INDICE:

3. Modos de funcionamiento

3.1 Modo Visual
3.2 Modo Muestras finito
3.3 Modo Continuo

Programa en Labview

1. Modulo de comunicación

2 . Modulo de Procesado e la señal

3. Módulos de representación de datos

Introducción

El objetivo de este proyecto era el de crear un sistema de captación y procesado de señales analógicas económico. Para la captación de señales optamos por una solución hardware basada en un microcontrolador que se comunicaría por puerto serie con un PC. La parte del procesado digital de la señal la llevaríamos acabo con el programa Labview y sus módulos para el procesado digital de señales analógicas.

Por lo tanto el proyecto se puede dividir en tres fases y así dividiremos esta documentación. Comenzaremos con la explicación del hardware. Se hará un análisis de la placa creada para albergar al microcontrolador y el convertidor analógico digital. Posteriormente se analizara el programa cargador en el microcontrolador a fin de controlar el conversor analógico digital y comunicarse con el PC. Para terminar veremos como se controla el microcontrolador y se reciben datos de el en el PC a través de un programa en labview.

Como anexo se incluirán todas las hojas de datos de los distintos integrados utilizados.

Documentación Hardware

El propósito de este sistema es el de captación de datos analógicos y el filtrado posterior de estos, utilizando distintos tipos de filtros conseguimos eliminar señales no deseadas.

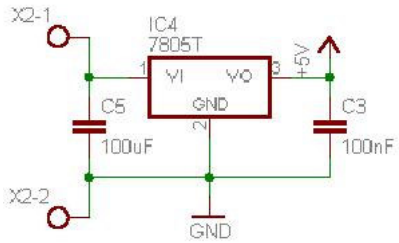
Para lograr esto utilizamos una tarjeta con la que adquirimos los datos, hasta ocho canales de entrada. Esta tarjeta debía transmitir los datos al ordenador que se iba a encargar de procesarlos. Los datos recogidos por la tarjeta son procesados y filtrados, utilizando el programa de control *LabView*.

El microcontrolador elegido incluye un puerto serie que utilizaremos para comunicarnos. Por el contrario no incorpora conversor analógico digital y es por ello que debemos incluir un integrado que haga tal función. Para terminar tenemos el circuito programador y un LCD con 2 pulsadores que permite operar de forma autónoma al PC.

Analizaremos el circuito bloque a bloque según el siguiente orden:

Alimentación
Programación
Conversión A/D

Alimentación:



Elegimos este circuito por su sencillez de montaje y fiabilidad, lo que puede causarnos algún contratiempo a la larga con respecto a las potencias que soporta, se calienta en exceso cuando lleva mucho tiempo conectado o iluminamos el LCD por lo que lo dotamos de radiador, que ayudará a disipar este calor.

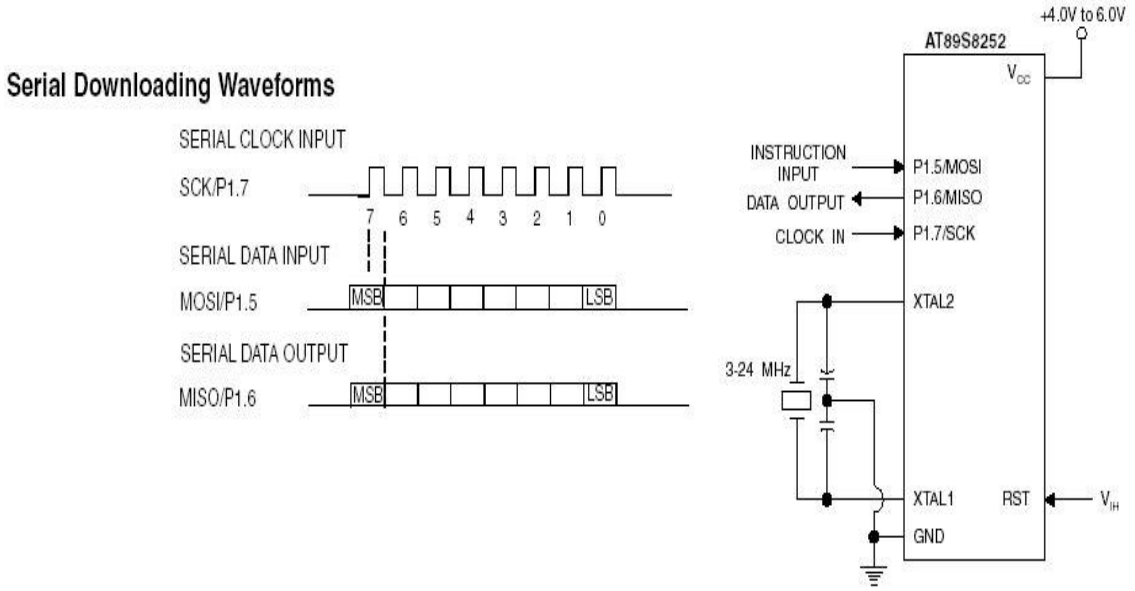
Es importante alimentar por encima de los 5v que queremos que nos de y no sobrepasar de los 12v de alimentación, pues podríamos quemar no sólo el 7805T si no que también se pueden quemar los demás componentes.

Para que nos de una señal lo mas continua posible, filtramos tanto la señal de entrada como la de salida utilizando para ello condensadores de 100nF.

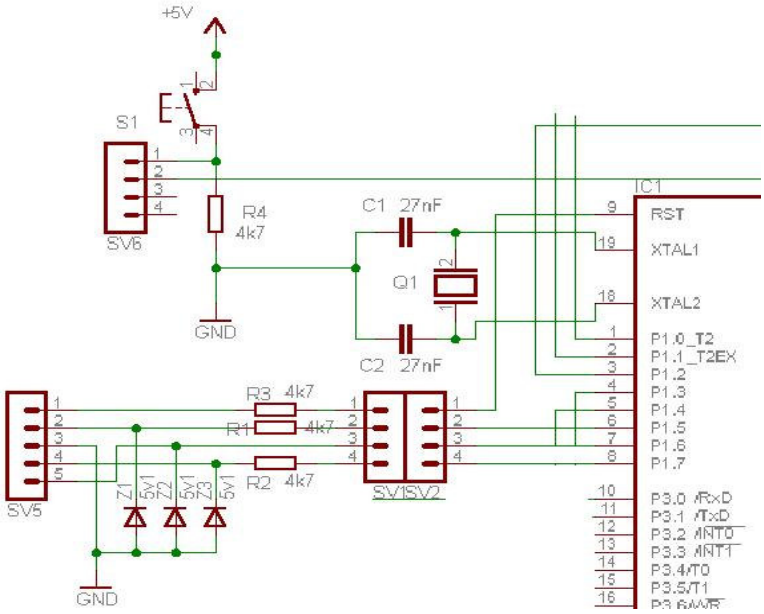
Programador:

Una de las mayores ventajas de este micro controlador es que permite su programación por interfase serie(SPI) esto nos permite integrar el circuito programador en el propio circuito de trabajo y hacer así el sistema ISP(programable en el sistema). Para la programación se utilizan 4 líneas como puede verse: *reset*, *mosi*, *miso* t *sck*. Sus funciones son las siguientes:

Iniciar el inicio de la programación, introducción del programa, lectura del programa y reloj de sincronismo. La programación se lleva acabo según el siguiente cronograma y esquema de conexión:



Seguimos buscando la sencillez de manejo y montaje así que buscando en Internet en la pagina www.lancos.com nos encontramos con este esquema de fácil montaje. Consiste en diodos zener (Z1-Z3) que limitan la tensión de 0 a 5v ahorrándonos un integrado, del estilo del max232, y resistencias de protección para el circuito, de 4k7 (R1-R3). Lo conectamos a través de un switch que en posición de programación aísla estas 4 líneas del resto del circuito, de forma que evitará posibles conflictos de tensiones por el puerto serie conectado al ordenador, con el cual programamos el micro controlador y la tensión de normal funcionamiento de nuestro circuito.

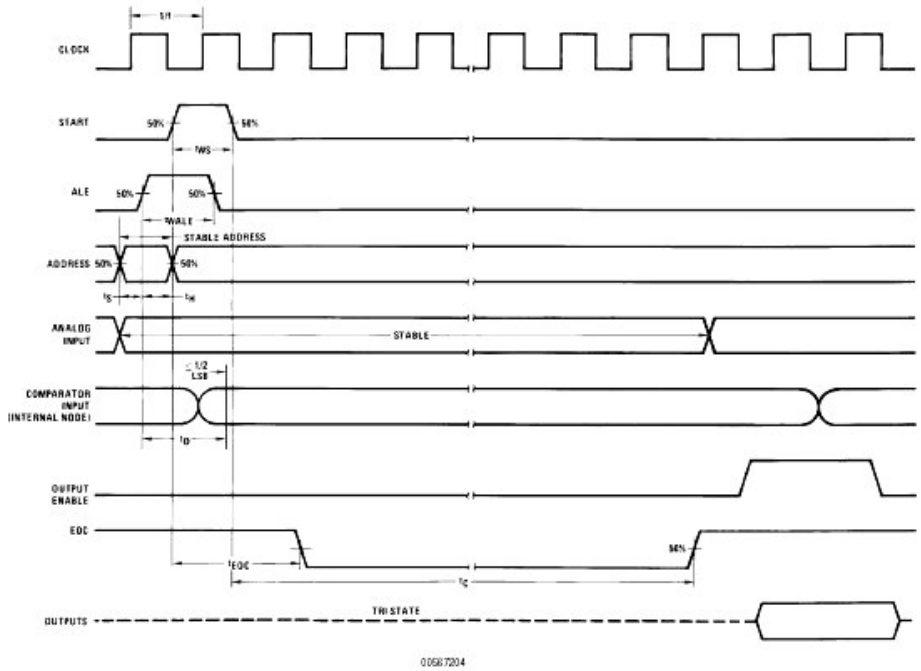


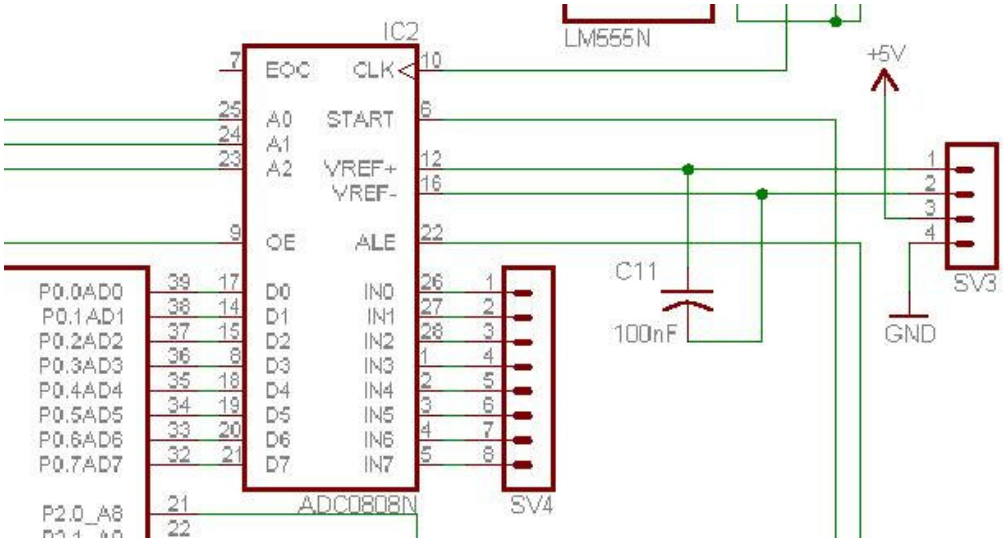
Para programar y descargar el programa en el micro hemos utilizado el PonyProg que encontramos en la pagina de Internet www.lancos.com, es de fácil manejo y rápido aprendizaje. Este programa permite volcar el archivo .HEX generado por el compilador, y además permite chequear el micro después de haber descargado el programa y comprobar que el programa se a descargado correctamente.

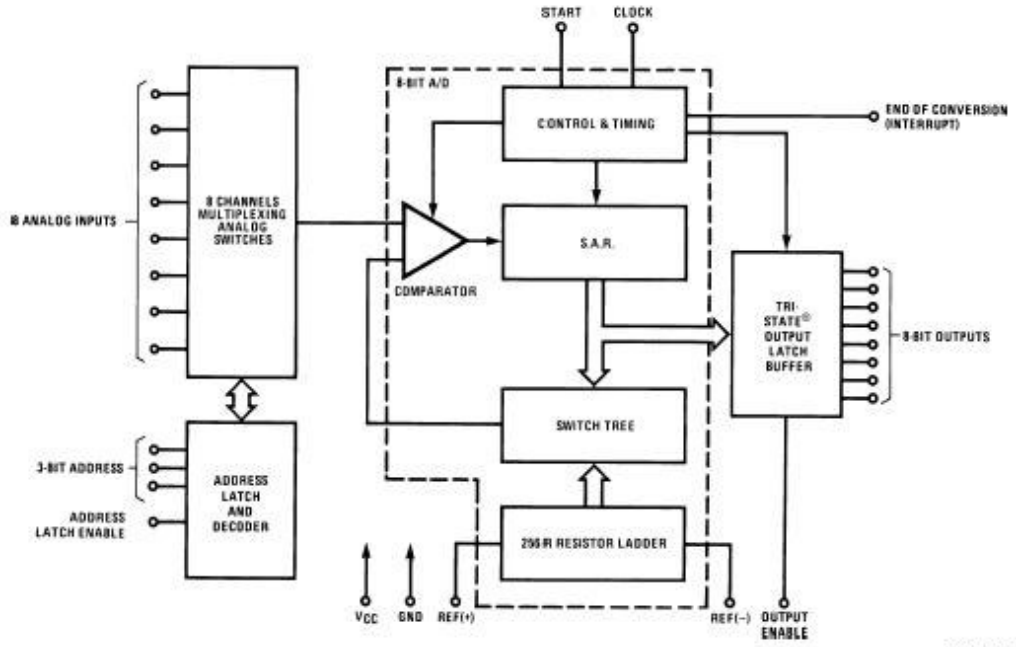
Adquisición de datos:

Para obtener los datos del exterior ya que el micro controlador no reconoce señales analógicas necesitamos un convertidor analógico a digital, *"ADC0808N"*, el cual traduce la señal analógica en datos que el micro controlador sea capaz de interpretar. Este es un conversor analógico digital de 8 canales de aproximaciones sucesivas, por lo cual necesita una señal de reloj y cierto tiempo para finalizar la conversión.

Timing Diagram

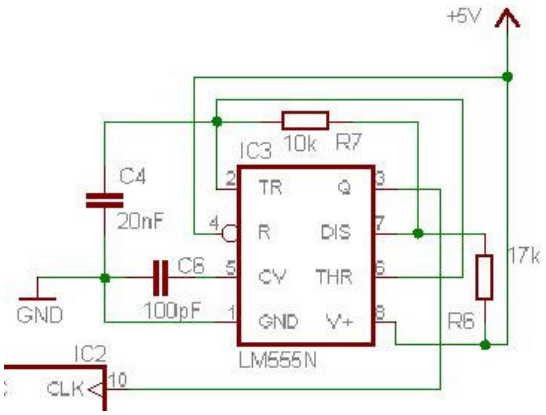






See Ordering Information

Es importante tener en cuenta la calibración de las tensiones de referencia de 0-5v, utilizando un condensador electrolítico de desacoplo de *100nF*, que colocamos entre las entradas de referencia del convertidor, de no hacer esto el convertidor nos provocará serios problemas con la lectura e interpretación de los datos porque no tendrá una referencia fiable.



Este convertidor necesita adicionalmente una señal externa de reloj, utilizando un integrado 555 en modo astable conseguimos una señal aproximada de 1KHz en la patilla 3 suficiente para la lectura de datos.

El microcontrolador Atmel 89S8252

Para el desarrollo de este proyecto hemos empleado un microcontrolador de la casa Atmel, el AT89S8252. Este microcontrolador esta basado en el núcleo del 8051 siendo compatible con el a nivel de instrucciones. Permite operar en el rango de los 3 a los 24Mhz habiendo elegido una frecuencia de trabajo de 11,0592MHz por su idoneidad a la hora de trabajar con el puerto serie. Hay que remarcar que este microcontrolador tiene un ciclo de maquina de 12 pulsos de reloj, lo que a la frecuencia indicada nos da un tiempo de instrucción cercano al microsegundo y suficiente para los propósitos deseados.

Hablando de las capacidades de este microcontrolador hay que decir que incluye 8k de flash para programa, lo cual nos da un gran flexibilidad a la hora de crear un programa complejo. Así mismo incluye 2k de EEPROM,que no será aprovechada en este proyecto, y 256bytes de RAM, suficiente para nuestros propósitos, de los cuales 128 son de acceso directo.

Una de las facilidades de este microcontrolador es que permite la programación en serie(SPI), siendo extremadamente sencillo programarlo incluso en el circuito de trabajo. Incluye también una interfase serie estándar, la cual con un convertidor a voltajes RS232 nos permitirá intercomunicarnos con el PC.

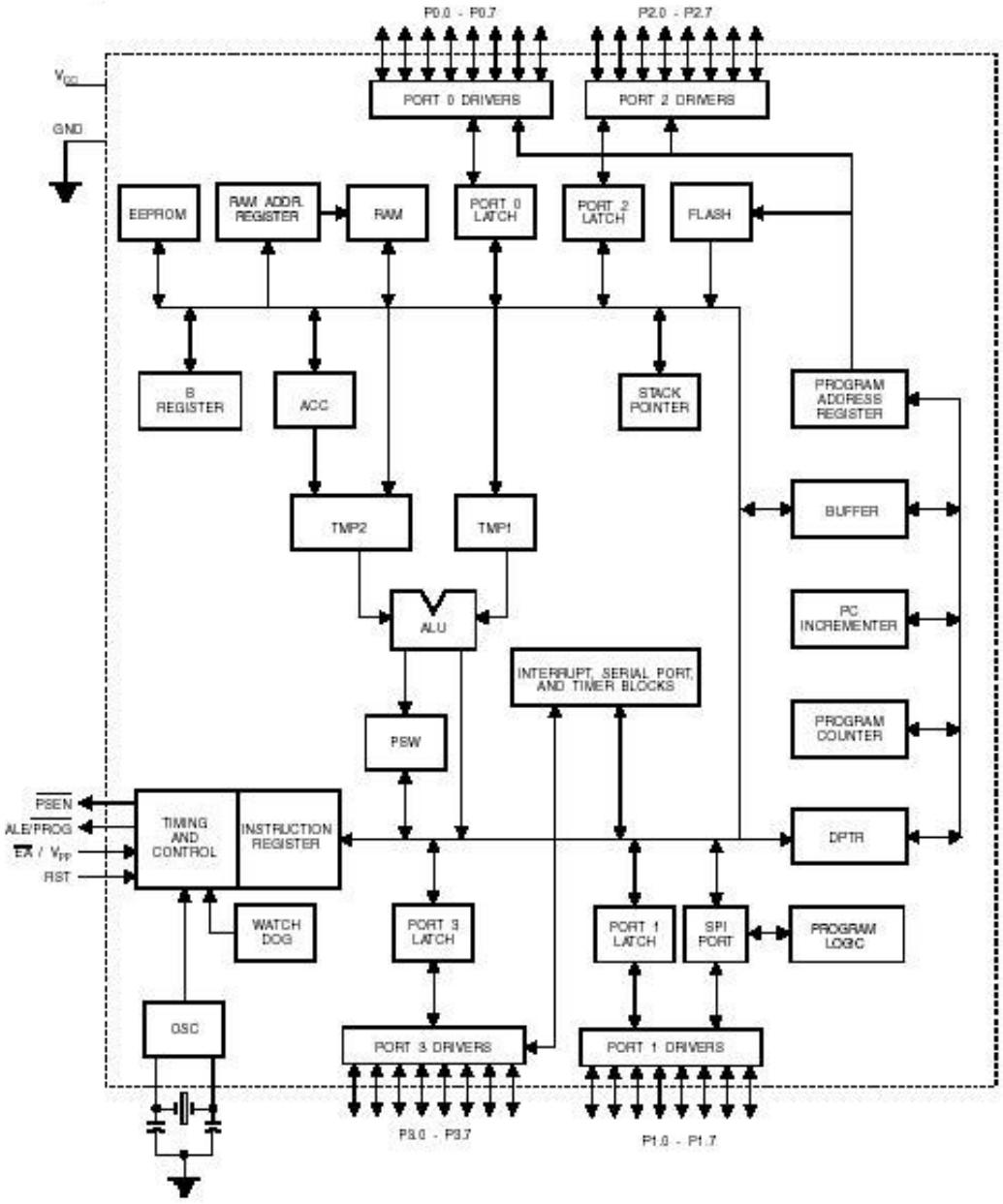
Por contra este microcontrolador no incluye conversor analógico digital y por ello nos vemos obligados a usar uno externo. Esto no será un problema debido a la gran cantidad de entradas / salidas de las que dispone(hasta 32).

Algunas características de este micro son:

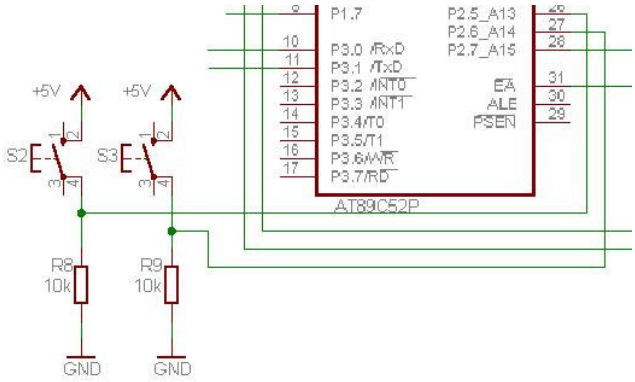
4 puertos de 8 bits, 32 líneas de entrada salida.
Puerto serie integrado.
3 Timers de 16 bits.

Todos estos recursos pueden verse en el siguiente esquema por bloques.

Block Diagram



Pulsadores:



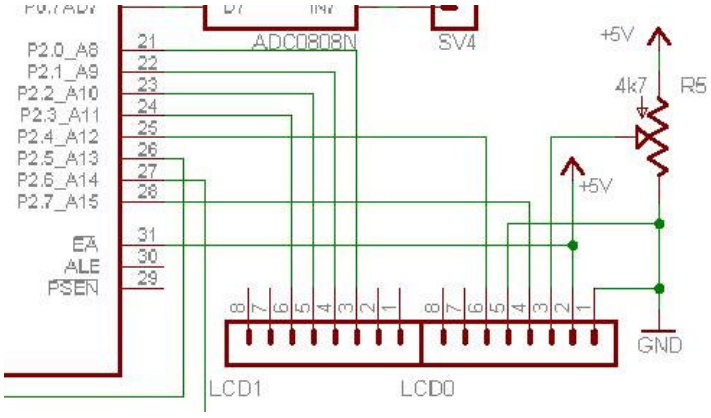
Tenemos tres pulsadores de pull down, uno de *RESET* que va con la parte de programación y otros dos para moverse en la pantalla del LCD, son normalmente abiertos y como son de pull down, introducen un cero en modo continuo y un uno cuando los pulsamos, así que hay que tener cuidado con l valor de la resistencia de pull down porque un valor demasiado alto o demasiado bajo, al pulsar no tendrás la señal que buscas.

Switch:

Permite el aprovechamiento de las 4 líneas de programación que de otra forma las perderíamos y así aísla el programador del exterior.

También a través del conmutador tenemos el circuito de reset, consistente en un pulsador y una resistencia “R4” de 4k7 de pull up, quedando conectado a las patillas de mayor peso del puerto 1 del micro (P1.7-P1.5) “AT89C52P”.

Visualizador LCD:

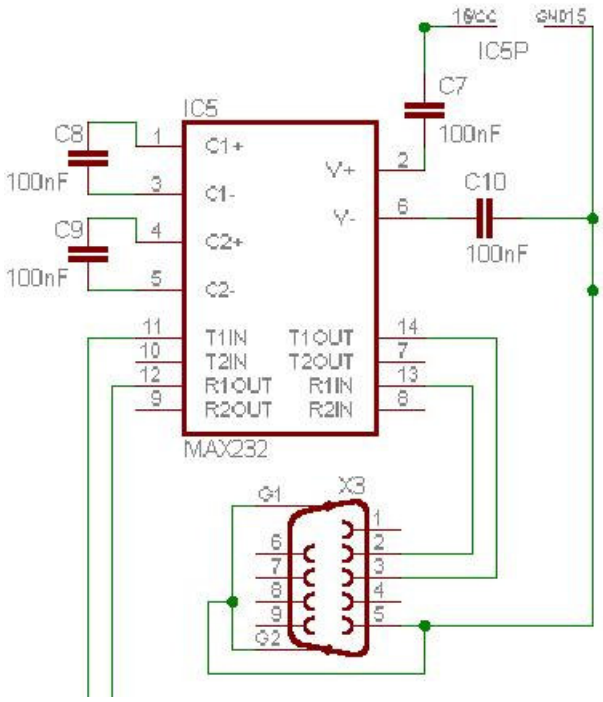


Bus de 4 líneas de datos de control, se desprecia uno de los de control de lectura.

El control de contraste se hace intercalando un potenciómetro de 4k7Ω

(R5), no usamos retro-iluminación debido al alto consumo que supone.

Comunicación con puerto serie:



Las salidas del micro controlador son de 0v-5v y la tensión a la que trabaja el puerto serie es de –12v/+12v así que es necesario transformar la tensión entre las salidas del micro controlador y el DB9 (conector serie), y esto se hace a través del RS232 conectado al micro (P3.0 /RxD, P3.1 /TxD) y utilizando I2C, logramos la transformación y correcta comunicación de datos.

El montaje*:*

El montaje en sándwich de la placa está provocado por los medios de los que hemos hecho uso, el programa que utilizamos para el diseño de la placa es el Eagle en una versión Demo, con el handicap del tamaño de algunos componentes como son el LCD, el micro o el convertidor, esto nos a llevado al diseño modular que nos resuelve el problema de la reducida dimensión del conjunto, pero a la hora de su manejo, teniendo una fácil lectura del LCD y manejo de los controles.

Conectores:

Tenemos una serie de conectores que actúan como puente entre las dos placas que conforman el sistema, de forma que casen correctamente y se amolden al conjunto.

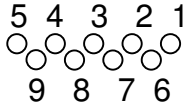
Estos conectores hay que reforzarlos mecánicamente con algunos puntos de anclaje que mejoren la rigidez mecánica del conjunto, que se ve

amenazada por la continua conexión y desconexión de los puertos de programación y control.

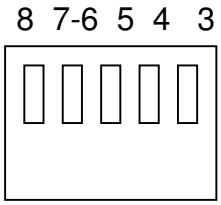
Puerto de programación:

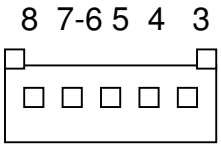
Utilizamos un cable con cinco conexiones porque necesitamos dos para control, dos para datos y una para reset.

Conectados de la siguiente manera:



8 Marrón (MISO)
7-6 Azul (SCK)
5 Verde (MASA)
4 Gris (MOSI)
3 Amarillo (RESET)





Puerto de comunicación serie:

La placa a sido diseñada para utilizar un cable puerto serie cruzado del cual solo necesitamos tres líneas una de masa y dos para entrada y salida de datos.

La disposición de las líneas en el conector están situadas de la siguiente forma.

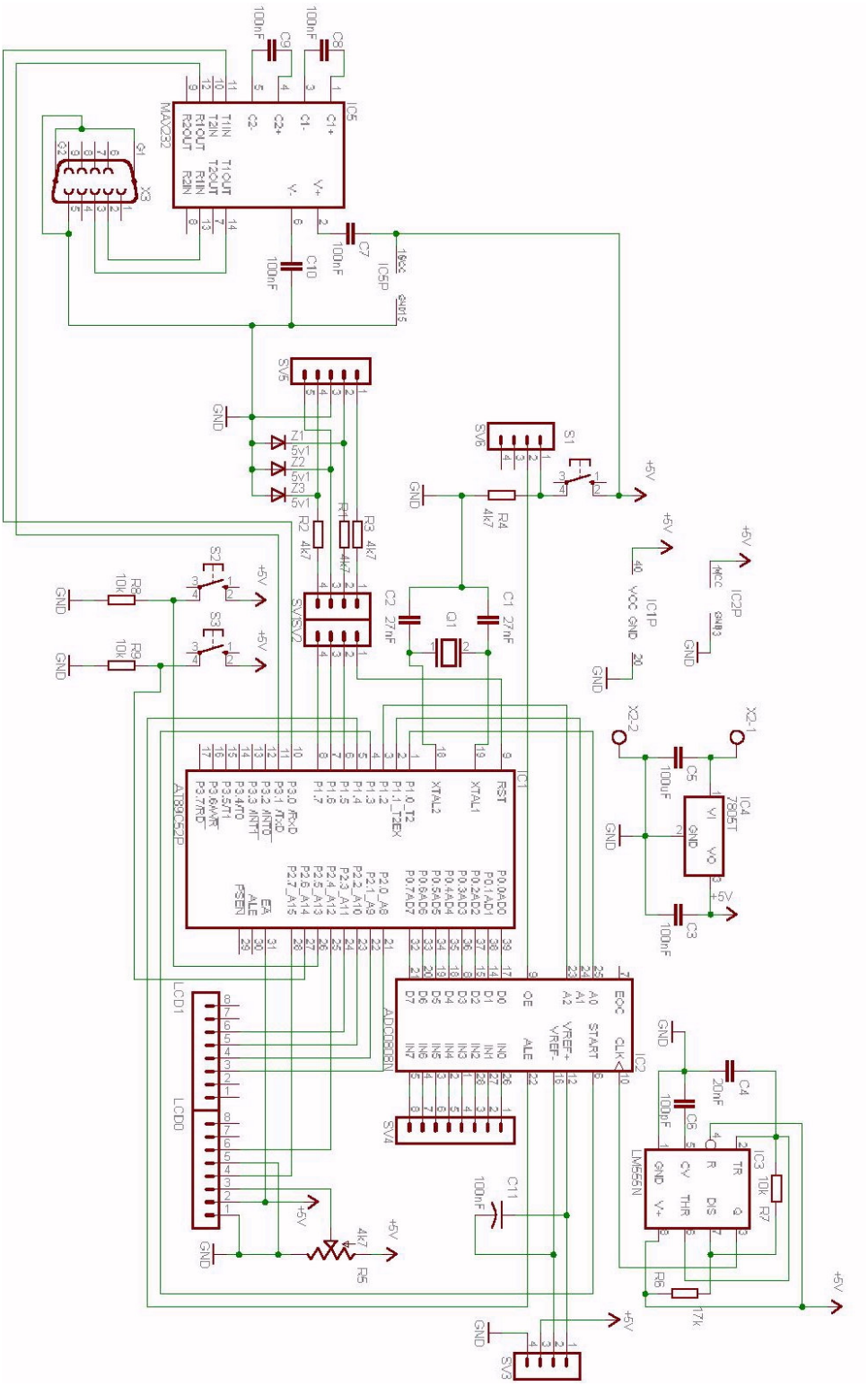
La salida de datos en la patilla tres del DB9 y la de entrada de datos en la dos y la patilla cinco será masa.

***Lista de componentes**:*

R1 4K7
R2 4K7
R3 4K7
R4 4K7
R5 4K7
R6 17K
R7 10K
R8 10K
R9 10K
C1 27nF
C2 27nF
C3 100nF
C4 20nF
C5 100nF
C6 100pF
C7 100nF
C8 100nF
C9 100nF
C10 100nF
Z1 Diodo zener 5v1
Z2 Diodo zener 5v1
Z3 Diodo zener 5v1

S1 pulsador
S2 pulsador
S3 pulsador
SV1 conmutador
SV2 conmutador
SV3 Conexiones al exterior GND y +Vcc.
SV4 Borne de entradas Analógicas.
SV5 Borne de programación
SV6 conmutador
SV7 Puente placa superior del LCD.
SV8 Puente placa superior del LCD.
SV9 Puente para puerto serie
IC1 AT89C52P Micro controlador.
IC2 ADC0808N Convertidor analógico digital.
IC3 LM555N
IC4 7805T Limitador de tensión.
IC5 MAX232

Esquema eléctrico



Programación del microcontrolador Atmel.

SUMARIO

1. Introducción

1.1 Por que usamos C

Aunque tradicionalmente se ha empleado el lenguaje ensamblador para la programación de microcontroladores, los actuales compiladores de C ofrecen un grado tal de optimización que es difícil de superar. Además el empleo de un lenguaje de uso común como es C permite cierto grado de abstracción ya que no hay q conocer el juego completo de instrucciones de cada microcontrolador. Nos bastara pues con conocer el hardware y los recursos de los que dispone el microcontrolado así como el modo de acceder a ellos.

El lenguaje C además permite el uso de ficheros cabecera permitiendo crear pequeñas librerías que pueden ser reutilizadas con gran facilidad en distintos proyectos.

1.2 Keil Micro Vision

Para el desarrollo de este proyecto hemos utilizado el IDE(Entorno de diseño Integrado) de Keil, "Keil Micro Vision". Keil ofrece un implementación altamente cercana a la especificación POSIX de C, lo cual permite que con unos conocimientos generales de C se pueda comenzar a desarrollar rápidamente.

El Keil aparte del compilador incluye un depurador que permite la ejecución del código programado. Esta función es extremadamente útil ya que nos posibilita la ejecución instrucción a instrucción y el visionado del contenido de la memoria en todo momento.

2. módulos del programa

Pasaremos ahora a detallar los distintos módulos o librerías de los que se compone el proyecto. Cada librería esta compuesta por un fichero de cabecera .h y su correspondiente fichero de código .c . Cada modulo tiene un cometido especifico, siendo generalmente el acceder a cierto hardware o recurso. Estos módulos funcionan a modo de "driver" ofreciendo unas funciones que permiten abstraerse del hardware que hay debajo. También hay módulos que nos ofrecen funciones de mas alto nivel como, son el modulo visual o el del protocolo.

La idea del empleo de pequeños módulos o librerías es el que estos puedan ser útiles para otros proyectos. Es por ello que pueden incluir funciones que en este proyecto no se utilicen pero que puedan ser útiles en un futuro.

Comenzaremos con las rutinas que están mas estrechamente ligadas al hardware, para pasar a las de mas alto nivel y finalmente llegar al programa general. Como se vera el programa principal es realmente simple, ya que todas las funciones complicadas se han ido resolviendo en los distintos módulos.

Antes de comenzar con el análisis del código decir que para hacer uso del un modulo y sus funciones únicamente es necesario incluir la siguiente línea al comienzo del código:

```
#include "modulo.h"
```

En ese fichero .h deben estar declaradas todas las funciones que se deseen sean accedidas desde fuera y las variables que deban ser publicas.

2.4 Interfaz con el puerto serie

Este modulo es un modulo típico y muy útil para todo tipo de proyectos. Esta formado por los ficheros rs232.h y rs232.c. En el se incluyen las rutinas para el manejo del puerto serie(UART) con interrupción tanto como sin ella. Es por ello que todas las rutinas(inicialización, envió y recepción) aparecen duplicadas para las 2 maneras de trabajar. Si optamos por utilizar interrupciones aparecen una serie de flags para que el programa principal sepa en que estado se encuentra el puerto serie. Se incluye también la función que se hace cargo de la petición de interrupción del puerto.

Por ultimo se incluye una función que envía una cadena entera y una función que devuelve lo recibido por el puerto serie y que únicamente tiene fines de comprobación del puerto.

Un parámetro importante a la hora de usar el puerto serie, que recordemos que es asíncrono, es la velocidad de transmisión, la cual se define en:

```
#define BAUDRATE      57600
```

Y además no hay que definir la frecuencia del cristal que usamos:

```
#define OSC           11059200L      //Frecuencia del cristal
```

Con estos parámetros se calculara el valor necesario a introducir en el contador 1.

Veamos como se inicializa el puerto serie usando interrupciones y la función de gestión de la interrupción necesaria para enviar y recibir con interrupciones. El código esta altamente comentado y es por ello que no se harán mucho mas comentarios.

```
-----------------------------------------------------------------------------------------------------------
//Inicializamos el puerto serie al BAUDRATE y con interrupciones
void init_serie_int(void)
{
F_NUEVODATO=0;      //Bajamos los flags
F_MANDADO=0;        //Bajamos el flag de carácter mandado
EA=0;               //Deshabilitamos las interrupciones mientras configuramos
ET1=0;              //Enmascara la interrupción del timer 1
TR1=0;              //Timer 1 parado
PCON |= 0x80;       //Configuración de la uart
SCON = 0x52;  //Configuración del puerto serie
TMOD &= 0x0F;       //Configuración del timer 1 para la uart
TMOD |= 0x20;
TH1 = (UCHAR)  (256 - (FAUX / BAUDRATE));   //cargamos el timer con el valor adecuado
                                            //la velocidad de trabajo elegida
TR1 = 1;            //Timer 1 en marcha

//Configuración de la interrupción de la uart
ES=1;           //Habilitamos la interrupción serie
PS=0;           //Con prioridad baja
EA=1;           //Habilitamos interrupciones globalmente
}//fin de init_serie_int()
-----------------------------------------------------------------------------------------------------------
```

Después de inicializar el puerto (hay que llamar a la rutina de inicialización en el programa principal antes de utilizarlo) definimos que función se hará cargo cuando se reciba una interrupción, en este caso la numero 4.

```
-----------------------------------------------------------------------------------------------------------
//Cuando el puerto serie crea una interrupción ejecutamos esta función
void int_serie() interrupt 4                //Esta función se ejecutara cuando el micro
{                                           //reciba una interrupción numero cuatro
if (RI==1)     //Si la interrupción ha sido debida a la recepción de un caracter...
        {
        RI=0;                   //Bajamos el flag ya que hay que hacerlo por software
        recibir_serie_int();    //Llamamos a la función que recibe el dato
        F_NUEVODATO=1;          //y activamos el flag de nuevo dato para que el programa
                                //principal sepa que tiene un nuevo dato
        }
if (TI==1)     //Si la interrupción es debida a la finalización del envió...
        {
        TI=0;                   //bajamos el flag
        F_MANDADO=1;            //Indicamos a través del flag mandado
                                //que ya hemos finalizado el envió
        }
}
-----------------------------------------------------------------------------------------------------------
```

Como se puede ver se hace uso de dos flags(F_NUEVODATO y F_MANDADO). Esto se hace para que el programa principal sepa en que estado se encuentra el puerto. Se ha llamado en la recepción a la función *recibir_serie_int(),* veamos lo que hace:

```
---------------------------------------------------------------------------------------------------------
void recibir_serie_int(void)
{
      dato_en_serie=SBUF ;
      salir=1;
}
---------------------------------------------------------------------------------------------------------
```

Como se ve es una función sencilla que únicamente almacena el contenido del buffer serie en una variable(que deberá ser global para que se pueda acceder a ella desde cualquier parte del programa) y activa el flag de salir para que si no encontramos en la parte de envió del protocolo demos por finalizado el mismo y atendamos a la nueva petición.

Para finalizar ojeemos como se mandaría un dato:

```
---------------------------------------------------------------------------------------------------------
void enviar_serie_int(UCHAR dato)
{
      while( F_MANDADO == 0); //Esperamos si aun hay un dato siendo enviado
      SBUF=dato;                              //Pasamos el caracter que queremos mandar al buffer serie
      F_MANDADO=0;              //Indicamos que se esta mandado un dato
}
---------------------------------------------------------------------------------------------------------
```

2.2 Rutinas de retardo y temporización

En este modulo se encuentran las funciones encargadas de manipular los timer con fines de control del tiempo. Se incluye una función de retardo genérica y una especifica para generar el retardo necesario para un muestreo a una frecuencia dada. así mismo se incluyen las funciones de inicialización del timer con interrupción y la función encargada de atender a la misma.

Para el el control temporal usaremos el timer 0, recordemos que el timer 1 es usado para el puerto serie. Para usar el timer 0 lo pondremos en modo 1(16bits) y activaremos las interrupciones necesarias(la especifica del timer y la general).

Veamos como es esta inicialización:

```
---------------------------------------------------------------------------------------------------------
void init_timer_int(void)
{
//Configuramos el timer
TR0 = TF0 = 0;          //garantizar las condiciones iniciales adecuadas, timer parado
TMOD |= 0x01;
TMOD &= 0xF1;                   //Timer en modo1, temporizador, control soft.

//Pasamos a habilitar las interrupciones
ET0=1;                          //Habilitamos la interrupción del Timer0
EA=1;                           //Habilitamos interrupción global

}
---------------------------------------------------------------------------------------------------------
```

Ahora vamos a ver como hay que utilizar el timer para lograr los muestreos a una frecuencia dada. Lo primero es pasar de frecuencia de muestreo a tiempo de muestreo. Lo siguiente es pasar ese tiempo de muestreo a numero de ciclos de maquina con el cristal que se este usando, un ciclo de maquina son 12 pulsos de reloj en este microcontrolador. Para terminar como ese numero de ciclos a cumplir puede ser mayor que el máximo que puede contener el timer (16bits, por lo tanto 65535) habrá que calcular el numero de desbordes que deberá cumplimentar. Todo esto en código queda de la siguiente manera:

```
---------------------------------------------------------------------------------------------------------
void init_timer_freq_s(UCHAR frecuencia)
{
        unsigned long tiemp_sample;     //Declaramos algunas variables auxiliares
        UINT resto,div,resto_div;       //Algunas UNIT, 16 bits
        F_TIMER=0;                      //bajamos la flag del timer
        frecuencia=frecuencia*10;       //Por el puerto serie nos llega
                                        //la décima parte de la frec. deseada

        //Lo primero es pasar de frecuencia a tiempo, ósea hacer la inversa.
        tiemp_sample=(unsigned long)1000000/frecuencia ;
        //Esto nos devuelve en microsegundos el tiempo de muestreo

        //Ahora tendremos q pasar el tiempo de Ts de us a ciclos de maquina
        //1 ciclo maq= 1,085069444444444444444444444444 us (11,0592MHz)
        tiemp_sample=(unsigned long)(tiemp_sample*1000)/1085;

        //Calculamos el numero de veces que deberá desbordar
        //el timer si es que no es suficiente con una
        div=tiemp_sample/65536;
        desbordes = div + 1; /* incluye desborde del resto */

        //Y también calculamos el resto que deberá contar después de los desbordes necesarios`
```

```
        resto_div=tiemp_sample%65536;
        resto = 65536 - resto_div;

        //Empezamos por contar el resto y luego seguiremos con los desbordes necesarios.
        TL0 = resto & 0x00FF;
        TH0 = resto >> 8;
        TR0 = 1;                    //Echamos el timer a correr
}
-------------------------------------------------------------------------------------------------------------
```

Y para terminar observemos la función que se encarga de la gestión de la interrupción del puerto serie. En esta función iremos decrementando el numero de desbordes hasta que este sea cero y entonces subiremos las flag que indica que se ha cumplido el tiempo de muestreo y se puede iniciar un nuevo muestreo.

```
-------------------------------------------------------------------------------------------------------------
void InteTimer_0() interrupt 1  //Rutina de interrupción del timer0
{
        if(desbordes--==0)              //Decrementos los desbordes y si es cero activamos las flag
        {
                F_TIMER=1;
        }
}
-------------------------------------------------------------------------------------------------------------
```

2.3 Interfaz del conversor analógico-Digital

Este modulo compuesto por lo ficheros adc0808.h y adc0808.c contiene todas las rutinas necesarias para el uso del conversor ADC0808. Para ello el conversor ha sido conectado con el microcontrolador de la siguiente manera:

Bus de datos de ADC:	Puerto 0 del microcontrolador
Bus de direcciones del ADC:	Los 3 bits de menos pesos
línea de START(Inicio de la conversión):	BIT 3 del Puerto 1
línea de ALE(Habilitar lectura de dirección):	BIT 4 del Puerto 1
línea de OE(Habilitar salida):	BIT 5 del Puerto 1

Conexión que es declarada en el fichero adc0808.h. Como se puede ver no se ha empleado la línea que indica el fin de la conversión. En su lugar se empleara un tiempo de espera de tal manera que esperaremos el tiempo suficiente para el el ADC termine la conversión.

Pasemos a ver las funciones. Las funciones *inicio*() y *dirección*() únicamente se encargan de mandar un pulso por la líneas de START y ALE.

La función *leer_dato* recibe como parámetro el numero del canal y devuelve un UCHAR con el dato leído del ADC en el canal correspondiente. La secuencia es sencilla y es la indicada en la hoja de datos del conversor:

*Se escribe en el bus de direcciones los 3 bits de menos peso del parámetro canal

*Se manda un pulso ALE para indicar que el ADC ya tiene la dirección disponible

*Se le manda seguidamente un pulso START para que comiendo la conversión

*Se espera un tiempo a que la conversión haya finalizado, ya que no disponemos de la línea EOF que nos lo indicaría

*Se activa la salida del ADC poniendo en alto OE

*Se lee el valor y se baja OE

Todo esto en código C queda de la siguiente manera:

```
-----------------------------------------------------------------------------------------------------------
UCHAR leer_dato(UCHAR canal)
{
UCHAR i,var;

DIRE_ADC &= 0xF8 ;          //Usamos una mascara ya que solo nos interesan
DIRE_ADC |= (canal&0x07);   //los 3 bits de menos peso como dirección

dirección();                //Hacemos que el ADC nos lea la dirección
inicio();                   //Le decimos que inicie la conversión
while(i++);                 //aprox. 255us de espera,
                                    //mientras se lleva acabo la conversión
OE=1;                               //Activamos la salida

i=220;
while(i++);                         //aprox. 35 us de espera, para que se estabilice la salida

var=VALOR_ADC;                      //Almacenamos el valor en var

OE=0;                               //Desactivamos la salida

return(var);                        //Devolvemos el valor de var
}
-----------------------------------------------------------------------------------------------------------
```

Se incluyen además otras 2 funciones: *barrido*() y *barrido_con_mascara*() las cuales realizan un barrido de los 8 canales. Para ello se hace uso de la función genérica de lectura de un canal.

Veamos por ejemplo el barrido con mascara.

```
-------------------------------------------------------------------------------------------
void barrido_con_mascara(UCHAR * buffer, UCHAR MASQ)
{
    UCHAR i;
    for (i=0;i<8;i++)           //Vamos a leer los 8 canales en principio
    {
        if (MASQ^1==1)          //Si la posición correspondiente esta a uno no leemos
        {
            buffer[i]=0x00;     //ponemos a cero el correspondiente dato del buffer
        }
        else                            //Sino, leemos el canal y metemos el valor en el buffer
        {
            buffer[i]=leer_dato(i);
        }
        MASQ>>1;                        //Rotamos un BIT a la derecha para en la siguiente
                        //iteración comprobar la mascara del siguiente canal
    }
}//Fin barrido con mascara
-------------------------------------------------------------------------------------------
```

2.4 Interfaz del LCD a nivel hardware

En este modulo se encuentran las funciones que acceden directamente al hardware del LCD. El LCD usado es uno de 16 caracteres y 2 líneas. Este LCD permite 2 tipos de operación: con 8 líneas para datos o con solo 4. En nuestro programa usaremos el modo de 4 líneas de dato mandando sucesivamente los 2 nibbles del byte del caracter. El empleo de este modo nos permite disminuir el numero de pistas siendo solo necesarias 6, 4 para datos y 2 para control.

En cuanto a líneas de control se utilizaran solo 2 de las 3 disponibles ya que la línea que permite entrar en modo lectura del LCD no a utilizaremos. Resumiendo, se usaran 6 líneas conectadas de la siguiente manera al microcontrolador:

Control Dato/Comando:	BIT 7 del Puerto 2
Enable:	BIT 2 del Puerto 2
4 líneas para datos	Los 4 bits de menos peso del Puerto 2

Esta asignación se encuentra dentro del fichero cabecera "lcd_4bit.h" dentro del cual también se definen las siguientes funciones:

```
---------------------------------------------------------------------------------------------------------
        void init_lcd (void);
        void lcd_envia (UCHAR d);
        void cursora_xy(UCHAR x,UCHAR y);
        void lcd_caracter(UCHAR c);
        void display(UCHAR *msg);
        void habilita();
        void lcd_clrscr (void);
        void lcd_cursoron (void);
        void lcd_cursoroff (void);
        void poner_linea_1(void);
        void poner_linea_2(void);
---------------------------------------------------------------------------------------------------------
```

Explicaremos en detalle solo *init_lcd*() y *lcd_envia*() ya que el resto de funciones hacen uso de ellas y sus nombre son suficientemente explicativos.

*** init_lcd() :**

En esta función se sigue la secuencia de inicio indicada en la documentación del LCD.

Seguidamente se configura para el uso de solo 4 bits de datos, se apaga el indicador del cursor, desactivamos el desplazamiento del LCD y borramos la pantalla.

```
---------------------------------------------------------------------------------------------------------
void init_lcd (void)
{
unsigned int c;
//Ciclo de inicialización indicado en la documentación el LCD
for(c=0;c<10000;c++);
PORT_LCD &= 0X40;
PORT_LCD |= 0X03;
habilita();
for(c=0;c<1000;c++);
habilita();

for(c=0;c<1000;c++);
habilita();
for(c=0;c<1000;c++);
//Fin del ciclo de inicialización. Pasamos a la configuración

lcd_envia(0X28);        /* bus de 4 bits */
lcd_envia(0X0C);        /* cursor apagado */
lcd_envia(0X06);        /* lcd no desplazado */
lcd_envia(0X01);        /* limpiar pantalla */
}
---------------------------------------------------------------------------------------------------------
```

*** lcd_envia()**

Esta función se encarga de mandar un dato, un byte, al LCD. Se le envía el dato separado en los 2 nibbles. Para ello se manda primero la parte alta, se rota el dato 4 posiciones y se envía la parte baja. Posteriormente se manda un pulso de ENABLE y espera el tiempo indicado en las especificaciones del LCD.

```
---------------------------------------------------------------------------------------------------------
void lcd_envia (unsigned char d)
{  int i;
   PORT_LCD &= 0XF0;            //Primero limpiamos la parte de puerto que vamos a usar
   PORT_LCD |= d >> 4; //mandamos primero el nibble alto y rotamos el dato 4 posiciones,
                                //el nibble bajo pasa a estar ahora en la mitad alta del byte

   habilita();                  //mandamos un pulso de ENABLE para que lea el nibble

   PORT_LCD &= 0XF0;            //Volvemos a limpiar el puerto antes de mandar
   PORT_LCD |= d & 0x0F;        //mandamos el nibble bajo, que ahora esta en la posición alta

   habilita();                  //mandamos un pulso de ENABLE para que lea el nibble

   for(i=0; i<300;i++);       //Esperamos un poco
   }
---------------------------------------------------------------------------------------------------------
```

El resto de funciones se encarga de modificar el estado del LCD haciendo para ello uso de las anteriores funciones. Sus nombres deberían ser suficientemente aclaratorios. Lo único que hacen es enviar el comando o series de comando necesarios y explicados en la hoja de datos.

2.5 Modulo de representación de datos en el LCD

Este modulo se encarga de la representación de los datos en el LCD de una manera amigable así como de la ínter actuación con el usuario a través de los 2 botones. En este modulo se incluye una rutina para pasar de un valor en binario(UCHAR) a su representación decimal en el intervalo 5-0v. así mismo se incluye una matriz para la conversión de estos dígitos decimales a sus correspondiente códigos ASCII. Veamos estos primero.

```
---------------------------------------------------------------------------------------------------------
//Matriz para pasar de dígitos hexadecimales a decimales ASCII
UCHAR code conversión[]={0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
---------------------------------------------------------------------------------------------------------
```

Como se ve para pasar a codificación ASCII usamos una matriz. Esta matriz es llamada con el numero decimal que deseamos como posición y nos devuelve su correspondiente código ASCII.

Para convertir el valor binario (0-255) al voltaje que representa se multiplica por el escalón quántico y se van sacando sucesivamente el valor de la unidad, el decimal y la centésima. Con estos valores se invoca a la matriz de conversión ASCII y se envía como carácter al LCD.

```
---------------------------------------------------------------------------------------------------------
//Esta función coge un valor y muestra el voltaje correspondiente por el LCD
//Nota: referenciado a 5V!!!!
//escalón quántico 19,6mV, por eso se multiplica 196(para no trabajar con decimales)
void mostrar_voltage(UINT dato)
{
lcd_caracter( conversión[dato*196 / 10000]);                //Mostramos las unidades
lcd_caracter(0x2C);                                                    //Mostramos la coma separadora de las
unidades y los decimales
lcd_caracter( conversión[(dato*196 % 10000) / 1000]);     //Mostramos las décimas
lcd_caracter( conversión[dato*196 % 1000/100]);             //Y finalmente las centésimas
}
---------------------------------------------------------------------------------------------------------
```

2.6 Protocolo de comunicación con el PC

Como el microcontrolador tenia que comunicarse con el PC por puerto serie nos vimos en la necesidad de desarrollar un pequeño protocolo de comunicaciones.

Este protocolo es extremadamente sencillo. Nada mas arrancar el microcontrolador se queda a la espera de comandos. El comando esta formado por 4 bytes que representan los siguientes parámetros: Modo de funcionamiento, Mascara de canales a leer, Frecuencia de muestreo y Numero de muestras a tomar. No todos los parámetros tiene importancia en todos los modos. Se han contemplado los siguientes modo de funcionamiento:

Modo Visual:
En este modo de funcionamiento el microcontrolador lee todo lo rápido que puede y muestra el valor de 2 canales en el LCD. El usuario puede desplazarse por lo distintos canales con los pulsadores. Este modo permite un funcionamiento autónomo separado del PC.

Modo Continuo:
En este modo el microcontrolador lee a la frecuencia indicada los canales

indicados de forma continua hasta que le llega un nuevo comando.

Modo de Muestras Finitas:
En este modo el microcontrolador lee solo la cantidad de muestras especificada a la frecuencia deseada y de los canales indicados en la mascara.

Las funciones de manejo del protocolo se pueden separar en 2: La recepción de comando y el envió o visualización de datos.

Veamos el código que se encarga de la recepción de comandos. Es una maquina de estados que va recibiendo sucesivamente los parámetros de configuración.

```
---------------------------------------------------------------------------------------------------------
void recibir(void)
{
if (F_COM==1)
        {
        COM=dato_en_serie;
        F_COM=0;            //Ya hemos recibido el comando...
        F_MASQ=1;           //...y lo siguiente que esperamos es la mascara
        }
        else
        {
        if (F_MASQ==1)
                {
                MASQ=dato_en_serie;
                F_MASQ=0;           //Ya hemos recibido la mascara...
                F_FREQ=1;           //...y lo siguiente que esperamos es la frecuencia
                }
                else
                {
                if (F_FREQ==1)
                        {
                        FREQ=dato_en_serie;
                        F_FREQ=0;           //Ya hemos recibido la frec....
                        F_N_SAMPL=1;//...lo siguiente será el numero de muestras
                        }
                        else
                        {
                        if (F_N_SAMPL==1)
                                {
                                N_SAMPL=dato_en_serie;
                                F_N_SAMPL=0;//ya tenemos el numero de muestras
                                //Si llega algo será un comando.
                                //Además como ya tenemos los 4 parámetros
                                //podemos empezar(subir la flag de arranque)
                                F_COM=1;F_ARRANCAR=1;
```

```
                                        }
                                }
                        }
                }
}//Fin de recibir
```

Echemos un vistazo a la parte de envió de datos del protocolo. Como es un poco largo lo iremos analizando en cada modo de funcionamiento.

```
void mandar(void)
{
UCHAR i;
UCHAR cont;
if (COM==MOD_N_SAMPL) //Si tenemos que funcionar en modo numero de muestras
{
lcd_clrscr();                   //Limpiamos el LCD
display("Modo N.S.");           //Sacamos un texto explicando en que modo estamos
poner_linea_2();        //vamos a la líneas 2 del LCD
mostrar_parametros();   //Mostramos los parámetros que hemos recibido
init_serie_int();       //Inicializamos el puerto serie para enviar con interrupción
init_timer_int();       //inicializamos el timer para controlar el tiempo de muestreo
for (;N_SAMPL>0;N_SAMPL--)      //Mientras queden muestras por mandar
{
        init_timer_freq_s(FREQ);        //Cargamos la frecuencia que estamos usando
        barrido_con_mascara(MASQ); //Leemos los canales que nos han pedido
        for(cont=0;cont<8;cont++)
        {
                enviar_serie_int(buffer[cont]); //Mandamos los 8 bytes de los 8 canales
        }
        while (F_TIMER==0);     //Esperamos hasta que se cumpla Ts(tiempo de muestreo)
        }
        return;         //Ya hemos hecho todas las lecturas y salimos
}
```

Esta es la parte del modo de funcionamiento de muestras finitas. Como se ve se hace un barrido de los 8 canales con mascara y a continuación se mandan los 8 datos. Esta operación se repite mientras queden muestras por mandar.

Veamos como funciona el modo continuo.

```
-----------------------------------------------------------------------------------------------------------
if (COM==MOD_CONT)
{
lcd_clrscr();
display("Modo continuo");
poner_linea_2();
mostrar_parametros();
init_serie_int();
init_timer_int();

while(salir!=1)              //Mientras no nos llegue un nuevo comando seguimos leyendo
{
        init_timer_freq_s(FREQ);
        barrido_con_mascara(MASQ); //Leemos los canales que nos han pedido
        for(cont=0;cont<8;cont++)
        {
                enviar_serie_int(buffer[cont]); //Mandamos los 8 bytes de los 8 canales
        }
        while (F_TIMER==0); ///Esperamos hasta que se cumpla Ts(tiempo de muestreo)
        //si llega un nuevo dato dejamos de mandar
        //y volvemos a la rutina de recepción de comandos
        if (F_NUEVODATO) { salir = 1;}
}
return;//Hemos recibido un nuevo dato y salimos
}//Fin del if de modo continuo
-----------------------------------------------------------------------------------------------------------
```

En este modo se hace un barrido con mascara y se repetirá infinitamente hasta que llegue un nuevo comando y levante la flag F_NUEVODATO.

Por ultimo veamos el código del modo visual.

```
-----------------------------------------------------------------------------------------------------------
if (COM==MOD_VISUAL)
{
while(salir!=1)
{
        i=ver(i);
        //si llega un nuevo dato dejamos de mandar
        //y volvemos a la rutina de recepción de comandos
        if (F_NUEVODATO) { salir = 1;}
}
return;//Hemos recibido un nuevo dato y salimos
}//Fin del if del modo visual
}//Fin de mandar()
-----------------------------------------------------------------------------------------------------------
```

2.7 Programa principal

El programa principal es muy sencillo. básicamente se inicializa el puerto serie y el LCD, se reciben comandos y se actúa en consecuencia. Lógicamente antes de llegar a la parte de código hay que hacer los include-s pertinentes y declarar unas variables. Es de resaltar el hecha de que muchas de estas variables son declaradas extern. Esto es así porque estas variables son declaradas en otras partes del programa(en los módulos) y no se deben declarar 2 veces.

```
/************************LICENCIA*****************************

Copyright (C) 2003  Jon Latorre Martinez & Fernando Cabrera

Este programa es software libre; usted puede redistribuirlo y/o
modificarlo bajo los términos de la Licencia Publica General GNU
tal y como lo publica la Free Software Foundation en su versión 2.

Este programa se distribuye con la idea de ser útil, pero
SIN NINGUNA GARANTIA

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*********************************************************************/

//Este include es especifico del microcontrolador que usamos
#include <at898252.h>

//Ahora van los includes con nuestras cabeceras
#include "lcd_4bit.h"
#include "visual.h"
#include "rs232.h"
#include "protocolo.h"

#define  UCHAR unsigned char
#define  UINT unsigned int

//Donde almacenamos los datos(en visual.h esta definido como extern)
UCHAR idata buffer[8] ;
```

```
//Texto que se mostrará al esperar comandos
UCHAR code * saludo="Espero Comando";

//Definimos las flags, las declaramos como externas porque ya
//han sido declaradas en otras partes
extern bit F_NUEVODATO;
extern bit F_COM;
extern bit F_MASQ;
extern bit F_FREQ;
extern bit F_N_SAMPL;
extern bit F_MANDADO;
extern bit F_ARRANCAR;
extern bit F_TIMER;
extern bit salir;

//Variables donde almacenamos los datos y comandos

extern UCHAR COM;
extern UCHAR MASQ;
extern UCHAR FREQ;
extern UCHAR N_SAMPL;



UCHAR dato_en_serie;

/******programa principal*********/
void main(void)
{

//Lo primero que esperamos es un comando
F_COM=1;F_MASQ=F_FREQ=F_N_SAMPL=0;
//Inicializamos el puerto serie con interrupción para recibir el comando
      init_serie_int();
//Inicializamos el LCD
     init_lcd();
while(1) //Bucle principal del que no saldremos
      {
      lcd_clrscr();                  //Limpiamos el LCD y mostramos el mensaje de espera
      display(saludo);
      poner_linea_2();
            //Esperamos ha que hallamos recibido el comando y sus parámetros
            while(F_ARRANCAR==0)
            {
                  while(F_NUEVODATO==0);        //Esperamos a que llegue un dato
                  recibir();            //Entonces lanzamos la rutina de recepción del protocolo
                  F_NUEVODATO=0;
```

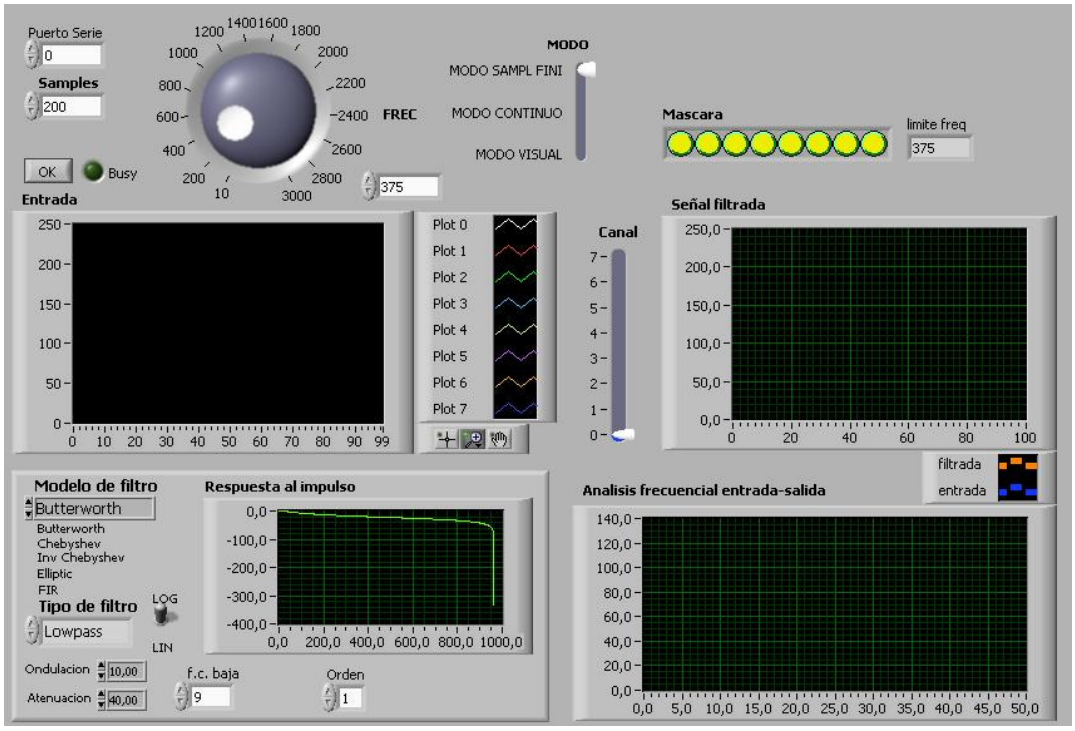
```
            }
            F_ARRANCAR=0;       //Ya podemos arrancar y
                                        //hay q bajar la flag para la próxima ves
            salir=0;
            mandar();                   //Una vez hemos recibido el comando empezamos a mandar

      }//Fin del while continuo donde recibimos comandos y mandamos datos
}
```

PROCESADO DE SEÑALES ANALOGICAS CON LABVIEW

INTRODUCCIÓN:

Mediante el siguiente programa en LabVIEW, se comunica a la placa de adquisición de datos, (en base al protocolo establecido), el control o modo de funcionamiento deseado. Se reciben los datos enviados por la tarjeta. Se muestran las respuestas de diferentes tipos y modelos de filtros FIR e IIR y la opción de elegir uno de ellos modificando sus parámetros. Con el filtro elegido se muestra una comparativa espectral de la señal de entrada-salida. El programa se divide prácticamente en los bloques citados.

PANEL FRONTAL:



En la parte superior se encuentran los controles para el manejo de la placa. La grafica de entrada registra las señales captadas en los diferentes canales del ADC. La parte inferior izquierda nos muestra la respuesta al impulso de los distintos filtros pudiendo ser modificada al variar los parámetros de los mismos, así nos hacemos una idea de cómo vamos a actuar sobre la señal de entrada. Con el selector de canal elegimos cual de las entradas se va a procesar. Las graficas de la derecha nos muestran, arriba la señal tras ser filtrada, y la inferior superpone los análisis espectrales de la entrada y la salida.

En el programa (completo.vi), se distinguen varios bloques:

Bloque de protocolo, comunicacion.vi.

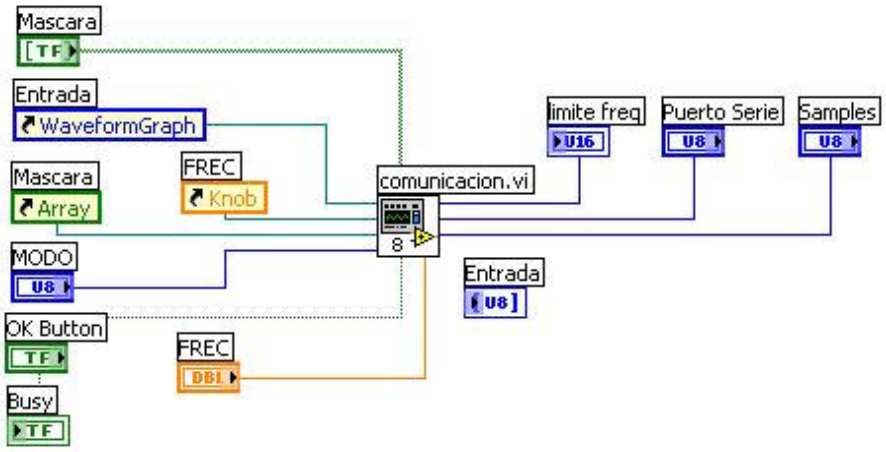
Bloque de filtros-procesado.

Bloque de representación.

insertar diagrama de dependencia de frames y vi´s*

BLOQUE DE PROTOCOLO:

Este bloque está compuesto por comunicación.vi.



Las entradas de esta subVI le proporcionan todos los parámetros necesarios para comunicarle a la placa cómo queremos que muestree la señal de entrada.

Mascara: es un array booleano que indica los canales que queremos muestrear.

Modo: es un control numérico con el que seleccionamos la manera de tomar las muestras: numero finito de muestras; de manera continua; ó modo visual, que solo enseña las lecturas en el LCD de la placa.

FREC: determina la frecuencia de muestreo para todos los canales.

Samples: numero de muestras que toma por canal.

Puerto Serie: Selección del puerto.

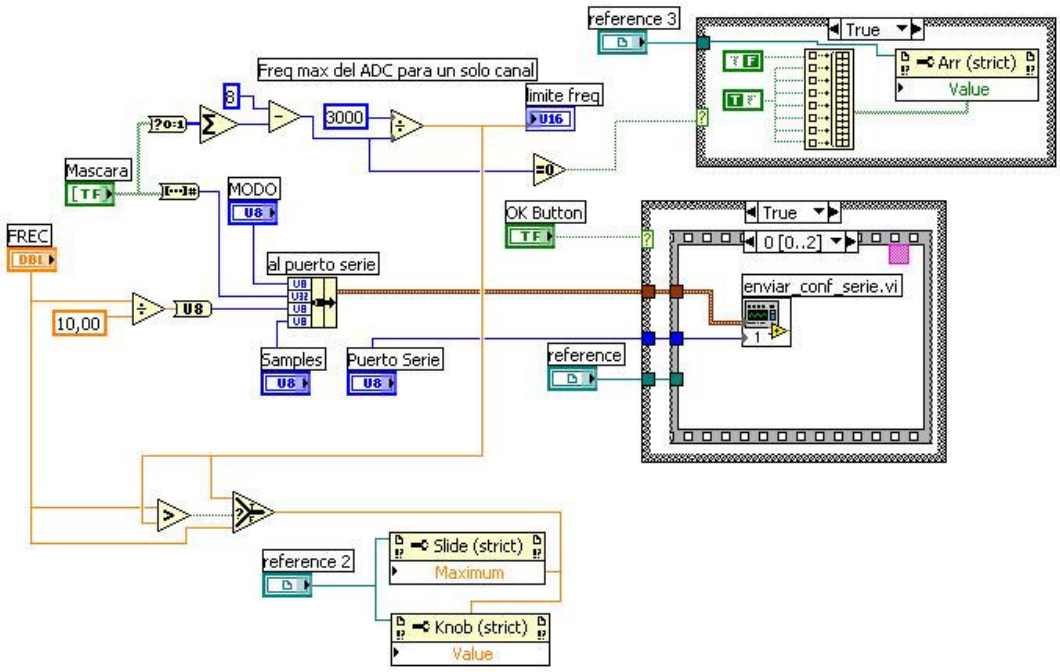
OK Button: al actuar sobre este pulsador se validarán todos los parámetros. El pulsador da un impulso al soltarle, que dura mientras se está ejecutando la subVI.

Las salidas de la subVI son:

Limite frec: entrega un calculo de la frecuencia máxima de muestreo en función del numero de canales elegidos.

Entrada: (es el resultado de esta subVI), son los datos recibidos de la placa por el puerto serie.

Internamente esta subVI está constituida a su vez por otras subVI´s incluidas en una estructura secuence que permite seguir los pasos del protocolo en el orden adecuado.



comunicacion.vi

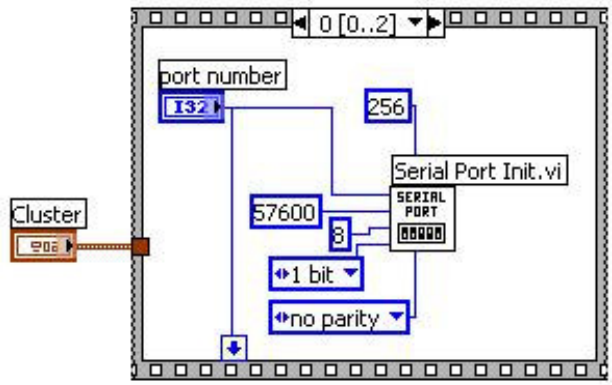
Los “0” contenidos en el array booleano son los canales usados. El array se convierte a decimal, se suman los “1”, (canales no usados), y se restan de 8 para contar el numero de canales seleccionados. La frecuencia teórica máxima es 3000, al dividirla entre el numero de canales se obtiene la máxima frecuencia de funcionamiento. Si no se ha seleccionado ninguno, se entra en el caso verdadero del case superior, que se encarga de forzar a 0 el primer canal (elegirlo) y modificar la mascara mediante reference 3.

Al pulsar el botón de OK se entra en el caso verdadero del case inferior, mientras no se pulse entra en el falso, que no hace nada.

El valor del MODO, la Mascara (en decimal), un décimo de la frecuencia (en decimal), y el numero de samples, se meten en un cluster.

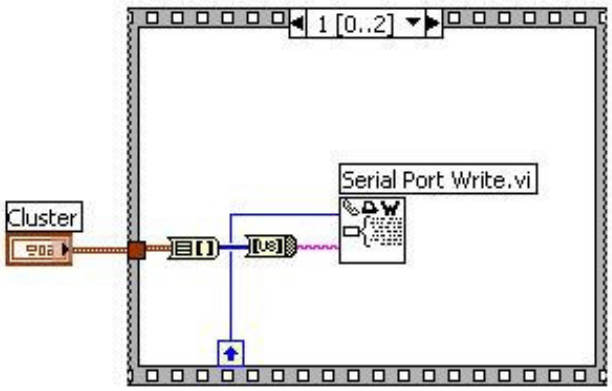
Se ha programado un limitador que actúa sobre el control de la frecuencia de muestreo mediante reference 2 para que no se pueda sobrepasar la frecuencia máxima calculada.

•En el frame 0 de comunicacion.vi se ejecuta la subVI enviar_conf_serie.vi. Esta subVI envía el protocolo seguido por el microcontrolador,(que es la forma en que queremos realizarlas medidas), para ello realiza las dos siguientes secuencias:



Frame 0 de enviar_conf_serie.vi

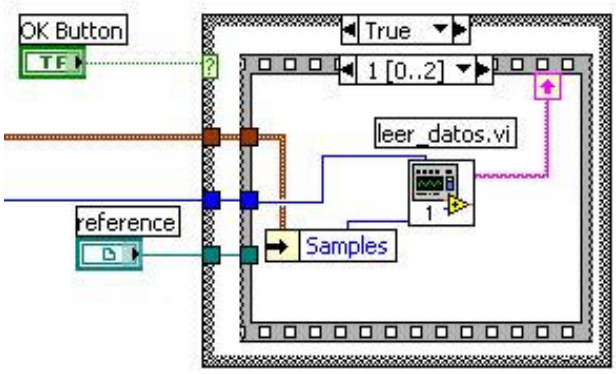
- Inicializamos el puerto seleccionado por el control numérico “port number” (0=COM 1), con 8 bits a 57600 baudios, reservando un buffer de 256 bytes (suficientes para nuestro propósito), sin control de flujo (flow control) ni paridad, y con un bit de stop.



Frame 1 de enviar_conf_serie.vi

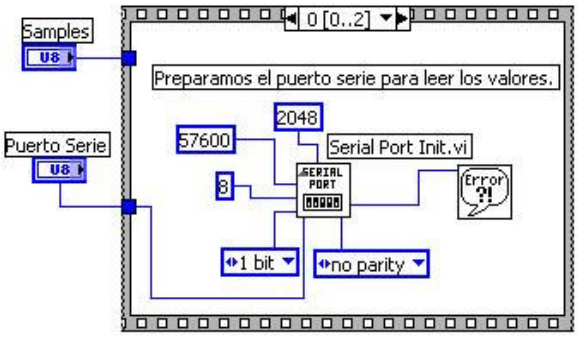
- En este segundo frame se escribe el comando de control para el microprocesador en el bus serie. Del frame anterior le llega el numero del puerto. Y del exterior le llega el cluster con los parámetros modo, mascara, frec., y samples. El cluster se transforma en un byte array de enteros sin signo y luego en un string para ser escrito en el puerto.

•Volviendo a la VI de configuración, en el frame 1 se ejecuta la subVI leer_datos.vi. Esta subVI inicializa el puerto serie, lee los datos del puerto enviados por el microcontrolador, y libera el puerto. Le siguen entrando el cluster con la configuración y el puerto elegido, pero del cluster sólo aprovecha la línea con el valor de los samples.



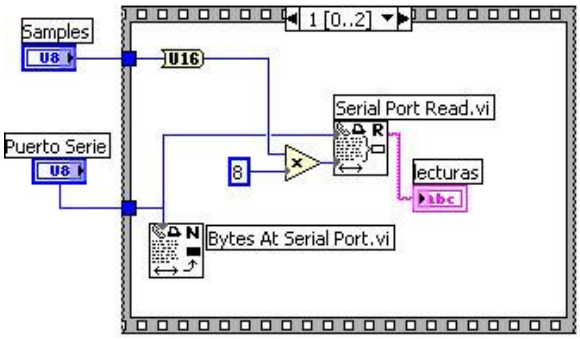
Frame 1 de comunicacion.vi

Para ello realiza las tres siguientes secuencias:



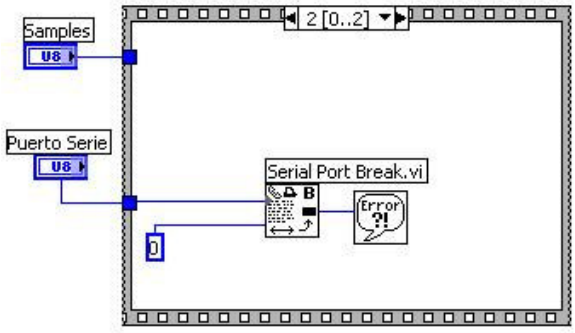
Frame 0 de leer_datos.vi

- Inicializamos el puerto seleccionado por el control numérico “port number” (0=COM 1), con 8 bits a 57600 baudios, reservando un buffer de 2048 bytes (suficientes para los datos que se puedan recibir), sin control de flujo (flow control) ni paridad, y con un bit de stop.



Frame 1 de leer_datos.vi

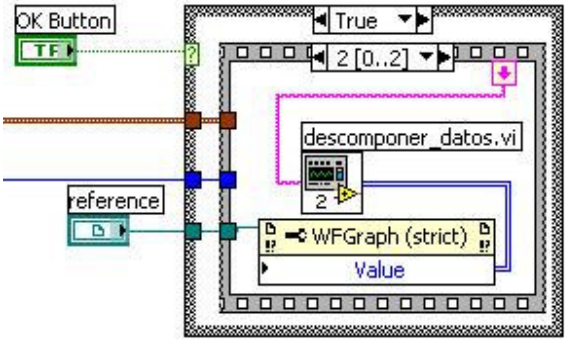
- Se manda leer del puerto tantos bytes como numero de muestras queremos multiplicados por los ocho puertos. Detecta los bytes que quedan por leer en el puerto. La lectura se lleva a un indicador tipo string (que está oculto en el panel frontal).



Frame 2 de leer_datos.vi

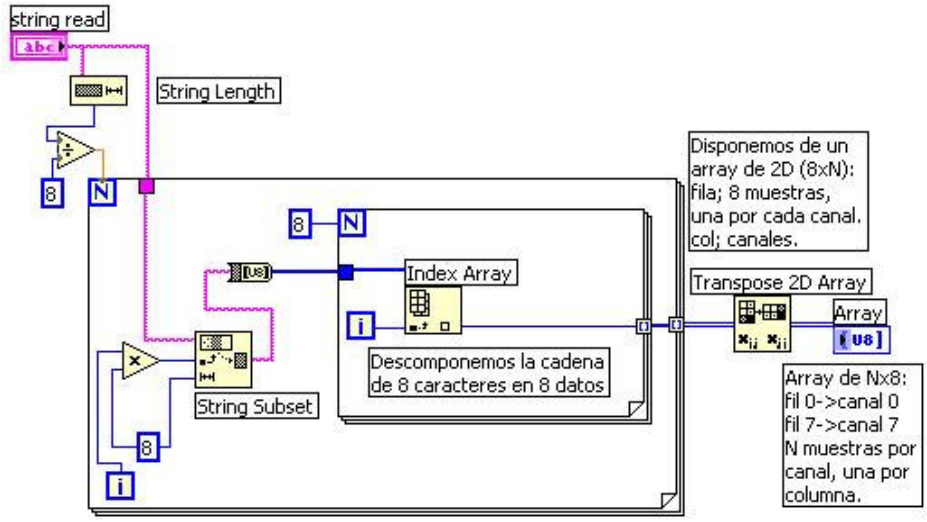
- Finalmente liberamos el puerto.

•Volviendo a la VI de configuración, en el frame 2 se ejecuta la subVI descomponer_datos.vi. Esta subVI fragmenta el string con las lecturas recibidas, para extraer los datos correspondientes a cada canal.



Frame 2 de comunicacion.vi

Internamente la subVI descomponer_datos.vi es como sigue:

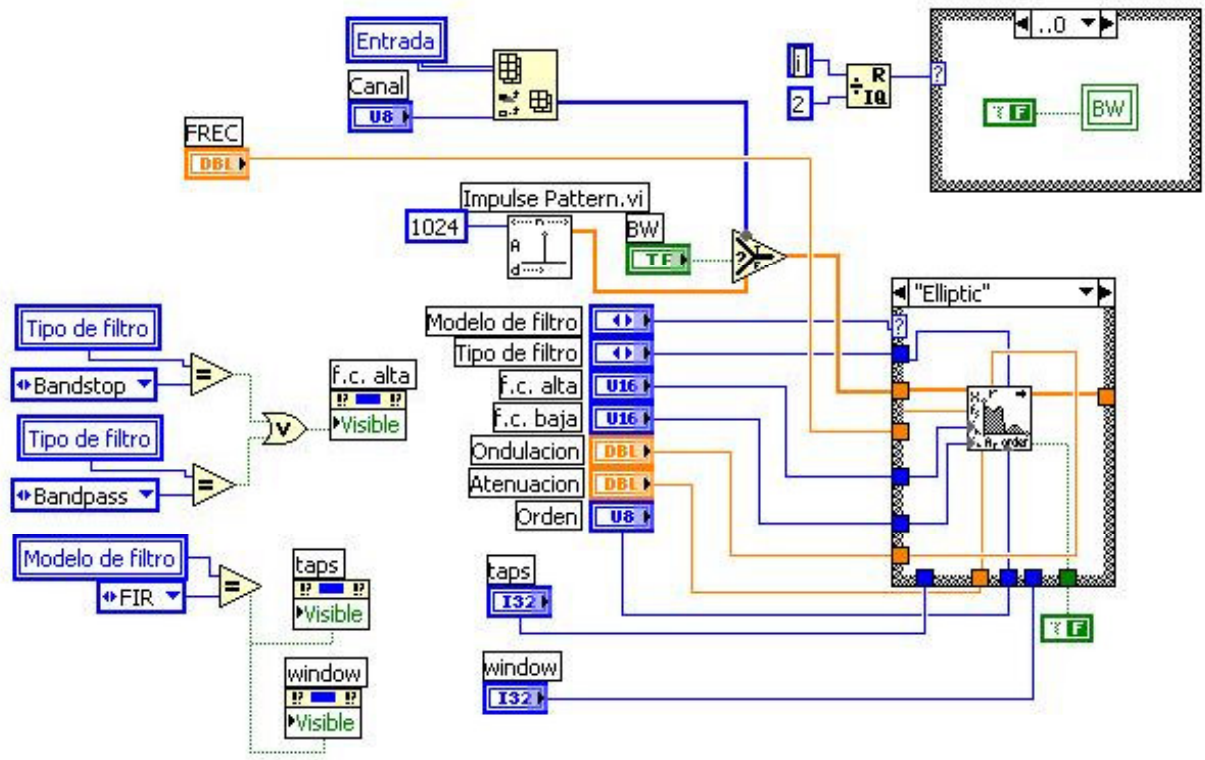


descomponer_datos.vi

A esta Vi se le pasa un string con todos los datos leidos ene l puerto serie. El for

externo se divide en grupos de 8 caracteres(el numero de canales) y en tantos grupos como muestras se han pedido. En el for interno estos grupos de 8 caracteres, tras convertidor a sus respectivos valores numéricos, son almacenados en una matriz. Con ello a la salida del for externo tenemos un array de 8xN, numero de canales por número de muestras, pero nos interesa tener lo contrario, Nx8 para luego poder procesarla fácilmente, es por ello que transponemos la matriz para que se ajuste a nuestras necesidades.

BLOQUE DE FILTROS:



En este bloque se incluyen los distintos tipos de filtro que podemos aplicar a la señal captada.

el elemento central es un case en el cual podemos observar los distintos tipos de filtro(butterword.....). El tipo de filtro se elegida en base al valor del modelo de filtro.

Dentro de cada modelo podemos elegir el tipo de filtro entre 4 posibles: De paso bajo, de paso alto, rechazo de banda y pasa banda. evidentemente también podremos elegir las 2 frecuencias de corte, siendo la frecuencia alta únicamente usada en los 2 últimos tipos de filtro nombrados.

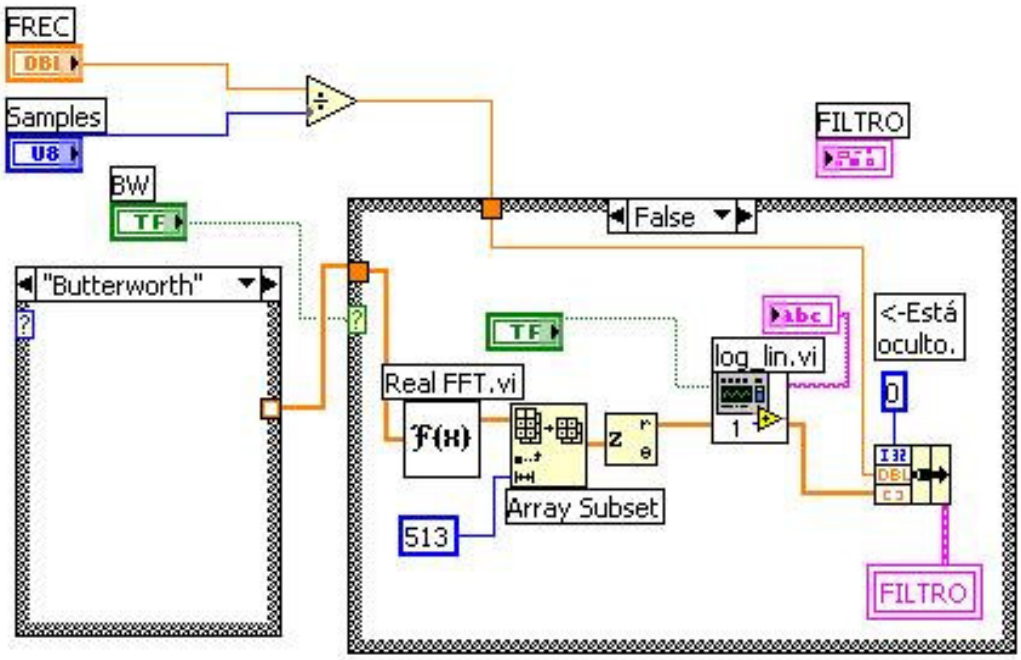
Se pueden también ajustar otros parámetro(atenuación, ondulación, taps) y elegir el tipo de ventana para el filtro FIR.

Como no todos los parámetros son validos para todos los modelos de filtro, los controles innecesarios desaparecen según el filtro elegido. Esto se logra a través de los "properti nodes" de la izquierda.

En la parte derecha alta puede verse un case. Su funcionalidad no es mas que la de generar un cambio continuo en una variable boleana(falso en caso de que el numero de iteración sea par y verdadero en el contrario).Esta boleana pulsante se utiliza con el propósito de que en cada iteración se refresque por un lado la señal de salida filtrada y por otro la respuesta al impulso del filtro elegido. Es por ello que al filtro de le pasa alternamente la señal captada y un pulso logrado a través de la Vi correspondiente.

la señal de entrada puede ser elegida de entre los 8 canales disponibles a través del "slice" canal.

Siguiendo con la repsuesta del filtro al impulso para representarla se utilza la siguiente programacion:



Como vemos a la respuesta del filtro al impulso se le aplica la transformada rápida de fourier. Posteriormente se selección las primeras 513 posiciones y se pasan a coordenadas polares para proceder a su representación, ya sea línea o logarítmica. Con ello logramos el diagrama de Bode del filtro seleccionado.

BLOQUE DE REPRESENTACION:

En este ultimo bloque se representa el análisis frecuencial de la señal de entrada(del canal seleccionado) y de la señal filtrada. Con ello podemos ver los componentes frecuenciales de los que se componen la señal captada así como el efecto del filtro sobre los mismo. Para ello se hace uso del Vi "Amplitude & Phase espectrum" proporcionado por el propio Labview. Los 2 análisis de muestran en la misma graph después de ser almacenado en un cluster con los correspondientes parámetros de valor inicial y espaciado entre muestras de la matriz.7