Facultad de Ingeniería

UNIVERSIDAD NACIONAL
de Mar del Plata

“Sistema de monitoreo y adquisición de pulsos eléctricos para fusiones celulares”

Proyecto Final de Ingeniería Electrónica.

Autor:

Fittipaldi Marcos Javier

Directores:
Ing. Gemin, Walter
Ing. Rivera, Raúl

Año 2007

Facultad de Ingeniería

UNIVERSIDAD NACIONAL
de Mar del Plata

“Sistema de monitoreo y adquisición de pulsos eléctricos para fusiones celulares”

Proyecto Final de Ingeniería Electrónica.

Autor:

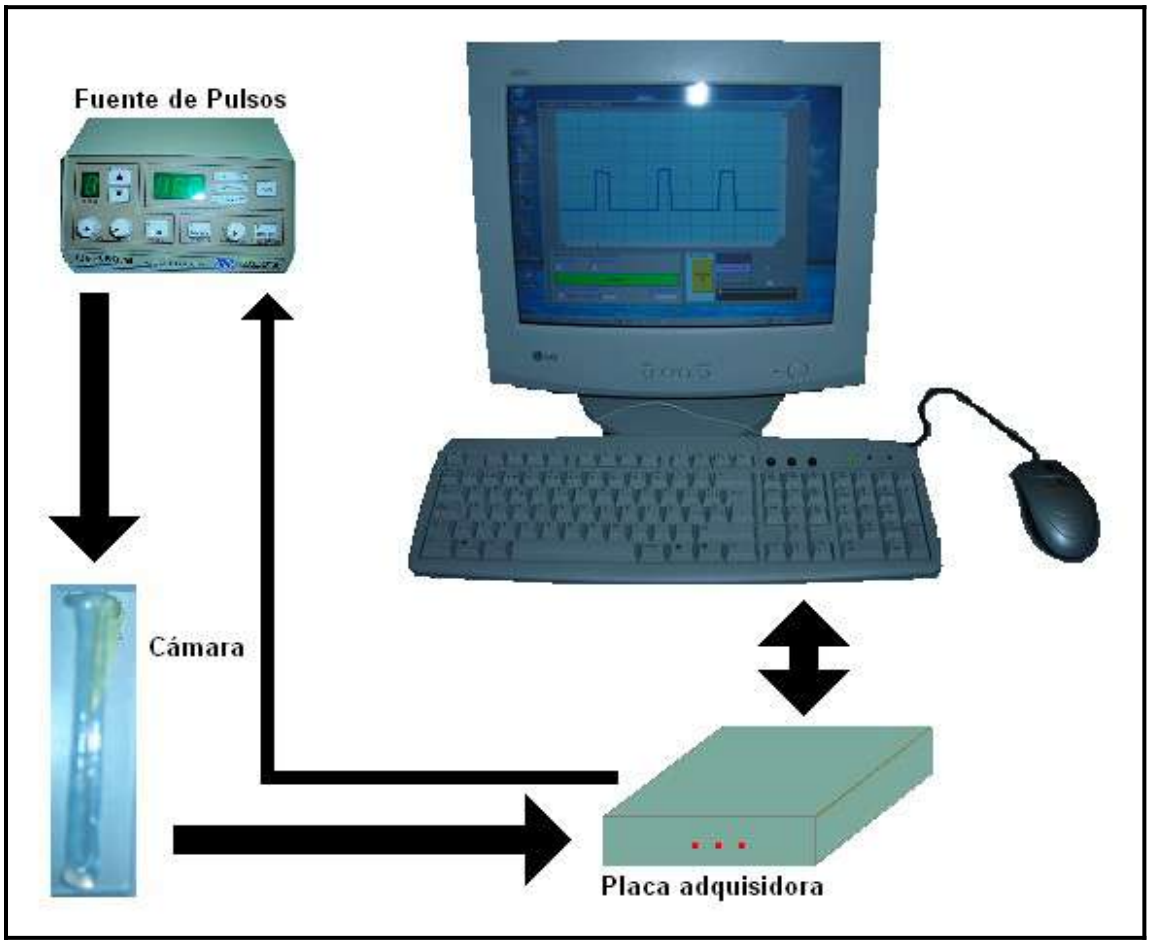
Fittipaldi Marcos Javier

Directores:
Ing. Gemin, Walter
Ing. Rivera, Raúl

Año 2007

Índice

Equipo



2.-Resumen

El presente proyecto se basa en una placa adquisición digital de datos, con el objeto de capturar un pulso eléctrico generado por una fuente de fusión celular.
Los pulsos son aplicados en el proceso de clonación de células; generalmente aplicados sobre óvulos y células de ovinos, dicho sistema biológico esta sumergido en un electrolito del tipo solución salina, en una “cámara” con electrodos conductores.
Los pulsos de la fuente USCF150B pueden ser modificados tanto en amplitud como en duración. Si las dimensiones son correctas, las membranas de dichos cuerpos biológicos son perforadas, lográndose la transferencia del ADN Celular al Óvulo en el caso particular de clonación.
Los datos son adquiridos a una velocidad de 20Msamples por segundo y almacenados en placa en una memoria FIFO, luego son transferidos al CPU por medio del microcontrolador PIC18F4550 a través del puerto USB 2.0.
Una vez en la PC los datos son procesados y representados en pantalla.
El software fue desarrollado en lenguaje C#, un lenguaje orientada a objetos con el propósito de tener al alcance el software y herramientas de programación de última generación.
La información suministrada en la pantalla es capaz de ofrecer al usuario en forma simple, las dimensiones y características de la señal adquirida, necesarias en el desarrollo de las prácticas de laboratorio.

Características generales:

- Alimentación 5volts.
- Consumo 228mA.
- Velocidad de muestreo 20Msps.
- Ancho de banda 5Mhz.
- Impedancia de entrada 10MΩ
- Resolución 488 microVolts.
- Memoria en placa 4096 palabras de 18bits.
- Velocidad de transferencia (USB) 12Mbps.
- Sistema operativo Windows 95/98/2000/XP.

Se obtuvo una placa adquisidora de alta velocidad de muestreo, logrando un bajo costo total del equipo, comparado con osciloscopios de prestaciones similares, con la capacidad de controlar el proceso de fusión celular y clonación.

3.-Introducción

Dado que es posible la visualización de formas de ondas por medio de un osciloscopio, no siempre se tiene a disposición el instrumental y se pueden extraer datos precisos de la pantalla, por tal motivo se propuso el diseño de una placa adquisidora, que registra los pulsos, los representa en la pantalla y almacena en la PC; además de poder controlar el instante de disparo del mismo.
Se utilizo como fuente de información al " Biological Laboratory Equipments Maintenance and Service Ltd." fabricante de equipamiento de laboratorio, osciloscopios Teltron, Microchip, Texas Instruments, Cypress, Altera, CCSC, Microsoft e información obtenida en Internet.

Para todo ello se dividió la investigación sistema en partes:

En primer lugar el circuito de entrada fue diseñado para obtener mayor ancho de banda posible, menor nivel de ruido y alta impedancia de entrada para no afectar al sistema bajo prueba.

En segundo lugar se diseño un control integrado y sincronizado de las partes fundamentales del desarrollo, como son el control del conversor analógico-digital y la memoria de datos tipo FIFO, ambas de alta velocidad. Todo el

control se llevo a cabo por medio de un microcontrolador de ultima generación de la serie “PIC18” de Microchip.

En tercer lugar se desarrollo un protocolo de comunicación por puerto USB y se diseño un software de PC que posibilita gestionar la comunicación, control y representación en pantalla con información adicional del proceso.

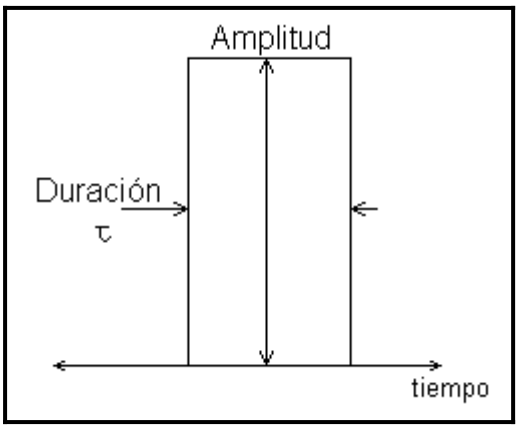
El proceso de investigación y desarrollo del software se dividió en tres partes:

- La investigación de los modos de transferencia del puerto USB
- Desarrollo y configuración soft del núcleo usb del microcontrolador PIC encargado de la comunicación.
- El desarrollo de un paquete de software compatible con las plataformas Windows 9x, XP encargado de la gestión de comunicación y control con el dispositivo de adquisición.

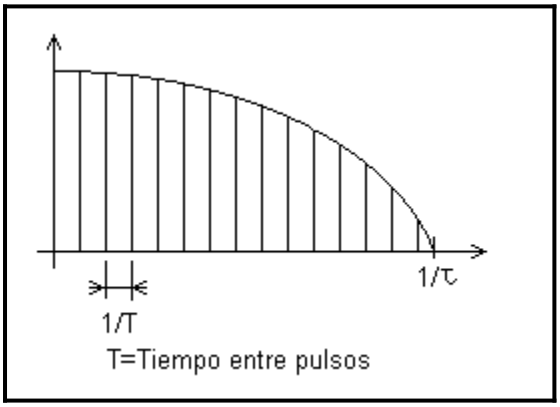
4.-Anteproyecto
4.1-Requisitos

Para comenzar el desarrollo de cada una de las partes del sistema, se analizó el tipo de señal a ser muestreada. Las dimensiones de las señales pueden ser modificadas en amplitud y duración antes de ser aplicado el pulso durante las pruebas.
Las características son las siguientes:

Amplitud mínima 10 Volts.
Amplitud máxima 155 Volts.
Duración mínima 10 microsegundos.
Duración máxima 150 microsegundos.
Forma del pulso estándar ideal rectangular.
Gráficamente el pulso es de la siguiente forma:



Haciendo un análisis espectral de la señal podemos asegurar que el ancho de banda requerido en el peor caso esta dado por:



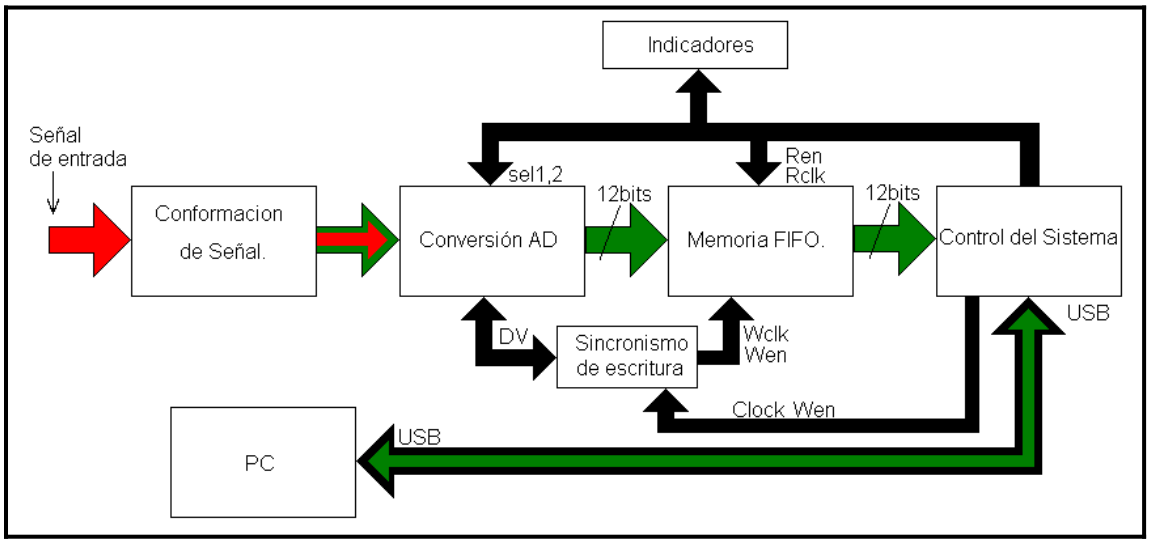
$$Tpulso = 10\mu seg = \tau \min$$

$$1/\tau = 100Khz$$

Según la teoría de muestreo y el análisis espectral tendremos que adquirir datos a una velocidad superior a 250Ksps y tener un ancho de banda superior a los 500Khz.
Podemos concluir primeramente que nuestro sistema debe contemplar las características anteriormente mencionadas.

4.2-Diagrama en bloques:

En el diagrama en bloques siguiente se puede contemplar cada una de las etapas que conforman la placa de adquisición propuesta; la etapa de entrada de conformación de la señal, adquisición y conversión digital, almacenaje en memoria, control de sincronismo de escritura, control del sistema, puerto de comunicación y visualización en pantalla.



4.3-Circuito de entrada

Esta parte del circuito comprende desde la punta de pruebas atenuadora, preamplificador en configuración seguidor y hasta el preamplificador con salida diferencial.

El circuito de entrada primeramente debe adaptar la impedancia entre la punta de prueba y el preamplificador filtro pasa-bajos.
Se experimento con dos circuitos básicos, un circuito adaptador implementado en los osciloscopios, y otro desarrollado de conceptos teórico-prácticos; adaptando ambos a las necesidades requeridas.
El primer circuito es un adaptador de impedancias realizado con un transistor de efecto de campo (FET) el cual tiene un ancho de banda considerablemente amplio, alta impedancia de entrada y de bajo costo de mercado. Una de las desventajas observadas a los fines prácticos es que para su correcta polarización, su alimentación debe provenir de una fuente partida y el prototipo es alimentado desde el puerto USB el cual entrega una tensión de 5Volts y una corriente máxima de 500mA.
Esta dificultad dio paso a la segunda propuesta implementada con un CI TL082. Es un circuito integrado con un ancho de banda muy inferior comparado al caso anterior, es de solo 5Mhz en ganancia unitaria, pero también tiene alta impedancia de entrada y es necesaria una fuente de alimentación partida pero con bajo consumo para su funcionamiento. Su costo es también bajo dentro unos pocos centavos.
La señal luego de atravesar el circuito de entrada, entra al preamplificador THS4503 el cual tiene su salida del tipo diferencial. Además de alcanzar un ancho de banda muy grande su figura de ruido es baja. Este CI es recomendado por el fabricante del conversor análogo-digital (ADS809 de TI.).

4.4-Conversión Análoga-Digital

La etapa siguiente es la conversión de la señal analógica a digital llevada a cabo por el ADS809 de Texas Instruments.
Este dispositivo es elegido dado que cumple los requerimientos de velocidad de hasta 80Msps, consumo inferior a 170mA, resolución de 12bits,tensión de alimentación de 5Volts y su costo reducido; además dicho dispositivo fue donado por ex alumno de la facultad de ingeniería UNMDP.
Dentro de los dispositivos elegidos estaban el ADC14L040, ADC12L020 de TI y otro como TSA1401 de ST que reúnen similares características al ADS809.

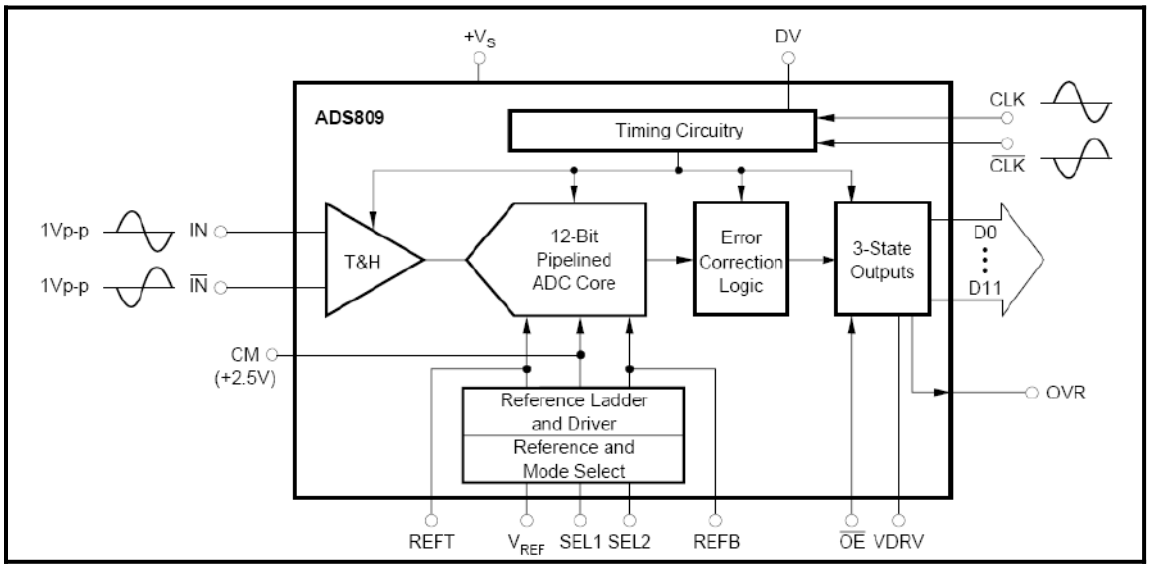
Las características principales de este dispositivo son:

Rango Dinámico:
- SNR: 65dB a 10MHz de frecuencia de entrada.
- SFDR: 68dB a 10MHz de frecuencia de entrada.

Compensación de muestreo y retención:
- Bajo Jitters: 0.5ps rms.
- Diferencial o simple entrada de señal.
- Selección de rango de escala de entrada.

Flexibilidad de señal de CLOCK:
- Diferencial o Simple
- Acepta clock de tipo senoidal o cuadrado inferior a 0.5Vp-p.
- Nivel de umbral variable.

El chip esta organizado de la siguiente manera:



4.5-Memoria

En la investigación de que tipo de dispositivo de almacenamiento de datos se ajustan al desarrollo, se optaron por las siguientes ideas:

- Banco de memorias sRAM 2Mbyts. X16bits.
- Un chip de memoria sRam de 16Kbyts. X16bits.
- Chip de memoria FIFO de 4Kwords de 18bits.

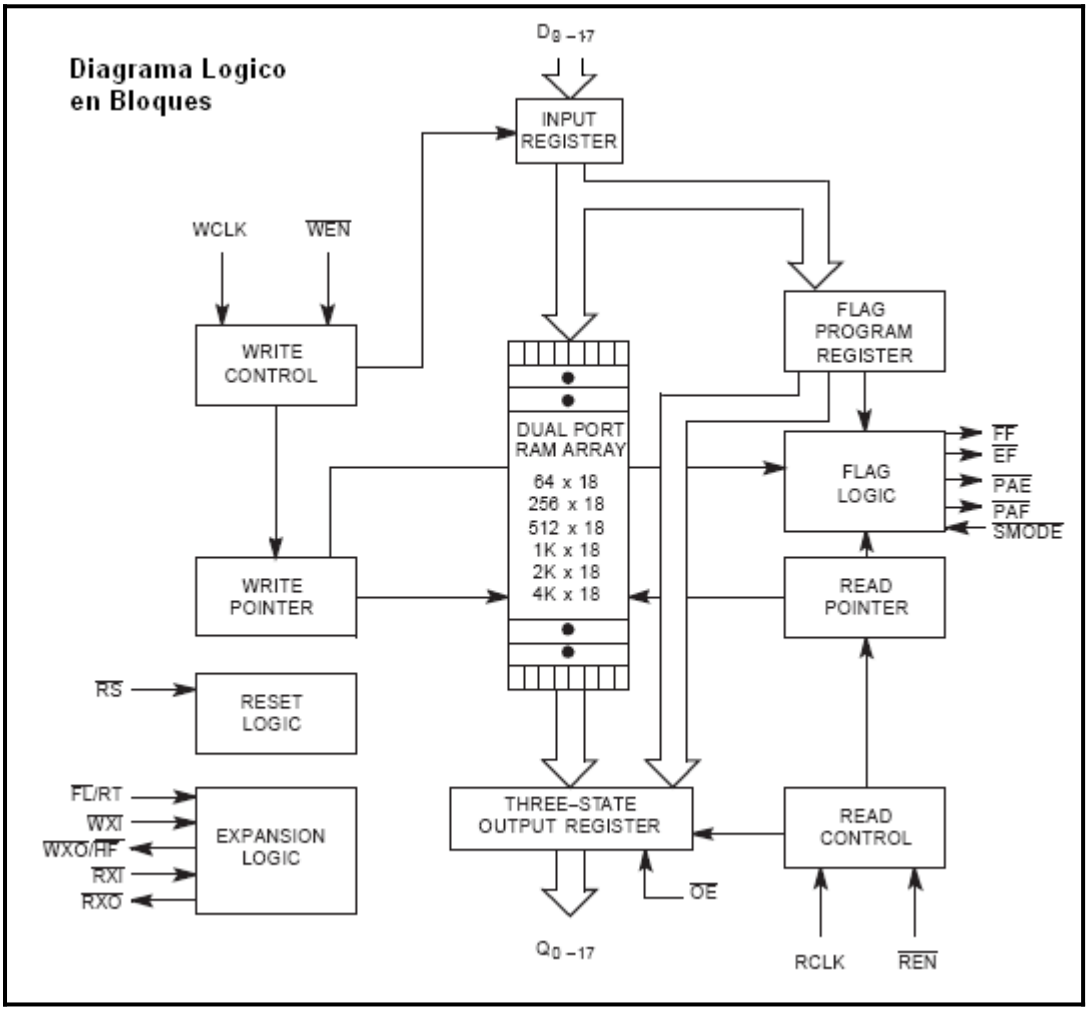
Comenzamos a estudiar la primer alternativa logrando un circuito completo de control del flujo de datos. Este consistía de cuatro chips de memorias sRam de 512K palabras de 8bits cada uno. El control era por medio de un CPLDs de la familia MAX3000A, precisamente el EPM3064ALC44-10, un dispositivo lógico programable de 64 macro celdas de 600 compuertas cada una.
El circuito elaborado fue evaluado y simulado lógicamente con el programa Quartus versión student de Altera. Los resultados obtenidos fueron aceptables, pero el circuito a desarrollar era de un volumen y complejidad exagerada, cuyo consumo de potencia superaba lo máximo admisible suministrado por el puerto USB.

La segunda opción fue disminuir la capacidad de memoria reduciendo el número de chips de almacenamiento a solo uno de 64K-bits de 4Kx18 de capacidad.
Con chip seleccionado logramos bajar el consumo total del circuito.
Haciendo mediciones sobre la red de suministro del puerto USB encontramos que al aumentar el consumo de corriente, la tensión de 5Volts disminuía cerca del 5% y en caso de un pico de corriente de puesta en marcha o consumo inesperado de algún sector podría peligrar el buen funcionamiento del resto del circuito y de probables averías en la fuente de alimentación del CPU.
Otro de los inconvenientes encontrados fue que los dispositivos CPLDs no están disponibles en para su exportación desde EEUU sin rigurosos controles

de aprobación, dado que son dispositivos de fácil reprogramación y de su probable peligrosidad en circuitos ensamblados en vuelos aéreos.

La ultima opción adoptada fue la utilización de una memoria FIFO de 4K-words de 18bits. es el CY7C4245. Dicha memoria simplifica notablemente el circuito dado que no es requerido un desarrollado control lógico como son los contadores que incrementan la posición de memoria para su direccionamiento. Los datos ingresan por un puerto de entrada, son almacenados en un registro de tipo flip-flops y son trasladados sincrónicamente con cada pulso de reloj hacia el puerto de salida. El control necesario para su funcionamiento es solo Wen, Ren, Wclock, Rclock y CS; luego solo son requeridos conectar los puertos de entrada y salida de datos; además el consumo es reducido a solo 36mA. y el tamaño del circuito completo es reducido a solo el encapsulado de 10mm cuadrados.

El costo del dispositivo es inferior a las otras propuestas y su diagrama circuital es:

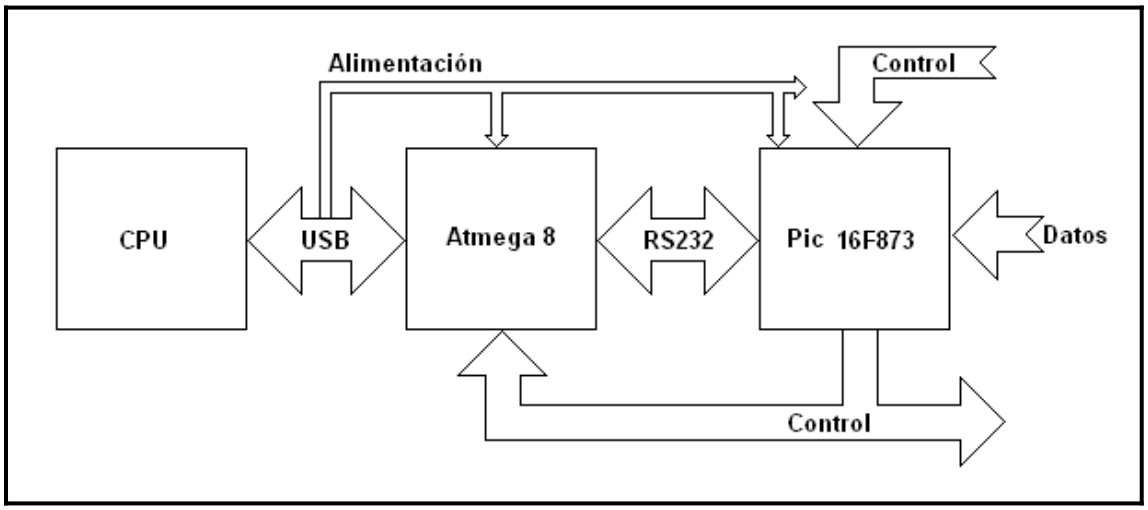


4.6-Microcontroladores

Dentro de las opciones posibles de controlar todo el sistema, optamos primeramente en un sistema compuesto por un microcontrolador que se encargue exclusivamente a la comunicación por USB y otro microcontrolador muy íntimamente ligado a este, encargado del control del proceso.

4.6.1-Atmega8 y Pic16f873

Una sección de lo propuesto es el siguiente diagrama en bloques:



El microcontrolador de Atmel, Atmega8 alojado en el prototipo, es el encargado de las comunicaciones entre el CPU y el microcontrolador de Microchip PIC16F873.
Este dispositivo realiza la comunicación haciendo una emulación del protocolo USB por medio de firmware, a su vez este convierte los datos provenientes del CPU del protocolo USB, en un protocolo RS232 de propósitos generales La

comunicación por USB requiere además un software exclusivo en el CPU con su correspondiente DLL.
La velocidad de transmisión del Atmega8 por USB es FullSpeed de 12Mbps.

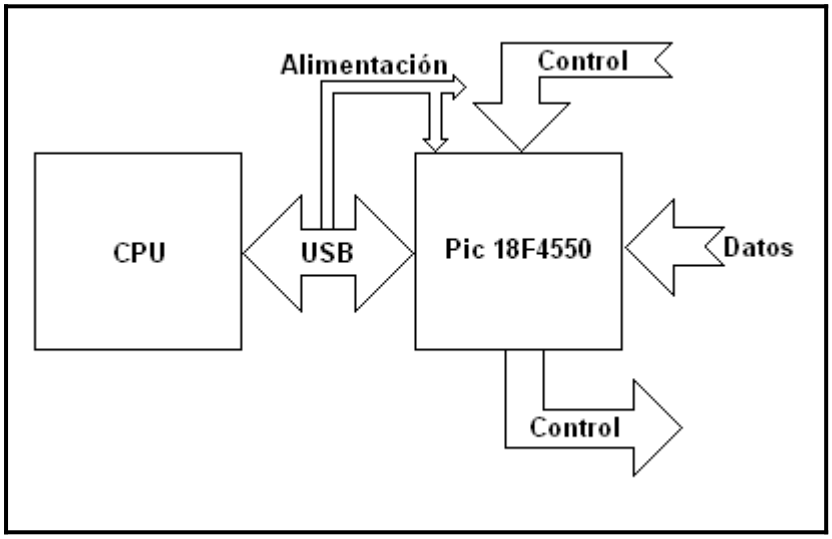
Dificultades encontradas:

- Diferentes velocidades de procesamiento de datos de cada microcontrolador.
- Una alta velocidad de transferencia de datos por USB y muy baja por RS232 entre los microcontroladores, da como resultado que la velocidad de transferencia se ajuste al sector más lento.
- Cada uno de los microcontroladores al ser de diferentes fabricantes, son programados en distintos niveles de lenguaje. El Atmega8 se programa en ensamblador con el AVR Studio conjuntamente con el Ispprog para grabar la ROM del chip y el Pic18f873 por su parte puede ser programado en C con Mplab o CCS-C y winpic800 para grabar la ROM del chip. Cada uno de ellos con diferentes hardware para la programación, STK200 Y GTP entre otros respectivamente.
- Se realizo un esquema de prueba y los cálculos de corriente arrojaron un exceso de consumo para todo el prototipo.

Otra de las opciones estudiadas fue un esquema integrado por solo un chip que realice todas las funciones.

4.6.2-PIC 18F455

El siguiente esquema muestra la idea:



El PIC18F4550 es un nuevo chip de Microchip, que incorpora en el mismo encapsulado dos núcleos procesadores, uno principal o maestro que controla todo el sistema y otro núcleo dedicado exclusivamente a la comunicación por USB FullSpeed a 12Mbps. Ambos núcleos pueden compartir 1Kbyte de memoria RAM de libre acceso.

Esta característica da un aumento notable en la velocidad de transferencia, unificación y acceso a los datos.
Esta nueva topología soluciona todas las dificultades anteriores con nuevas ventajas

- Única y alta velocidad de procesamiento.
- Rapidez en la comunicación y control.
- Mayor cantidad de puertos de entrada de datos y salidas de control.
- Mayor capacidad de memoria RAM de datos y ROM de programa.
- Programación del firmware en un nivel superior “C”.
- Menor nivel de ruido irradiado por el oscilador.
- Único programador y programación en circuito.
- Bajo consumo de corriente.
- Volumen reducido.
- Menor costo.

4.7-Lenguaje de programación de interfase gráfica

Dentro de las opciones a la hora de elegir un lenguaje de programación, se optó por C#.
Es un moderno lenguaje que fue presentado por Microsoft en Julio del año 2000.
El lenguaje de programación C# fue creado con el mismo espíritu que los lenguajes C y C++. Esto explica sus poderosas prestaciones y su fácil curva de aprendizaje. No se puede decir lo mismo de C y C++. Como fue creado desde cero, Microsoft se tomo la libertad de eliminar algunas de las prestaciones más pesadas (como los Punteros)

Este lenguaje combina las mejores ideas de C, C++ y Java con las mejoras de .NET Framework de Microsoft.

.Net Framework se compone de cuatro partes:

- Entorno común de ejecución.
- Un conjunto de bibliotecas de clases.
- Un grupo de lenguajes de programación.
- El entorno ASP.NET.

Y fue diseñado con tres objetivos en mente:

- Debía lograr aplicaciones de Windows más estables y proporcionar una aplicación con mayor grado de seguridad.
- Debía simplificar el desarrollo de aplicaciones y servicios Web.
- El entorno fue diseñado para proporcionar un solo grupo de bibliotecas que pudieran trabajar con varios lenguajes.

4.7.1-Desarrollo de aplicaciones

Aparte del desarrollo Web con .NET Framework también puede construir las tradicionales aplicaciones de Windows. Estas aplicaciones creadas con .NET Framework se basan en Windows Forms. Windows Forms es una especie de cruce entre los formularios de Visual Basic 6 y los formularios de Visual C++. Aunque los formularios son iguales a los predecesores, están completamente orientados a objetos y basados en clases de forma muy parecida a los formularios objeto de Windows Foundation Class.
Estos Windows Forms ahora admiten muchos de los controles clásicos que aparecían en Visual Studio como Button, TextBox y Label, junto con los Actives.

4.8-Edición del firmware del PIC.

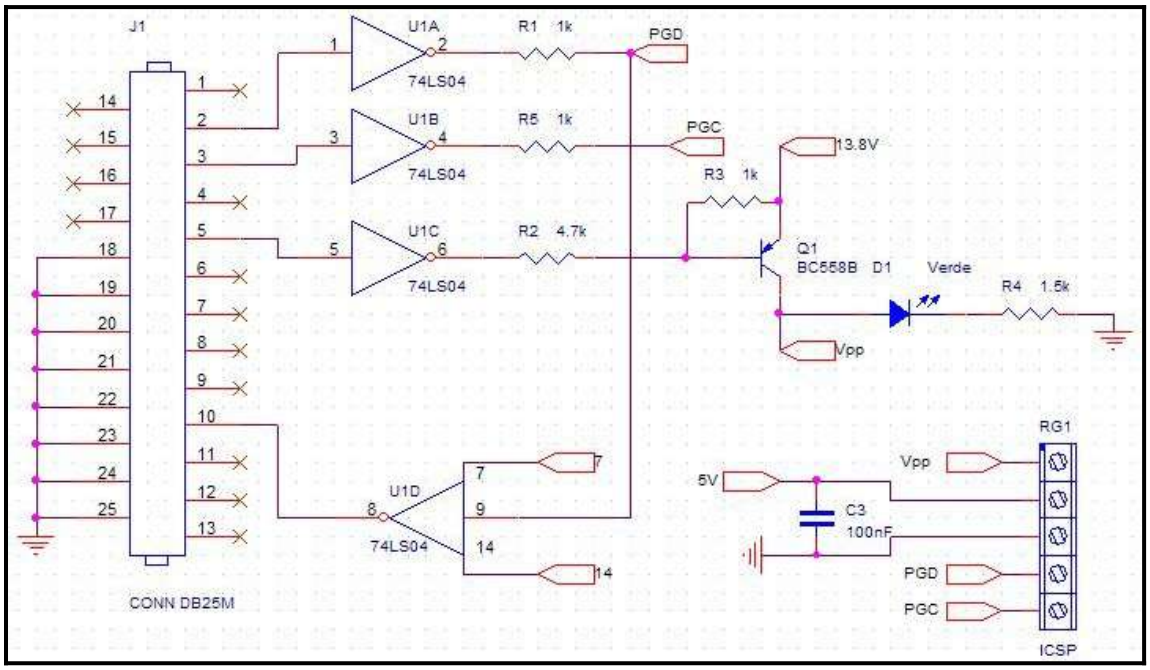
El compilador de C que vamos a utilizar es el PCW de la firma CCS Inc.
El compilador lo integramos en un entorno de desarrollo integrado (IDE) que nos va a permitir desarrollar todas y cada una de las fases que se compone un proyecto, desde la edición hasta la compilación, pasando por la depuración de errores. La ultima fase a excepción de la depuración será la programación del Pic.
Al igual que el compilador de Turbo C, este “Traduce” el código C del archivo fuente (.C) a lenguaje máquina para los microcontroladores PIC, generando así un archivo en formato hexadecimal (.HEX).

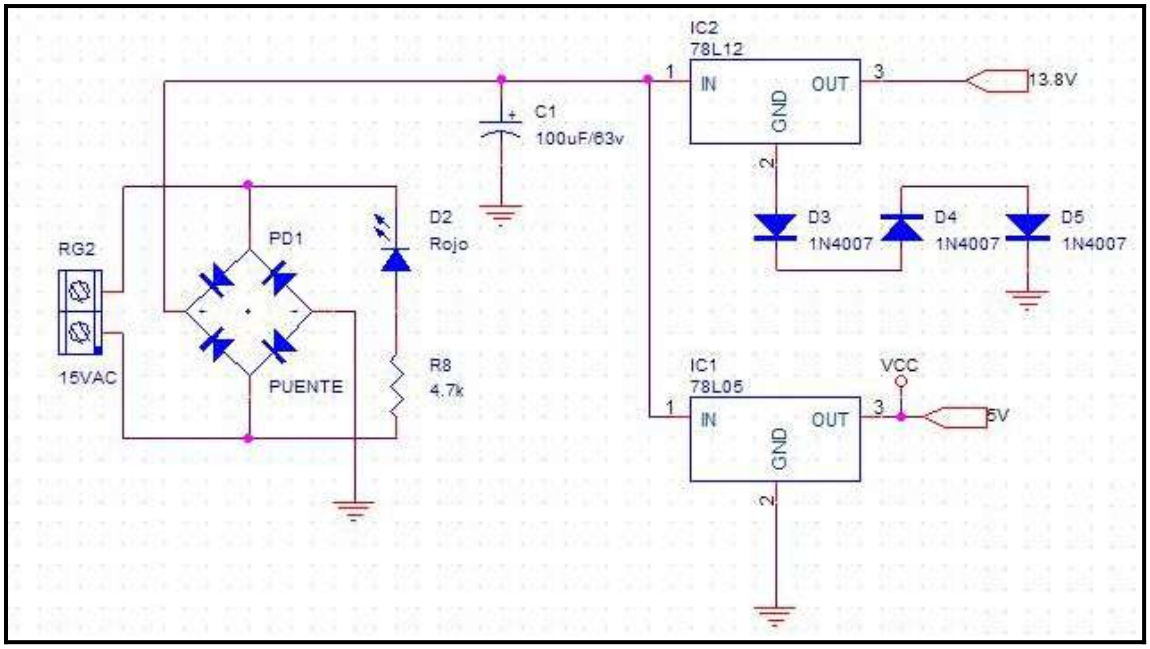
4.8.1-El programador de microcontroladores PIC.(Winpic800 GTP Lite)

El Winpic800 es el programa encargado de trasladar el programa a la memoria del microcontrolador. Los recursos de hardware utilizados para tal fin son el puerto paralelo conjuntamente con el programador GTP Lite.
El GTP Lite es un grabador ICSP de microcontroladores PIC y memorias, haciendo uso del puerto paralelo y de una alimentación externa. Utiliza el método ICSP (In Circuit Serial Programming) que solo necesita tres conexiones de habilitación y de sincronismo de datos, además de las dos de alimentación.

4.8.2-Hardware del programador

El circuito esquematico que se observa abajo es el GTP Lite y más abajo su fuente de alimentacion


J1
U1A
74LS04
R1 1k
PGD
U1B
74LS04
R5 1k
PGC
13.8V
R3 1k
U1C
74LS04
R2 4.7k
Q1
BC558B
D1
Verde
R4 1.5k
Vpp
U1D
74LS04
RG1
5V
C3
100nF
Vpp
PGD
PGC
ICSP
CONN DB25M

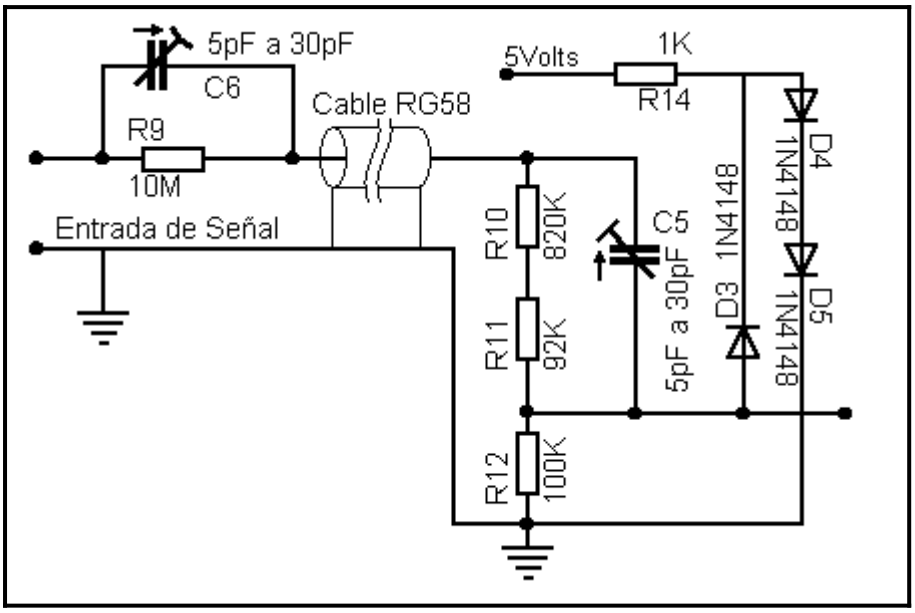

IC2
78L12
IN
OUT
GND
13.8V
C1
100uF/63v
D3
1N4007
D4
1N4007
D5
1N4007
RG2
15VAC
PD1
PUENTE
D2
Rojo
R8
4.7k
IC1
78L05
VCC
5V

.

5.-Proyecto

5.1-Circuito de entrada

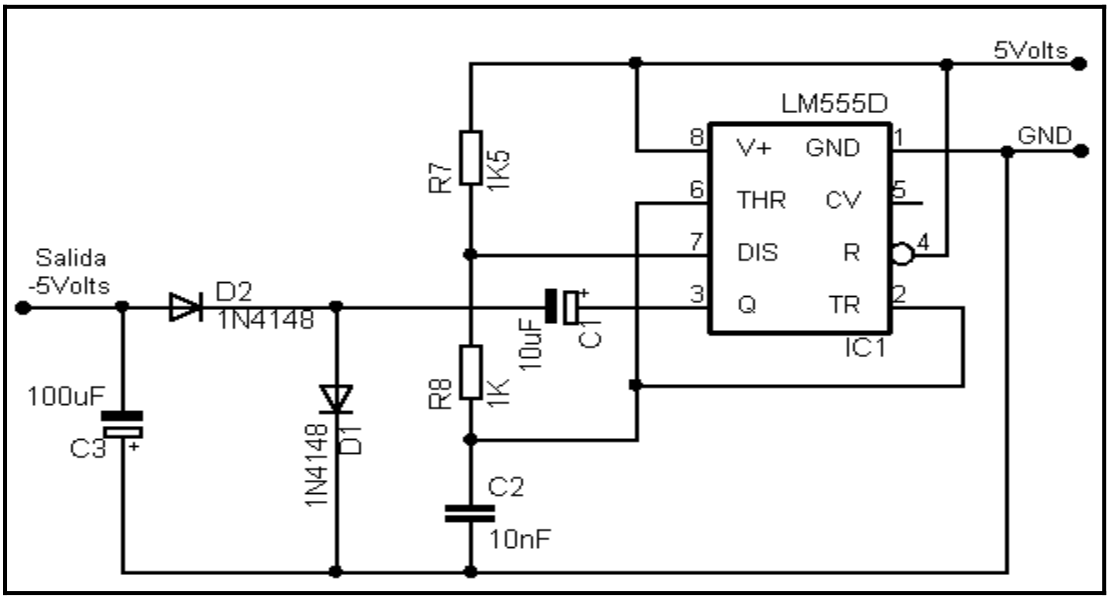
La señal a medir debe ser adquirida por un sistema cuya impedancia de entrada sea alta para no modificar las condiciones del sistema bajo prueba (DUT).
El circuito de entrada está compuesto por un cable conductor de conexión al sistema bajo prueba o también llamado punta de prueba, donde primeramente ingresa la señal y luego en segundo lugar, un circuito atenuador y adaptador de impedancias, con una red de protección contra sobre tensiones.
Dada la característica de máxima señal de entrada, la punta de prueba junto con el atenuador de entrada localizado en la placa principal, dan una atenuación de señal de 100 veces (X100).



El divisor resistivo R9, R10, R11 y R12 y los capacitores trimmer de compensación C5 y C6 aseguran una reducción de la amplitud de la señal.
El circuito de protección actúa cuando la señal supera los 1,9 Volts de amplitud. Esto se logra al polarizase el diodo D3 en directa luego de sobrepasarse esa tensión. Los diodos D4 y D5 son conectados en directa a través de R14 logrando una V_γ total de 1,3 Volts, obteniendo así el funcionamiento equivalente de un diodo zenner de esa característica.

5.2-Primer amplificador operacional:

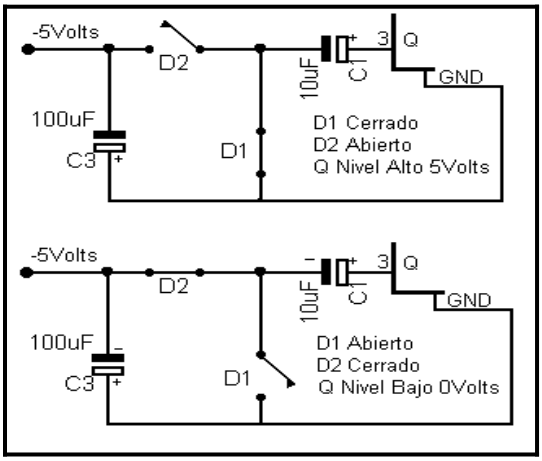
La segunda etapa del circuito de entrada esta compuesta por un circuito integrado, el TL082 es un amplificador operacional de alta impedancia de entrada, constituido internamente por un par de transistores JFET J1 y J2 conectados en configuración par diferencial que garantizan la alta impedancia de entrada, bajo consumo y rápido flanco de subida.
Este circuito integrado requiere fuente de alimentación partida, es decir con $\pm$Vcc.
Dado que nuestra placa es alimentada de una fuente simple de +5Volts y una corriente máxima de 500mA, por medio de un circuito enclavador recreamos la tensión negativa de requerida como muestra el siguiente circuito.



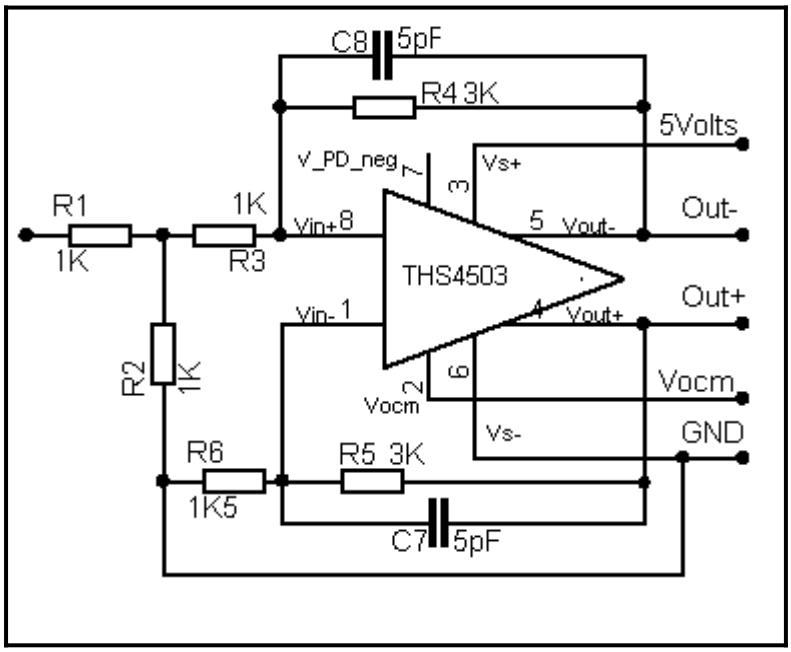
Al ser el circuito integrado TL082 de bajo consumo, la tensión pico a pico es mantenida constante y estable en 7Volts logrando la polarización correcta de los JFETs y TBJ internos al TL082.
Como solo adaptamos impedancias la configuración circuital es seguidor común, obteniendo el máximo ancho de banda del dispositivo y además dándole una de las características más importantes al sistema de adquisición en cuestión.

5.3-Circuito enclavador como fuente complemento:

El circuito enclavador esta constituido por principalmente por el CI LM555 y su conexionado como oscilador.



La frecuencia de trabajo es de 13,5 Khz dada por las resistencias R7, R8 y el capacitor C2, Rx da el tiempo de carga o estado en alto de la salida (pin 3).
El mecanismo de obtención de -Vcc es por medio de la carga de C2 y el cambio de nivel dado por el CI en el pin 3.
Una vez adaptada la impedancia de entrada al sistema, el circuito de carga del TL082 es una red resistiva formada por R1, R2 y R3 del circuito de más abajo. Esta red es el circuito de entrada al segundo amplificador operacional THS4503.
Este dispositivo THS4503 es el encargado de suministrar la señal en forma diferencial al conversor análogo digital ADS809Y.

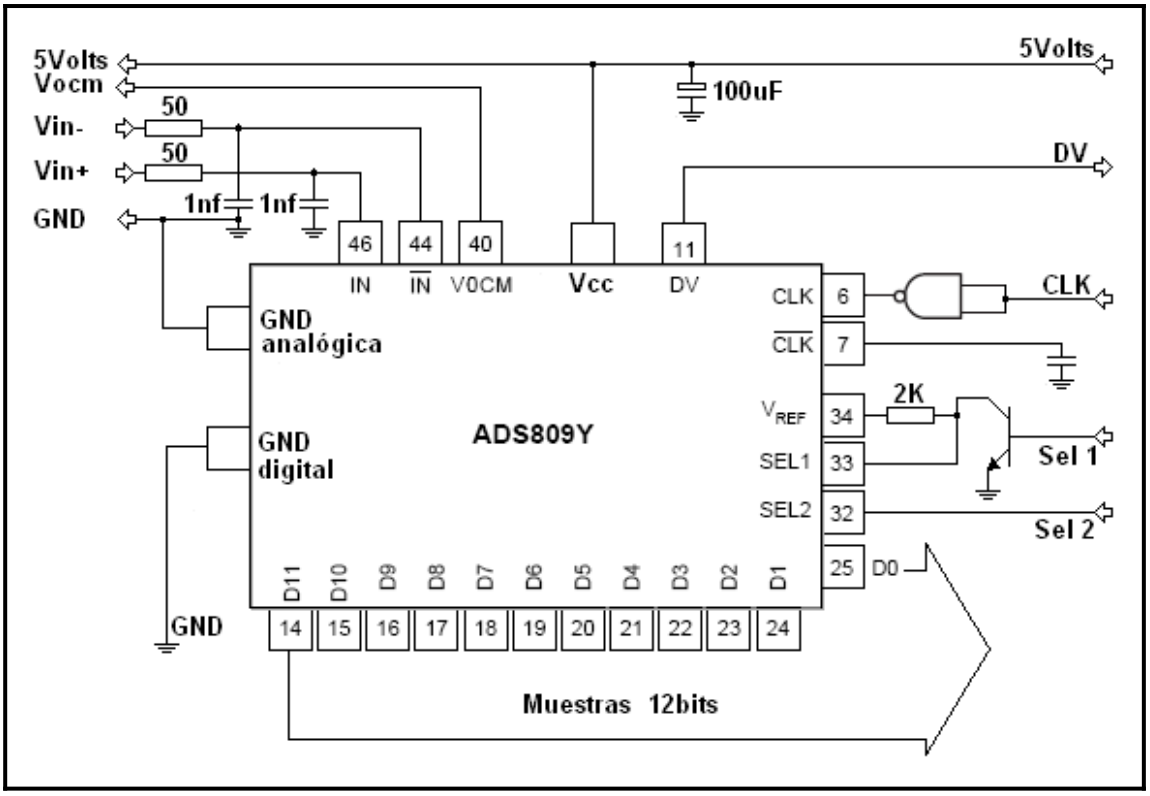


La ganancia de este circuito es G=2 y esta dada por R4=R5=Rf y R3 más R2 en paralelo con R1. Además existe un polo dominante que limita el ancho de banda y el nivel de ruido de la señal al ingresar al ADC.
Este polo esta ubicado a los 10Mhz y esta dado por Rf y Cf.

$$f = \frac{1}{2*\pi*Rf*Cf} = \frac{1}{2*\pi*R_{4,5}*C_{7,8}} = \frac{1}{2*\pi*3K\Omega*5pF} = 10{,}6103Mhz$$

5.4-Conversor ADS809

El conversor análogo-digital es el ADS809Y, es un conversor de 12bits y puede alcanzar una velocidad de 80 millones de muestras por segundo.
El diagrama esquemático parcial de esta etapa del proyecto es la siguiente:



Se puede observar que la señal proveniente del amplificador operacional THS4503, ingresa al conversor previamente pasando por un filtro pasabajos, cuya finalidad es definir el corte en frecuencia y limitar el nivel de ruido. El capacitor en la alimentación es para limitar el ruido de conmutación hacia la alimentación.

Entre las diferentes topologías circuitales y formas de funcionamiento, se configuro el chip de la siguiente manera:

Pin	I/O	Designación	Descripción	Nota
26	Input	VDRV	Tensión de alimentación del driver de salida de datos	Es conectada a 5Volts concordante con la alimentación
28	Input	OE-NEG	Habilitación de salida de datos	Conectada a 0Volts. Habilitado
29	Input	PD	Alimentación en reposo. Power Down	Conectada a 0Volts. Funcionamiento normal
30	Input	BTC/SOB	Datos en complemento a dos o 12 bits	Datos de 12bits conversión normal, conectado a 0volts

La referencia del rango de escala del ADS809Y esta dada por:

Rango de escala	**Pin SEL1**	**Pin SEL2**	**Vref Interno**
1Vp-p	Vref	GND	0.5Volts
1.5Vp-p	GND	5Volts	0.75Volts
2Vp-p	GND	GND	1.0 Volts

Con el control de estos 3 pines podemos obtener diferentes rangos de escala en la digitalización de las muestras, logrando así un cambio en la resolución dependiendo de la amplitud de la señal.
Si “N” es el numero de bits, la resolución del conversor estará dada por:

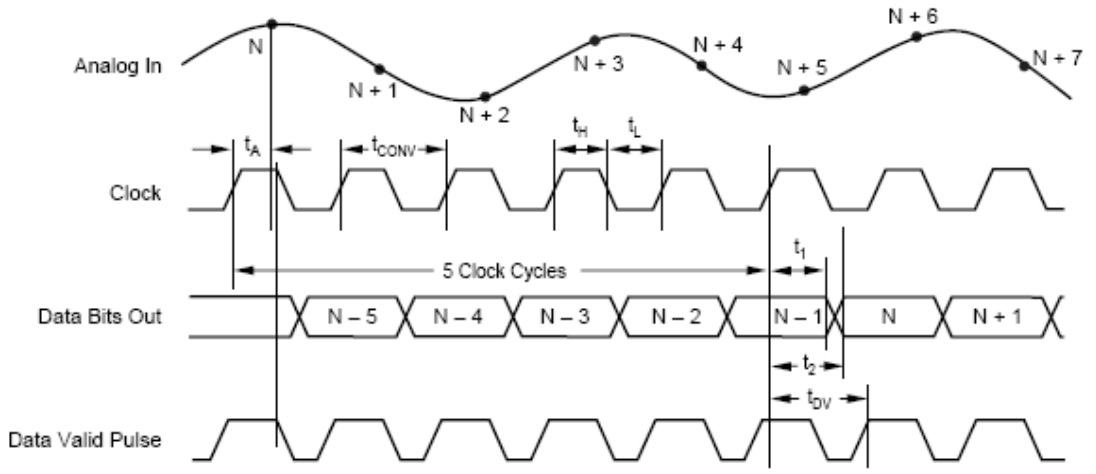
$$resolucion = \frac{Vref}{2^N} = \frac{Vref}{4096}$$

En la siguiente tabla damos algunas opciones en la elección del rango de escala:

Tensión de Entrada	SEL1	SEL2	Nota
Vin<100Volts	0Volts	0Volts	Valores de tensión en las entradas en el esquema propuesto
100Volts<Vin<150Volts	5Volts	5Volts	
Vin<200Volts	5Volts	0Volts	

El tiempo de conversión es de cinco ciclos de reloj, es decir que la primer muestra va a estar disponible en el bus de datos luego de ese tiempo.

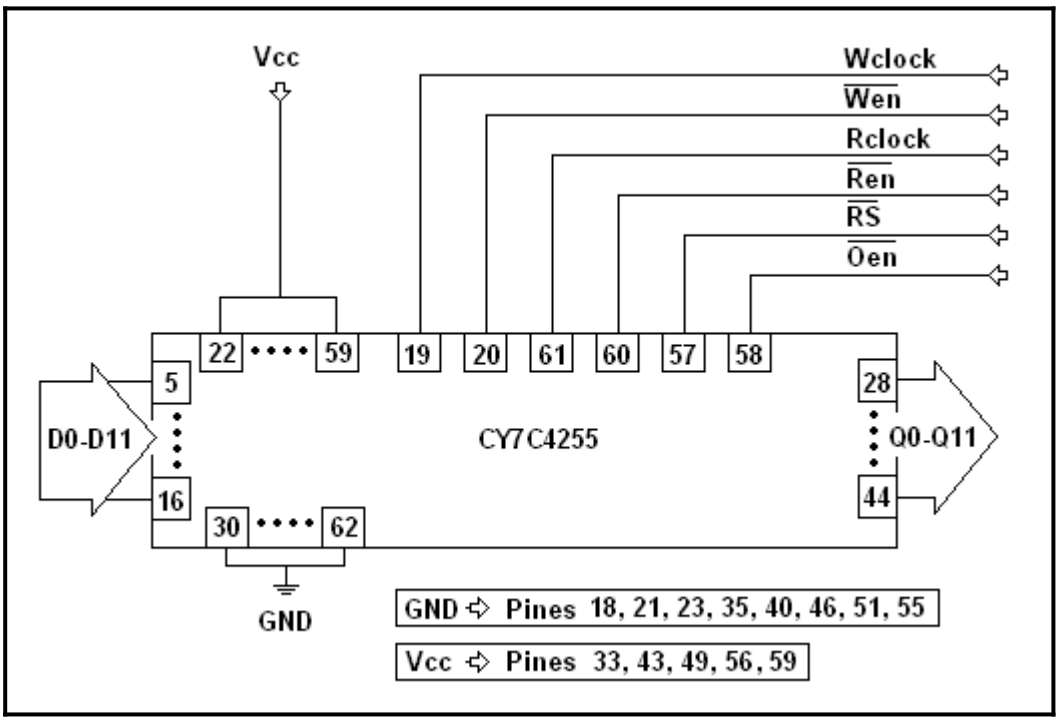
El siguiente diagrama muestra los tiempos necesarios para la adquisición:



SYMBOL	DESCRIPTION	MIN[1]	TYP	MAX[1]	UNITS
t_{CONV}	Convert Clock Period	12.5		1µs	ns
t_H	Clock Pulse HIGH	6.2	$t_{CONV}/2$		ns
t_L	Clock Pulse LOW	6.2	$t_{CONV}/2$		ns
t_A	Aperture Delay		4.6	6.1	ns
t_{DV}	Data Valid Pulse Delay[2]		10	12	ns
t_1	Data Hold Time, C_L = 0pF	4	5.8		ns
t_2	New Data Delay Time, C_L = 15pF max		9	11	ns

5.5-Memoria FIFO

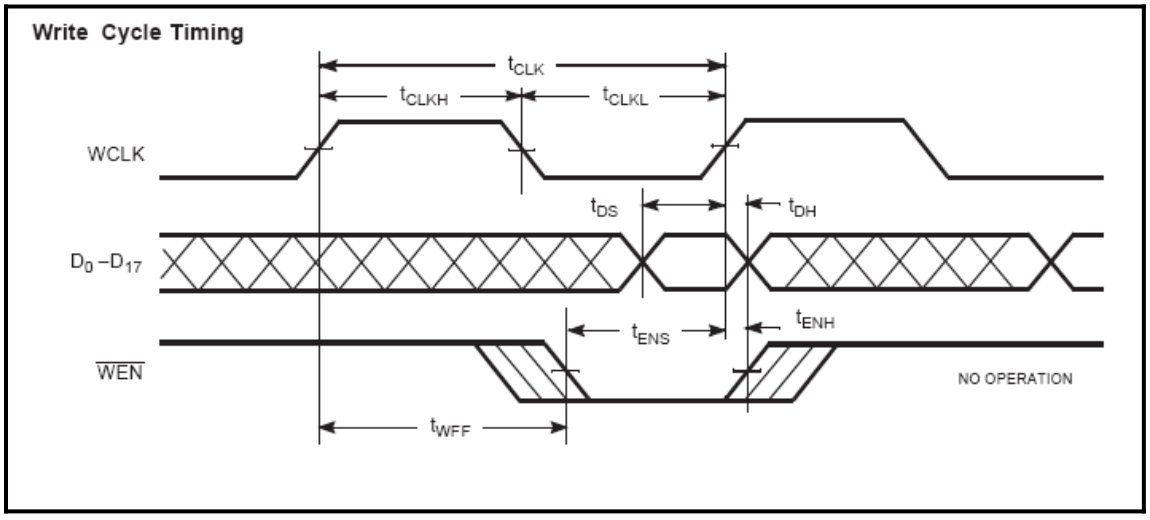
La CY7C4255 es una memoria FIFO de alta velocidad y bajo consumo que es escrita y leída en cada ciclo de reloj.
El esquema de conexiones del dispositivo es el siguiente



La configuración de funcionamiento que le hemos dado al chip es “Single Mode”. Es decir que es controlado y sincronizado por un control central. Este tipo de funcionamiento es de control muy ágil y es suficiente para la aplicación dada. En caso de un expansión del banco de memoria, se deberá hacer una interconexión tipo cascada entre chips.
Entre los modos de operación posibles, el proceso de escritura y lectura de los datos es realizado en etapas independientes.
Eléctricamente el bus de entrada siempre presenta alta impedancia de entrada.
Para la escritura de datos se requiere fundamentalmente que:

- Se encuentre habilitada la escritura $\overline{Wen}$ en estado bajo.
- Los datos estén presentes en el bus de entrada de los mismos.
- El flanco de subida de $Wclk$ se encuentre entre t_{ENS} y t_{ENH} .

El fabricante de este dispositivo da el siguiente diagrama de tiempos:



El tiempo de acceso de datos es una de las características importantes a tomar en cuenta a la hora del diseño.
Los tiempos citados en el diagrama pueden obtenerse de la hoja de datos del dispositivo http://www.digikey.com/.
Los datos convertidos por el ADC están presentes en el bus de datos en cada periodo de reloj. Luego que es escrito un dato en la posición “0” en el próximo flanco de reloj se incrementa la posición de memoria, dejando la primer posición “0” libre para el nuevo dato y trasladando el primer dato a la segunda posición, lo mismo sucede si hay datos ya existentes en posiciones siguientes; es decir que son trasladados desde la entrada en la posición “0” hasta la salida en la posición 4095 de este dispositivo.
Si los datos no son recolectados o leídos a la salida no se podrán recuperar.
Este proceso de almacenado de datos es realizado con todas las posiciones de memoria del dispositivo y es el principio de funcionamiento de las memorias FIFO.

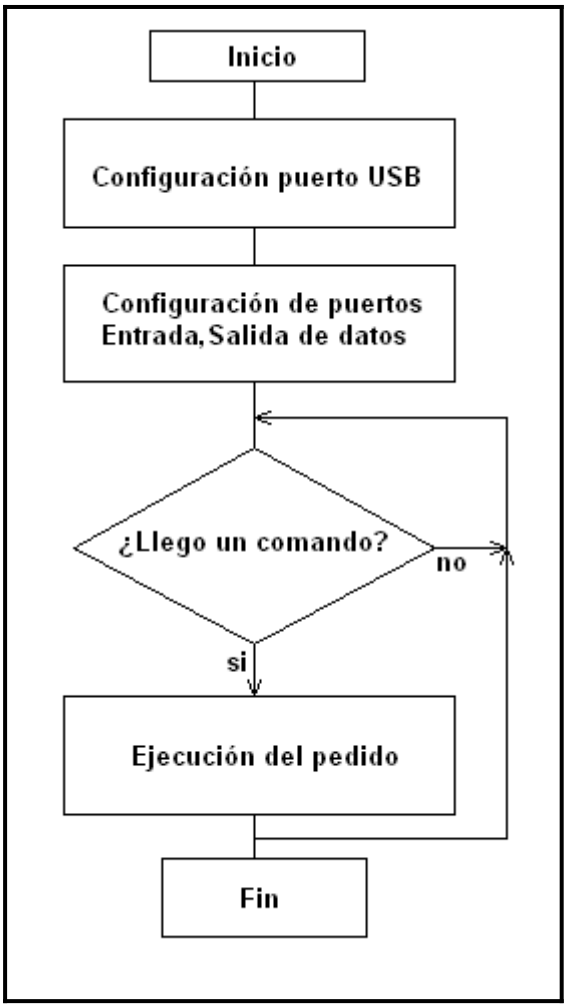
En el proceso de lectura, se utiliza un control independiente de habilitación e incremento de posiciones. Fundamentalmente se requiere que:

- Se encuentre habilitada la lectura $\overline{\mathrm{Re}\,n}$ en estado bajo.
- El flanco de subida de $Rclk$ se encuentre entre t_{ENS} y t_{ENH} .
- El Pin $\overline{Oen}$ este en estado bajo.

Los tiempos y momentos de lectura requeridos distan muy poco de los de escritura y son citados en la hoja de datos correspondiente.
Luego de leer los 4096 datos, los próximos datos son los llamados datos basura que son obviamente producidos por el mismo sistema.

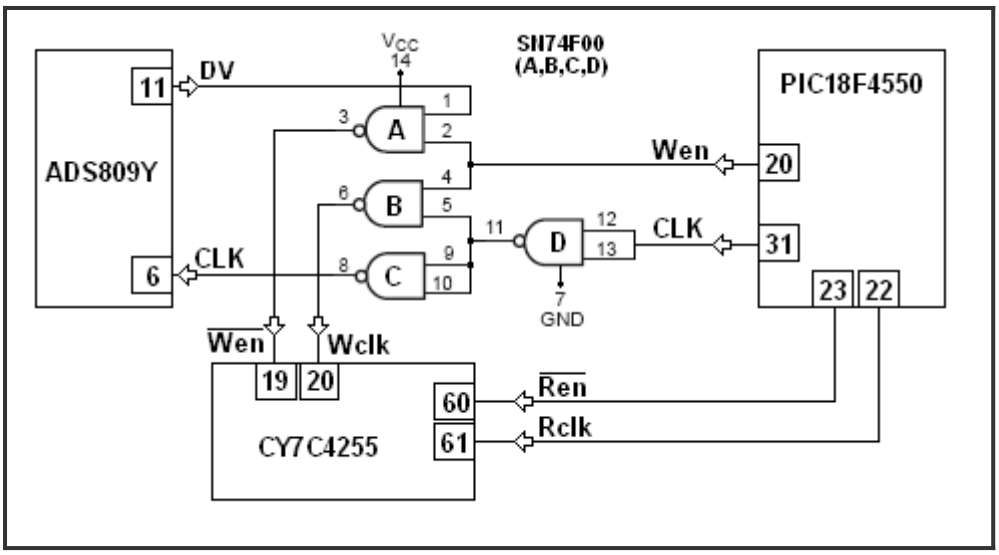
5.6-Control del sistema

El mecanismo de control de flujo de datos y control de todo el proceso lo lleva a cabo el microcontrolador PIC18F4550 y un pequeño circuito asociado.
Para desarrollar y poder entender todas las funciones, primeramente nos ubicaremos a partir del diagrama en bloques visto, en un diagrama de flujo.
El diagrama de flujo muestra:



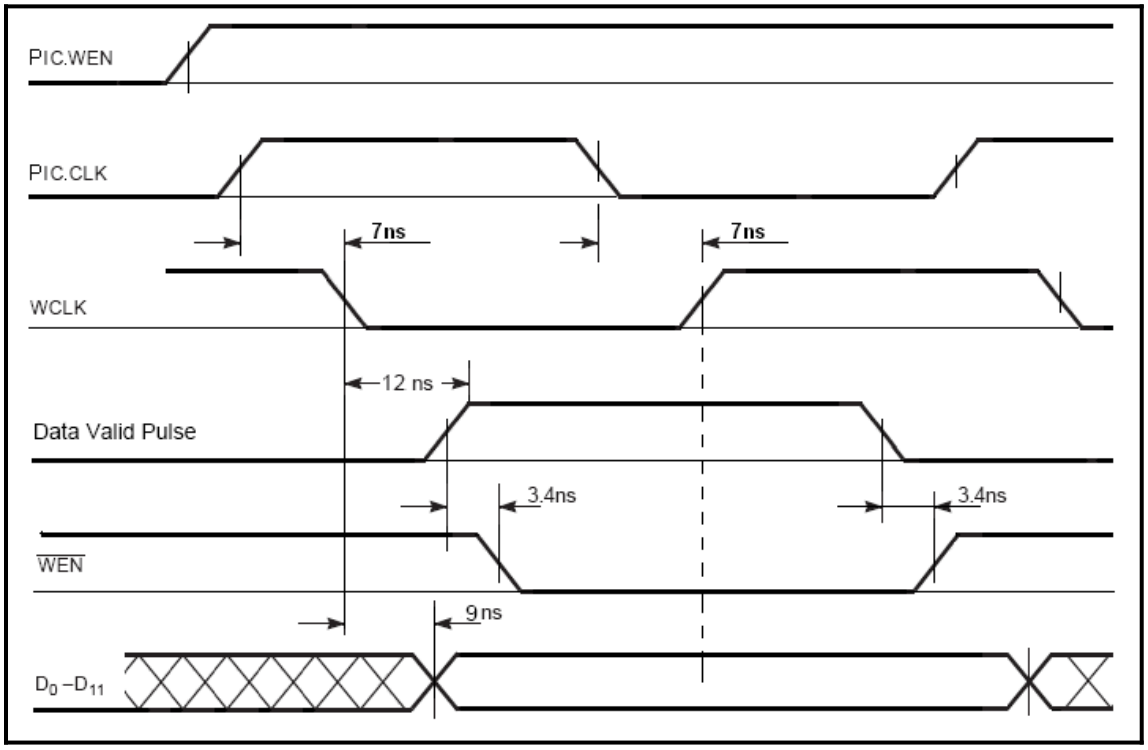
En la configuración del puerto USB, el microcontrolador establece la velocidad, el espacio de memoria requerido para los datos a ser recibidos y enviados, y un paso previo de encuesta de sistema conectado al PC.
Para la configuración de los puertos de entrada de datos y salida de control, se realizo tomando en cuenta la posición y disposición de los pines del chip.
Los datos muestreados ingresan por puertos RB0-RB7 para la parte menos significativa y RD0-RD3 para la parte más significativa.
Para el bus de control, utilizamos los pines PA0-PA4 el puerto A, entre ellos están WEN, RS, OE, REN y RCLK.

Para el proceso de escritura de la memoria FIFO a alta velocidad, se emplea un circuito adicional que reúne las señales provenientes del conversor y del PIC18F4550.
El circuito se muestra a continuación:



Se emplea el circuito integrado SN74F00 de Texas Instruments, este dispositivo esta compuesto por cuatro compuertas NAND de bajo tiempo de propagación del nivel lógico cercano a los 3,4 nanosegundos.
La compuerta “D” del diagrama superior, cumple la función de conformar el ciclo de reloj proveniente del cristal usado por el microcontrolador. Esta señal debe ser conformada dado que el cristal oscila con una señal senoidal y nosotros requerimos que sea de forma cuadra o que se aproxime mucho a esa forma, con flancos de subida y bajada bien definidos.
Las compuertas “A” y ”B” son requeridas para darle el funcionamiento lógico a lo pretendido.
Con una señal de nivel alto en el Pin 20 del microcontrolador habilitamos las compuertas “A” y “B”, por lo tanto un nivel alto de tensión en el Pin 1 del SN74F00 proveniente del Pin 11 DV del conversor, habilitamos la escritura de datos. Además con los pulsos conformados que ingresan al Pin 5 del SN74F00 obtenemos los pulsos de reloj necesarios para que la memoria FIFO grabe el dato en el registro de entrada y desplace los siguientes datos a las siguientes posiciones.
Por último la compuerta “C” da el retardo de tiempo necesario para la sincronización de los estados lógicos.

En el siguiente diagrama de tiempos, se puede visualizar los periodos de cada uno de los estados lógicos dados para el control de escritura.
Se pueden apreciar también, los tiempos de retardo transcurridos luego de un cambio de estado lógico de alto a bajo y viceversa.
Mas detalles de los tiempos del CI ADS809 pueden encontrarse en http://www.TI.com/



El proceso de lectura de datos se logra de una manera más sencilla.
Una vez grabados los datos en la memoria FIFO, se deshabilita con nivel bajo el Pin 20 “Wen” del PIC y luego se procede a una rutina donde una secuencia de estados lógicos en el bus de control llevan a cabo dicho proceso.
Un ciclo de la rutina de lectura se describirá en los siguientes pasos a continuación:

Pin A4 Rclk	Pin A3 Ren	Pin A2 OE	Pin A1 RS	Pin A0 Wen	Notas
1	1	1	1	1	Deshabilitados los procesos de lectura y escritura
1	1	0	1	1	Habilito las salida de datos.
1	0	0	1	1	Habilito la lectura de datos.
0	0	0	1	1	Bajo el nivel del Rclk (flanco descendente).
1	0	0	1	1	Subo el nivel de Rclk.

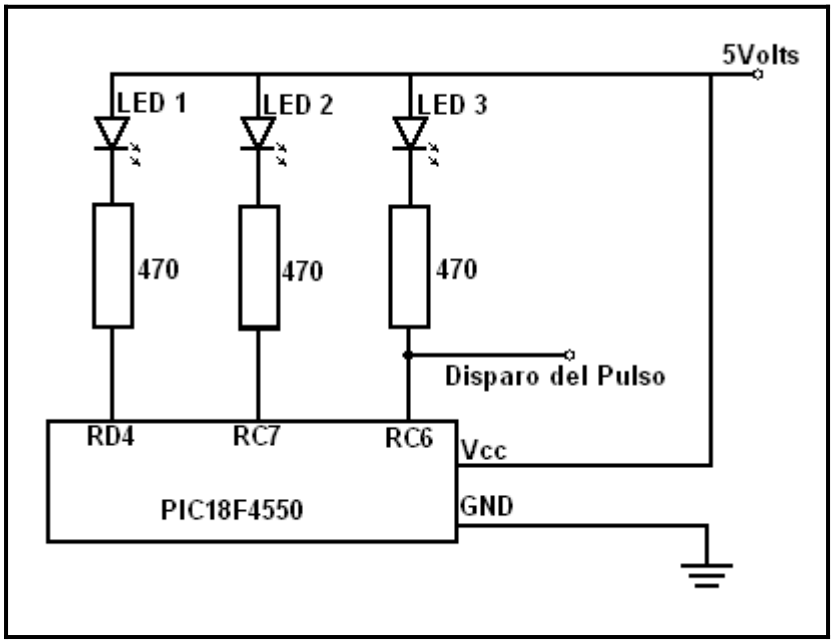
Volviendo al diagrama de flujo, entrando en un ciclo infinito donde se evalúa la entrada de un comando proveniente del CPU, podemos ejecutar la operación deseada.
Los comandos están definidos en un protocolo.
El paquete transmitido por el CPU es un vector de 20 Bytes que son almacenados en un lugar reservado de la memoria del microcontrolador. A los fines prácticos solo son usados 9 Bytes.
El primer Bytes de ese vector define la operación a realizar y el resto de los Bytes definen los parámetros de cada operación y el control de los posibles errores producidos en la comunicación.

En el pedidos de los datos capturados, se realizan los siguientes pasos:

- Se dispara la fuente y se espera 74 milisegundos para empezar a adquirir.
- Se habilita la escritura de la memoria.
- Al llenar la memoria, se deshabilita la escritura.
- Se leen 100 datos de 12 Bits y se envían 200 Bytes de 8 Bits.
- El CPU envía la petición de los próximos 100 datos y en bucles reiterados los siguientes.
- Culminada la lectura de los 4000 datos queda el microprocesador en espera.

El panel indicador es un simple circuito con 3 leds que indican lo siguiente:

- Encendido. Power on. (LED 1)
- Enlace por USB. (LED 2)
- Señal de disparo. (LED 3)



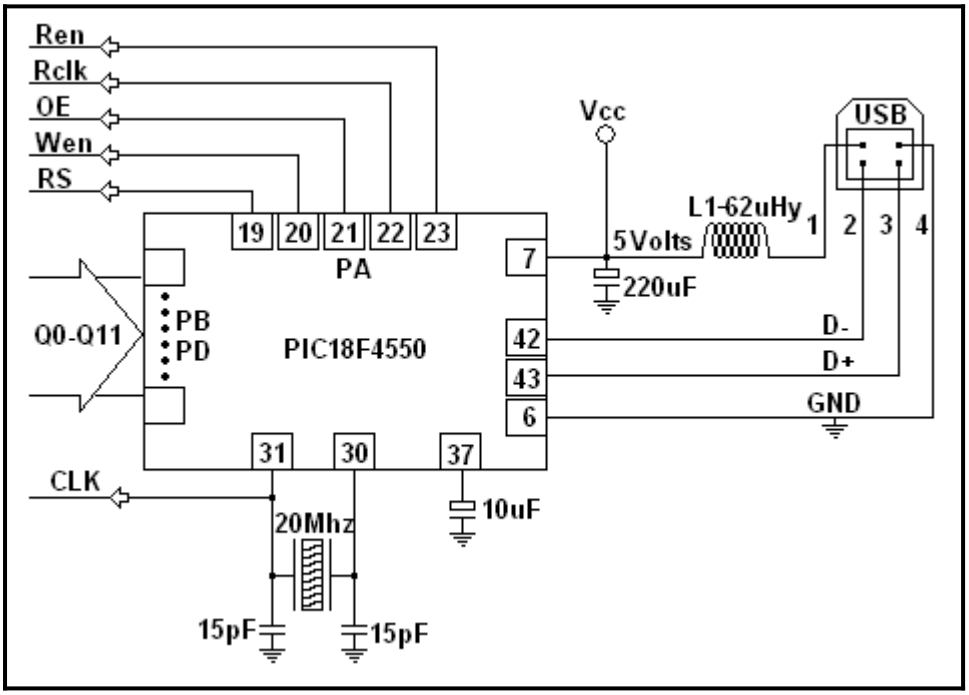
La señal de disparo es conectada a la fuente generadora del pulso fusor.
Dicha fuente tiene un conector plug que es utilizado para comando a distancia, que con solo bajar el nivel de tensión a 0Volts aprox. por un lapso de 2 milisegundos, puede ser disparada con toda facilidad.

5.6.1-Alimentación del sistema

La fuente de energía es suministrada por el CPU a través del puerto USB.
Este puerto entrega una tensión continua de 5Volts y una corriente máxima de 500mAmpers, suficiente para el propósito de este proyecto.
El diseño fue desarrollado en función de lograr un bajo consumo total, disminuyendo el consumo particular de determinadas secciones del prototipo.
Dividiendo el esquema electrónico en bloque específicos, determinamos el consumo general de todo el prototipo como muestra el cuadro siguiente:

Sección del esquema	Consumo de corriente
Circuito de entrada	21mA
Circuito conversor	133mA
Memoria	45ma
Procesador	29mA
Consumo total de todo el sistema	**228mA.**

El siguiente esquema muestra el circuito de alimentación.



Los pines 37, 42 y 43 corresponden al puerto USB, dicho puerto suministra como ya comentamos la alimentación general del dispositivo y nos suministra de un sistema de comunicación confiable y de alta tasa de transferencia de datos (12Mbps).
El Pin 37 es conectado a un capacitor electrolítico para estabilizar la tensión de referencia en la transacción de datos.
El Pin 1 del conector USB, es conectado a una bobina de 62uHy y un capacitor electrolítico de 220uF para filtrar el posible ruido existente en la línea de transmisión.

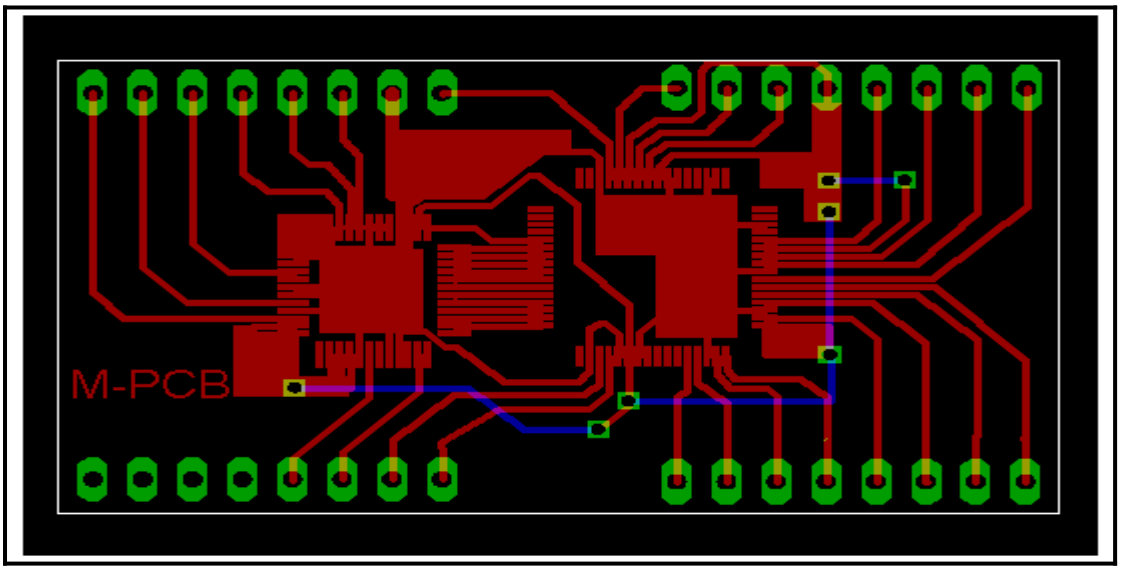
El oscilador del microcontrolador esta constituido por un cristal de 20Mhz y un par de condensadores cerámicos de 15pF; de aquí se obtiene la señal de reloj para el resto del sistema para ser conformada.

5.7-Plaqueta de circuito impreso

El diseño del circuito impreso es un punto crucial a la hora del desarrollo de un sistema que trabaja a altas velocidades de transferencia de datos. Las emisiones por radiación son elevadas y las capacidades distribuidas en las pistas del circuito comienzan a ser relevantes.
Para ello se tuvo en cuenta:

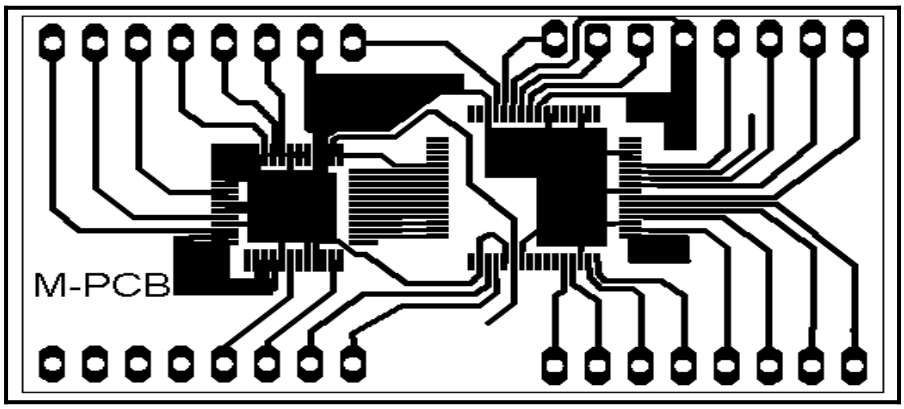
- Método de diseño de circuitos impresos.
- La distancia entre dispositivos.
- El tipo de material a emplear.
- El espesor de la lamina de cobre sobre el pertinax o placa de epoxi.
- también el costo por centímetro cuadrado de plaqueta.

Los circuitos integrados SMD de montaje superficial empleados, como el conversor analógico-digital y la memoria FIFO, son de dimensiones notablemente reducidas; podemos observar que la distancia física entre pines es de 200 micras, es decir de 0.2 milímetros de separación y el espesor de cada Pin es de 0.3 milímetros.
Estos dispositivos mencionados son una parte crítica del desarrollo y para ello se diseño una plaqueta de dimensiones reducidas, donde la distancia entre los buses de salida del conversor y de entrada de la memoria son mínimas para evitar la capacidad distribuida de los conductores y las emisiones al resto del circuito.

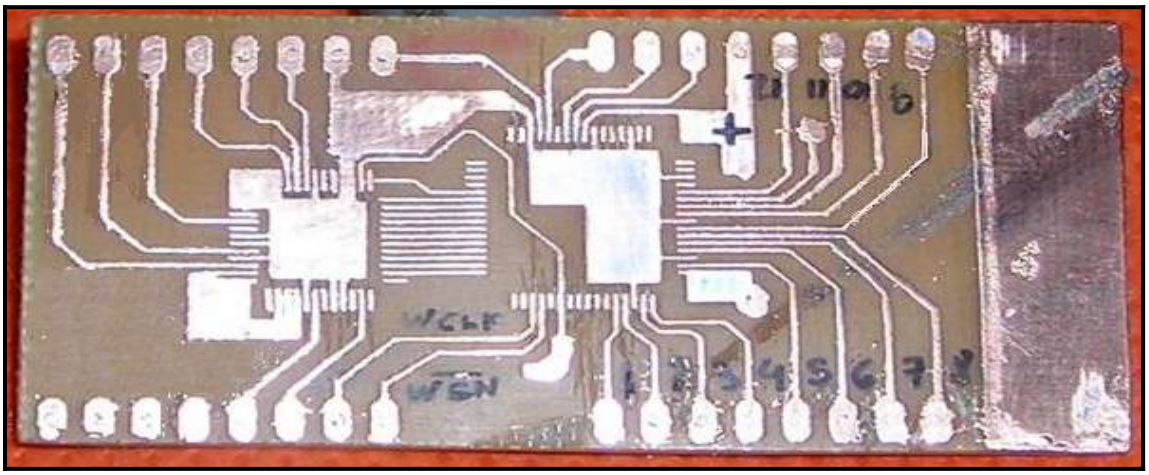


En la imagen de arriba se puede observar el diseño del PCB realizada con el programa Eagle. Es el esquema circuital de la placa que contiene al conversor ADC y la memoria FIFO. Es un imagen ampliada al 350% del tamaño real, dado que por el tamaño reducido de las pistas y la resolución de un monitor

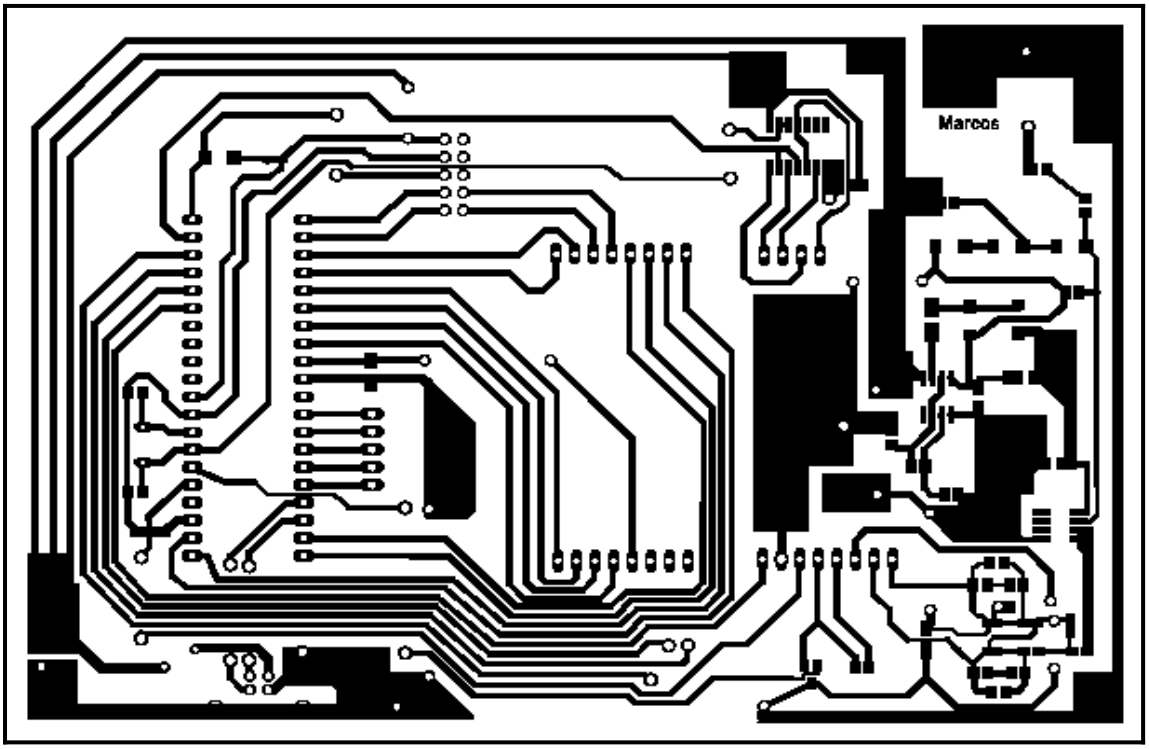
estándar no se logra alcanzar a visualizar con exactitud donde cada Pin debe ser soldado.
Insolación es el método empleado para la construcción de la plaqueta.
En pocas palabras vamos a describir éste método.
Primeramente realizamos el circuito esquemático en un programa de diseño como en nuestro caso Eagle. En dicho programa además del esquemático, se pueden editar cada uno de los componentes y se diseña también el PCB correspondiente.
Una vez obtenido el archivo de extensión .bmp con el diagrama del circuito deseado, se imprime el positivo o el negativo del circuito, como muestra la imagen de abajo. Se imprime sobre una filmina con una impresora o fotocopiadora láser que nos brinda una alta resolución de 600dpi o superior.
En la figura de abajo se puede observar la imagen positiva del circuito

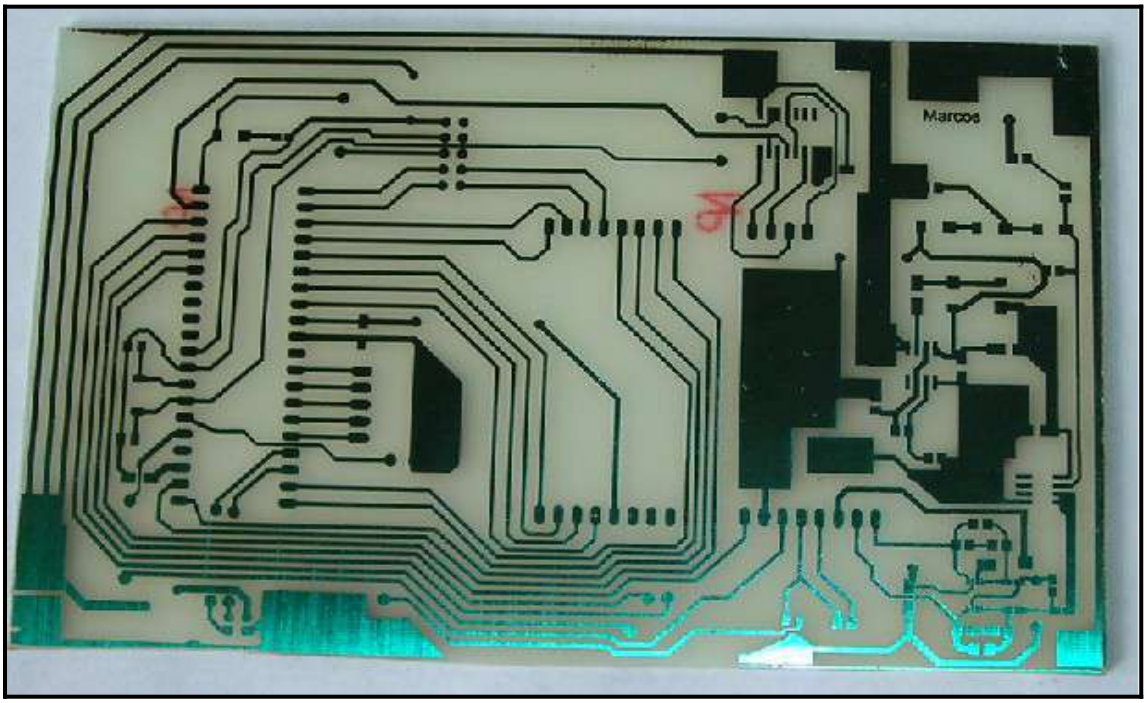


La filmina se apoya sobre la placa virgen, se adhiere con cintas adhesivas y luego se somete a la exposición de una fuente de luz ultravioleta. La placa de epoxi se adquiere ya tratada con un fotolito o se puede tratar en forma artesanal con un fluido fotosensible. Luego de un tiempo aproximado de 10min se retira la filmina y se sumerge la placa sobre una solución de agua y soda cáustica hasta que se descubra el circuito requerido. Una vez impreso el circuito sobre la placa de epoxi, se la introduce en una cuba con percloruro férrico, que reacciona sobre el cobre que se encuentra al descubierto.
El resultado se puede apreciar en las siguientes fotos.
En esta foto se ve la placa que alojara el ADC ADS809 y la memoria FIFO CY7C4425



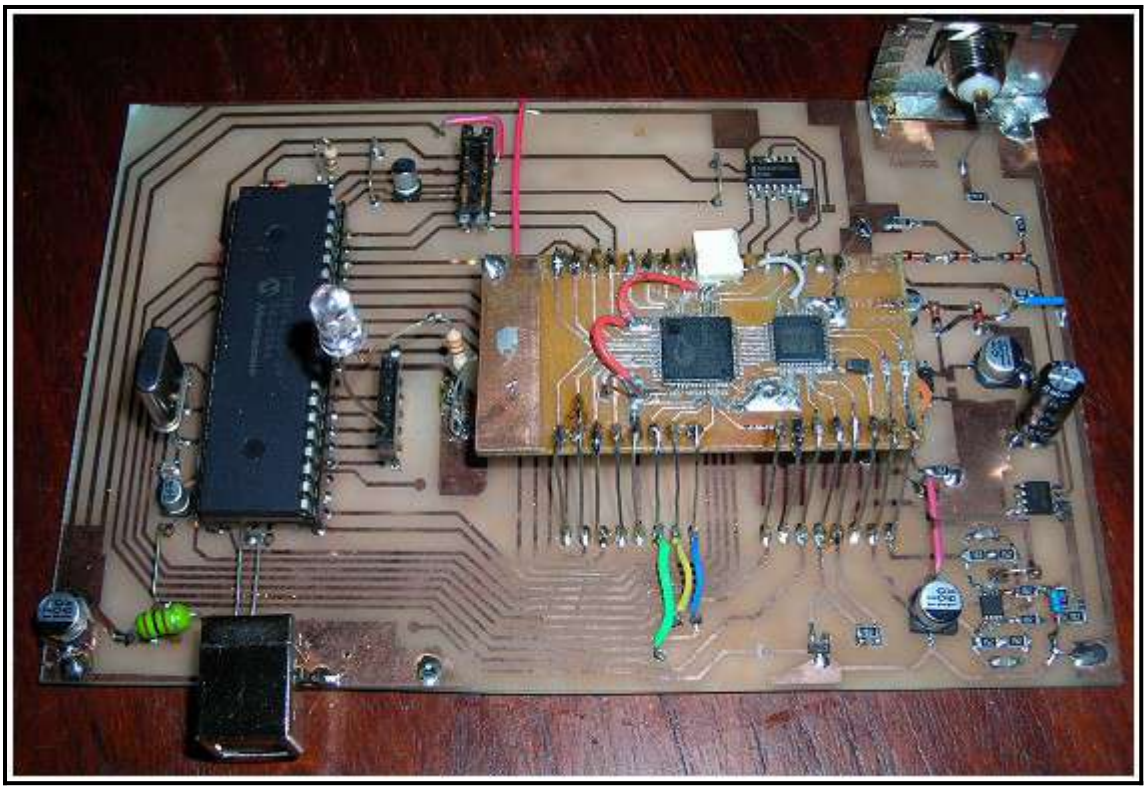
Aquí abajo se puede ver el segundo circuito ha ser impreso sobre la placa de fibra de epoxi





En la foto de arriba se puede apreciar la segunda placa resultante del proceso descrito, donde se van a soldar el microcontrolador PIC 18F4550 a la izquierda del diseño y el circuito de entrada con los amplificadores operacionales a la derecha de la misma.

5.7.1-Placa final



Se puede ver en la foto de arriba todo el circuito terminado, donde se encuentran sobre la placa principal:

- el puerto USB abajo a la izquierda y más arriba el microcontrolador.
- En la parte media central la segunda plaqueta con el ADC y la memoria FIFO.
- Abajo a la derecha el AOp. THS4503 y un poco más arriba el TL082.
- En el centro a la derecha, el circuito enclavador y el conector BNC de entrada.

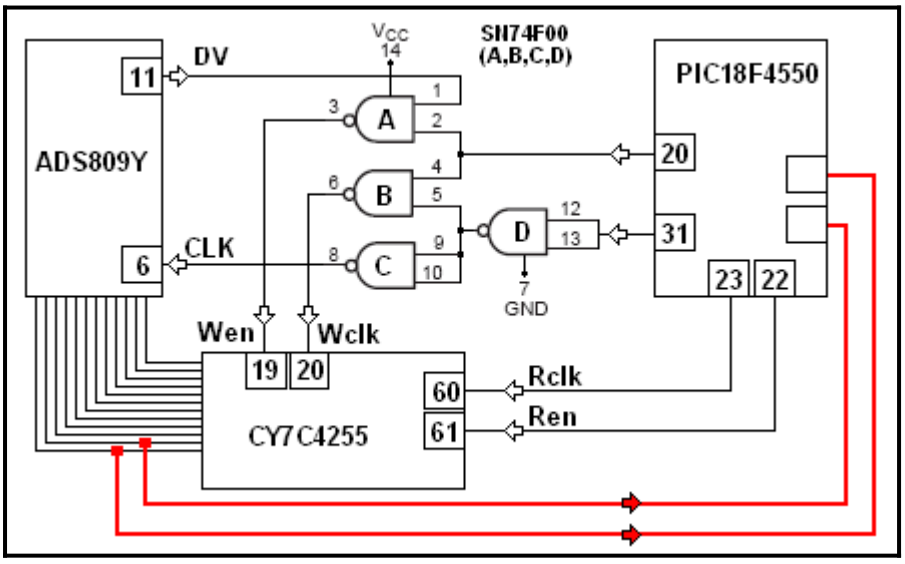
Comparando a simple vista el tamaño de los componentes estándar con el resto de los componentes SMD, podemos darnos una idea de lo dificultoso y laborioso que es trabajar con los mismos.

Las emisiones de tipo electromagnética se propagan a otros componentes cercanos y fundamentalmente por los conductores de alimentación de todo el sistema.
En el caso del conversor analógico-digital los pulsos provenientes del oscilador son irradiados por la alimentación; se pueden observar con un osciloscopio que a la tensión de alimentación se le suma un ruido coherente con los pulsos de reloj, dado el gran consumo de corriente requerida en cada conversión de datos. Además éste dispositivo es capas de disipar hasta 1Watts de potencia en un área de encapsulado de solo unos 5 milímetros cuadrados; para ello se le agrego al diseño, una superficie de cobre en la parte inferior del dispositivo, logrando por conducción la disipación del calor hacia la plaqueta.

Ambas plaquetas desarrolladas son aisladas de EEM (Emisiones Electromagnéticas) por medio de una base metálica de cobre. Se diseñaron los circuitos sobre placas doble faz, donde la placa inferior se conecta al potencial de masa. Dicho circuito de masa (GND) en lugares críticos, se produce una división entre masa analógica y digital, fundamental para evitar o disminuir problemas de ruido.

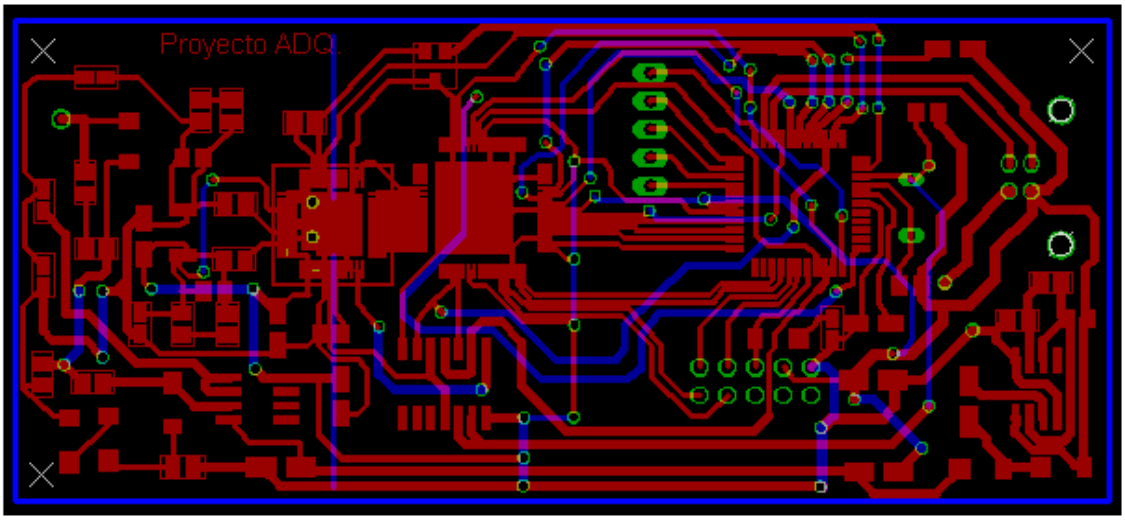
5.7.2-Problemas ha ser superados.

Los pulsos generados por la fuente USCF150B, son generados en diferentes tiempos, es decir se hacen presentes con un pequeño jitter que oscila en los 2milisegundos.
Esto acarrea el hecho que si se hace un retardo de 76 milisegundos para comenzar la adquisición justo cuando se genera el pulso, puede ocurrir que el pulso ya se produjo, comenzó y solo tomamos una porción de este o finalmente que comencemos a adquirir y el pulso no se produzca en el tiempo justo.
Para solucionar este problema se puede comenzar la adquisición a los 74 milisegundos y tomando como referente los bits 10 y 11 del conversor para comenzar una cuenta de 3500 pulsos. Esto es que si el pulso se produce en algún tiempo posterior, tenemos 500 muestras que anteceden al pulso y luego tendremos 3500 muestras ha ser registradas.
Estos dos bits son censados por el microcontrolador, que en una rutina de interrupción, cuenta los 3500 pulsos de reloj
El circuito final seria:



Los conductores color rojo representan la modificación futura.
Con ésta modificación del circuito y una pequeña modificación del firmware del PIC podemos superar el problema descripto.

Además todo el circuito puede ser construido sobre una plaqueta de dimensiones más reducidas. El siguiente diseño del PCB muestra las nuevas dimensiones de la placa de doble faz.

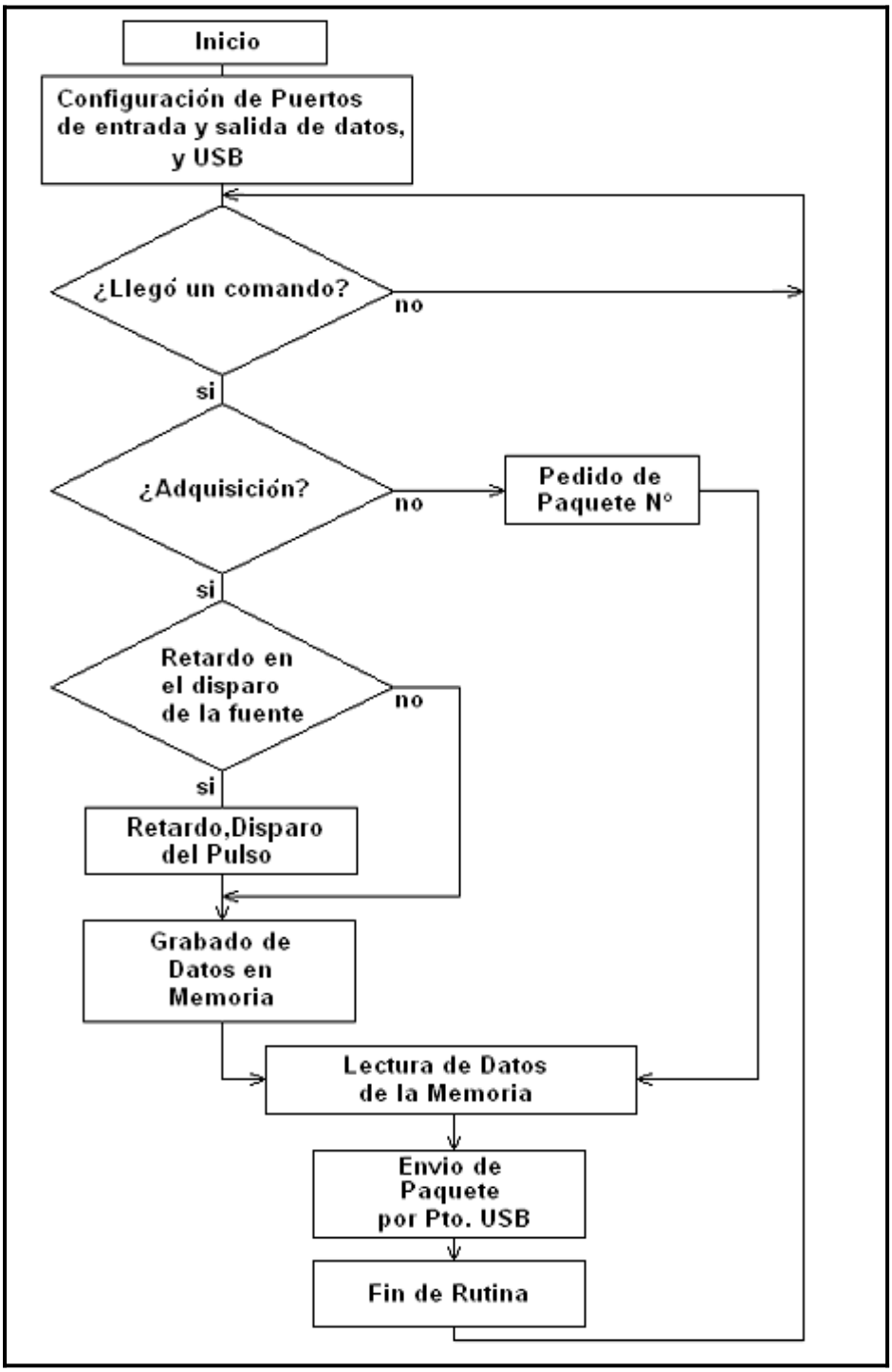


Aquí se deberán utilizar componentes SMD en su totalidad

5.8-Software

5.8.1-Desarrollo del software del microcontrolador.

La programación del software se realizo en lenguaje C. La rutina que ejecuta el programa es descripta en el diagrama de flujo:



El siguiente programa denota el diagrama de flujo anteriormente descrito.

Programa del Microcontrolador

```
#include <18F4550.h>
```

```
#fuses
HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL1,CPUDIV1,V
REGEN
#use delay(clock=4000000)
////////////////////////////////////////////////////////////////////////
//
// CCS Library dynamic defines.  For dynamic configuration of the CCS Library
// for your application several defines need to be made.  See the comments
// at usb.h for more information
//
////////////////////////////////////////////////////////////////////////
#define USB_HID_DEVICE     FALSE            //deshabilitamos el uso de
                                  //las directivas HID
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_TX_SIZE    210              //size to allocate for the tx
#define USB_EP1_RX_SIZE    20               //size to allocate for the rx

////////////////////////////////////////////////////////////////////////
//
// Include the CCS USB Libraries.  See the comments at the top of these
// files for more information
//
////////////////////////////////////////////////////////////////////////

#include <pic18_usb.h>     //Microchip PIC18Fxx5x Hardware layer for CCS's
                           //PIC USB driver
#include <PicUSB.h>        //Configuración del USB y los descriptores para
                           //este dispositivo
#include <string.h>
#include <math.h>          //Funciones Matemáticas Trigonometricas.
#include <STDLIB.H>
#include <usb.c>

////////////////////////////////////////////////////////////////////////
#define LED_ON  output_high
#define LED_OFF output_low

////////////////// Configuración de pines del bus de control


#define WEN     PIN_A0     /// Pin conectado al 74f00
#define RS      PIN_A1      /// Cable Blanco
#define OE      PIN_A2      /// Cable Negro
#define REN     PIN_A3      /// Cable Marrón
#define RCLK    PIN_A4     /// Cable Rojo
#define DISP    PIN_C6      /// Cable Rojo


#define comando   recibe[0]
```

```
#define param1     recibe[1]    /// Adquirir  ().
#define param2     recibe[2]
#define param3     recibe[3]
#define param4     recibe[4]    /// Nº de paquete pedido.
#define param5     recibe[5]    /// Cantidad de paquetes.
#define param6     recibe[6]    /// Retardo en milisegundos o microsegundos
                                /// 1=milisegundos ; 2=microsegundos.
#define param7    recibe[7]     /// Retardo del disparo.
#define param8    recibe[8]    /// Errores de transmisión. ???? ===> va también
                                ///en el paquete de transmisión

#int_ccp2
void isr()
{
 disable_interrupts(INT_CCP2);       ///  Deshabilito la interrupción por
                                     ///flanco ascendente ///
 enable_interrupts(INT_TIMER1);      ///  habilito la interrupción por
                                     ///cuenta de pulsos  ///

}

#int_TIMER1
TIMER1_isr()
{

 output_A(0b11111110);               // Termino la escritura de la FIFO es decir
                                     //( deshabilitar la FIFO).
 disable_interrupts(INT_TIMER1);

}
void main(void)

{

   int8 recibe[20];             //declaramos variables
   int8 envía[210];
   int8 y;
   int8 t;
   int8 r;
   int8 x;
   int8 paquete;
   int8 error;
   int8 data;
   int16 i;
   float c;
   float argument;
   float n;
   /////// Configuración de puertos ///////

SET_TRIS_A(0b11100000);     //Configuro el puerto 'A' con las respectivas
```

```
                                    //entradas y salidas
SET_TRIS_B(0b11111111);     //Configuro el puerto 'B' como entradas
SET_TRIS_C(0b00001111);     //Configuro el puerto 'C' como salidas
SET_TRIS_D(0b11111111);     //Configuro el puerto 'D' como entradas la parte
baja
                                    // y salidas la parte alta

/////// Configuración de la interrupción
Setup_ccp2(CCP_CAPTURE_FE);
Setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_1 );

//////////////////////////////////////////////////////////////////////////

usb_init();                         //inicializamos el USB
usb_task();                          //habilita periférico usb e interrupciones
usb_wait_for_enumeration();     //esperamos hasta que el PicUSB  sea
LED_ON(DISP);                       //configurado por el host

//////////////////////////////////////////////////////////////////////////

i=0;
x=0;
c=0;
paquete=0;

//// Aquí configuramos el inicio del puerto de control del sistema   //////

output_A(0b11111110);
///   En alto deshabilito el ciclo de clock de LECTURA

while (TRUE)

  {
    if(usb_enumerated())     //si el PicUSB está configurado
    {
      if (usb_kbhit(1))   //si el endpoint de salida contiene datos del host
      {

        usb_get_packet(1, recibe, 9); //tomamos el paquete de
                    //tamaño 9_bytes del EP1 y almacenamos en recibe

        ////////////////////////////////////////////////////////////

        if (comando == 0) // Modo_Suma

        {
         envía[200]=0x01;

         usb_put_packet(1, envia,210, USB_DTS_TOGGLE)
 //enviamos el paquete de tamaño 1byte del EP1 al PC
```

```
        }

////////////////////////////////////////////////////////////////

         if (comando == 1) // Modo_Led
          {

           if (param1 == 0)  //// Apagamos los leds  //////

          {

           if ((param4 == 1) & (param8 ==0))
                ///// Primer paquete y no hay error //////////
          {

//////////////////////// Aca con "param6" entramos si hay retardo ////////////////

   if (param6 != 0)          //// Si hay retardo entra ////
     {

        if (param6==1)
          {

          delay_ms(param7);         // Retardo de X milisegundos  /////////
          output_C(0b01000001);  // Apago el disparo de la fuente de pulsos
          output_C(0b00000001);  // Disparo la fuente de pulsos
          delay_ms(500);
          output_C(0b01000001);  // Apago el disparo de la fuente de pulsos
          delay_ms(param7);         // Retardo de X milisegundos  /////////

          }
        else
          {

          }

        if (param6==2)
          {
          output_C(0b01000001);   // Apago el disparo de la fuente de pulsos
          output_C(0b00000001);   // Disparo la fuente de pulsos

          delay_ms(500);

          output_C(0b01000001);   // Apago el disparo de la fuente de pulsos
          delay_us(param7);          // Retardo de X microsegundos /////////

          }
     }
```

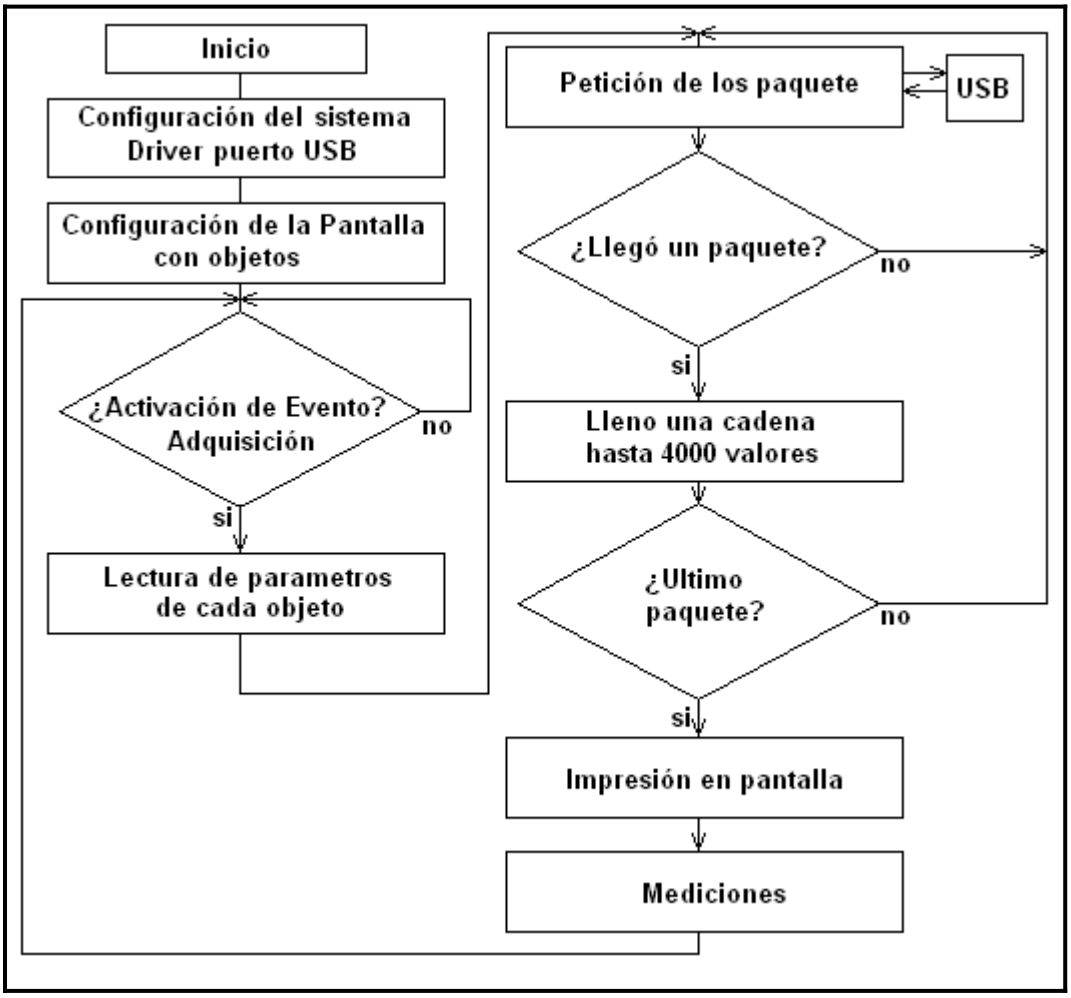
```
//////////////////////   Disparo la fuente de pulsos   ////////////////////

        else
          {
            LED_ON(DISP); // Apago el disparo de la fuente de pulsos
            LED_OFF(DISP); // Disparo la fuente de pulsos

            delay_ms(70);

          }

////////////////////////////////////////////////////////////////////////////////////////////

////////////////////// Aqui Escribo la FIFO con 4096 valores ////////////////////


        setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);

        set_timer1(64661);
        ////  Ejemplo del help -----> 256-(.000035/(4/20000000))

        output_A(0b11111110); ////  Reset   /////
        output_A(0b11111100);
        output_A(0b11111110);

        output_A(0b11111111); ////  Comienza la escritura de la FIFO ////

        enable_interrupts(INT_CCP2);////  Espero que haya un pulso de subida
        enable_interrupts(GLOBAL);  ////  Habilito las interrupciones

        delay_ms(20);  /// Retardo para que actue la interrupcion del TIMER_1
        disable_interrupts(INT_TIMER1);

        output_A(0b11111110);
        delay_ms(600);
        LED_ON(DISP); // Apago el disparo de la fuente de pulsos
          // all interrupts OFF

//////////////// ---Aqui comienza la  lectura de los puertos que leen la FIFO ///

//////////////// Con "parame8" entra si no hay error de paquete //////

    if (param8 != 1)
    {
     t=0;
     for (i=0;i<=99;++i)
     ////  Leo los 200 datos 100 parte alta y 100 parte baja  (HSB Y LSB)
     {
     output_A(0b11111110);     // Todas las salidas del bus de
                                  // control deshabilitadas
```

```
      output_A(0b11111010);     // Habilito las salidas de datos
      output_A(0b11110010);     // Habilito la lectura de datos
      output_A(0b11100010);     // Bajo el clock de lectura
      output_A(0b11110010);     // subo el clock de lectura
      envia[t]= input_b();           // Leo  el dato en el puerto parte Baja

      t=t+1;                        // Incremento la posicion del vector
                                     // Bajo el clock de lectura
                                     // subo el clock de lectura
      envia[t]= input_d();      // Leo  el dato en el puerto parte Alta
      envía[t]&=(0b00001111);
      t=t+1;                         // Incremento la posición del vector

      }  //// Fin de llenado del arreglo "ENVIA[200]"

      output_A(0b11111110);
                                      // Todas las salidas del bus de control
deshabilitadas

      paquete=paquete+1;     //  Cuento las entradas a la rutina  ////
      error=0;                      //  No hay error ////

      }

////////////  Aquí enviamos el paquete correspondiente a la entrada  //////////

        else
        {
          error=1;    //////Al salir del else coloca en envia[201] 0x01 /////
        }

        if (paquete==param5)
        {

        envía[200]=paquete;      //// Numero de Paquete enviado ////
        envía[201]=error;          //// Numero de Error generado  ////
        envía[202]=1;              //// Fin de la transacción     ////

        usb_put_packet(1, envia,210, USB_DTS_TOGGLE);

        ////enviamos el paquete de tamaño 1byte del EP1 al PC

        paquete=0;
        envía[202]=0;
        error=0;

        }


        else
```

```
        {
        envía[200]=paquete;      //// Numero de Paquete enviado ////
        envía[201]=error;        //// Numero de Error generado  ////
        envía[202]=0;            //// Fin de la transacción     ////

        usb_put_packet(1, envia,210, USB_DTS_TOGGLE);
        ////enviamos el paquete de tamaño 1byte del EP1 al PC
        }
        }
       }
       }
     }
   }
}
```

5.8.2-Desarrollo del software para el CPU.

En el diagrama de flujo siguiente podemos ver los pasos principales que desarrolla el software en el transcurso de compilación



Para la programación de la aplicación para PC se ha optado por usar el Visual C# Express 2005 y puede ser descargado gratuitamente de la página de Microsoft:
Tenemos tres etapas de diseño que incluye una parte de entorno visual y otras abocadas a la interpretación de cada comando activado y a los requerimientos específicos de toda la aplicación.
Ahora se denota la parte en la cual el programa hace uso de la clase "PICUSB", la cual es como un recipiente para todo el código de aplicación, donde se sitúan las variables de uso local y publicas, además de las funciones que son fracciones de código que pueden devolver o no al código que lo invoco en un principio.
También un controlador de evento que es un procedimiento en el código que determina las acciones que deben llevarse a cabo cuando ocurra un evento, como es el pulsado de algún botón.
También se pude ver el uso de librerías como son: System.Collections.Generic, System.ComponentModel, System.Data etc.

Programa

```
sing System;
using System.Collections.Generic;
using System.ComponentModel;
using System. Data;
using System. Drawing;
using System. Text;
using System.Windows.Forms;
using System.Diagnostics; //Clase para abrir página web
using PVOID = System.IntPtr;
using DWORD = System.UInt32;

namespace PicUSB
{

   public partial class PicUSB : Form

   {

      //public event EventHandler HScroll;
      //protected override CreateParams CreateParams { get; }
      public void AddMyScrollEventHandlers()
      {
         // Create and initialize a VScrollBar.
         VScrollBar vScrollBar1 = new VScrollBar();

         // Add event handlers for the OnScroll and OnValueChanged events.
         //vScrollBar1.Scroll += new ScrollEventHandler(
         //   this.vScrollBar1_Scroll);
         //vScrollBar1.ValueChanged += new EventHandler(
         //   this.vScrollBar1_ValueChanged);

      }

      public int x1;
      public int x2;
      public int x3;
      public double x8;
      public double x9;
      public double x10;
      public double x11;
      public double x12;
      public double x13;
      public int envia;
      public int t;
      public int y ;

      public uint paquetes_1;
      public uint parametro_1;
      public uint comando_1;
      public uint retardo_1;
```

```
        public uint Tiempo_1;
        public uint nºpaquete_1;
        public uint paquetes_3;
        public uint error_3;
        public uint error_1;
        public int x4; ////Bandera de cantidad de paquetes de almacenaje
        public  bool entro;
        static System.Windows.Forms.Timer myTimer = new
System.Windows.Forms.Timer();
        static System.Windows.Forms.Timer Timer2 = new
System.Windows.Forms.Timer();
        //static bool exitFlag = false;

        public uint[] paquete = new uint[400];
        public uint[] muestra_1 = new uint[4096];
        private Pen gridPen;
        Grilla Grilla = new Grilla(); //Formulario extra

        PicUSBAPI usbapi = new PicUSBAPI();

        public PicUSB()
        {
            InitializeComponent();
        }

        private void leds_off_Click(object sender, EventArgs e)
        {

            usbapi.LedPIC(0x00); //codigo para apagar los leds

        }

        private void led_verde_Click(object sender, EventArgs e)
        {
            usbapi.LedPIC(0x01);    //codigo para encender led verde
        }

        private void led_rojo_Click(object sender, EventArgs e)
        {
            usbapi.LedPIC(0x02);    //codigo para encender led rojo
        }

        private void adquirir_Click(object sender, EventArgs e)
        {

            comando_1 = uint.Parse(comando.Text);
            parametro_1 = uint.Parse(parametro.Text);
            nºpaquete_1 = 1;

            paquetes_1 = uint.Parse(paquetes.Text);
```

```
            retardo_1 = uint.Parse(Retardo_Disparo.Text);
            Tiempo_1 = uint.Parse(Tiempo.Text);
            error_1 = 0;
            uint fin = 0;
            uint flags_1=0;


           ///// Borro el vector de datos /////

            uint k = 0;

            while (k < 4096)
            {
               muestra_1[k] = 0;
               k++;
            }
            k = 0;  //////fin del Borrado////////

          usbapi.adquirir(comando_1, parametro_1, nºpaquete_1, paquetes_1,
retardo_1,Tiempo_1,error_1);
            this.paquete = (usbapi.ResultadoPIC(flags_1));


            uint contador = 0;
            uint y = 0;
            while (y < 100)
            {
               muestra_1[k] = paquete[y];
               y++;
               k++;

            }

            paquetes_3=paquete[200];
            error_3=paquete[201];
            fin=paquete[202];

            while ((paquetes_3 < paquetes_1))
            {

               if (error_3 == 1)
               {                  //////Vuelvo a pedir el mismo paquete //////
                  usbapi.adquirir(comando_1, parametro_1, nºpaquete_1,
paquetes_1, retardo_1,Tiempo_1, error_1);
                  /////k = k - 150;
               }

               if (error_3 == 0)
               {                //////pido el paquete siguiente //////
```

```
                nºpaquete_1 = nºpaquete_1 + 1; //// Incremento el paquete a
pedir //////
                usbapi.adquirir(comando_1, parametro_1, nºpaquete_1,
paquetes_1, retardo_1,Tiempo_1, error_1);
                contador = contador + 1;

            }

            this.paquete = ((usbapi.ResultadoPIC(flags_1)));

            paquetes_3 = paquete[200];
            error_3 = paquete[201];
            y = 0;
            k =  contador * 100  ;
            while (y < 100)
            {
                muestra_1[k] = paquete[y];
                y++;
                k++;
            }
          }
          entro = true;

        }
        public void Un_paqute_Click(object sender, EventArgs e)
        {
          paquetes_1 = 0;
          // usbapi.adquirir(0,1, 0);
        }
        private void avatar_Click(object sender, EventArgs e)
        {
            Process.Start("http://www.hobbypic.com");
        }
        private void PicUSB_Load(object sender, EventArgs e)
        {
        }
        private void groupBox1_Enter(object sender, EventArgs e)
        {
        }

    ////////////////////////scope/////////////////////////

        private void Redraw()
        {
          Refresh();
            ////cStatusReceived.Text = ((int)(scope.CurrentBPS / 1000)).ToString() +
"             kBps";
        }
```

```
  ////////////////////////scope/////////////////////////

      private void Dibujo_Paint(Object sender,
System.Windows.Forms.PaintEventArgs            e)
      {
        if (entro == true)
        {

            Graphics G = Dibujo.CreateGraphics();
            gridPen = new Pen(Brushes.Gray);

            int horizDivs = 8;
            int vertDivs = 10;
            int scopeHeight = 400;
            int scopeWidth = 700;

            float height = scopeHeight / horizDivs;
            float width = scopeWidth / vertDivs;

            int yOffset = (int)((horizDivs * height) / 2) - scopeHeight / 2; // for
some reason often the centrelines aren't lined up, this corrects that
            int xOffset = (int)((vertDivs * width) / 2) - scopeWidth / 2;

            // draw a line down the middles
            //g.DrawLine( new Pen( Brushes.Blue, 2 ), 0, scopeHeight / 2,
scopeWidth, scopeHeight / 2 );
            //g.DrawLine( new Pen( Brushes.Blue, 2), scopeWidth / 2, 0,
scopeWidth                      / 2, scopeHeight );

            // draw the horizontal lines (voltage base)
            for (int i = 1; i <= horizDivs; i++)
               e.Graphics.DrawLine(gridPen, 0, i * height - yOffset, scopeWidth, i
* height - yOffset);

            // draw the vertical lines (time base)
            for (int i = 1; i <= vertDivs; i++)
               e.Graphics.DrawLine(gridPen, i * width - xOffset, 0, i * width -
xOffset, scopeHeight);

            int dato6 = 0;
            int dato7 = 0;

            //x1 = 0; //////Referncia Horizontal---->x1
            //x2 = 0; //////Offset en el bucle
            //x3 = 0; //////Referncia vertical------>x3
            //int x1 = (Tiempo_Muestra.Value);
            x2 = 0;
            t = 0;
            while (t < 3900)
            {
```

```
            //t = t + 1;
            x2 = t -x1;
            if (x2 > 0)
            {
               int dato0 = dato6;
               int dato1 = dato7;
               int dato2 = x2 + 1;
               int dato4 = x2 + 2;
                  dato6 = x2 + 3;

                  dato1 = 2000-(UInt16)muestra_1[t] + x3;
                     t = t + 1;
               int dato3 = 2000-(UInt16)muestra_1[t] + x3;
                     t = t + 1;
               int dato5 = 2000-(UInt16)muestra_1[t] + x3;
                     t = t + 1;
                  dato7 = 2000-(UInt16)muestra_1[t] + x3;

               Point[] points ={
            new Point(dato0, dato1),
            new Point(dato2, dato3),
            new Point(dato4, dato5),
            new Point(dato6, dato7)};
               Pen pen = new Pen(Color.FromArgb(255, 0, 0, 255));
               G.DrawCurve(pen, points, 0.6f);
               //t = x2+x1;
            }
            else
            {
               t = t + 1;
            }

         }

       }

         Timer2.Interval = 100;
         Timer2.Start();
         myTimer.Interval = 1;
         myTimer.Start();
        //myTimer.Stop();
      }

      private void backgroundWorker1_DoWork(object sender,
DoWorkEventArgs e)
      {

      }

      private void pictureBox1_Click_1(object sender, EventArgs e)
```

```
        {

        }
        private void PicUSB_MouseMove(object sender, MouseEventArgs e)
        {
            this.Text = (e.X + x1) * 0.00000025 + "segundos , " + (2000-e.Y - x3) *
0.000488 + "mV";
        }
        private void Pantalla_MouseMove(object sender, MouseEventArgs e)
        {

            x8 = (e.X + x1) * 0.00000025;
            Punto1.Text = x8+" ";
            Punto2.Text = x10+" ";
            x9 = x10 - x8;
            x9 = Math.Abs(x9);
            Diferencia_de_Tiempo.Text = x9 +" ";
            x10 = x8;
            Frecuencia.Text = 1 / x9 +" ";
            x11=(2000-e.Y - x3) * 0.000488;
            Tensión.Text = x11 + " ";
            x12=x13-x11;
            x12 = Math.Abs(x12);
            difTensión.Text=x12+" ";
            x13 = x11;
        }
        private void cHorizDivs_ValueChanged(object sender, EventArgs e)
        {
            Redraw();
        }
        private void Abrir_Grilla_Click(object sender, EventArgs e)
        {
            //Form1 Grilla = new Grilla();
            //Grilla.Visible = true;
            //Grilla ActiveForm;
            //ActiveForm.Show;
            //Show;//Grilla.Show;
            //Form1.Hide;
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            this.Invalidate(true);
        }
        private void comando_TextChanged(object sender, EventArgs e)
        {
        }
        private void progressBar1_Click(object sender, EventArgs e)
        {
        }
        private void Parametro_TextChanged(object sender, EventArgs e)
```

```
        {

        }
        private void Dibujo_VisibleChanged(object sender, EventArgs e)
        {
        }
        private void Dibujo_EnabledChanged(object sender, EventArgs e)
        {
        }
        private void Tiempo_Muestra_Scroll(object sender, ScrollEventArgs e)
        {
            x1 = Tiempo_Muestra.Value;
           //this.Text = e.X;
        }
        private void vScrollBar1_Scroll(object sender, ScrollEventArgs e)
        {
          x3 = Nivel_Vertical.Value;
        }
       private void label2_Click(object sender, EventArgs e)
        {
        }
        private void timer2_Tick(object sender, EventArgs e)
        {
        }
        private void paquetes_TextChanged(object sender, EventArgs e)
        {
        }
        private void Retardo_Disparo_TextChanged(object sender, EventArgs e)
        {
        }
        private void Con_Grilla_CheckedChanged_1(object sender, EventArgs e)
        {
           ///Boolean Grilla  = (Triggering.RisingEdge);
        }
        private void Tiempo_TextChanged(object sender, EventArgs e)
        {
        }
        private void Diferencia_de_Tiempo_TextChanged(object sender,
EventArgs e)
        {
        }
        private void groupBox2_Enter(object sender, EventArgs e)
        {
        }
    }
}
```

Ahora veremos otra porción del programa donde que describe una función llamada desde un evento.

Función picUSBapi.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Runtime.InteropServices; // Clase para importar DLL
using System.Drawing;
using PVOID = System.IntPtr;
using DWORD = System.UInt32;

namespace PicUSB
{
   public enum Triggering
   {
      RisingEdge,
      FallingEdge,
      None
   }

   unsafe public class PicUSBAPI
   {
      #region Definición de los Strings: EndPoint y VID_PID
      string vid_pid_norm = "vid_04d8&pid_0011";

      string out_pipe = "\\MCHP_EP1";
      string in_pipe = "\\MCHP_EP1";
      #endregion

      #region Funciones importadas de la DLL: mpusbapi.dll
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBGetDLLVersion();
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBGetDeviceCount(string pVID_PID);
      [DllImport("mpusbapi.dll")]
      private static extern void* _MPUSBOpen(DWORD instance, string
pVID_PID, string pEP, DWORD dwDir, DWORD dwReserved);
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBRead(void* handle, void* pData,
DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBWrite(void* handle, void* pData,
DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
      [DllImport("mpusbapi.dll")]
      private static extern DWORD _MPUSBReadInt(void* handle, DWORD*
pData, DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
      [DllImport("mpusbapi.dll")]
      private static extern bool _MPUSBClose(void* handle);
```

```
        #endregion


        void* myOutPipe;
        void* myInPipe;

        //private int horizDivs = 8;
        //private int vertDivs = 8;

        public int paquete;
        public int nº_paquete;
        public int y;
        //public int bloq;  ////Variable de cantidad de bloques de pedido al pic
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new PicUSB());

        }

        public void OpenPipes()
        {
            DWORD selection = 0;

            myOutPipe = _MPUSBOpen(selection, vid_pid_norm, out_pipe, 0, 0);
            myInPipe = _MPUSBOpen(selection, vid_pid_norm, in_pipe, 1, 0);
        }

        public void ClosePipes()
        {
            _MPUSBClose(myOutPipe);
            _MPUSBClose(myInPipe);
        }

        private void SendPacket(byte* SendData, DWORD SendLength)
        {
            uint SendDelay = 1000;
            DWORD SentDataLength;
            OpenPipes();
            _MPUSBWrite(myOutPipe, (void*)SendData, SendLength,
&SentDataLength, SendDelay);
            ClosePipes();
        }

        private void ReceivePacket(byte* ReceiveData, DWORD* ReceiveLength)
        {
            uint ReceiveDelay = 1000;
            DWORD ExpectedReceiveLength = *ReceiveLength;

            OpenPipes();
```

```
        _MPUSBRead(myInPipe, (void*)ReceiveData, ExpectedReceiveLength,
ReceiveLength, ReceiveDelay);
        ClosePipes();
    }



    public void adquirir(uint comando_2, uint parametro_2, uint nºpaquete_2,
uint paquetes_2, uint Retardo_2, uint Tiempo_2,uint error_2)
    {
        byte* send_buf = stackalloc byte[9];
        send_buf[0] = (byte)comando_2;
        send_buf[1] = (byte)parametro_2;
        send_buf[2] = 0x00;
        send_buf[3] = 0x00;
        send_buf[4] = (byte)nºpaquete_2;
        send_buf[5] = (byte)paquetes_2;
        send_buf[6] = (byte)Retardo_2;
        send_buf[7] = (byte)Tiempo_2;
        send_buf[8] = 0x00;                 //// Error  //////
        SendPacket(send_buf, 9);

    }

    public /*string*/ uint[] ResultadoPIC(uint flags_2)
    {

        uint[] myIntArray = new uint[210];
        //byte* result = stackalloc byte[300];
        byte* send_buf = stackalloc byte[9];
        byte* receive_buf = stackalloc byte[210];
        DWORD RecvLength = 210;
        ReceivePacket(receive_buf, &RecvLength);


        uint dato_Bajo;
        uint dato_Alto;
        uint suma_total;

        ////////int nº_paquete = receive_buf[300];

        uint cont = 0;
        uint p = 0;

        while (cont < 200)   ////Bucle de carga del array de 200 datos
        {
            dato_Bajo = receive_buf[cont];
            cont++;
            dato_Alto = receive_buf[cont];
            dato_Alto = dato_Alto * 256;
```

```
            suma_total = dato_Bajo + dato_Alto;
            myIntArray[p] = suma_total;
            p++;
            cont++;
        }

        //// Aqui cargamos los datos de confirmacion de datos enviados por el
pic para verificar si hay error/////
        p = p + 100;
        myIntArray[p] = receive_buf[200];   //// Numero de Paquete enviado
        p++;
        myIntArray[p] = receive_buf[201];   //// Numero de Error generado  ////
        p++;
        myIntArray[p] = receive_buf[202]; ; /////fin de transaccion //////


        return myIntArray;

    }


    public void LedPIC(uint led)
    {
        byte* send_buf = stackalloc byte[2];

        send_buf[0] = 0x01;            // Código de Entrada a Modo_Led
        send_buf[1] = (byte)led;
        SendPacket(send_buf, 2);
    }

  }
}
```

6.-Manual de Operación.

6.1-Instalación del equipo

El equipo es conectado al conector USB del computador por medio del cable USB universal de impresora.
Al conectar el equipo por primera vez, aparecerá el asistente para la instalación de nuevo hardware. Se marca la opción de "instalar desde una lista o ubicación específica". Luego se pincha sobre siguiente y en la siguiente pantalla se marca la opción de "Buscar el controlador más adecuado". En estas ubicaciones, se pincha sobre "Incluir esta ubicación en la búsqueda" y a continuación sobre Examinar. Entonces se selecciona la carpeta PicUSB_Driver, y se le da a Aceptar, y a Siguiente. Aparecerá entonces una pantalla de advertencia, se hace clic sobre Aceptar, y la instalación del Driver estará terminada; el led del Prototipo habrá pasado de estar rojo a verde y estará listo para ser usado.

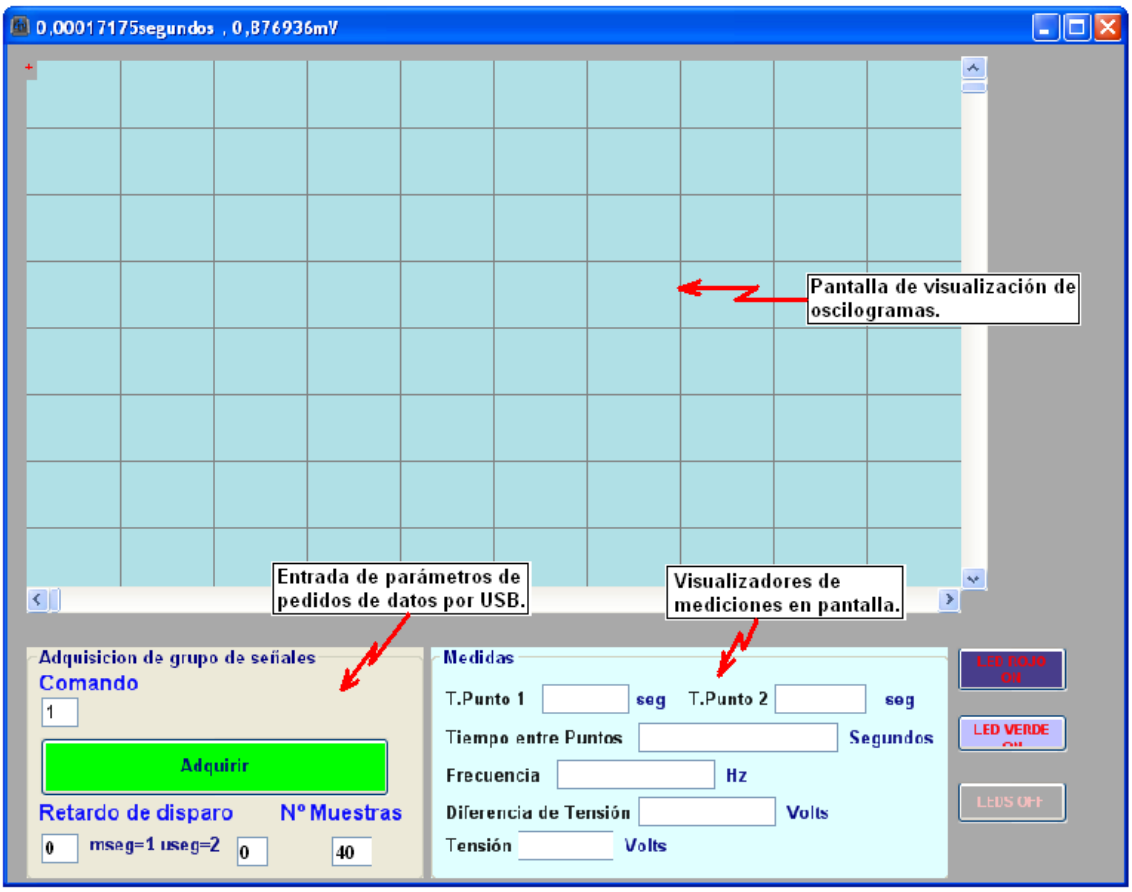
Una vez instalado el driver con la correspondiente DLL, podemos hacer uso del proyecto haciendo doble clic sobre PicUsbProyec.exe donde se abrirá el proyecto.
En la pantalla se pueden ver bien localizados dos tres sectores:

- Entrada de parámetros de pedidos de datos por USB.
- Pantalla de visualización de oscilogramas.
- Visualizadores de mediciones en pantalla.

La configuración inicial permite hacer uso de una adquisición con solo apretar el botón de “Adquirir”.
De inicio es seteada una configuración con:

- Máximo numero de muestras: 4000.
- Máximo nivel de entrada de tensión: 175Volts.
- Máxima resolución: 488 microVolts.

La pantalla que se observará es la siguiente:



En el sector “Entrada de Parámetros” se ingresan los siguientes datos:

- Comando:

1. Retardo, Adquisición y Disparo.
2. Adquisición
3. Disparo

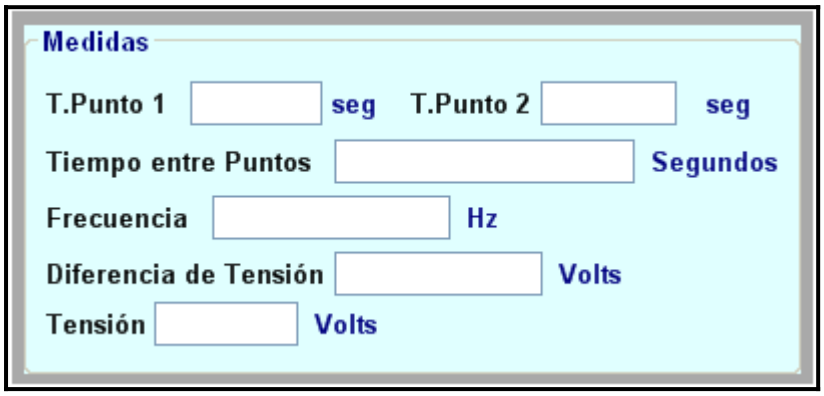
- Retardo de Disparo:

Tiempo de disparo requerido
1. milisegundos
2. microsegundos

- Numero de muestras:

Paquetes de 100 muestras
1 a 40 Paquetes

En el siguiente dibujo se visualizaran las medidas realizadas sobre la pantalla:



Con solo pinchar con el puntero sobre la cuadricula se obtiene el dato del tiempo y tensión correspondiente a dicho punto, y de la misma manera en otra posición de la cuadricula, se obtendrán posteriormente:

- Tiempos entre Puntos.
- Frecuencia correspondientes.
- Diferencias de tensión.

Los indicadores sobre el dispositivo son:

- Encendido.
- Se ha realizado el enlace por el puerto USB.
- Disparo de la fuente de pulsos.

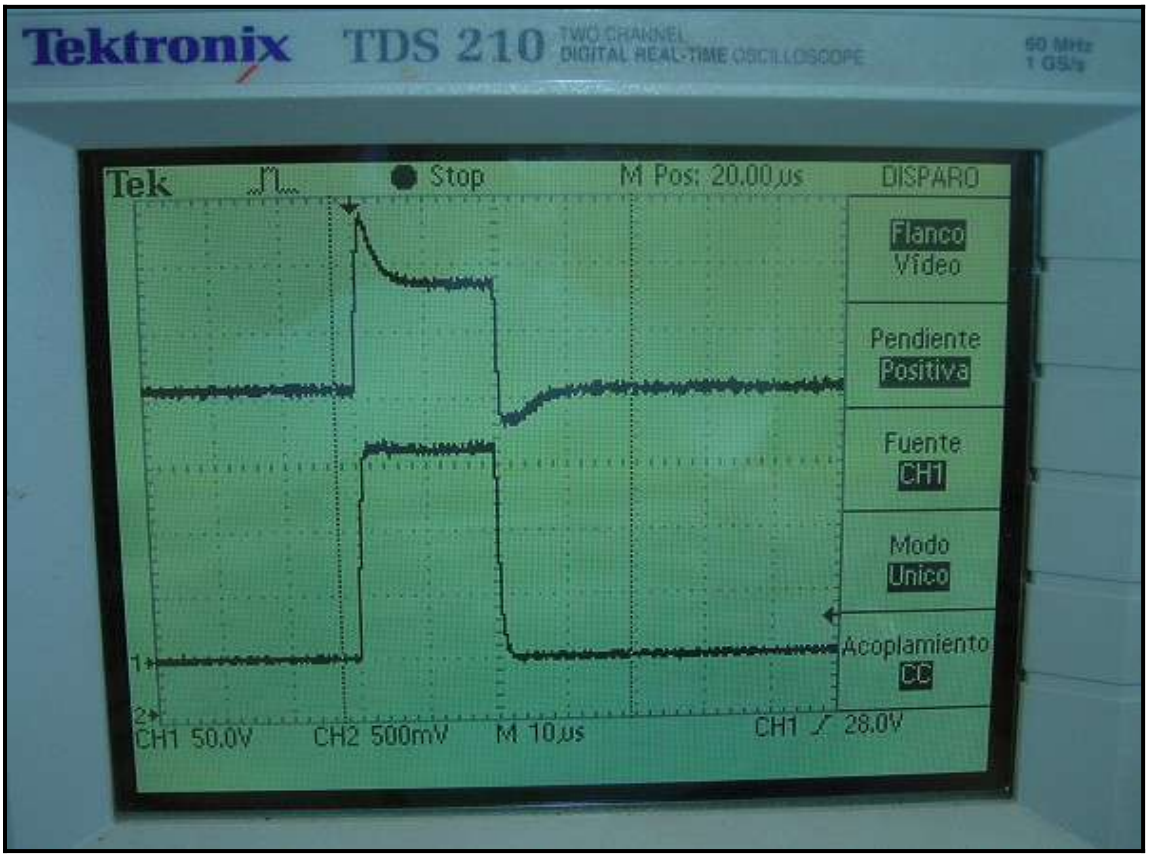
La pantalla puede representar 700 puntos en sentido horizontal y 400 en sentido vertical. Los puntos restantes se pueden visualizar desplazando los respectivos cursores, logrando ver un máximo de 4096 niveles de tensión y 4000 muestras con sus respectivos tiempos.

7.-Mediciones.

Las mediciones necesarias que se llevaron a cabo en el diseño fueron:

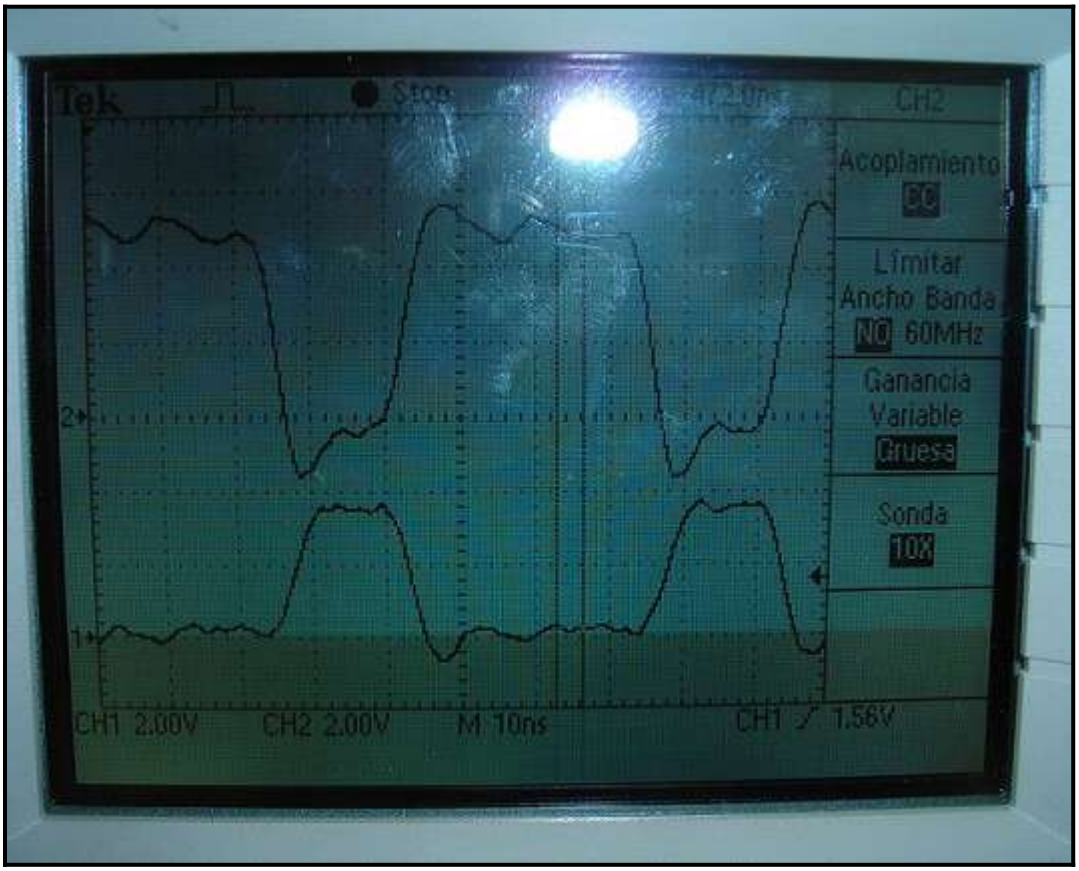
- Controles de tensiones de alimentación.
- Continuidad entre masas de distintas placas.
- Continuidad de pistas delicadas.
- Controles sobre posibles corto circuitos.
- Niveles de ruido en las primeras etapas de circuito de señales analógicas.
- Diferencias de tiempos en los correspondientes ciclos de escritura y lectura.
- Niveles lógicos .
- Compensación de punta de prueba.
- Ancho de banda resultante.
- Consumo de las diferentes etapas.

De todas las mediciones realizadas, fue relevante que las señales que ingresen por la punta de prueba, conserven la forma luego de pasar por todos los amplificadores operacionales. Es decir no sufran una distorsión por limitación del ancho de banda o falta de compensación entre otras.
Las siguiente foto muestra la punta sin compensar.



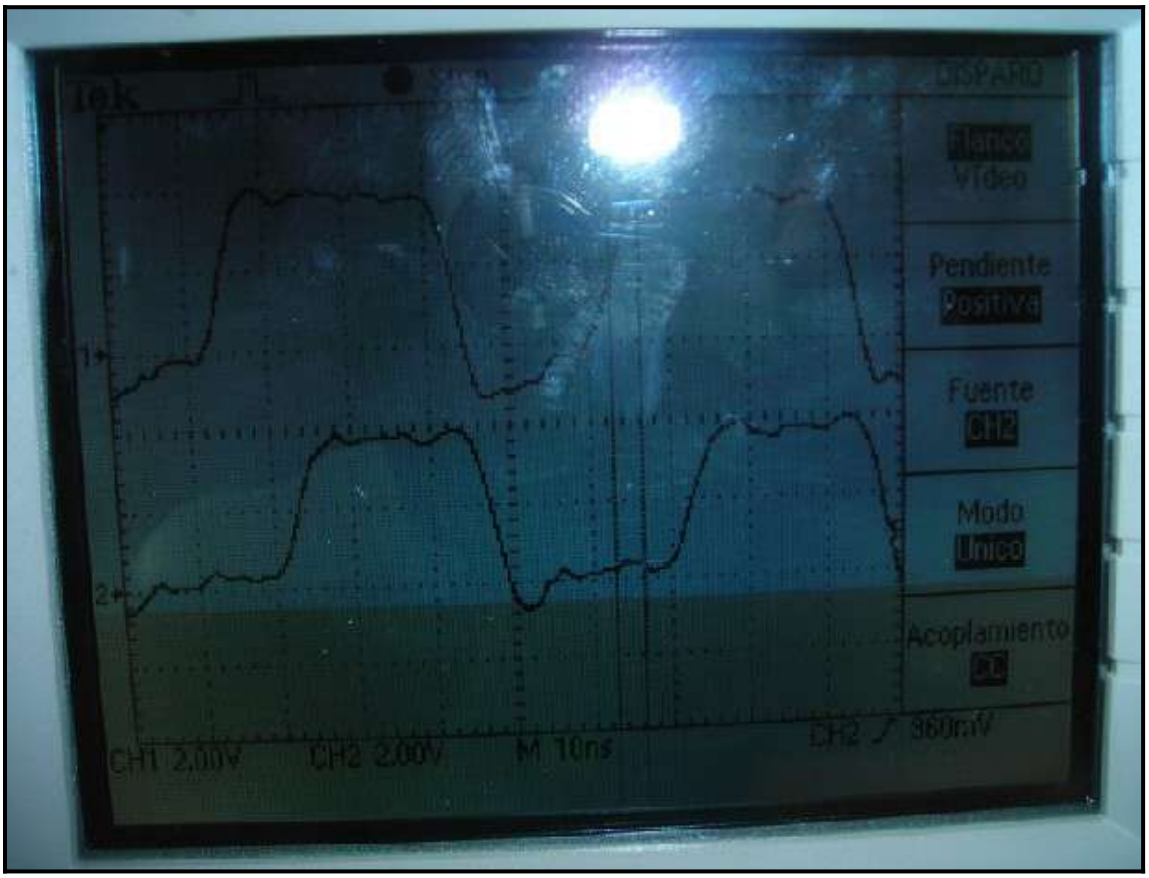
Se puede observar que la amplitud del pulso de 160Volts, puede llegar a tener una descompensación considerable en la etapa de entrada al conversor ADC.

Se ve en el canal 2 del osciloscopio que la señal tiene una componente de continua de 2.5Volts de referencia, propio de la salida diferencial.
Otra de las mediciones relevantes y cruciales a la hora del funcionamiento del equipo, fue la sincronización de los tiempos de habilitación de escritura de la memoria, cuyo nivel lógico proviene del Pin DV del conversor ADC.
Las señales puestas en juego fueron la señal de Wclock de la memoria y la de habilitación de escritura ya nombrada.
La fotos siguientes son un registro del problema:



En la foto de arriba se observan dos ondas, la superior es corresponde a la señal de Wclock y la dela parte inferior es la señal Wen proveniente del Pin DV del conversor.
Dadas las especificaciones de tiempos de escritura ya vistas en la sección de Memoria, el flanco ascendente del Wclock debe producirse en el momento en que el chip esta habilitado, es decir con Wen con nivel bajo.
Se corrigió este problema colocando una compuerta que retrace el tiempo de habilitación de chip (Wen).

La siguiente fotografía da cuenta de los resultados obtenidos.



Aquí se puede apreciar la onda superior Wclock con un retrazo temporal que proporciona los resultados requeridos.

8.-Conclusiones.

Concluimos finalmente que el equipo desarrollado arrojo en su gran mayoría los resultados deseados. De lo expuesto en este documento, cada parte pasó por un análisis de ingeniería minucioso, tomando en cuenta los materiales, tiempos de desarrollo, factibilidad de funcionamiento y costos.
En diferentes etapas se resolvieron problemas de toda índole.
La selección de componentes que trabajan con señales analógicas y digitales, con diferentes velocidades, generaron estudios interesantes e información útil acerca de la utilización de los mismos.
Se resolvieron problemas con componentes SMD, donde su acoplamiento al circuito impreso es muy dificultoso con herramientas convencionales.
El método por insolación de fabricación de circuitos impresos solucionó problemas de dificultosa conectividad y aislamiento electromagnético generado en las conversiones de datos.
La interacción entre el microcontrolador y el CPU por medio de puerto USB da un avance para futuros desarrollos, dado que las nuevas computadoras tienen problemas con los dispositivos que se conectan al puerto RS232 de muy baja velocidad y menor confiabilidad. Todos los nuevos dispositivos son conectados al puerto USB con gran facilidad.

9.-Bibliografía

- Programacion en C#. -La biblia de C# ANAYA .De Jeff Ferguso, Brian Patterson y Jasón Beres
- Programacion en C. –C Compiler for Mcrochip PICmicro MCUs. Andrés Canovas López.
- Microcontroladores PIC, diseño práctico de aplicaciones Tercera parte, José M.a Angulo Usategui, Ignacio Angulo Martínez, Susana Romero Yesa.
- Pic18F4550, Datasheets, Application Notes.
- ADS809Y, Datasheets, Application Notes.
- CY7C4245, Datasheets, Application Notes.
- Bibliografía de la cátedra de Adquisición digital de señales UNMDP
- Bibliografía de la cátedra de Mediciones Electrónicas UNMDP
- www.Digikey.com- http://www.ti.com/ http://www.national/ http://www.microchip.com/
- WWW.

Agradecimientos:

- A mi madre y mi familia.

- A los señores Germán Káiser y Martín Calveira por la ayuda y predisposición.

- Al Ing. Walter Gemín por su colaboración y a ayuda en el proyecto.

- A los profesores de la facultad de Ingeniería por su dedicación y enseñanza.