is not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF // Block 0 Table Read Protect (Block 0 is not protected from table reads execute
#pragma config EBTR1 = OFF // Block 1 Table Read Protect (Block 1 is not protected from table reads execute
#pragma config EBTR2 = OFF // Block 2 Table Read Protect (Block 2 is not protected from table reads execute
#pragma config EBTR3 = OFF // Block 3 Table Read Protect (Block 3 is not protected from table reads execute

// CONFIG7H
#pragma config EBTRB = OFF // Boot Block Table Read Protect (Boot block is not protected from table reads e

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#endif /* CONFIG_MAIN_H */
```



RESULTADOS

Los resultados fueron muy satisfactorios ya que se muestran las señales de forma correcta y en forma según los Push Botton, es decir dependiendo de qué botón presione la señal se mostrara en el osciloscopio con la frecuencia que se mostrara, en el proyecto se colocó un potenciómetro para poder variar la frecuencia y que la señal cambiara su amplitud, dando una señal diferente visualmente, estos resultados se verán en las imágenes.

CONCLUSIONES

En este proyecto final se pudo llegar a sacar una señal de cada tipo, una senoidal y una PWM y una triangular, con el PIC 18F45K50, la programación, se logró con ayuda del software de Microchip que fue XC-8 y en sincronía con Proteus para ir viendo cómo se comportaba, logrando resultados de una manera muy buena y dando los resultados esperados con la ayuda de la variación de frecuencia, el proceso que se lleva a cabo en este proyecto es algo largo tanto que fueron necesarias muchas sesiones del curso para terminarlo, desde el soldar la placa hasta empezar con la programación y terminarlo sincronizándolo con Proteus.

El manejo de los pines del microcontrolador con los registros fue algo complejo pero gracias al Data Sheet se pudo hacer que funcionara, de esta forma se logró cargar el programa a la memoria de programa del PIC para que se ejecutara el programa cada vez que se utilizaría, concluimos que las señales que nos mostro fue la implementación entre muchas cosas del convertidor analógico digital ya que fue necesaria para sacar las señales e especial la señal PWM y que se mostrara con la sucesión de puntos.

El desarrollo del proyecto del generador de señales fue un reto ya que para lograr su implementación tuvimos que soldar varias piezas en la placa que compramos, me gustó mucho porque pude aprender a usar más componentes electrónicos y saber manejarlos,



Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar



Docente: **Dr. Gustavo Cerda Villafaña** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

por otro lado la programación también fue un reto porque teníamos que investigar acerca de los puertos y sus configuraciones, algunos ya venían definidos y solo era sacarlos del datasheet, otro punto importante fueron los push button porque la pantalla LCD al ser muy sensible a veces se brincaba o entraba a un estado sin ser seleccionado, la implementación de las tres funciones y como programarlas para que nos mostrara una señal a la vez sin mostrar las otras dos, incluso los delay para que los puntos pudieran dibujarse de la manera adecuada para ser mostrada en el osciloscopio, primero usamos for y después se nos hizo mejor implementar un while para que se mantuviera en ese ciclo hasta que nosotros cambiáramos los botones, con la ayuda de Proteus, pudimos avanzar para hacer las pruebas en nuestras casas y en la clase antes de probarlas en el laboratorio, la materia me permitió conocer a más detalle cómo funcionan los microprocesadores en la actualidad y como trabajar con ellos.

BIBLIOGRAFIA

-  http://ww1.microchip.com/downloads/en/DeviceDoc/PIC18F2X_45K50-30000684B.pdf
-  [file:///C:/Users/Juan/Downloads/BeeDev_UG_v1%20\(1\).pdf](file:///C:/Users/Juan/Downloads/BeeDev_UG_v1%20(1).pdf)
-  <https://es.wikipedia.org/wiki/Microcontrolador#Caracter%C3%ADsticas>

UNIVERSIDAD DE
GUANAJUATO



Universidad de Guanajuato

División de Ingenierías
Campus Irapuato – Salamanca

Docente: **Dr. Gustavo Cerda Villafaña** // gcerdav@ugto.mx
Materia: **Microprocesadores y Microcontroladores**

Jueves, 30 de Mayo de 2019
Juan Monserrat González García
Marco Antonio Flores Rivera
Mariana Arce Aguilar