CAPÍTULO 1

1. DESCRIPCIÓN GENERAL DEL PROYECTO

1.1 Antecedentes

Los Analizadores Lógicos se desarrollaron casi simultáneamente al mismo tiempo que los primeros microcontroladores que salieron al mercado. Los ingenieros que diseñaban sistemas basados en estos nuevos dispositivos pronto notaron que para la implementación de diseños con microcontroladores y chips digitales hacía falta una herramienta para visualizar más entradas de las que podían ofrecer los osciloscopios.

Los analizadores lógicos, con sus distintas entradas, eran la solución perfecta a este problema, dado que varias señales podían ser analizadas al mismo tiempo a diferencia del osciloscopio. Estos instrumentos han aumentado gradualmente tanto su velocidad de adquisición como el recuento de canales para mantenerse a la altura de los rápidos avances en la tecnología digital. El analizador lógico es una herramienta clave para el desarrollo de los sistemas digitales, aunque existen similitudes entre los osciloscopios y los analizadores lógicos.

1.2 Descripción del Proyecto

Nuestro proyecto consiste en un Analizador Lógico de 4 canales, para lo cual vamos a usar el PIC16F887, programado en MikroC PRO el cual será el encargado de tomar las distintas señales digitales, para luego ser visualizadas en una interfaz grafica que en este caso será una pantalla GLCD que facilitara el posterior análisis de las entradas digitales, que es el propósito de este proyecto.

FIGURA 1.1: PIC 16F887 y la Pantalla GLCD

1.3 Aplicaciones

Los analizadores son empleados principalmente para la detección de errores y comprobación de prototipos antes de su fabricación, comprobando las entradas y analizando posteriormente el comportamiento de sus salidas.

Un analizador lógico se inicia cuando en el circuito digital a analizar se da una determinada condición lógica, cuando los analizadores lógicos empezaron a utilizarse, era común conectar varios cientos de "clip" a un sistema digital.

Muchos diseños digitales, incluso circuitos integrados, se simulan para encontrar fallos antes de ser construidos. Aunque tales simulaciones no reproducen exactamente las señales como un analizador lógico las puede obtener de un prototipo real, cubren la mayoría de las necesidades reales en cuanto a la depuración de un programa, además los Analizadores Lógicos más avanzados cuentan con una conexión al PC donde se pueden ver los datos mediante un software

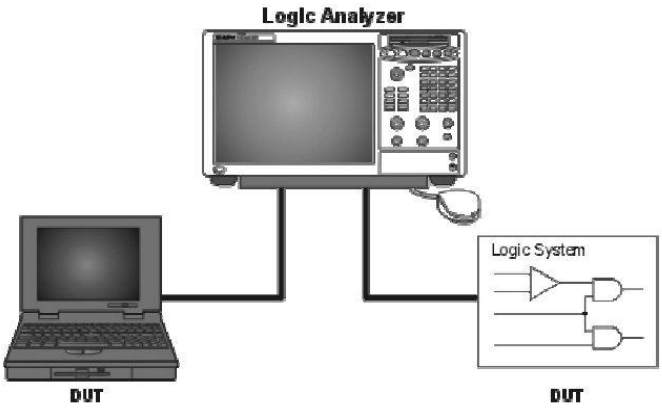

FIGURA1.2 : Analizador Lógico conectado a PC y Sistema Digital

1.3.1. Uso de Analizador Lógico en FPGA

El gran crecimiento en el tamaño y la complejidad del diseño, hace que el proceso de verificación sea crítico para los sistemas actuales basados en FPGAs, el acceso limitado a las señales internas, los paquetes avanzados de FPGA y el ruido eléctrico de la tarjeta del circuito impreso (PCB) contribuyen en su conjunto a que la eliminación de errores y la verificación sean los procesos más difíciles del diseño.

Se puede fácilmente pasar más del 50% de la duración del diseño depurando y verificándolo, por lo que para ayudar en el proceso se requieren nuevas herramientas que ayuden a eliminar los errores de diseño mientras la FPGA funciona a plena velocidad.

Una de las decisiones más importantes que es necesario tomar en la fase de diseño es la elección de la metodología a utilizar para depurar la FPGA. Hay dos metodologías básicas de depuración de FPGAs cuando están ya instaladas en el circuito: la primera consiste en la utilización de un analizador lógico embebido y la segunda en la utilización de un analizador lógico externo.

La selección de la metodología a utilizar depende de las necesidades de depuración del proyecto. Debido a las limitaciones de la metodología del analizador lógico embebido, muchos diseñadores de FPGAs han adoptado una metodología que utiliza la flexibilidad de la FPGA y la potencia de un analizador lógico externo.

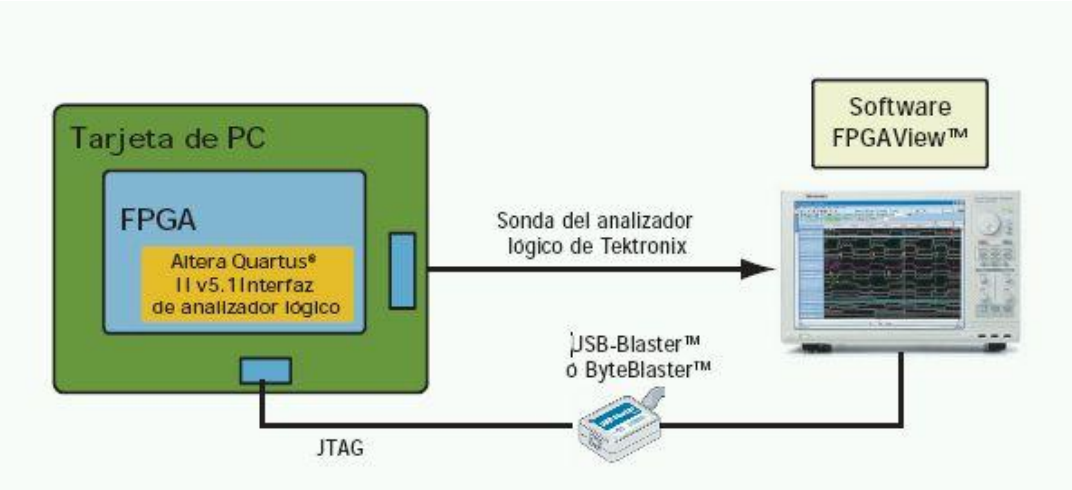

FIGURA 1.3: Esquema de Conexión Tarjeta FPGA.

1.3.2. Analizador Lógico para medir integridad de la Señal

La observación y la medida directa de la señal es la única manera de descubrir las causas de los problemas relacionados con la integridad de la señal. La elección de la herramienta correcta simplificará el trabajo. En la mayoría de los casos, las medidas de la integridad de señal se realizan por los mismos instrumentos familiares encontrados en casi cualquier laboratorio electrónico de ingeniería.

Entre estos instrumentos se incluyen el analizador lógico y el osciloscopio. Las sondas y el software completan el conjunto básico de herramientas, además, las fuentes de la señal se pueden utilizar para proporcionar señales distorsionadas para las pruebas de tolerancia y la evaluación de nuevos dispositivos y sistemas.

Cuando se debe ajustar un sistema de medida de la integridad de la señal las consideraciones principales se centran alrededor de:

– La utilización de sondas

– El ancho de banda y la respuesta al escalón

– La resolución temporal

– La longitud del registro

– El disparo

– La integración

Cuando se buscan problemas de integridad en señales digitales, especialmente en sistemas complejos con numerosos buses, entradas y salidas, el analizador lógico es la primera línea de defensa, este instrumento dispone de un elevado número de canales, una memoria profunda y un disparo avanzado para adquirir la información digital de muchos puntos de prueba y después visualizarla de una forma coherente, puesto que es un instrumento verdaderamente digital, el analizador lógico detecta los cruces del umbral de tensión por las señales que está supervisando y después muestra las señales lógicas según son vistas por los chips lógicos.

El análisis de la integridad de la señal requiere la utilización de alguno de los analizadores lógicos de más altas prestaciones disponible.

CAPÍTULO 2

2. FUNDAMENTO TEÓRICO

2.1 Requerimientos para aplicación del Proyecto

El desarrollo del proyecto se lo puede dividir en Software y Hardware. El software para el programa del PIC 16F887 es MIKRO C PRO FOR PIC de MIKROELECTRONICA y a su vez el software para cargar este programa en el PIC es el programa PICKIT 3, que permite cargar el archivo en hexadecimal que fue generado por MIKRO C al PIC desde un puerto USB de la PC.

Los componentes para desarrollar el proyecto, es decir el hardware usado son principalmente el PIC 16f887 que tiene 40 pines y además la pantalla táctil GLCD que nos permitirá visualizar las señales digitales.

2.2 Herramientas de Software

2.2.1 MikroC PRO

MikroC PRO para PIC es una herramienta muy útil y con todas las características para microcontroladores PIC de Microchip. Está diseñado para desarrollar, construir y depurar aplicaciones embebidas basadas PIC. Este entorno de desarrollo cuenta con una amplia variedad de características tales como IDE fácil de usar, muy compacta y eficiente código, hardware y software de bibliotecas, documentación completa, simulador de software, soporte de hardware depurador.

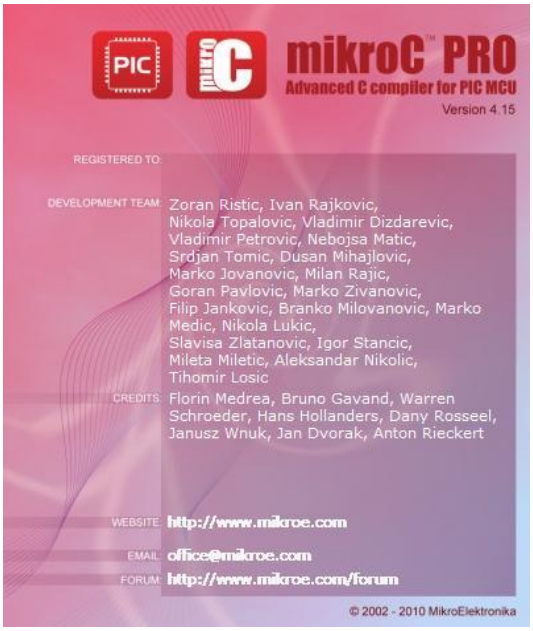

FIGURA 2.1: Identificación Software MikroC PRO 4.15v.

2.2.2 PROTEUS

Este software ofrece una amplia gama de prestaciones al realizar diferentes tipos de proyectos electrónicos, en todas sus etapas: diseño, simulación, depuración y construcción, ya sea analógica, digital o combinada, usando diferentes componentes electrónicos, como también puertos existentes en el mercado, herramientas de medición como multímetros, osciloscopios, analizadores lógicos y demás.

Esta herramienta nos permite simular y depurar el funcionamiento de todo el sistema paso a paso, de tal manera que podremos observar el contenido de registros y diferentes posiciones de memoria (en circuitos digitales), que nos facilitara determinar si la respuesta del hardware es la correcta o en su defecto existen errores.

Proteus posee una herramienta llamada ARES que la utilizamos para el diseño de circuitos impresos (PCB) que se pude generar automáticamente a partir de nuestro diseño.

FIGURA 2.2: Identificación Software Proteus.

2.3 Herramientas de hardware

2.3.1 PIC 16F887

Entre las ventajas principales nos ofrece:

1. Oscilador interno seleccionable de 8 a 31MHz.
2. Hasta 36 pines de entrada/salida usando el oscilador interno.
3. 14 entradas A/D contra 8 del PIC16F877A.
4. Dos comparadores.
5. Más económico.

Características:

- DIP 40 pines.
- Memoria FLASH para programa: 14KB
- Memoria RAM para datos: 368 bytes.
- Memoria EEPROM para datos: 256 bytes.
- Capacidad de Interrupciones.
- Stack de 8 niveles.

- Oscilador interno de 8 a 31 MHz.
- Modos de direccionamiento directo, indirecto y relativo.
- Hasta 36 pines de entrada/salida.

- Convertidor A/D de 14 canales, 10 bit.

- Tres timers/counters (8, 16 y 8 bits).

- 1 Módulo CCP (capture, compare y PWM).

- EUSART/SCI.

- In circuit serial programming (ICSP).

- Power On Reset, Power Up Timer, Oscillator Start-up Timer.

- Watchdog Timer, Brown-out detect.

- Code protection.

- Modo SLEEP para ahorro de energía

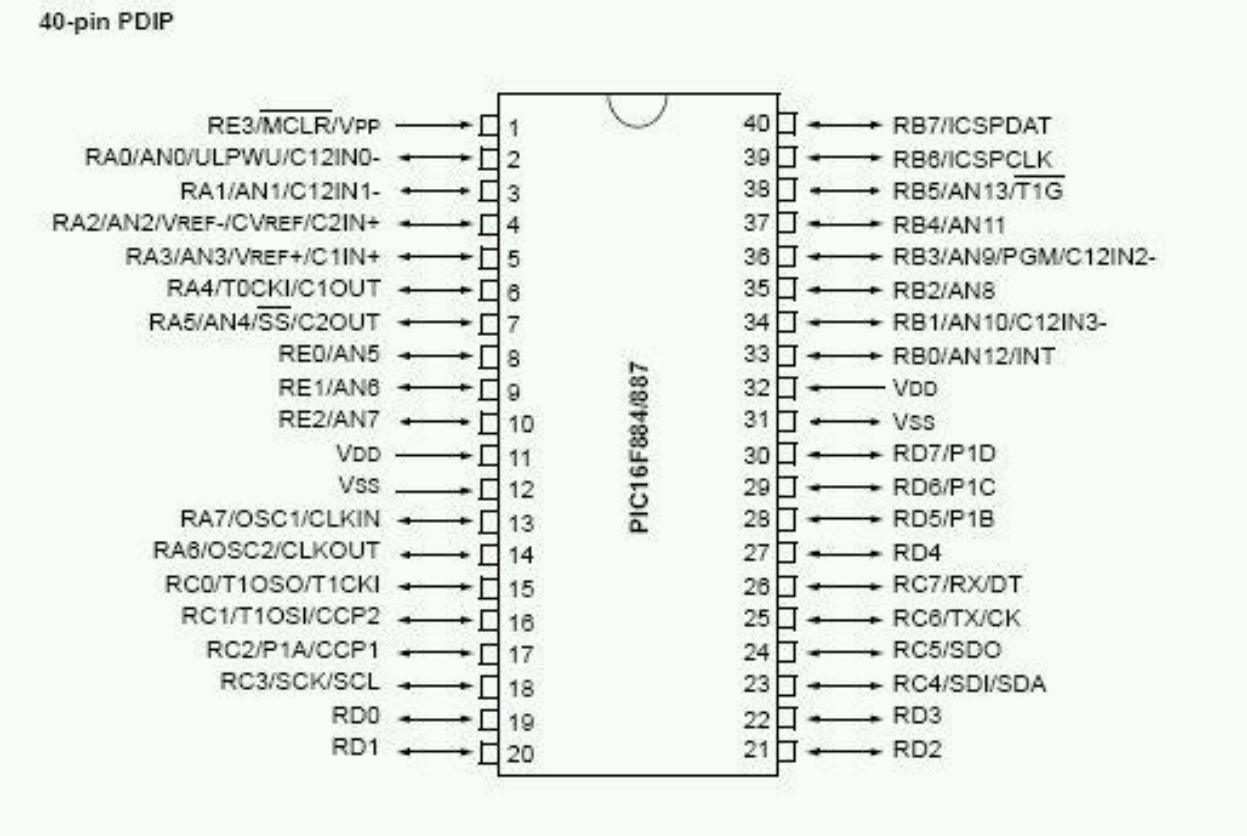

FIGURA 2.3: Configuración de pines PIC 16F887.

2.3.2 Pantalla Tactil GLCD

Las pantallas táctiles resistivas son más accesibles por su durabilidad ante elementos como polvo o agua, razón por la que son el tipo de pantallas más usado en la actualidad, aunque pero tienen una pérdida de aproximadamente el 25% del brillo debido a las múltiples capas necesarias. Otro inconveniente que tienen es que pueden ser dañadas por objetos afilados.

Interface pin connections

PIN NO	Symbol	Function
1	/CSA	Chip select for IC1
2	/CSB	Chip select for IC2
3	VSS	GND
4	VDD	+5V
5	V0	Constrast adjustment
6	D/I	H/L Register select signal
7	R/W	H/L Read/Write signal
8	E	H/L Enable signal
9 to 16	DB0 to DB7	H/L Data bus line
17	RST	Reset signal
18	VEE	Negative voltage output
19	A	Power supply for BKL(4.2V)
20	K	Power supply for BKL(GND)

TABLA 2.1: Configuración de pines Pantalla GLCD.

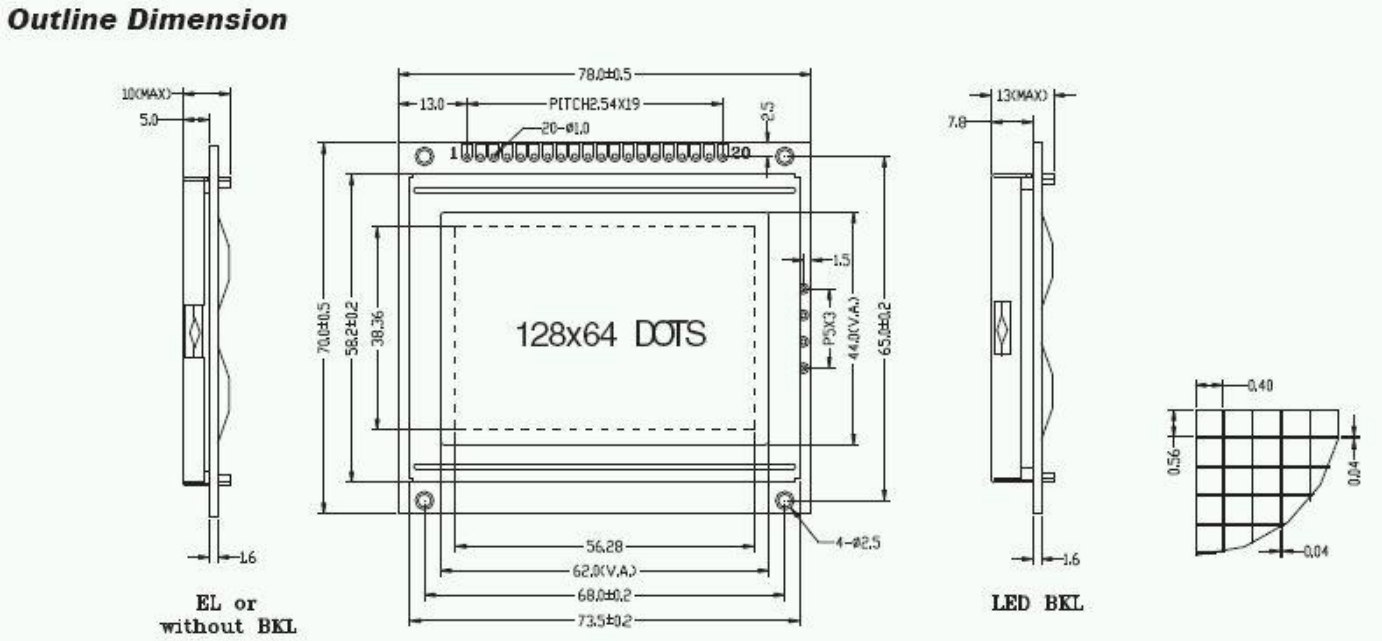

FIGURA 2.4: Dimensiones Pantalla GLCD.

2.3.3 PICKIT 3

El PICkit 3 es un programador fabricado por Microchip© para programar toda su línea de micro-controladores PIC's® desde los PIC10, PIC12, PIC14, PIC16, PIC18, PIC24, dsPIC30 y dsPIC33 (todos los micro-controladores con memoria Flash sin excepción). El programador fue diseñado para programar los microcontroladores en circuito (ICSP) lo que significa que puede programar los microcontroladores montados directamente en tu aplicación y/o protoboard sin necesidad de tener que sacarlo y meterlo cada vez que se modifica el programa.

FIGURA 2.5: Hardware y Software PICkit 3.

CAPITULO 3

3 DESCRIPCION E IMPLEMENTACION DEL PROYECTO

Este capítulo muestra los procesos por los cuales tuvo que pasar nuestro proyecto para poder desarrollarlo como tal.

3.1 Descripciones Detalladas del Proyecto

Este proyecto nos permite visualizar por medio de una pantalla hasta 4 señales digitales que coloquemos a la entrada. Las señales pueden variar en frecuencia y

para poder mostrarlo en la GLCD usamos dos potenciómetros para cambiar el tiempo de muestreo por periodo.

Las 4 señales son ingresadas por los puertos RB4 - RB7, y el control de tiempo de muestreo lo realizamos internamente en el pic con una conversión A/D de los potenciómetros en los puertos RA1 y RA2, que nos permitirá cambiar valores en TIMER1, logrando así interrupciones a distintos tiempos para poder visualizar diferentes frecuencias de las señales de entrada.

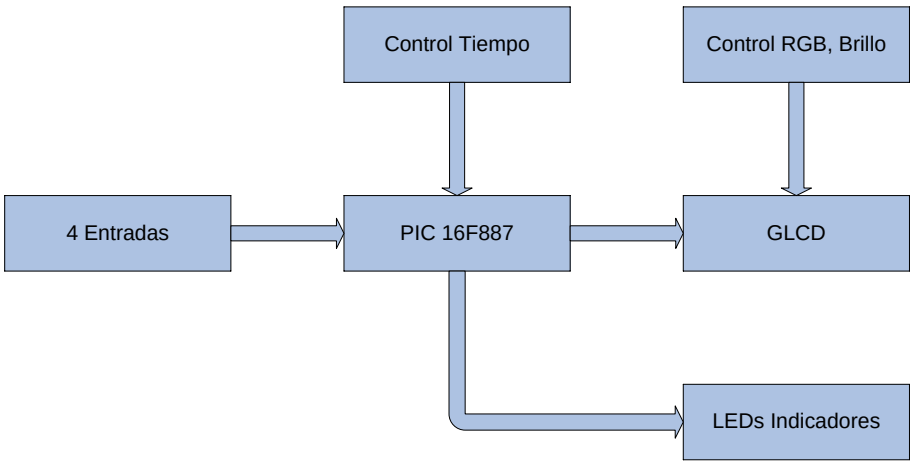

FIGURA 3.1 : Diagrama de bloques del Proyecto

3.1.1 Descripción Uso Convertidor A/D

Usamos un potenciómetro en cada puerto RA1 y RA2 del pic, ya que están configurados como entradas analógicas para poder usar la conversión A/D, al realizar la conversión se almacenan 10 bits, es decir un valor decimal entre 0 y

1023, dando así intervalos para lograr tener un control tipo switch entre los dos potenciómetros.

Se lo realizo de esta manera debido a que en total deberíamos representar 9 estados diferentes, lo que nos representaría desperdicio de recursos del pic si utilizábamos más puertos E/S, por lo que su funcionamiento esta dado por la siguiente tabla.

Frecuencia (Hz)	Periodo (ms)	Valor Conversión A/D	Intervalo Voltaje
5	0,20	0 - 341	1,66v
10	0,10	342 - 682	3,33v
20	0,05	683 - 1023	5,00v

TABLA 3.1: Factor de conversión A/D Potenciómetro 1

Factor	Valor Conversión A/D	Intervalo Voltaje
1	0 - 341	1,66v
10	342 - 682	3,33v
100	683 - 1023	5,00v

TABLA 3.2: Factor conversión A/D Potenciómetro 2

Por ejemplo si queremos mostrar una señal de **1KHz o 1ms**, el potenciómetro 1 debería tener un valor a su salida entre 1.66 y 3.33 voltios y el potenciómetro 2 debería tener un valor entre 3.33 y 5.00 voltios dando así 10 x 100 = **1KHz** o 0.1/100 = **1ms**.

3.1.2 Descripción Uso Timer1

Como es de conocimiento, para uso del temporizador en este caso TIMER1 a mas de configurar los bits para habilitar la interrupción se debe configurar los bits de valor inicial como son TMR1L y TMR1H ya que pueden alcanzar juntos un valor máximo en hexadecimal de 0xFFFF = 65535 instrucciones, además a esto, el uso en este proyecto un cristal externo de 4MHz, es decir 1us cada instrucción, dando como total 65,535ms como tiempo máximo para generar la interrupciones sin usar el pre-escalador.

Configurando el registro T1CON, podremos cambiar el valor del pre-escalador, pudiendo así aumentar el tiempo máximo de estas interrupciones hasta en 8 veces, es decir a 524,28ms.

Tomando estas consideraciones podemos empezar a determinar diferentes valores de TMR1L, TMR1H y T1CON para generar interrupciones de acuerdo a la selección que el usuario de al equipo.

En este proyecto para tratar de generar lo más exacto posible las señales a la salida, lo que hacemos es tomar 4 muestras de un periodo completo de cada señal colocada a la entrada, por lo que si queremos tomar datos de una señal de 20Hz, es decir con periodo de 0,05s, generaremos interrupciones cada 12,5ms. Este principio básico es conocido como Teorema de Nyquist.

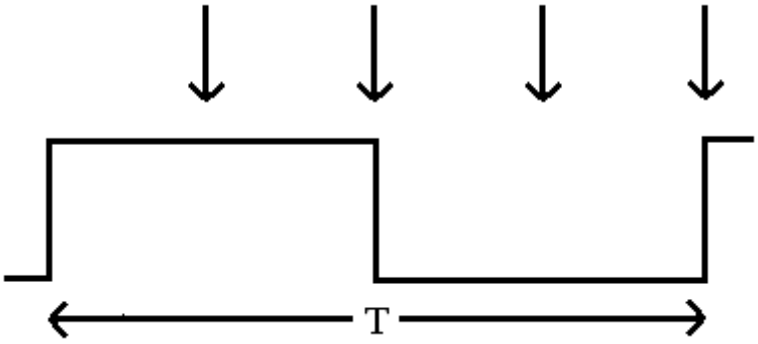

FIGURA 3.2: Interrupciones generadas por periodo

Este proceso fue posible realizarlo para mostrar señales desde 5Hz hasta los 20Hz, por limitaciones del equipo que demostraremos en el siguiente punto.

Dada esta limitación para poder mostrar señales de más de 20Hz, ahora lo que haremos será tomar muestras saltando uno o varios periodos dependiendo de la señal que deseemos observar en la GLCD como mostraremos en la siguiente grafica.

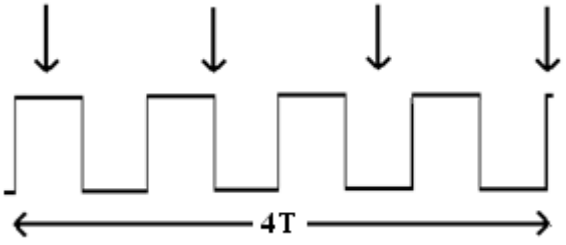

FIGURA 3.3: Interrupciones generadas saltando un periodo

Para el ejemplo de la figura anterior realizamos los cálculos debidos para obtener el tiempo al que debe generarse la interrupción y generaríamos un periodo completo graficado en la GLCD a partir de 4 periodos en la señal de la entrada.

De tal manera que variando los valores a los que deseemos la interrupción lograríamos mostrar señales mayores a los 20Hz hasta una frecuencia máxima de 2KHz, teniendo así un rango de 5Hz hasta 2KHz en los 4 canales de entrada.

3.1.3 Gráficos en la GLCD

Para graficar en la GLCD usamos funciones de la propia librería de MikroC, principalmente lo que se hace es un barrido horizontal hacia la derecha, en cada interrupción se graficará dos pixeles en la GLCD por cada una de las señales de la entrada, censando el valor de esta señal, ya que los gráficos son en tiempo real, es decir, no se guarda información alguna sobre las señales de la entrada, se las grafica directamente, por lo que con lo explicado anteriormente tendremos que un periodo completo de la señal será equivalente a 8 pixeles y como la GLCD tiene 128 pixeles a lo ancho, se graficarán 16 periodos completos.

Graficamos con la función **Glcd_Dot(x,y,color)**, cada pixel dando como valores la coordenada en X, Y y además el color.

3.1.4 Limitaciones de el Proyecto

Por ahora ya explicamos la limitación de las interrupciones hasta un máximo de 524,28ms usando el pre-escalador, pero hasta ahora no hemos considerado el

tiempo que le toma al microcontrolador ejecutar los diferentes procedimientos del programa.

En pruebas realizadas en los simuladores, observamos que la función "Glcd_Dot" tarda aproximadamente 525us en ejecutarse y en la función interrupción se llama 16 veces a este procedimiento, por lo que la función interrupción tarda 8,4ms aproximadamente sin considerar demás código de programación en esta función, lo que extiende este tiempo hasta aproximadamente 8,6ms., es decir podríamos graficar señales desde 1.91Hz hasta 116,28Hz, (8,6 – 524,28ms).

Es así que deberíamos por lo menos esperar 8,6ms para poder generar otra interrupción a fin de no generar errores al graficar las señales.

Estas consideraciones son muy importantes para el diseño de este proyecto, ya que debemos ser muy cuidadosos en cuanto a los tiempos, caso contrario al no haber sincronismo con la señal de la entrada mostraríamos señales erróneas en la GLCD.

Para las señales mayores a 20Hz, como lo dijimos anteriormente para poder graficar estas señales lo hacemos saltando varios periodos, y debimos extender cada interrupción hasta un tiempo de 10 veces su periodo.

Esto se lo realizo con la finalidad de al colocar señales de inferior frecuencia a la señal seleccionada no se superpongan o generen errores en la imagen a la salida.

Ponemos como ejemplo, al seleccionar la frecuencia de 50Hz, calculamos un tiempo de interrupción cada 5ms, tomando 4 muestras, pero la limitación de que debe ser entre 8,6ms y 524,28ms no nos permite programarlo así, ya que obtendríamos errores, si saltamos un periodo para tomar cada muestra entonces deberíamos sumar 20ms de un periodo, obteniendo 25ms, valor que se encuentra dentro del rango, pero para frecuencias menores a esta existirán errores.

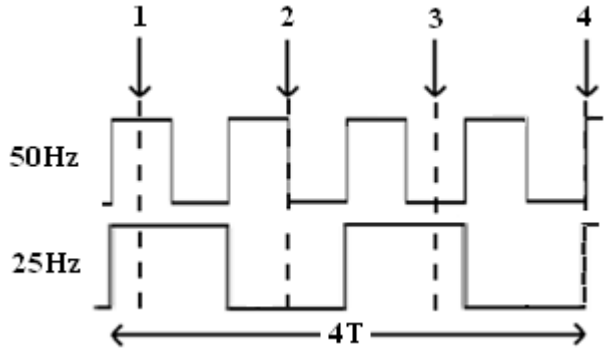

FIGURA 3.4: Interrupciones que generaran error

Analizamos la imagen anterior y con tan solo reducir a la mitad la frecuencia de una de las señales ya obtendríamos errores.

La primera muestra en los 50Hz y 25Hz la obtendríamos como un 1 lógico, estaría correcto para ambas señales, la segunda muestra en la señal de 50Hz tomaría 1 lógico pero para la señal de 25Hz tomaría como dato un cero lógico y seria ya el primer error y así sucesivamente, es por esto que extendemos la interrupción hasta 10 veces su periodo a fin de conseguir poder observar hasta 10 señales menores a la frecuencia seleccionada.

Para la señal de 50Hz entonces se determinó un valor de 205ms, es decir, estamos saltando 10 periodos lo que nos da opción a poder graficar señales de hasta 10 veces menos esta señal en rangos de 5 en 5, es decir desde 5Hz, 10Hz, 15Hz…así hasta los 50Hz y con valores superiores, solo graficará el doble de la frecuencia.

frecuencia	tiempo de interrupción (ms)	periodos saltados
50Hz	205,00	10
100Hz	102,.50	10
200Hz	51,25	10
500Hz	200,50	10
1KHz	100,25	10
2KHz	50,125	10

TABLA 3.3: Tiempo de cada Interrupción considerando periodos omitidos

Por lo expuesto anteriormente, entonces la limitación general que tiene nuestro proyecto es que para frecuencias mayores a 20Hz, las gráficas pueden ser de señales con intervalos de frecuencia (f), 1*f/10; 2*f/10; 3*f/10……9*f/10; f y además 2f.

Este inconveniente se podría solucionar saltando por lo menos 100 o más periodos en cada señal, pero no es una solución viable ya que con este cambio tardaría mucho mas en graficar toda la GLCD y nuestro proyecto tarda 13,12s en graficar pantalla completa para la frecuencia de 50Hz que es el caso que más toma tiempo.

3.2 Código Programación del PIC

A continuación detallaremos paso a paso el código usado para programar nuestro pic 16F887.

3.2.1 Variables y Configuraciones Globales

```
#include <built_in.h>

unsigned int adc_1, adc_2;

int timer1H=0, timer1L=0;

int i=0, estado=1, estado_ant=1;

int rb_4=0, rb_5=0, rb_6=0, rb_7=0;

int flag_switch=1, flag=0;


// Glcd module connections

char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RC0_bit;

sbit GLCD_CS2 at RC1_bit;

sbit GLCD_RS at RC2_bit;

sbit GLCD_RW at RC3_bit;

sbit GLCD_EN at RC4_bit;
```

```
sbit GLCD_RST at RC5_bit;
sbit GLCD_CS1_Direction at TRISC0_bit;
sbit GLCD_CS2_Direction at TRISC1_bit;
sbit GLCD_RS_Direction at TRISC2_bit;
sbit GLCD_RW_Direction at TRISC3_bit;
sbit GLCD_EN_Direction at TRISC4_bit;
sbit GLCD_RST_Direction at TRISC5_bit;
// End Glcd module connections
```

La librería “built_in.h” es usada para la conversión A/D, propia de MikroC.

Las variables adc_1 y adc_2 son utilizadas para almacenar el valor de la conversión A/D de tamaño 10 bits.

Las variables timer1H, timer1L, son usadas para almacenar el valor al que se cambiará la interrupción, ya que luego son asignadas directamente a las variables TMR1H y TMR1L.

La variable “i” es usada para incrementar el valor de la coordenada en X durante el proceso del barrido hacia la derecha para graficar en la GLCD; la variable estado y estado_ant sirven para guardar un valor entre 1 y 9 de acuerdo a los rangos de la conversión A/D.

Las variables rb_4, rb_5, rb_6 y rb_7, son de gran ayuda debido a que son los valores de las 4 entradas justo un instante antes de ejecutar la interrupción, ya que si no los usamos no hay sincronía, por el tiempo que demora el microcontrolador en ejecutar el proceso de graficar en la GLCD.

La variable **flag_switch** es usada como bandera cuando exista cambio en los potenciómetros que nos harán cambiar los tiempos de interrupción y la variable **flag,** también usada como bandera para usar el botón de mantener la pantalla estática.

3.2.2 Funciones usadas en el Programa Principal

Función: adc_dato()

Esta función nos ayuda para determinar intervalos de operación a la conversión A/D de los potenciómetros usados a manera de switch, y a la vez control de 6 puertos E/S en los que encendemos y apagamos leds, según sea el caso para que el usuario pueda conocer el tiempo por periodo de muestreo de la señal en la GLCD.

```
void adc_dato(){

    PORTA.RA3 = 0; PORTA.RA4 = 0; PORTA.RA5 = 0;

    PORTB.RB0 = 0; PORTB.RB1 = 0; PORTB.RB2 = 0;
```

```
//POT2 x 1
if(adc_2 <= 341){
   if(adc_1 <= 341){                //caso 5Hz
        estado = 1; PORTA.RA3 = 1; PORTB.RB0 = 1;}
   if(adc_1 >= 342 & adc_1 <= 682){    //caso 10Hz
        estado = 2; PORTA.RA4 = 1; PORTB.RB0 = 1;}
   if(adc_1 >= 683){                //caso 20Hz
        estado = 3; PORTA.RA5 = 1; PORTB.RB0 = 1;}
}
//POT2 x 10
if(adc_2 >= 342 & adc_2 <= 682){
   if(adc_1 <= 341){                //caso 50Hz
        estado = 4; PORTA.RA3 = 1; PORTB.RB1 = 1;}
   if(adc_1 >= 342 & adc_1 <= 682){    //caso 100Hz
        estado = 5; PORTA.RA4 = 1; PORTB.RB1 = 1;}
```

```
    if(adc_1 >= 683){                //caso 200Hz
        estado = 6; PORTA.RA5 = 1; PORTB.RB1 = 1;}
    }
    //POT2 x 100
    if(adc_2 >= 683){
    if(adc_1 <= 341){                //caso 500Hz
        estado = 7; PORTA.RA3 = 1; PORTB.RB2 = 1;}
    if(adc_1 >= 342 & adc_1 <= 682){    //caso 1KHz
        estado = 8; PORTA.RA4 = 1; PORTB.RB2 = 1;}
    if(adc_1 >= 683){                //caso 2KHz
        estado = 9; PORTA.RA5 = 1; PORTB.RB2 = 1;}
    }
if (estado_ant != estado){
    flag_switch = 1;
    }
```

```
}
```

Función: switch_rot()

Esta función depende de la variable “estado”, que será un valor numérico entre 1-9 producto de la función anterior y esto determinará los valores de configuración de TMR1L, TMR1H y T1CON en esta función.

```
void switch_rot(){
    if(flag_switch){
        switch(estado){
        case 1: timer1H=0x3B; timer1L=0x88; T1CON=1; break;
        case 2: timer1H=0x9D; timer1L=0xC3; T1CON=1; break;
        case 3: timer1H=0xCE; timer1L=0xE1; T1CON=1; break;
        case 4: timer1H=0x36; timer1L=0x70; T1CON=0x21; break;
```

```
        case 5:  timer1H=0x36;  timer1L=0x70;  T1CON=0x11;
break;

        case 6:  timer1H=0x36;  timer1L=0x70;  T1CON=1;

break;

        case 7:  timer1H=0x3A;  timer1L=0xDE;  T1CON=0x21;
break;

         case 8:  timer1H=0x3A;  timer1L=0xDE;  T1CON=0x11;
break;

         case 9:  timer1H=0x3A;  timer1L=0xDE;  T1CON=1;
break;

          default:  break;

         }

         flag_switch=0;

       }

}
```

Función: interrupt()

Función interrupción que es llamada a ejecutarse dependiendo de los valores anteriores, según el usuario quien será quien pueda controlarlo dependiendo de la señales a la entrada para su visualización.

```
void interrupt(){

  if (PIR1.TMR1IF = 1){    // Reiniciar el bit TMR1IF

     TMR1H = timer1H;     // El valor inicial se devuelve en los registros

     TMR1L = timer1L;     // del temporizador TMR1H y TMR1L

      rb_4 = PORTB.RB4;

     rb_5 = PORTB.RB5;

     rb_6 = PORTB.RB6;

     rb_7 = PORTB.RB7;

     PIR1.TMR1IF = 0;

         //ENTRADA 1

        if(rb_4==0){

            Glcd_Dot(i,4,1);
```

```
            Glcd_Dot(i+1,4,1);

            Glcd_Dot(i,13,0);

            Glcd_Dot(i+1,13,0);

        }else{

            Glcd_Dot(i,13,1);

            Glcd_Dot(i+1,13,1);

            Glcd_Dot(i,4,0);

            Glcd_Dot(i+1,4,0);

        }

      //ENTRADA 2

    if(rb_5==0){

            Glcd_Dot(i,19,1);

            Glcd_Dot(i+1,19,1);

            Glcd_Dot(i,28,0);

            Glcd_Dot(i+1,28,0);
```

```
        }else{
            Glcd_Dot(i,28,1);
            Glcd_Dot(i+1,28,1);
            Glcd_Dot(i,19,0);
            Glcd_Dot(i+1,19,0);
      }
    //ENTRADA 3
if(rb_6==0){
            Glcd_Dot(i,35,1);
            Glcd_Dot(i+1,35,1);
            Glcd_Dot(i,44,0);
            Glcd_Dot(i+1,44,0);
        }else{
            Glcd_Dot(i,44,1);
            Glcd_Dot(i+1,44,1);
```

```
        Glcd_Dot(i,35,0);

        Glcd_Dot(i+1,35,0);

  }

//ENTRADA 4

    if(rb_7==0){

        Glcd_Dot(i,50,1);

        Glcd_Dot(i+1,50,1);

        Glcd_Dot(i,59,0);

        Glcd_Dot(i+1,59,0);

    }else{

        Glcd_Dot(i,59,1);

        Glcd_Dot(i+1,59,1);

        Glcd_Dot(i,50,0);

        Glcd_Dot(i+1,50,0);

  }
```

```
    i = i+2;
    if(i>=128){
      i=0;
    }switch_rot();
  }
}
```

Función: init_analizador()

Función que asigna valor a puertos analógicos y digitales, ya sea como entrada o salida y además habilita interrupción Timer1.

```
void init_analizador(){
  ANSEL = 0x04;       // Configurar AN2 (RA2)como pin analógico
  ANSEL = 0x02;       // Configurar AN1 (RA1)como pin analógico
  ANSELH = 0;         // Configurar los demás pines AN como E/S digitales
  TRISB = 0xF0;       // RB4-RB7 as input, rest is output
  TRISD = 0x00;       // Configurar puerto D como output
```

```
TRISA = 0x00;       // Configurar puerto A como output

PORTB = 0x00;       // Valor inicial de los bits del puerto PORTB

PORTD = 0x00;       // Valor inicial de los bits del puerto PORTD

PORTA = 0x00;       // Valor inicial de los bits del puerto PORTA


T1CON = 1;          // Configurar el temporizador TMR1

PIR1.TMR1IF = 0;    // Reiniciar el bit TMR1IF


TMR1H = 0x00;       // Ajustar el valor inicial del temporizador TMR1

TMR1L = 0x00;

PIE1.TMR1IE = 1;          // Habilitar la interrupción al producirse un
desbordamiento

INTCON = 0xC0;      // Interrupción habilitada (bits GIE y PEIE)


C1ON_bit = 0;                       // Disable comparators
```

```
  C2ON_bit = 0;
}
```

3.2.3 Programa Principal

Programa principal que hace uso de todas las funciones descritas anteriormente y adicionalmente valida el botón HOLD, que mantiene estática la GLCD con la señal mostrada para su análisis.

```
void main() {
  init_analizador();     // carga valores iniciales registros y puertos
  Glcd_Init();           // inicia GLCD

  while (1){
     estado_ant = estado;
     adc_1 = ADC_Read(1);   // Obtener el resultado de 10 bits de la
     adc_2 = ADC_Read(2);   // conversión ADC
```

```
    adc_dato();           // Con datos de la conversión escoge un "estado"
                          // según selección del usuario
    switch_rot();         // con la variable "estado" determina valores
                          // de inicio para la interrupción TMR1
    delay_ms(500);
    while(PORTA.RA0==1){    // proceso para usar boton HOLD
      INTCON = 0x00;
      flag = 1;
    }
    if(flag==1){
      INTCON = 0xC0;
      flag = 0;
    }
  }
}
```

CAPITULO 4

4 SIMULACIÓN Y PRUEBAS EXPERIMENTALES

4.1 Implementación en Protoboard.

Para realizar el proyecto, primero se hicieron pruebas en protoboard, básicamente las conexiones entre el pic16F887 y la GLCD, para verificar comunicación desde el pic hacia la GLCD con ejemplos básicos y calibrando color de fondo con los tres potenciómetros RGB y además el control de brillo hasta operar correctamente para así avanzar al siguiente paso.

Una vez que logramos una programación del pic con interrupciones a un mismo tiempo, usamos botoneras a las entradas de las señales, para así simular una señal digital 0-5v.

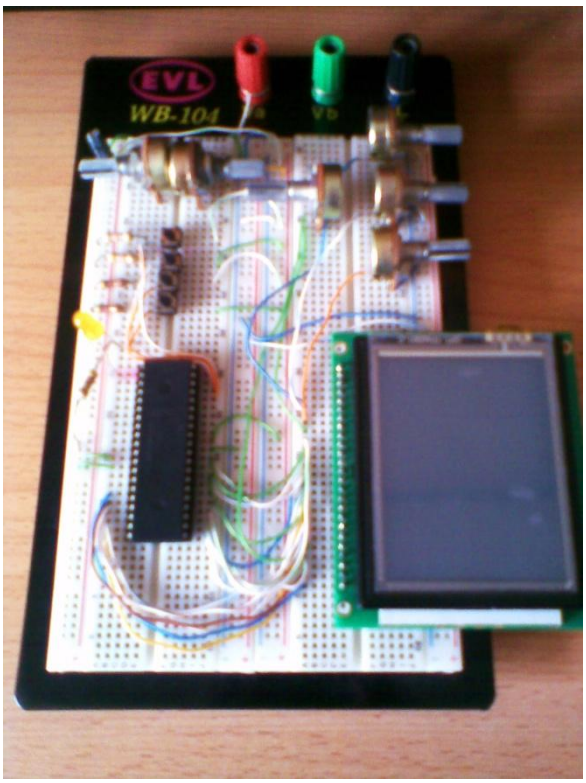

FIGURA 4.1: Conexiones en Protoboard

4.2 Simulación en Proteus.

A través del software Proteus simulamos nuestro proyecto a fin de observar su correcto funcionamiento. Aquí se puede probar los botones Reset del pic y

además el botón Hold para mantener la GLCD estática y poder analizar las señales mostradas en la pantalla.

Simulamos un primer ejemplo, en el que seleccionamos una frecuencia de muestreo de 5Hz o 0,2s, a la entrada colocamos señales de 5Hz, 3Hz, 1Hz y 10Hz respectivamente como podremos observar en la siguiente figura.

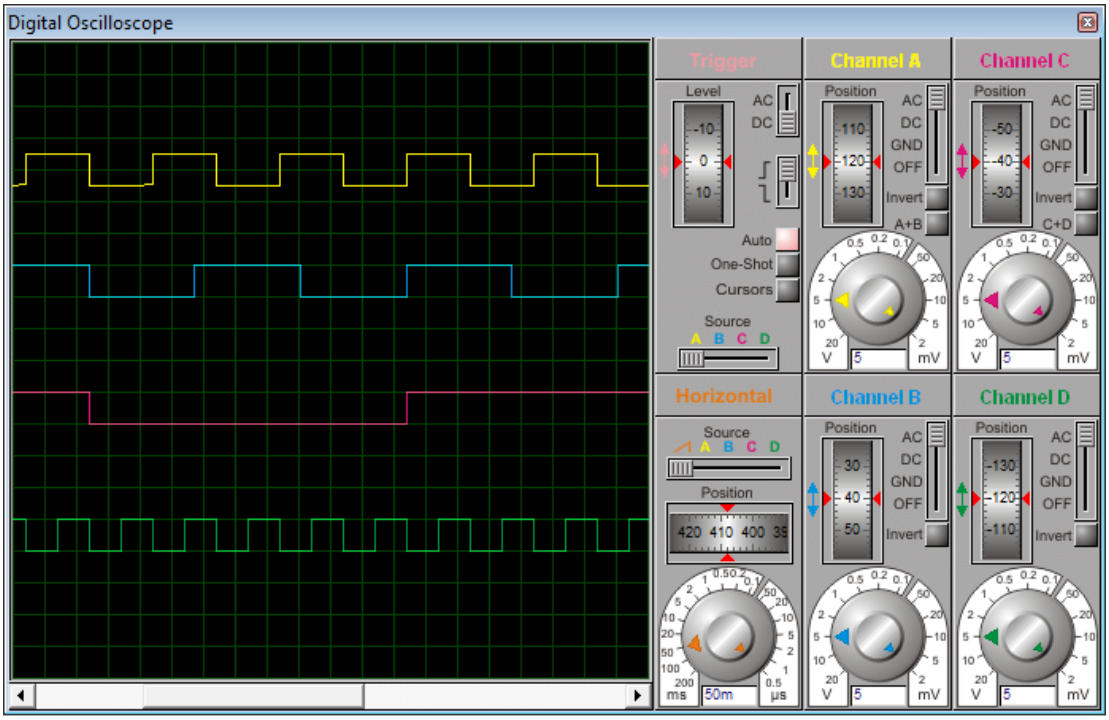

FIGURA 4.2: Osciloscopio en Proteus ejemplo 1

Ahora observaremos la grafica a la salida en la GLCD y además la representación mediante leds de la frecuencia seleccionada por el usuario.

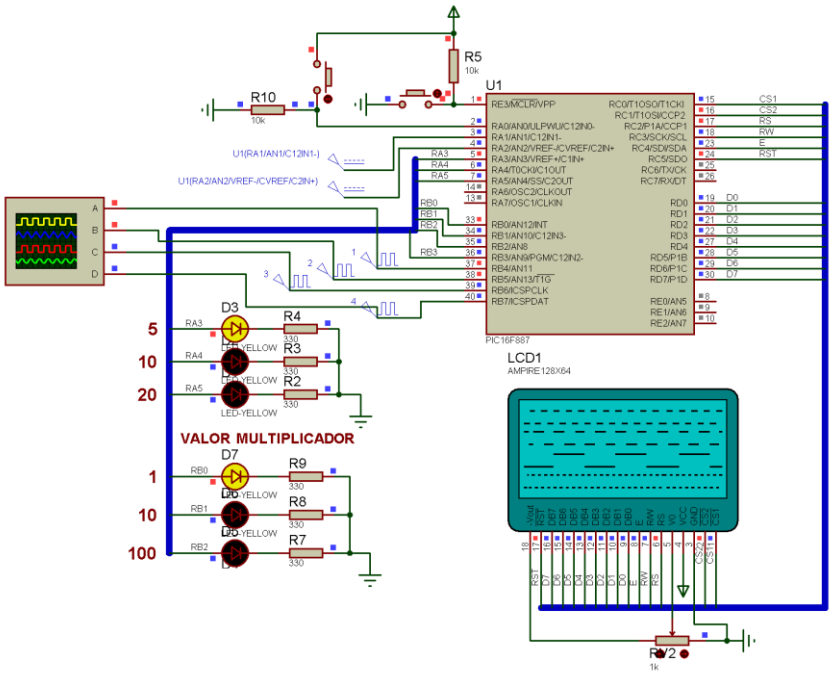

FIGURA 4.3: Señales en GLCD ejemplo 1

Ahora simulamos un segundo ejemplo, en el que seleccionamos una frecuencia de muestreo de 1KHz o 1ms, a la entrada colocamos señales de 1KHz, 500Hz, 100Hz y 800Hz respectivamente como podremos observar en la siguiente figura.

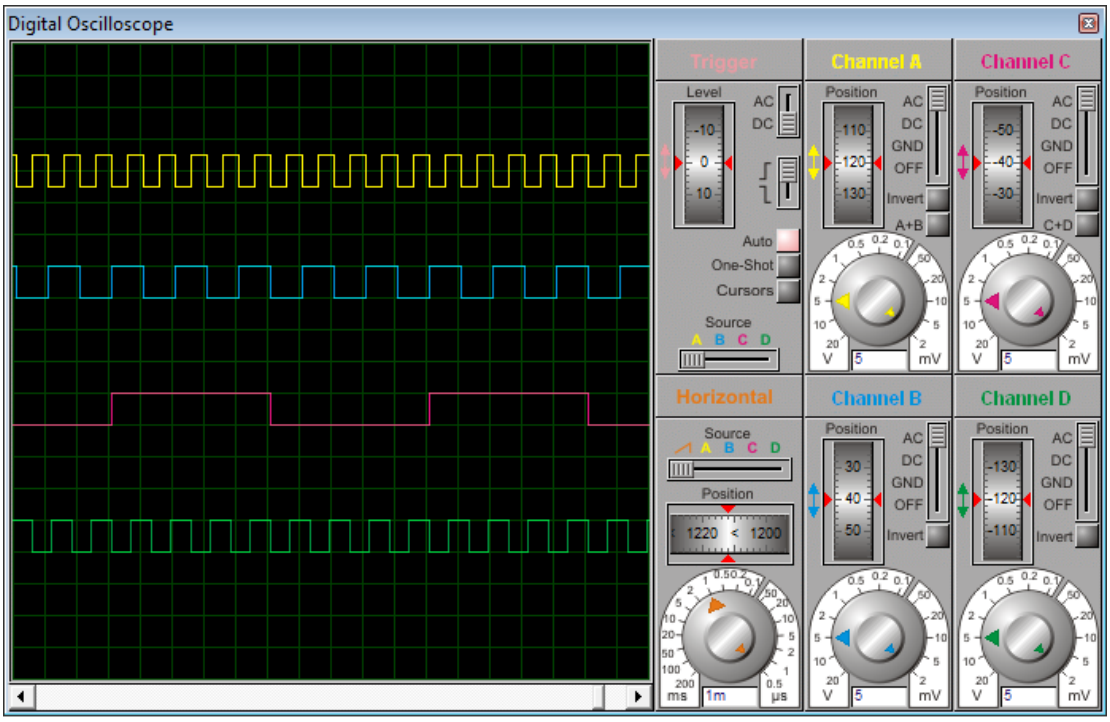

FIGURA 4.4: Osciloscopio en Proteus ejemplo 2

Y observamos en la GLCD y además la representación mediante leds de la frecuencia seleccionada.

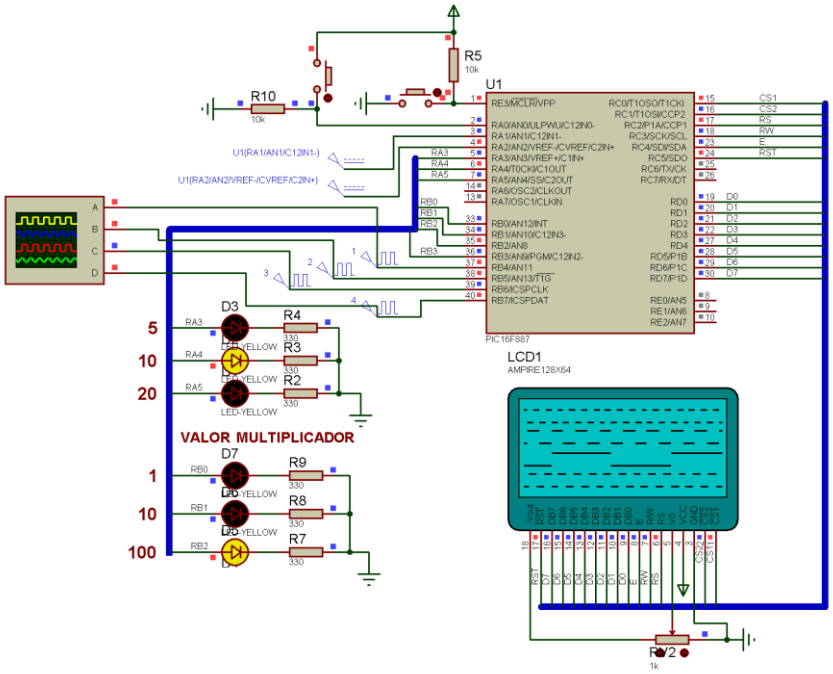

FIGURA 4.5: Señales en GLCD ejemplo 2

4.3 Tarjeta Electrónica PCB

El diseño de la tarjeta se realizó en PROTEUS y ARES de LAB CENTER ELECTRONICS. Es un diseño elaborado a una sola cara.

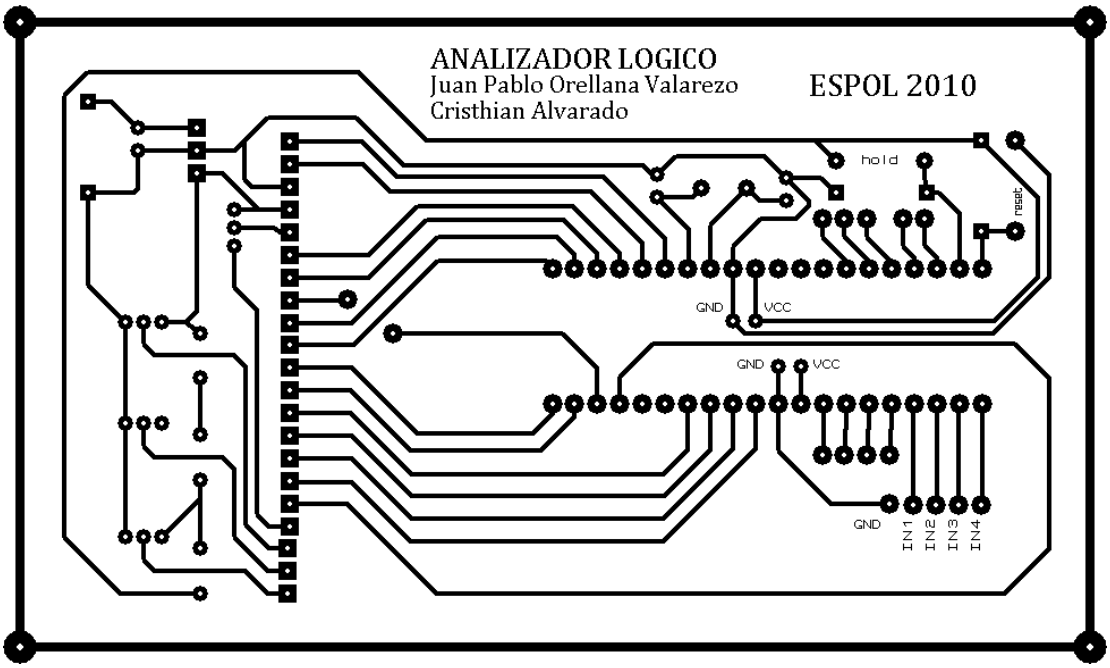

FIGURA 4.6: Diseño PCB

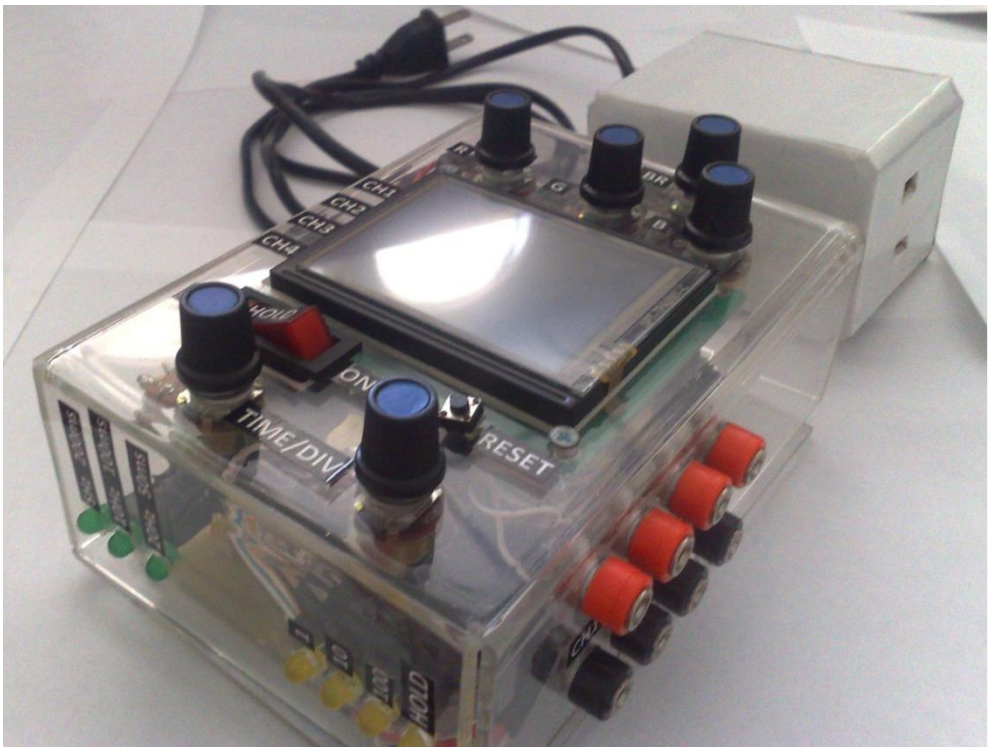

FIGURA 4.7: Proyecto Final