UNIVERSIDAD PANAMERICANA

ESCUELA DE INGENIERIA

Con estudios incorporados a la Universidad Nacional Autónoma de México

DISEÑO DE UNA TARJETA DE ADQUISICION DE DATOS APLICADA AL AUDIO

T E S I S

PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

Presenta:

RAUL URBINA VILLALPANDO

Revisor: ING. RODOLFO BRAVO DE LA PARRA

MEXICO, D.F. 1998

Espero que este trabajo sirva como agradecimiento a toda mi familia por el apoyo recibido y motivación para los que vienen después de mí.

Finalmente, un agradecimiento especial para todas las personas que proporcionaron la información y el material utilizados para realizar este trabajo, así como a los que dieron la asesoría y cónocimientos para finalizar mi labor.

INTRODUCCIÓN

CAPÍTULO 1

INTRODUCCIÓN AL PROCESAMIENTO DIGITAL DE SEÑALES

CAPÍTULO 2

AUDIO DIGITAL

INTRODUCCIÓN

El gran desarrollo del Internet lo ha llevado en pocos años ha ser uno de los medios de comunicación más importantes. Ahora muchas ventas se realizan por este medio llevando a las empresas a crear páginas más atractivas para atraer ventas, gastando en el proceso importantes sumas de dinero.

El sonido añade una nueva dimensión a las páginas de Internet. Puede dar a una página un lado competitivo y puede incrementar dramáticamente el interés del visitante, las visitas repetidas y la información retenida. El sonido puede estimular poderosas respuestas emocionales que no pueden activarse de por medio de las palabras escritas.

¿Qué sonido utilizar? Tal vez se necesitará de una voz, una música de fondo, efectos de sonido o ruidos graciosos que añadan el elemento de humor a la página. El diseño de sonido es tanto como el arte del diseño gráfico Es mucho mejor tener sonido apropiado y de gran calidad a uno inapropiado y de baja calidad. Cerremos los ojos. Recordemos cómo la radio integró efectos de sonido para crear imágenes sin el beneficio del reforzamiento visual.

Una de las herramientas para crear sonido de calidad es el del procesamiento digital de señales. Las técnicas digitales se han convertido en el método elegido en el procesamiento de señales gracias a que las computadoras digitales han incrementado su poder y velocidad.

Por esta razón se decidió realizar el diseño de una tarjeta de adquisición de audio. Aunque

existen muchas tarjetas en el mercado éstas tienen generalmente precios elevados, haciéndolas inaccesibles para el público en general y en otros casos adolecen de mala fidelidad. También se buscó desarrollar un diseño sencillo que pudiera aplicarse en otras áreas.

Este trabajo involucra el uso de compuertas lógicas programables (CPLD y FPGA). Esta herramienta puede ser virtualmente en cualquier sistema lógico digital, desde supercomputadoras a instrumentos de mano, de sistemas de dirección de misiles a sintetizadores de guitarra.

El primer capítulo trata sobre las bases del procesamiento digital de señales. Se exponen tipos de señales, así como el análisis de Fourier para señales periódicas y apariódicas.

El segundo capítulo expone las bases al audio digital. Expone temas como "aliasing", "dither" y una pequeña exposición sobre la transformada z, sin tratarla exhaustivamente. Incluye también una pequeña exposición sobre convertidores A/D y D/A y filtros digitales, utilizados para la adquisición de señales de audio.

El tercer capítulo expone los elementos en el diseño, así como su desarrollo. Se exponen temas como los tipos de interfaces hacia la computadora, los distintos elementos de control, las memorias y los convertidores A/D y D/A utilizados, así como el porqué de su utilización.

En el apéndice se incluyen los listados del software, la programación del circuito integrado y documentación técnica de los "chips" que complementan la información expuesta en los capítulos.

CAPÍTULO 1

INTRODUCCIÓN AL PROCESAMIENTO DIGITAL DE SEÑALES

1.1 SEÑALES Y SISTEMAS

Una señal está definida como una cantidad física que varía con el tiempo, espacio o cualquier otra variable o variables independientes. Matemáticamente, describimos una señal como una función de una o más variables independientes[1]. Por ejemplo, las funciones

$s_1(t) = 5t$ **ecuación 1.1.1**

$s_2(t) = 20t^2$

describen dos señales, una que varía linealmente con la variable independiente t (tiempo) y la segunda que varía cuadráticamente con t.

$s(x,y) = 3x + 2xy + 10y^2$ **ecuación 1.1.2**

Esta función describe una señal con dos variable x y y que pueden representar dos coordenadas

espaciales en un plano.

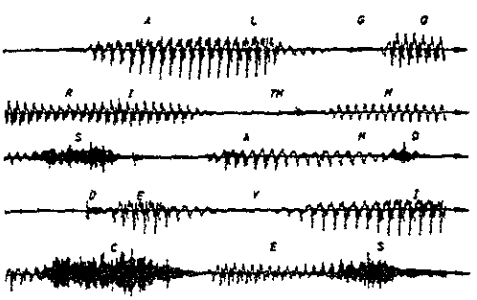
Las señales descritas por las **ecuaciones 1.1.1** y **1.1.2** pertenecen a una clase de señales que se definen precisamente especificando la dependencia funcional en una variable independiente. Pero, existen casos que esta relación funcional se desconoce o son extremadamente complicados para ser de alguna utilidad.

Por ejemplo, una señal de una conversación no puede describirse funcionalmente por expresiones como **ecuación 1.1.1**. En general, un segmento de conversación puede representarse con un grado alto de exactitud como una suma de varias senoides de diferente amplitud y frecuencias, esto es, como

$$\sum_{i=1}^{N} A_i(t)\,\mathrm{sen}(2\pi k F_i(t) + \theta_i(t)) \qquad \textbf{ecuación 1.1.3}$$

donde $\{A_i(t)\}, \{F_i(t)\}$ y $\{\theta_i(t)\}$ son los valores de (posiblemente variable con el tiempo) amplitudes, frecuencias y fases, respectivamente, de las senoides. De hecho, una forma de interpretar el contenido de información en cualquier segmento de tiempo corto de la señal de conversación es el medir las amplitudes, frecuencias y fases contenidas en el segmento corto de tiempo de una señal.

Figura 1. Ejemplo de una señal de voz.



Otro ejemplo de una señal natural es la de un electrocardiograma (ECG). Esta señal provee a un doctor de información sobre la condición del corazón de un paciente. Similarmente, la señal de un electroencefalograma (EEG) provee información acerca de la actividad de un cerebro.

Las señales de conversación, electrocardiograma y electroencefalograma son ejemplos de las señales que contienen información que se desarrollan como funciones de una variable llamada de tiempo. Un ejemplo de una señal que es una función de dos variables independientes es la de una señal de imágenes. Estos son algunos ejemplos de un número incontable de señales naturales encontradas en la práctica.

Asociadas con las señales naturales son los medios por las cuales dichas señales son generadas. Por ejemplo, las señales de voz son generadas forzando aire a través de las cuerdas vocales. Las imágenes son obtenidas exponiendo una película fotográfica a una escena o un objeto. Entonces la generación de señales es usualmente asociada con un *sistema* que responde a un estímulo o fuerza. En una señal de voz, el sistema consiste de las cuerdas vocales el tracto vocal, también llamada cavidad vocal. El estímulo en combinación con el sistema es llamado la fuente de la señal.

Un sistema también se define como un dispositivo físico que desarrolla una operación en una señal. Por ejemplo, el filtro usado para reducir el ruido y la interferencia que corrompe una señal que lleva información es llamado un sistema. En este caso el filtro desarrolla una o varias operaciones en la señal, con el efecto de reducir el efecto del ruido y la interferencia de la señal deseada.

Cuando pasamos una señal a través de un sistema, como lo hacemos en el filtrado, decimos que hemos procesado la señal. En este caso el procesamiento de la señal envuelve el filtrado del ruido e

interferencia de la señal deseada. En general, el sistema se caracteriza por el tipo de operación que ésta desarrolla en la señal. Por ejemplo, si una operación es lineal, el sistema se llama lineal. Si la operación de la señal es no lineal, el sistema se dice que es no lineal, y así sucesivamente. Dichas operaciones son usualmente referidas como el procesamiento de señal.

1.2 CLASIFICACIÓN DE SEÑALES

Los métodos que usamos para procesar una señal o analizar la respuesta de un sistema en una señal dependen grandemente de los atributos característicos de la señal específica. Existen técnicas que se aplican solamente a familias específicas de señales. Consecuentemente, cualquier investigación u procesamiento de señal envuelve de la aplicación específica.

1.2.1 SEÑALES MULTICANALES Y MULTIDIMENSIONALES

Como se explicó en la sección anterior, una señal se describe como una función de una o más variables. El valor de la función (la variable dependiente) puede ser un cantidad real escalar, una cantidad compleja o tal vez un vector. Por ejemplo, la señal

$$s_1(t) = A\,\mathrm{sen}(3\pi t)$$

es un señal con valor real. Pero, la señal

$$s_1(t) = Ae^{j3\pi t} = A\cos(3\pi t) + jA\,\mathrm{sen}(3\pi t)$$

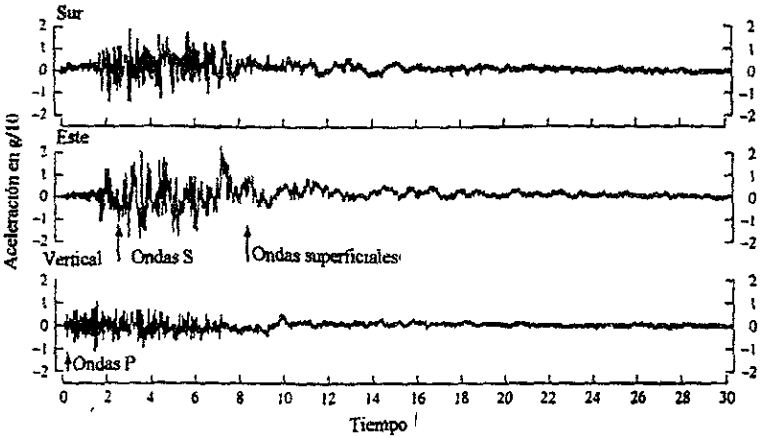
es de valor complejo.

En algunas aplicaciones, las señales son generadas por fuentes o sensores múltiples. En este caso, pueden representarse en forma de vector. La **figura** que se presenta a continuación muestra los tres componentes de una señal vectorial que representan la aceleración de la superficie dado un terremoto. Esta aceleración se debe a tres tipos diferentes de ondas elásticas. La primera onda (P) y la segunda (S) se propagan en el cuerpo de la roca y son longitudinal y transversal, respectivamente. El tercer tipo de onda elástica se llama la onda de superficie, porque se propaga cerca de la superficie terrestre. Si $s_k(t)$, k=1,2,3, denota una señal eléctrica del k-ésimo sensor como una función del tiempo, las 3 señales pueden representarse como un vector $S_3(t)$, donde

$$S_3(t) = \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

Nos referimos a dicho vector de señales como una señal multicanal. En electrocardiografía, por ejemplo, electrocardiogramas de 3 terminales y 12 terminales (ECG) son muy usados en la práctica, y resultan en señales de 3 y 12 canales. Pongamos nuestra atención en las variables independientes. Si una señal es función de una variable independiente, la señal se llama de *unidimensional.* Por otro lado, una señal es llamada de *dimensión M* si su valor es una función de M variables independientes.

Figura 2. Ejemplo de una señal de un sismo.



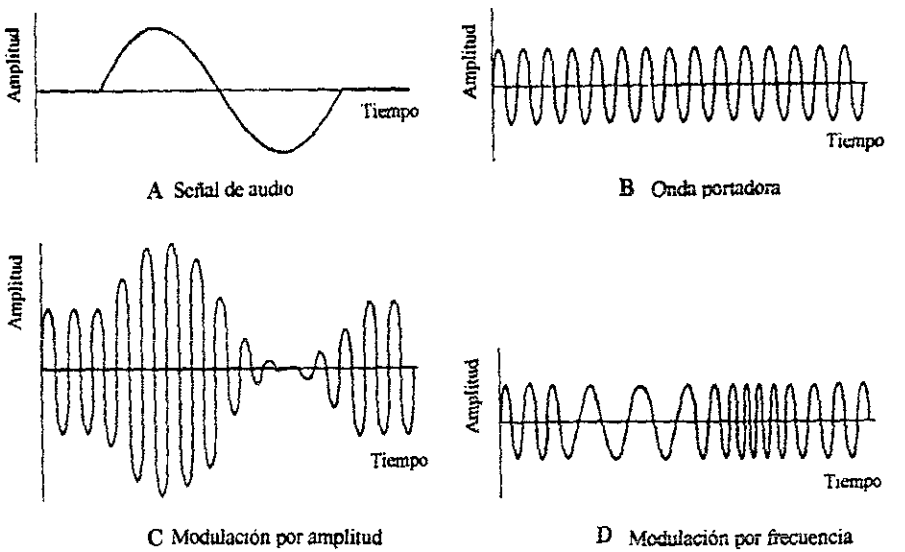
La figura mostrada es un ejemplo de una señal bidimensional, ya que la intensidad o brillo I(x,y) en cada punto es una función de dos variables independientes. Por otro lado, la pantalla de televisión blanco y negro puede representarse como I(x,y,t) ya que el brillo es una función del tiempo. Por lo tanto la pantalla de televisión puede ser tratada como una señal tridimensional. En contraste, la pantalla de una televisión a color puede describirse por tres funciones de intensidad de la forma Ir(x,y,t), Iv(x,y,t) y Ia(x,y,t), correspondiente al brillo de los tres principales colores (rojo, verde y azul) como funciones del tiempo. Por lo tanto la pantalla de una televisión a color es una señal tridimensional de 3 canales, que puede representarse por el vector

$$I(x,y,t) = \begin{bmatrix} I_r(x,y,t) \\ I_g(x,y,t) \\ I_b(x,y,t) \end{bmatrix}$$

1.3 TIPOS DE MODULACIÓN

En teoria, un número casi incontable de técnicas pueden ser usadas para codificar señales de audio digitalmente. Son fundamentalmente idénticas en su operación de representar señales analógicas como datos digitales, pero en la práctica difieren grandemente en su eficiencia relativa concerniente al ancho de banda requerido y al cociente de señal a ritmo. La modulación no es más que la manera de codificar información con el propósito de transmitirla o almacenarla[2]. Técnicas como la amplitud modulada (AM) y frecuencia modulada (FM) han sido largamente utilizadas para modular frecuencias portadoras de información analógica de audio para su transmisión por radio. Ya que son un tipo de modulación continua, son conocidas como modulación de parámetros de onda.

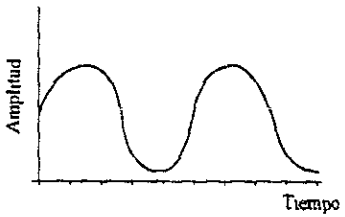
Figura 3. Las modulaciones AM y FM.



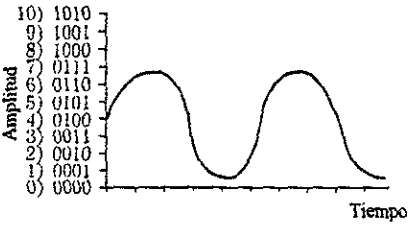
Cuando se transmite información de muestreo, varios tipos de modulación por pulsos son utilizados. Por ejemplo el ancho de pulso o su posición en el tiempo pueden representar la amplitud de la señal a un tiempo de muestreo; la modulación por el ancho de pulso (PWM), es un ejemplo de lo primeramente descrito y la modulación por posición de pulso (PPM;), es un ejemplo del segundo caso mencionado. En ambos casos la amplitud original de la señal es codificada y transmitida a través de pulsos de amplitud constante. La amplitud de la señal también puede ser transmitida directamente por la modulación de pulsos; la modulación por amplitud de pulsos (PAM) es un ejemplo de esta aproximación. La amplitud de los pulsos igual a la amplitud de la señal en un tiempo de muestreo. PWM, PPM y PAM se muestran en la figura. En otros casos las amplitudes de muestreo son transmitidas a través de métodos numéricos. Por ejemplo, en la modulación por número de pulsos (PNM), el modulador genera una secuencia de pulsos; la cuenta de los pulsos representa la amplitud de la señal en un tiempo de muestreo. De cualquier modo para una resolución alta, un número muy grande de pulsos es requerido. Aunque PWM, PPM, PAM y PNM son usados comúnmente en contexto de conversiones, no son adecuados para la transmisión o la grabación por sus limitaciones en su error y/o ancho de banda.

El método más comúnmente usado para la modulación es la modulación por código de pulsos (PCM). En PCM la señal de entrada debe pasar por el proceso de muestreo, cuantización y codificación. Al representar una amplitud analógica de muestras con un código de pulsos, números binarios pueden utilizarse para representar la amplitud. Las palabras binarias que representan amplitudes de las muestras son directamente codificadas en ondas PCM.

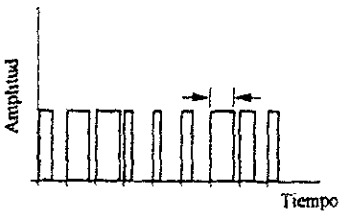
Figura 4. Ejemplo de modulación PWM, PNM, PPM, PCM y PAM.



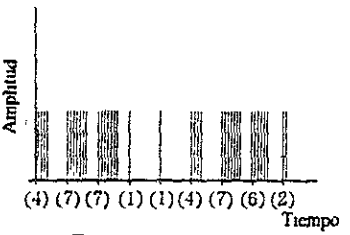
A Señal de audio



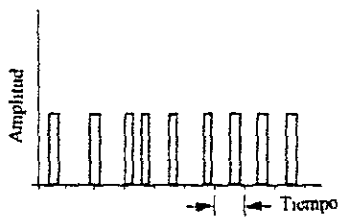
E Señal analógica cuantizada



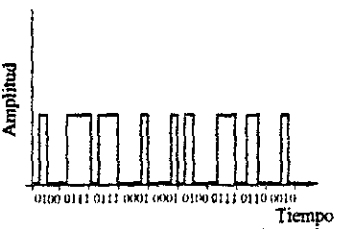
B Modulación por ancho de pulso



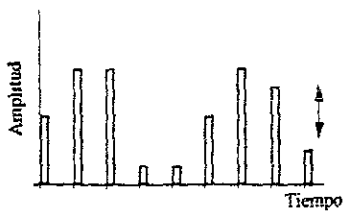
F Modulación por número de pulsos



C Modulación por posición por pulsos



G Modulación por código de pulsos



D Modulación por amplitud de pulsos

1.4 LAS SERIES DE FOURIER PARA SEÑALES PERIÓDICAS CONTINUAS

Algunos ejemplos de señales periódicas que se encuentran en la práctica son las ondas cuadradas, rectangulares, triangulares, y claro, las señales senoidales y exponenciales complejas.

Las bases para la representación de señales periódicas son las series de Fourier, que es una suma de senoides o exponenciales complejas relacionadas armónicamente. Juan Baptiste Joseph Fourier (1768-1830), un matemático francés, usó dichas series trigonométricas para describir el fenómeno de conducción de calor y la distribución de temperatura a través de un cuerpo. Aunque su trabajo fue motivado por el problema de conducción de calor, las técnicas matemáticas que él desarrolló durante la primera parte del siglo XIX ahora encuentran aplicación en una variedad de problemas que envuelven diferentes campos, incluyendo óptica, vibraciones de sistemas mecánicos, teoría de sistemas y electromagnetismo.

Una combinación de exponenciales complejas relacionadas armónicamente de la forma

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k F_0 t} \qquad \textbf{ecuación 1.4.1}$$

es una señal periódica con un período fundamental $T_p=1/F_0$. Entonces podemos pensar que las señales exponenciales

$$\{e^{j2\pi kF_0 t} \qquad k=0,\pm1,\pm2, \ldots..\}$$

como una base de "bloques de construcción" de donde podemos construir señales periódicas de diferentes tipos por medio de una relación adecuada de la frecuencia fundamental y los coeficientes $\{c_k\}$. F_0 determina el período fundamental de *x(t)* y los coeficientes $\{c_k\}$ especifican la forma de la onda.

Supongamos que tomamos una señal periódica x(t) con período T_p. Podemos representar la señal periódica por medio de series como la **ecuación 1.4.1**, llamadas *series de Fourier*[1], donde la frecuencia fundamental F_0 es seleccionada para ser el reciproco del período dado T_p. Para determinar la expresión para los coeficientes $\{c_k\}$ primero debemos multiplicar ambos lados de la **ecuación 1.4.1** por la exponencial compleja

$$e^{-j2\pi kF_0 lt}$$

donde *l* es un entero y después se integra ambos lados de la ecuación resultante a través de un período, digamos de 0 a T_p, o de forma más general, de t_0 a $t_0 + T_p$, donde t_0 es un valor arbitrario pero conveniente para empezar matemáticamente. Entonces obtenemos

$$\int_{t_0}^{t_0+T_p} x(t)e^{-j2\pi lF_0 t}dt = \int_{t_0}^{t_0+T_p} e^{-j2\pi lF_0 t}\left(\sum_{k=-\infty}^{\infty} c_k e^{+j2\pi kF_0 t}\right)dt \qquad \textbf{ecuación 1.4.2}$$

Para evaluar la integral del lado derecho de la **ecuación 1.4.2**, se intercambia el orden de la sumatoria e integral combinando las dos exponenciales y se obtiene

$$\sum_{k=-\infty}^{\infty} c_k \int_{t_0}^{t_0+T_p} e^{j2\pi F_0(k-l)t} dt = \sum_{k=-\infty}^{\infty} c_k \left[\frac{e^{j2\pi F_0(k-l)t}}{j2\pi F_0(k-l)} \right]_{t_0}^{t_0+T_p}$$ **ecuación 1.4.3**

Para $k \neq l$, el lado derecho de la **ecuación 1.4.3** evaluado en los límites inferior y superior, t_0 a $t_0 + T_p$, respectivamente, resulta cero. Por otra parte, si $k = l$, obtenemos

$$\int_{t_0}^{t_0+T_p} dt = t \Big]_{t_0}^{t_0+T_p} = T_p$$

Consecuentemente, la **ecuación 1.4.2** se reduce a

$$\int_{t_0}^{t_0+T_p} x(t) e^{-j2\pi F_0 t} dt = c_l T_p$$

y entonces la expresión de los coeficientes de Fourier en términos de la señal periódica se convierte en

$$c_l = \frac{1}{T_p} \int_{t_0}^{t_0+T_p} x(t) e^{-j2\pi F_0 t} dt$$ **ecuación 1.4.4**

Ya que t_0 es arbitrario, la integral puede evaluare a través de cualquier intervalo Tp, esto es, a través de cualquier intervalo igual al periodo de la señal x(t). Así, la integral de los coeficientes de las series de Fourier puede ser escrita como

$$c_l = \frac{1}{T_p} \int_{T_p} x(t) e^{-j2\pi F_0 t} dt$$

Como un importante punto que surge en la representación de la señal periódica x(t) por series de Fourier es si éstas convergen o no para cualquier valor de *t* en *x(t)*, esto es, si la señal x(t) y su representación en series de Fourier

$$\sum_{k=-\infty}^{\infty} c_k e^{j2\pi k F_0 t} \qquad \textbf{ecuación 1.4.5}$$

Son iguales para cualquier valor de *t*. Las *llamadas condiciones de Dirichlet*[1] garantizan que la **serie 1.4.5** es igual a *x(t)*, excepto en los valores de *t* que *x(t)* es discontinua. En estos valores de t, las series convergen al punto medio (el valor promedio) de la discontinuidad. Las condiciones de Dirichlet son:

1. La señal x(t) tiene un número finito de discontinuidades en cualquier período.
2. La señal x(t) contiene un número finito de máximos y mínimos durante cualquier período
3. La señal x(t) es absolutamente integrable en cualquier período, esto es,

$$\int_{T_p} |x(t)|dt < \infty \qquad \textbf{ecuación 1.4.6}$$

Cualquier señal periódica de interés práctico satisface estas condiciones.

Una condición más débil, que la señal tenga energía finita en un período,

$$\int_{T_p} |x(t)|^2 dt < \infty \qquad \textbf{ecuación 1.4.7}$$

garantiza que la energía en la señal de diferencia

$$e(t) = x(t) - \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k F_0 t}$$

sea cero, aunque x(t) y sus series de Fourier no sean iguales en todos los valores de *t*.

En suma, si *x(t)* es periódica y satisface las condiciones de Dirichlet, puede representarse como series de Fourier como en la **ecuación 1.4.4**, donde los coeficientes son especificados por la **ecuación 1.4.4**. Estas relaciones se resumen en

Análisis de Frecuencia de señales periódicas continuas

Ecuación de síntesis $x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k F_0 t}$ **ecuación 1.4.8**

Ecuación de análisis $c_k = \frac{1}{T_p} \int_{T_p} x(t) e^{-j2\pi k F_0 t} dt$ **ecuación 1.4.9**

En general, los coeficientes c_k de Fourier tienen valores complejos. Más aún, se muestra fácilmente que si la señal periódica es real, c_k y c_{-k} son complejos conjugados.

Como resultado, si

$$c_k = |c_k| e^{j\theta_k}$$

entonces

$$c_{-k} = |c_k| e^{-j\theta_k}$$

Consecuentemente, las series de Fourier pueden también representarse en la forma

$$P_x = \frac{1}{T_p} \int_{T_p} |x(t)|^2 \, dt \qquad \textbf{ecuación 1.4.10}$$

donde c_0 es un valor real cuando $x(t)$ es real.

Finalmente, debemos indicar que otra forma de series de Fourier puede obtenerse expandiendo la función coseno como

$$\cos(2\pi k F_0 t + \theta_k) = \cos(2\pi k F_0 t) * \cos(\theta_k) - \sin(2\pi k F_0 t) * \sin(\theta_k)$$

Consecuentemente, podemos escribir la **ecuación 1.4.10** de la forma

$$x(t) = a_0 + \sum_{k=1}^{\infty} (a_k \cos 2\pi k F_0 t - b_k \operatorname{sen} 2\pi k F_0 t) \qquad \textbf{ecuación 1.4.11}$$

donde

$$a_0 = c_0$$
$$a_k = 2|c_k|\cos\theta_k$$
$$b_k = 2|c_k|\operatorname{sen}\theta_k$$

1.4.1 ESPECTRO DE DENSIDAD DE POTENCIA DE UNA SEÑAL PERIÓDICA

Una señal periódica tiene energía infinita y potencia promedio finita, que está dada como

$$P_x = \frac{1}{T_p}\int_{T_p} |x(t)|^2 dt \qquad \textbf{1.4.12}$$

Si tomamos el complejo conjugado de la **ecuación 1.4.8** y substituimos por $x^*(t)$ en la **ecuación 1.4.12**, obtenemos

$$P_x = \frac{1}{T_p}\int_{T_p} x(t)\sum_{k=-\infty}^{\infty} c_k^* e^{-j2\pi kF_0 t} dt$$
$$= \sum_{k=-\infty}^{\infty} c_k^*\left[\frac{1}{T_p}\int_{T_p} x(t)e^{-j2\pi kF_0 t} dt\right] \qquad \textbf{ecuación 1.4.13}$$
$$= \sum_{k=-\infty}^{\infty} |c_k|^2$$

Entonces, establecemos la relación

$$P_x = \frac{1}{T_p}\int_{T_p} |x(t)|^2 dt = \sum_{k=-\infty}^{\infty} |c_k|^2 \qquad \textbf{ecuación 1.4.14}$$

que se llama la *relación de Parseval*[11] para señales de potencia.

Para ilustrar el significado físico de la **ecuación 1.4.14**, supongamos que $x(t)$ consiste en una exponencial compleja simple

$$x(t) = c_k e^{j2\pi kF_0 t}$$

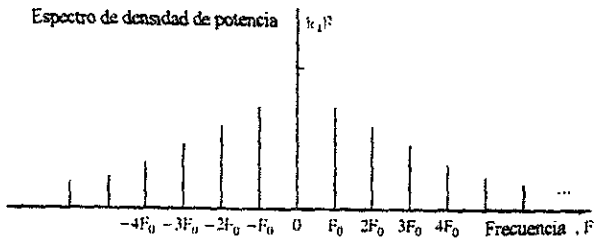
En este caso, todos los coeficientes de las series de Fourier excepto c_k son cero. Así, el promedio de potencia en la señal es

$$P_k = | c_k |^2$$

Es obvio que $| c_k |^2$ representa la potencia en el componente armónico k-ésimo de la señal. Por lo tanto la potencia promedio en la señal periódica es simplemente la suma del promedio de las potencias en todas las armónicas.

Si graficamos $| c_k |^2$ como una función de frecuencias kF_0, k=0, ±1, ±2, .., el diagrama que se obtiene muestra cómo la potencia de la señal periódica se distribuye a través de los diferentes componentes de frecuencia. El diagrama, que se ilustra, es llamado el *espectro de densidad de potencia** de una señal periódica $x(t)$. Ya que la potencia de una señal periódica existe solamente en valores discretos de las frecuencias (F_0=0, ±F_0, ±2F_0, ...), la señal se dice que tiene un *espectro lineal*. El espacio entre dos líneas consecutivas espectrales es igual al recíproco del período fundamental T_p, mientras que la forma del espectro (la distribución de potencia de la señal), depende en las características del dominio de tiempo de la señal

Figura 5. Espectro de densidad de potencia de una señal periodica.



Si la señal periódica tiene valor real, los coeficientes de las series de Fourier $\{ c_k \}$ satisface la condición

$$c_{-k} = c_k^*$$

Así $| c_{-k} | = | c_k^* |$. Por lo tanto el espectro de potencia es una función simétrica de la frecuencia. Esta condición también significa que la magnitud del espectro es simétrica del origen y el espectro de fase es una función impar. Como consecuencia de la simetría, es suficiente especificar que el espectro de una señal periódica real para un número positivo de frecuencias únicamente. También, el total del promedio de potencia puede expresarse como

$$P_x = c_0^2 + 2\sum_{k=1}^{\infty} |c_k|^2 \qquad \textbf{ecuación 1.4.15}$$

$$= a_0^2 + \frac{1}{2}\sum_{k=1}^{\infty}(a_k^2 + b_k^2) \qquad \textbf{ecuación 1.4.16}$$

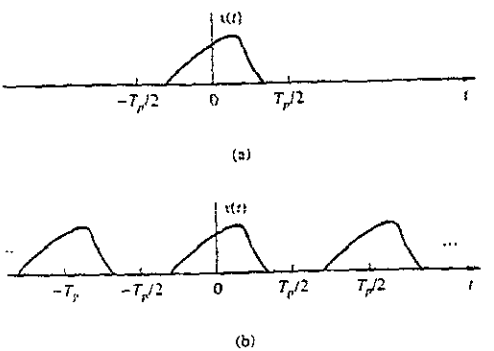
que se deduce de la relación dada en la sección anterior a lo largo de los coeficientes $\{a_k\}$, $\{b_k\}$ y $\{c_k\}$ en las expresiones de las series de Fourier.

1.5 TRANSFORMADA DE FOURIER PARA SEÑALES APERIÓDICAS CONTINUAS

Consideremos una señal aperiódica *x(t)* con duración finita como se muestra en la **figura mostrada**. De esta señal aperiódica, podemos crear una señal periódica $x_p(t)$ con periodo T_p, como se muestra en la **figura mostrada.** Ciertamente, $x_p(t) = x(t)$ y el limite como $T_p \rightarrow \infty$, que es,

$$x(t) = \lim_{T_p \rightarrow \infty} x_p(t)$$

Figura 6. (a) Señal aperiódica x(t) y (b) señal periódica $x_p(t)$ construida por repetir x(t) con un periódo T_p.



Esta interpretación implica que se puede obtener el espectro de $x(t)$ del espectro de $x_p(t)$ simplemente tomando el límite como $T_p \to \infty$.

Empezamos con la representación en series de Fourier de $x_p(t)$,

$$x_p(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k F_0 t} \quad F_0 = \frac{1}{T_p} \qquad \textbf{ecuación 1.5.1}$$

donde

$$c_k = \frac{1}{T_p} \int_{-T_p/2}^{T_p/2} x_p(t) e^{-j2\pi k F_0 t} dt \qquad \textbf{ecuación 1.5.2}$$

Ya que $x_p(t) = x(t)$ para $-T_p/2 \le t \le T_p/2$, y la **ecuación 1.5.2** puede expresarse como

$$c_k = \frac{1}{T_p} \int_{-T_p/2}^{T_p/2} x(t) e^{-j2\pi k F_0 t} dt \qquad \textbf{ecuación 1.5.3}$$

Es también verdad que $x(t)=0$ para $|t| > T_p/2$. Consecuentemente, los límites de la integral en la **ecuación 1.5.3** puede reemplazarse por $-\infty$ e ∞. Entonces

$$X(F) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi k F t} dt \qquad \textbf{ecuación 1.5.4}$$

Definamos una función $X(F)$, llamada *la transformada de Fourier* de $x(t)$, como

$$X(F) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi Ft} dt$$ **ecuación 1.5.5**

X(F) es una función de la variable continua *F*. No depende en T_p o F_o. Pero si comparamos la **ecuación 1.5.4** y la **ecuación 1.5.5**, es claro que los coeficientes de Fourier c_k pueden expresarse en términos de *X(F)* como

$$c_k = \frac{1}{T_p} X(kF_0)$$

o su equivalente

$$T_p c_k = X(kF_0) = X\left(\frac{k}{T_p}\right)$$ **ecuación 1.5.6**

Ya que los coeficientes de Fourier son muestras de *X(F)* tomadas como múltiplos de F_o y medidas por *Fo* (multiplicados por 1/ T_p). La substitución de c_k de la **ecuación 1.5.6** en **ecuación 1.5.1** resulta

$$x_p(t) = \frac{1}{T_p} \sum_{k=-\infty}^{\infty} X\left(\frac{k}{T_p}\right) e^{j2\pi kF_0 t}$$ **ecuación 1.5.7**

Deseamos tomar el límite de **ecuación 1.5.7** como T_p cuando se aproxima a infinito. Primero, definimos $\Delta F = T_p$. Con esta substitución, **ecuación 1.5.7** se convierte en

$$x_p(t) = \sum_{k=-\infty}^{\infty} X(k\Delta F)e^{j2\pi k\Delta Ft}\Delta F \qquad \textbf{ecuación 1.5.8}$$

Es claro que cuando el límite T_p se aproxima a infinito, $x_p(t)$ se reduce a x(t). También, ΔF se convierte en el diferencial dF y $k\Delta F$ se convierten en la variable continua de frecuencia F. La sumatoria en la **ecuación 1.5.8** se transforma en la integral sobre la variable de frecuencia F. Entonces

$$\lim_{T_p\to\infty} x_p(t) = x(t) = \lim_{\Delta F\to 0} \sum_{k=-\infty}^{\infty} X(k\Delta F)e^{-j2\pi k\Delta Ft}\Delta F \qquad \textbf{ecuación 1.5.9}$$

$$x(t) = \int_{-\infty}^{\infty} X(F)e^{j2\pi Ft}dF$$

Esta relación de integral resulta en $x(t)$ cuando $X(F)$ es conocida y se le llama *la transformada de Fourier inversa.*

Esto concluye nuestra derivación heurística del par de transformadas de Fourier dada por las **ecuaciones 1.5.5** y **1.5.9** para una señal aperiódica $x(t)$. Aunque la derivación no es rigurosamente matemática, nos lleva a las relaciones de la transformada de Fourier con argumentos relativamente de intuición sencilla. En suma, el análisis de frecuencia de señales aperiódicas continuas envuelve el siguiente par de transformadas de Fourier.

Análisis de frecuencia de señales aperiódicas continuas.

Ecuación de síntesis $x(t) = \int_{-\infty}^{\infty} X(F)e^{j2\pi Ft}dF$ **ecuación 1.5.10**

Transformada inversa

Ecuación de análisis $X(F) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi Ft}dt$ **ecuación 1.5.11**

Transformada directa

Es claro que la diferencia esencial entre las series de Fourier y la transformada de Fourier es que el espectro de la última es continuo y por lo tanto la síntesis de una señal aperiódica a partir de su espectro se logra por medio de una integral en vez de una sumatoria.

Finalmente, queremos indicar que el par de transformadas de Fourier en las **ecuaciones 1.5.10** y **1.5.11** pueden expresarse en términos de variables de frecuencia en radianes $\omega=2\pi F$. Ya que $dF=d\omega/2\pi$, las **ecuaciones 1.5.10** y **1.5.11** se convierten en

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega \qquad \textbf{ecuación 1.5.12}$$

$$X(\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt \qquad \textbf{ecuación 1.5.13}$$

Las condiciones que garantizan la existencia de la transformada de Fourier son las *condiciones de Dirichlet*, que pueden expresarse como:

1. La señal x(t) como un número finito de discontinuidades finitas.
2. La señal x(t) como un número finito de un máximo y un mínimo
3. La señal x(t) es absolutamente integrable, esto es

$$\int_{-\infty}^{\infty}|x(t)|dt<\infty \qquad \textbf{ecuación 1.5.14}$$

La tercera condición se deduce fácilmente de la transformada de Fourier dada en la **ecuación**, en efecto,

$$|X(F)| = \left|\int_{-\infty}^{\infty} x(t)e^{-j\,2\pi kF_0 t}\right| \le \int_{-\infty}^{\infty}|x(t)|dt<\infty$$

por lo tanto $|X(F)|<$ si la **ecuación 1.5.14** se satisface.

Una condición menos fuerte es la de la existencia de la transformada de Fourier es que x(t) tenga energía finita; esto es

$$\int_{-\infty}^{\infty}|x(t)|^2 dt<\infty \qquad \textbf{ecuación 1.5.15}$$

Nótese que si la señal *x(t)* es absolutamente integrable, si la energía es finita. Esto es, si

$$\int_{-\infty}^{\infty}|x(t)|dt<\infty$$

entonces

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 \, dt < \infty \qquad \textbf{ecuación 1.5.16}$$

Pero, lo contrario no es verdadero. Esto es, que la señal puede tener energía finita pero puede no ser absolutamente integrable. Por ejemplo, la señal

$$x(t) = \frac{\operatorname{sen} 2\pi t}{\pi t} \qquad \textbf{ecuación 1.5.17}$$

es integrable cuadráticamente pero no es absolutamente integrable. Esta señal tiene transformada de Fourier.

$$X(F) = \begin{cases} 1, & |F| \le 1 \\ 0, & |F| > 1 \end{cases} \qquad \textbf{ecuación 1.5.18}$$

Ya que esta señal viola la **ecuación 1.5.14,** es claro que las condiciones de Dirichlet son suficientes pero no necesarias para la existencia de la transformada de Fourier. En cualquier caso, casi todas las señales finitas de energía tienen transformadas de Fourier, entonces no necesitamos preocuparnos acerca de señales patológicas, que raramente se encuentran en la práctica.

1.5.1 ESPECTRO DE DENSIDAD DE ENERGÍA DE SEÑALES APERIÓDICAS

Tomemos *x(t)* como cualquier señal de energía finita con transformada de Fourier *X(F)* . Esta energía es

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 dt$$

que puede expresarse en términos de *X(F)* como sigue

$$\begin{aligned} E_x &= \int_{-\infty}^{\infty} x(t)x^*(t)dt \\ &= \int_{-\infty}^{\infty} x(t)dt\left[\int_{-\infty}^{\infty} X^*(F)e^{-j2\pi Ft}dF\right] \\ &= \int_{-\infty}^{\infty} X^*(F)dF\left[\int_{-\infty}^{\infty} x(t)e^{-j2\pi Ft}dt\right] \\ &= \int_{-\infty}^{\infty} |X(F)|^2 dF \end{aligned}$$

Entonces, concluimos que

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(F)|^2 dF \qquad \textbf{ecuación 1.5.19}$$

Esto es la *relación de Parseval* para señales de energía finita y expresa el principio de conservación de energía en el dominio de tiempo y frecuencia.

El espectro $X(F)$ de una señal es en general, un valor complejo. Consecuentemente, es usualmente expresado en forma polar como:

$$X(F) = |X(F)|e^{j\Theta(F)}$$

donde $|X(F)|$ es el espectro de magnitud y $\Theta(F)$ es el espectro de fase

$$\Theta(F) = \angle X(F)$$

Por el otro lado, la cantidad

$$S_{xx}(F) = |X(F)|^2 \qquad \textbf{ecuación 1.5.20}$$

que es el integrando en la **ecuación**, que representa la distribución de energía en la señal como una función de frecuencia. Por lo tanto $S_{xx}(F)$ es llamado el *espectro de densidad de energía* de x(t) . La integral de $S_{xx}(F)$ de todas las frecuencias da el total de energía en la señal. Visto de otra manera, la energía en la señal x(t) sobre la banda de frecuencias $F_1 \leq F \leq F_1 + \Delta F$ es

$$\int_{F_1}^{F_1+\Delta F} S_{xx}(F)dF$$

De la **ecuación** observamos que $S_{xx}(F)$ no contiene la información de fase [$S_{xx}(F)$ es puramente real y no negativa]. Ya que la fase del espectro de $x(t)$ no está contenida en $S_{xx}(F)$, es imposible de reconstruir la señal dada $S_{xx}(F)$.

Finalmente, como en el caso de las series de Fourier, se muestra fácilmente que si la señal x() es real, entonces

$$|X(-F)| = |X(F)| \qquad \textbf{ecuación 1.5.21}$$

$$\angle X(-F) = -\angle X(F) \qquad \textbf{ecuación 1.5.22}$$

Combinando las **ecuaciones 1.5.21** y **1.5.22**, obtenemos

$$S_{xx}(-F) = S_{xx}(F) \qquad \textbf{ecuación 1.5.23}$$

En otras palabras el espectro de densidad de energía de una señal real es simétrico.

CAPÍTULO 2

AUDIO DIGITAL

2.1 MUESTREO EN TIEMPO DISCRETO

Con la grabación analógica, una cinta se modula continuamente o una ranura se corta continuamente. Con la grabación digital, los números se utilizan. La primera pregunta es cómo crear dichos números. En otras palabras, ¿cómo grabaremos un grupo de datos de una onda cambiante? La digitalización utiliza muestreos de tiempo y cuantización de amplitud para codificar la onda analógica variable como valores de amplitudes discretas en el tiempo. Primero, consideremos la idea del muestreo en tiempo discreto como la esencia del audio digital.

El tiempo parece fluir continuamente. Las manecillas del reloj recorren todo el tiempo como pasa. Un reloj de lectura digital también nos dice el tiempo pero con una pantalla de valores discretos. En otras palabras exhibe un tiempo de muestreo. Similarmente la música varía continuamente en el tiempo y puede grabarse y reproducirse ya sea continua o discretamente. El muestreo en tiempo

discreto es esencialmente un mecanismo esencial que define un sistema de audio digital, permite la conversión de analógico a digital y lo diferencia de un sistema analógico.

Pero, una pregunta persistente se presenta a continuación. Si un sistema digital hace muestras discretamente, ¿qué pasa entre las muestras?, ¿no hemos perdido la información que ocurre entre los tiempos de muestreo? La respuesta sorpresivamente intuitiva es: NO. Dadas las condiciones correctas, ninguna información se pierde por el muestreo entre la entrada y la salida en un sistema de digitalización. Las muestras contienen la misma información que una señal no muestreada.

Aunque la analogía es algo inadecuada los cuadros discretos de una película crean una escena, similarmente las muestras de una grabación audio digital crean una señal, en los sistemas de audio digital debemos suavizar los brincos de la señal entrante. Específicamente la señal se pasa por un filtro pasabajas; esto es, las frecuencias demasiado altas para ser muestreadas debidamente deben ser removidas. Observamos que una señal con una frecuencia de respuesta finita puede ser muestreada sin pérdida de información[2]; las muestras contienen toda la información de la señal original. Ésta puede ser completamente recobrada de las muestras. Generalmente observamos que existe un método para reconstruir la señal a través de sus valores de amplitud tomados en puntos espaciados equitativamente.

2.1.1 TEOREMA DE MUESTREO

Dada una señal analógica, ¿cómo vamos a seleccionar el período de muestreo T_0, equivalentemente el ritmo de muestreo F_s? Para responder a esta cuestión debemos tener información sobre las características de la señal en la que vamos a realizar el muestreo. En particular, debemos tener

un poco de información general concerniente al contenido de frecuencia de la señal. Esta información generalmente está disponible. Por ejemplo, sabemos generalmente que los componentes de frecuencia mayores de una señal de una voz caen arriba de los 3000 Hz. Por otro lado, las señales de televisión, en general, contienen componentes de frecuencia importantes hasta los 5 MHz. El contenido de información de estas señales está dentro de las amplitudes frecuencias y fases de varios componentes de frecuencia, pero el conocimiento detallado de las características de éstas no está disponible para nosotros antes de obtener la señal. De hecho, el propósito de procesar las señales es usualmente el de extraer esta información detalladamente. Pero, si conocemos el contenido máximo de frecuencia de la clase general de señales (la clase de señales de voz, de video, etc.), podemos especificar el ritmo de muestreo necesario para convertir las señales analógicas a señales digitales.

Supongamos que la señal analógica puede ser representada por una suma de senoides de diferentes amplitudes, frecuencias y fases, esto es,

$$x_a(t) = \sum_{i=1}^{N} A_i \cos(2\pi F_i + \theta_i) \qquad \textbf{ecuación 2.1.1}$$

donde N representa el número de componentes de frecuencia. Todas las señales, como las de voz y video, permiten esta representación sobre cualquier período corto de tiempo. Las amplitudes, frecuencias y fases usualmente cambian lentamente con el tiempo de un segmento de tiempo a otro. Pero, supongamos que las frecuencias no exceden una frecuencia conocida, digamos F_{max}. Por ejemplo, F_{max} es igual a 3000 Hz para la clase de señales de voz y F_{max} es igual a 5 MHz para las señales de televisión. Ya que la frecuencia máxima puede variar mínimamente de muestra a muestra entre las señales dada una clase (p.j., variará ligeramente de locutor a locutor), desearemos asegurar que F_{max} no

exceda un valor predeterminado al pasar una señal analógica a través de un filtro que severamente atenúe los componentes de frecuencia arriba de F_{max}. En la práctica, este filtrado es usualmente usado antes del muestreo.

De este conocimiento de F_{max}, podemos seleccionar un ritmo de muestreo apropiado. Conocemos que la frecuencia más alta en una señal analógica que puede ser reconstruída sin ambigüedad cuando una señal es muestreada a un ritmo F_s=1/T es F_s/2. Cualquier frecuencia arriba de F_s /2 o debajo de – F_s /2 resulta en muestras que son idénticas con una frecuencia correspondiente en un rango – F_s /2 ≤ F ≤ F_s /2 [5]. Para evitar las ambigüedades resultantes del "aliasing", debemos de seleccionar un ritmo de muestreo suficientemente alto. Esto es, debemos escoger F_s /2 a ser mayor a F_{max}. Esto para evitar el problema del "aliasing", F_s selecciona para que

$$F_s > F_{max} \qquad \textbf{ecuación 2.1.2}$$

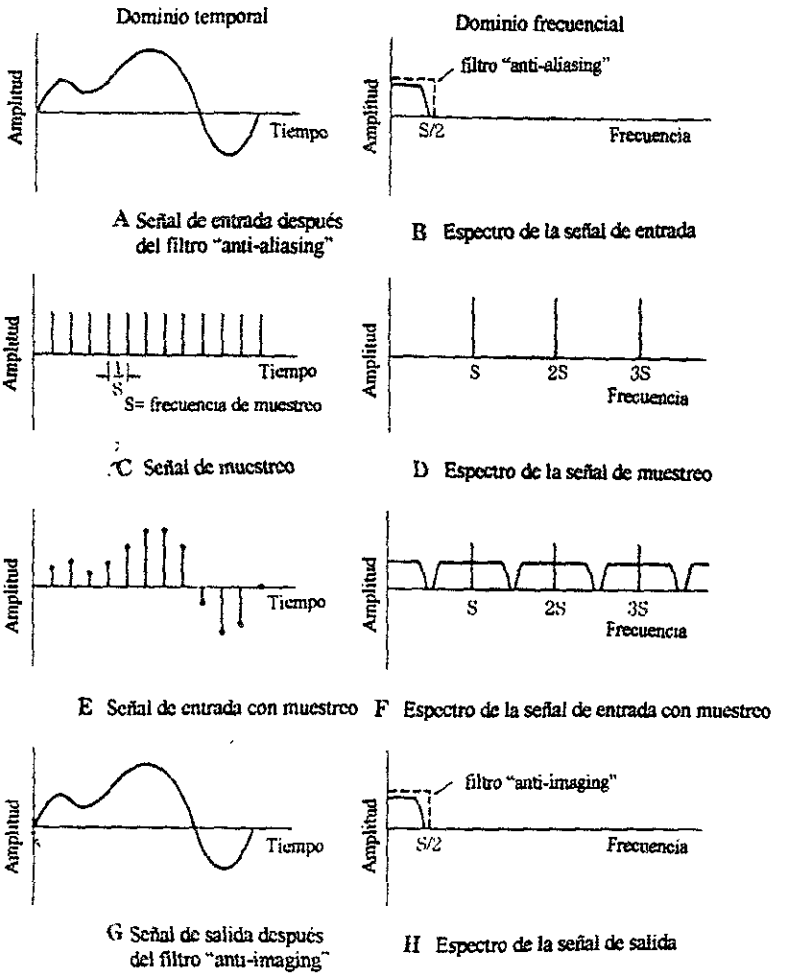
donde F_{max} es el componente de frecuencia más alto en una señal analógica, Con el ritmo de frecuencia seleccionado de esta manera, cualquier componente de frecuencia , digamos $|F_i| < F_{max}$ en una señal analógica se puede graficar en una senoide discreta en el tiempo con una frecuencia.

$$-\frac{1}{2} \le f_i = \frac{F_i}{F_s} \le \frac{1}{2} \qquad \textbf{ecuación 2.1.3}$$

o, equivalentemente

$$-\pi \le \omega_i = 2\pi f_i \le \pi \qquad \textbf{ecuación 2.1.4}$$

Figura 7. Señales de dominio temporal(columna izquierda) y dominio frecuencial(columna derecha) ilustra el efecto del muestreo en banda limitada y su reconstrucción



2.2 "ALIASING"

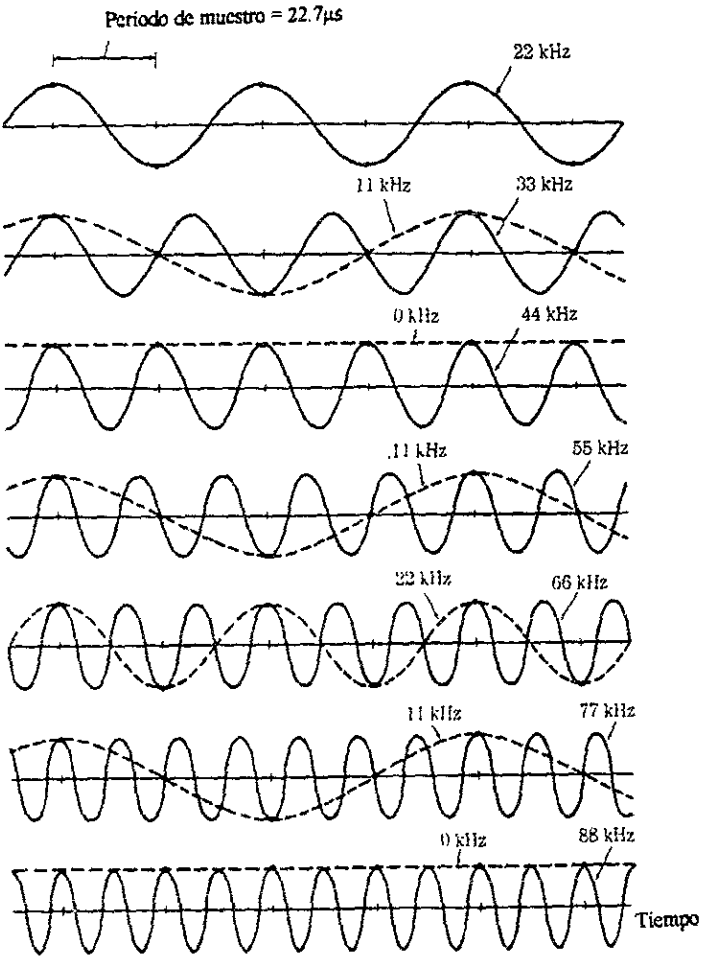
Un problema para la persona que diseña un sistema de audio digital es el de "**aliasing**", una forma de confusión en el muestreo que puede ocurrir en el lado que grabamos una cadena de la señal

Como un criminal, que puede tomar varios nombres y confundir su identidad, "**aliasing**" puede crear falsos componentes de la señal. Estas señales erróneas pueden aparecer dentro de ancho de banda de audio y ser imposible distinguirlos de señales legítimas. Obviamente, la obligación de la persona que diseña es la de prevenir que esta distorsión ocurra.

La señal debe ser limitada en su ancho de banda; esto es, un filtro pasabajas debe preceder al sistema de muestreo. Si esto no se puede hacer, la señal se muestrea menos y puede producir "**aliasing**".

"**Aliasing**" es una consecuencia de violar el teorema de muestreo[2]. La frecuencia de audio más alta en un sistema de muestreo debe ser igual o menor a la mitad de la frecuencia de muestreo. Si la frecuencia de audio es mayor que la mitad de la frecuencia de muestreo, "**aliasing**" ocurre. Como la frecuencia de audio aumenta, el número de puntos de muestreo por período decrece. Cuando la frecuencia de Nyquist se alcanza, existen dos muestras por período, el mínimo necesario para registrar la naturaleza bipolar de una onda. Si muestreáramos más altas frecuencias de audio, el circuito de muestreo continuará produciendo muestras a su ritmo fijado, pero éstas pueden crear falsa información en la forma de frecuencias alias. Si la frecuencia de audio aumenta, una frecuencia alias descendiente es creada. Específicamente, si S es la frecuencia de muestreo y F es la frecuencia más alta que la mitad de la frecuencia de muestreo y N es un entero, entonces la nueva frecuencia F_f es creada a $F_f = \pm N S \pm F$. En otras palabras, las frecuencias alias aparecen atrás en la banda de audio, envuelta en la frecuencia de muestreo. De hecho, "**aliasing**" es llamado a veces plegado de imagen.

Figura 8. Frecuencias "alias" (líneas punteadas) en la banda de audio entre 0 y 22 kHz mientas se incrementa la frecuencia de entrada (líneas sólidas).



2.2.1 PREVENCIÓN ALIAS

En la práctica, el problema de "**aliasing**" puede ser superado. De hecho, en un buen diseño de un sistema digital de grabación, "**aliasing**" no ocurre La solución es sencilla; la señal de entrada es limitada a un ancho, con un filtro pasabajas fuerte (filtrado "anti-aliasing") diseñado para proveer una

atenuación signifitiva a la frecuencia de Nyquist, para asegurar que la señal muestreada nunca exceda la frecuencia de Nyquist. Un filtro ideal tendrá una característica de pared con una atenuación instantánea e infinita en la banda de bloqueo. Sin embargo, en la práctica, un filtro no puede lograr esto. Más bien, este es diseñado con una banda de transición en donde la atenuación es lograda con una característica muy inclinada de pendiente. Más aún, un filtro provee una atenuación dentro de los límites de amplitud de resolución del sistema. Esto asegura que el sistema encuentre las demandas del teorema de muestreo; por lo que, "**aliasing**" no ocurre.

Es crítico observar el teorema de muestreo y el filtrado de la señal de entrada en un sistema de conversión digital. Si se deja que "aliasing" ocurra, no existe técnica que pueda remover las frecuencias alias del ancho de banda original del audio. Niveles extremadamente bajos de "aliasing" pueden ocurrir después del filtro "anti-aliasing" por el error de cuantización. Una señal de ruido llamada "dither" es usada para aliviar esta distorsión.

2.3 CUANTIZACIÓN

La medición de un evento variable sólo es significativa si el tiempo y el valor de la medición es guardado. El muestreo representa el tiempo de medición, y la cuantización representa el valor de medición y en el caso del audio, la amplitud de la forma de onda a un tiempo de muestreo. El muestreo y la cuantización son entonces componentes fundamentales en la conversión digital y juntos pueden caracterizar un evento acústico, siendo variables que determinan, respectivamente, el ancho de banda y la resolución de la caracterización[2]. Una señal analógica puede ser representada por una serie de pulsos; la amplitud de cada pulso entrega un número que representa el valor analógico de un instante

con la cuantización. Como en cualquier medición analógica, la exactitud está limitada por la resolución del sistema. Por la longitud finita de la palabra, la resolución digital de audio está limitada, y se introduce un error de medición. Este error es frecuentemente similar al ruido en un sistema analógico de audio; sin embargo, perceptualmente, es más intruso porque su carácter varía con la amplitud de la señal.

Con una cuantización uniforme, la amplitud de una señal analógica es proyectada en una parte numérica de igual altura. El número infinito de los puntos de amplitud de una señal analógica debe ser cuantizado por un número finito de partes de nivel; esto introduce un error. La representación de alta calidad requiere un número grande de niveles; por ejemplo, una señal de audio de gran calidad puede requerir 655365 niveles de amplitud o más. Sin embargo, solamente unos pocos niveles pueden llevar el contenido de la información; por ejemplo, dos niveles de amplitud pueden (apenas) transmitir una plática entendible.

Para un medidor digital la exactitud está limitada por la resolución del medidor que es el número de dígitos mostrados, Entre más dígitos, mayor la exactitud, pero el último sistema binario es usado para la medición; decimos que el error de resolución del sistema es la mitad de un LSB (bit menos significativo). Para ambos sistemas, analógico y digital, el problema de medir un fenómeno analógico como la amplitud nos lleva a un error. Nosotros obtenemos mayor información de un evento analógico cuando éste es caracterizado en términos de datos digitales.

La cuantización es entonces una técnica de medición de un evento analógico para formar un valor numérico. Un sistema digital utiliza un sistema binario. El número de posibles valores está determinado por la longitud de la palabra binaria que es el número de bits disponibles para formar la

representación. Justo como los números de los bits en un voltímetro digital determinan la resolución, el número de bits en una grabadora de audio digital determina la resolución. En la práctica, la resolución es primariamente influenciada por la calidad del convertidor analógico digital.

El muestreo de una señal limitada en su ancho de banda es teóricamente un proceso sin pérdidas, pero escoge el valor de amplitud a un tiempo de muestreo ciertamente no lo es. Cualquier decisión en escalas y códigos muestra que la conversión digital nunca puede completamente codificar una función analógica continua.

Una señal analógica tiene un número infinito de valores en su amplitud, pero un cuantizador tiene un número finito de intervalos. Todos los valores analógicos entre dos intervalos pueden ser representados por un sólo número asignado a este intervalo. Entonces, el valor cuantizado es sólo una aproximación del valor actual.

2.4 "DITHER"

Con señales complejas de amplitud larga, existe poca correlación entre la señal y el error de cuantización, aunque el error es aleatorio y perceptualmente similar al ruido blanco analógico. Con señales de niveles bajos, el carácter del error cambia porque se correlaciona con la señal y potencialmente resulta en una distorsión audible. Un sistema de conversión digital debe suprimir cualquiera de las cualidades audibles del error de cuantización. Obviamente, el número de bits en la palabra puede aumentar, resultando en un descenso del error de amplitud de 6 dB por bit adicional.

Esto resulta nada económico, y muchos bits son necesarios para reducir satisfactoriamente la audiobilidad del error de cuantización.

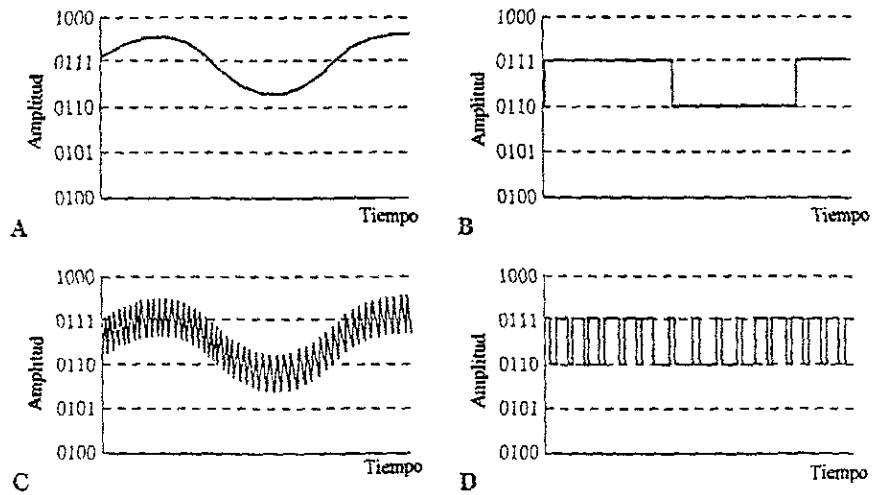
El "dither" es una técnica muy eficiente. Con el "dither", una pequeña cantidad de ruido es añadida a la señal de audio antes del muestreo para linealizar el proceso de cuantización. Esencialmente, con el "dither" la señal de audio se obliga a cambiar con respecto a los niveles de cuantización. El promedio del proceso suaviza el efecto en los niveles de cuantización incremental y rompe la correlación del error con la señal. Esto hace aleatorio el efecto del error de cuantización al punto de su total eliminación. Pero, aunque reduce grandemente la distorsión, añade un poco de ruido a la señal de salida.

El "dither" no oculta el error de cuantización; más bien, ayuda al sistema digital a codificar las amplitudes más pequeñas que el bit menos significativo, de una manera similar que el sistema analógico retiene señales abajo del nivel de ruido . Un sistema digital apropiado con "dither" excede el rendimiento en señal a ruido de un sistema analógico. Pero también, un sistema digital sin "dither" puede ser inferior a un sistema analógico, particularmente en condiciones de niveles muy bajos en la señal. Un sistema digital de audio de alta calidad demanda el "dither" antes de cuantizar la señal en el convertidor analógico- digital. En un sentido conceptual, el "dither" es similar a la tendencia a las frecuencias altas de una grabadora de cinta magnética analógica. En suma. Las computaciones digitales deben tener "dither" antes de recuantizar en un convertidor digital- analógico.

Matemáticamente, con el "dither", el error de cuantización no es más que una función determinística de la señal de entrada, pero más bien, se convierte en una variable aleatoria tendiente a cero. En otras palabras, más que cuantizar solamente la señal de entrada, el ruido "dither" y la señal

son cuantizados al mismo tiempo y esto hace aleatorio el error. Esto linealiza el proceso de cuantización. Esta técnica es conocida como "dither" no substractivo porque la señal y el "dither" son sumadas permanentemente a la señal de audio; el error total no depende estadisticamente de la señal de audio y los errores no dependen de muestra en muestra. Aunque, el "dither" no substractivo manipula las propiedades estadisticas del cuantizador reproduciendo estadisticamente los momentos condicionales del error total independiente de la entrada, rompiendo la correlación efectivamente del error de cuantización de las muestras de la señal y de cada una. El espectro de potencia del error total de la señal puede hacerse blanco. El "dither" substractivo, en el que la señal dither es removida después de la recuantización, teóricamente provee total independencia estadistica de error, pero es más difícil de implementar.

Figura 9. El efecto "dither" es usado para suavizar los efectos del error de cuantización. A señal de entrada sin "dither". B. Resultados de la cuantización. C. Una señalde entrada con "dither". La cauntización resultante en forma PWM.

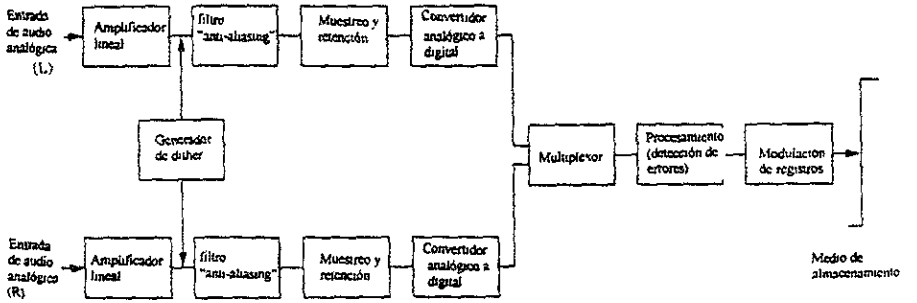


2.5 CÓDIGO DE MODULACIÓN DE PULSOS

La arquitectura de un sistema PCM sigue estos pasos para diseñar un sistema de transformación numérica[2]. La señal analógica es filtrada y muestreada y su amplitud es cuantizada por un convertidor analógico a digital. Números binarios son representados como una serie de código de modulación de pulsos que representan amplitudes de ondas en un tiempo de muestreo. Si dos canales son muestreados, los datos pueden ser multiplexados para formar un flujo de datos. Los datos pueden ser manipulados para proveer sincronización y corrección de errores, como también datos auxiliares pueden ser adicionados. Cuando se reproducen, los datos son demodulados, decodificados y corregidos los errores para recuperar la amplitud original a los tiempos muestreados y reconstruir la forma de la señal por un convertidor digital a analógico y un filtro pasabandas.

La sección de codificación para un aparato PCM registrador convencional consiste en amplificadores de entrada, un generador de ruido aleatorio, filtros pasabajas, circuitos de retención y muestreos, convertidores analógicos a digitales, un multiplexor, circuitos de modulación y procesamiento digitales y un medio de almacenaje como una cinta digital o un disco. Este diseño de "hardware" es una realización práctica del teorema de muestreo.

Figura 10. Diagrama lineal mostrando los principales elementos de la modulación por código de pulsos.



2.6 VENTAJAS DEL PROCESAMIENTO DE SEÑALES DIGITALES SOBRE LAS SEÑALES ANALÓGICAS

Existen muchas razones por las cuales el procesamiento digital de señal de cualquier señal analógica es preferible para procesar la señal directamente en el dominio analógico. Primero, los sistemas programables digitales permiten flexibilidad en reconfigurar las operaciones del procesamiento digital de señal simplemente cambiando el programa. Reconfigurar un sistema analógico usualmente implica un re-diseño del "hardware" seguido de las pruebas y verificaciones para ver si opera apropiadamente[1].

Las consideraciones en la precisión también juegan un importante papel en la forma del procesador de señal. Tolerancias en los componentes de un circuito analógico hacen extremadamente difícil para el diseñador el control de la precisión. Por otra parte, un sistema digital provee mejor control en los requerimientos de precisión. Éstos resultan en especificar la precisión de un convertidor A/D y el procesador digital de señal, en términos de longitud de palabra y factores similares

Las señales digitales son más fáciles de guardar en un medio magnético (disco o cinta) sin el deterioro y pérdida en la fidelidad de la señal más allá que el introducido por el convertidor analógico digital. Como una consecuencia, las señales se convierten en transportables y pueden ser procesadas en un laboratorio remoto. El método de procesamiento de señal digital también permite la implementación de algoritmos más sofisticados de procesamiento de señal. Es usualmente difícil desarrollar operaciones matemáticas precisas en señales en forma analógica pero estas mismas operaciones pueden ser implementadas rutinariamente en una computadora digital usando "software".

En algunos casos la implementación digital de un sistema de procesamiento de señal es más barato que su contraparte analógica. Su costo más bajo puede ser por el hecho que el "hardware" digital es más barato o quizás es el resultado de la flexibilidad para modificaciones que provee la implementación digital.

Como una consecuencia de las ventajas, el procesamiento de señal digital ha sido aplicado en sistemas prácticos que cubren una gran diversidad de disciplinas. Podemos citar, por ejemplo, la aplicación de técnicas en el procesamiento de voz y la transmisión de señales en canales de teléfono, el procesamiento de imágenes y su transmisión, en sismología y geofísica, la exploración de petróleo, la detección de explosiones nucleares, el procesamiento de señales recibidas del espacio exterior y una basta variedad de otras aplicaciones,

Pero también la implementación digital tiene sus limitaciones. Una limitación práctica es la velocidad de operación de los convertidores A/D y los procesadores de señal digital. Podemos ver que las señales que tienen anchos de bandas demasiado grandes requieren convertidores A/D con frecuencias de muestreo rápidos y procesadores digitales rápidos.

2.7 LA TRANSFORMADA Z

La transformada z de una señal de tiempo discreta X(n) está definida como la serie de potencias

$$X(z) \equiv \sum_{n=-\infty}^{\infty} x(n) z^{-n} \quad [1] \qquad \textbf{ecuación 2.7.1}$$

donde z es una variable compleja. La relación anterior es a veces llamada como la transformada z directa porque ella transforma la señal de tiempo discreta x(n) en su representación de su plano complejo X(z). El procedimiento inverso i.e., obteniendo x(n) de X(z) es llamado transformada z inversa.

Por conveniencia, la transformada z de una señal x(n) se denota por

$$X(z) \equiv Z[x(n)] \qquad \textbf{ecuación 2.7.2}$$

donde la relación entre x(n) y X(z) es indicada por:

$$x(n) \overset{z}{\longleftrightarrow} X(z) \qquad \textbf{ecuación 2.7.3}$$

Ya que la transformada z es una serie infinita de series, existe sólo para aquellos valores de z para los que la serie converge. La región de convergencia (ROC) de X(z) es un conjunto de todos los valores de z para los cuales X(z) tiene un valor finito.

La ROC de una señal de desviación finita es el plano z entero, excepto posiblemente los puntos z_0 y/o $z=\infty$. Estos puntos son excluidos por lo que $z^{k}(k>0)$ se hace ilimitado para $z=\infty$ y $z^{-k}(k<0)$ se hace ilimitado para $z=0$.

Desde un punto de vista matemático la transformada z es simplemente una representación alternativa de una señal.

En muchos casos podemos expresar la suma de una serie finita o infinita para la transformada z en una expresión cerrada. En estos casos la transformada z ofrece una alternativa compacta para representar una señal.

2.7.1 PROPIEDADES DE LA TRANSFORMADA Z

La transformada z es una herramienta muy poderosa para el estudio de señales y sistemas de tiempo discretos. El poder de esta transformada es una consecuencia de algunas de las muy importantes propiedades que la transformada posee.

1. Linealidad

Si

$$x_1(n) \xleftrightarrow{z} X_1(z) \qquad \textbf{ecuación 2.7.4}$$

y

$$x_2(n) \xleftrightarrow{z} X_2(z) \qquad \textbf{ecuación 2.7.5}$$

entonces

$$x(n) = a_1 x_1(n) + a_2 x_2(n) \xleftrightarrow{z} a_1 X_1(z) + a_2 X_2(z) \qquad \textbf{ecuación 2.7.6}$$

para cualquier constantes a_1 y a_2.

La propiedad de linealidad puede ser fácilmente generalizada para un número arbitrario de señales. Básicamente, esto implica que la transformada z de una combinación lineal de señales es la misma combinación lineal de sus transformadas z. Por lo que la transformada z nos ayuda a encontrar la transformada z de una señal expresándola como una suma de señales elementales, para las que, la transformada z es ya conocida.

2. Cambio de tiempo

Si

$$x(n) \xleftrightarrow{z} X(z) \qquad \textbf{ecuación 2.7.7}$$

entonces

$x(n-k) \xleftrightarrow{z} z^{-k} X(z)$ **ecuación 2.7.8**

La ROC de $z^{-k}X(z)$ es la misma que la de $X(z)$ excepto por $z=0$ si $k>0$ y $z=\infty$ si $k>0$. La prueba de esta propiedad se deduce inmediatamente de la definición de la transformada z dada en la **ecuación 2.7.5.**

La propiedad de linealidad y cambio de tiempo son la llave que hacen a la transformada z sea extremadamente útil para el análisis de sistemas discretos LTI.

2.8 CONVERTIDORES ANALÓGICO-DIGITAL (A/D)

Los convertidores analógico-digital convierten una cantidad analógica (usualmente voltaje) a una señal digital[3]. Existen dos métodos populares para construir un convertidor A/D. Ambos utilizan comparadores. Un comparador que tiene 2 entradas y una salida. La salida es esencialmente digital; es uno si la entrada A es mayor que la entrada B y cero de otra manera.

Un método para efectuar las conversiones A/D es el de aproximaciones sucesivas; para efectuar la conversión, un convertidor digital- analógico y un comparador son usados. Las entradas al comparador son el voltaje desconocido, que va a ser convertido a una señal digital y la salida a un convertidor A/D. Sucesivamente entradas digitales más largas son aplicadas al convertidor D/A hasta que la salida se aproxima lo más cerca posible al voltaje analógico. Entonces el comparador detiene la entrada digital para que no se incremente. El número digital hasta este tanto representa el voltaje analógico de entrada.

Por otro lado, los convertidores "flash" son los convertidores A/D más rápidos, pero tambien son los más complejos. Un convertidor "flash" para una salida de n-bits requiere 2n - 1 comparadores, pero realiza la conversión en un solo ciclo.

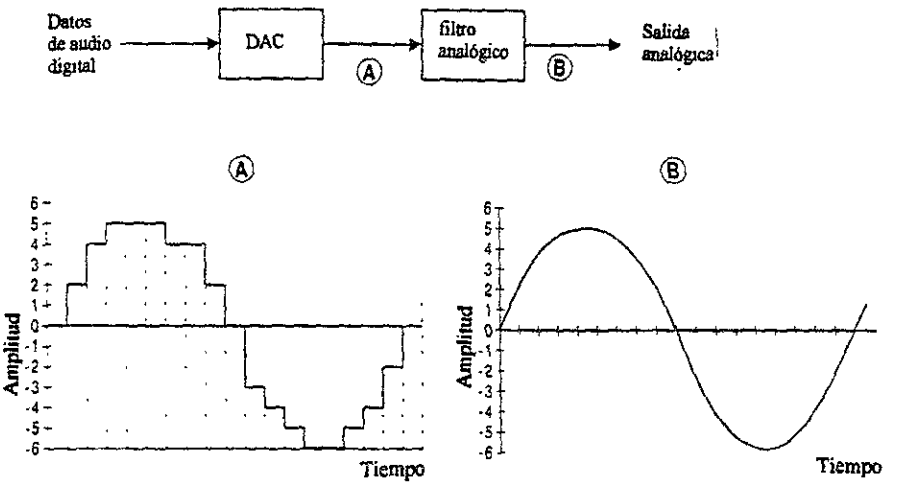
2.9 CONVERTIDORES DIGITAL A ANALÓGICO

En términos muy sencillos, un convertidor analógico a digital transforma una serie de palabras digitales de varios bits a una representación analógica "equivalente" de una señal[4]. La representación intuitiva producirá una onda en forma de escalera donde el ancho de cada "escalón" se definirá por un período de muestreo y la amplitud definida de la palabra digital.

Esto puede ser logrado convirtiendo cada palabra digital a un voltaje analógico que es entonces guardado por un circuito de muestreo y retención por un período definido de muestreo. La forma de escalera es entonces suavizada para producir una señal analógica.

El espectro contenido en esta "escalera" incluye la información original deseada que existe entre DC y $F_s/2$ así como imágenes de esta información en todos los múltiplos enteros de la frecuencia de muestreo. El ancho de cada pulso impone un *(sen x)/x* en el espectro de salida que añade ceros en los múltiplos enteros de la frecuencia de muestreo en toda la función de transferencia.

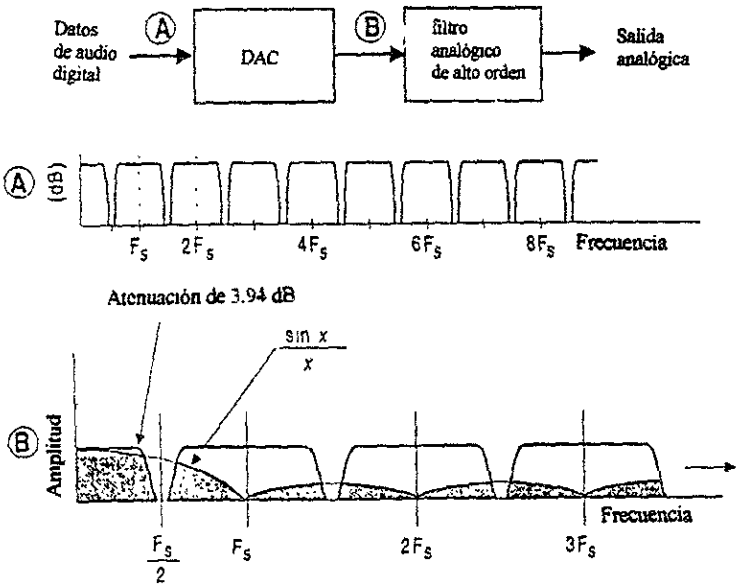
Figura 11. Diagrama de una conversión digital a analógica.



Si los componentes electrónicos y de reproducción fueran lineales a través de DC para "iluminar" las imágenes no habría problema. Pero, las no linealidades de los sistemas combinados con los componentes de imágenes de las frecuencias altas resultan en distorsión que modula la información de las frecuencias altas en el ancho de banda del audio.

Una técnica "clásica" analógica requiere de filtros de alto orden para remover las imágenes y corregir la señal para la atenuación en los 20 kHz. Este filtro es usualmente del orden de los doce polos e incluye todos los problemas asociados con los filtros analógicos; variación en la respuesta de fase, amplitud de respuesta, etc., debido a las tolerancias de los componentes y las dificultades en su construcción.

Figura 12. Conversión digital a analógica sin sobremuestreo.



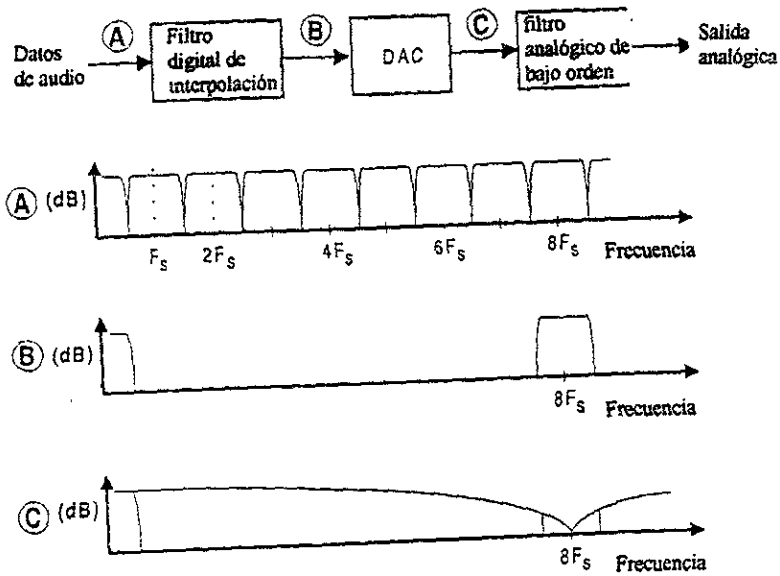
Otra alternativa, como en los convertidores analógico a digital, es el uso de técnicas de filtrado digital.

En vez de un filtro de corte para reducir la frecuencia de muestreo, un filtro digital de interpolación es usado para incrementar la frecuencia de muestreo. Cada muestra de datos de audio es introducida en el filtro de interpolación y seguido por *n* muestras de valores. Los datos son filtrados digitalmente para producir un conjunto "promedio" de puntos intermedios de datos.

El espectro interpolado de audio incluye solamente el espectro original de ancho de banda y los componentes de imágenes en los múltiplos de la frecuencia de muestreo interpolada.

El espectro interpolado también contiene ruido adicional de cuantización introducido por el filtro digital. Pero, el ruido es equitativamente distribuido entre DC y la frecuencia de muestreo de interpolación dividida entre dos, no el ancho de banda original de DC a F_s. El ruido de cuantización debido a la interpolación tiene poco efecto en el rendimiento en un filtro digital diseñado apropiadamente.

Figura 13. Conversión digital a analógico con sobremuestreo.



2.9.1 PARÁMETROS DE LOS CONVERTIDORES

Los parámetros más importantes de un convertidor A/D son[3]:

1. Resolución : el cambio de un voltaje analógico reflejada en el cambio de 1 bit en la salida digital
2. Número de bits: el número de bits binarios de salida.
3. Tiempo máximo de conversión: el peor caso de tiempo de conversión.
4. Código de salida: los convertidores A/D están disponibles ya sea en código binario o BCD
5. Rango de entrada analógica: el rango de entrada analógica que será convertida.
6. Impedancia de entrada.
7. Disipación de potencia.
8. Precio.

Los parámetros que aplican a los convertidores D/A son :

1. Resolución
2. Tamaño de bits
3. Error : el peor caso de error de salida como un porcentaje de la escala entera.
4. Modo de salida: corriente o voltaje de salida están disponibles para los convertidores D/A.
5. Tiempo de establecimiento: el tiempo requerido para que el voltaje de salida analógico sea confirmado.
6. Código de entrada: los convertidores D/A pueden aceptar entradas binarias o BCD.
7. Rango de salida analógico
8. Referencia.
9. Precio.

2.10 FILTROS DIGITALES

El concepto de filtrado es una analogía entre la acción de un depurador físico o tamiz a la acción de un operador lineal en secuencias cuando el operador es visto en dominio de frecuencias[5]. Este filtro puede permitir que ciertos componentes de frecuencia en la entrada al pasar sin cambios a la salida mientras bloquea otros componentes. Naturalmente cualquiera de estas acciones tendrá su resultado correspondiente del dominio del tiempo. Este modo de ver los operadores lineales abre un área grande en el análisis teórico y provee gran entendimiento de los sistemas digitales.

Existen dos clases extensas de filtros digitales. Existe la ecuación de diferencias como un operador general:

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q) - \sum_{p=1}^{P-1} a_p y(n-p) \qquad \textbf{ecuación 2.10.1}$$

Nótese que la suma infinita ha sido reemplazada por sumas finitas. Esto es necesario para que los filtros puedan ser realizados físicamente.

La primera clase de filtros digitales tiene un a_p igual a cero para todas las p. El nombre común para este tipo de filtros es el de filtros de respuesta finita de pulsos (FIR), ya que su respuesta a los impulsos disminuya en un número finito de muestras. Estos filtros también son llamados de promedio movible (MA), ya que su entrada es simplemente un promedio de los valores de entrada.

$$y(n) = \sum_{q=0}^{Q-1} b_q x(n-q)$$ **ecuación 2.10.2**

Existe una ventana para estos pesos (b_q) que toman exactamente el valor más reciente Q de *x(n)* y los combina para producir la salida.

La segunda clase de filtros digitales es la de los filtros de respuesta infinita de pulsos (IIR). Esta clase incluye los filtros autoregresivos (AR) y la forma más general ARMA: en el caso AR todas las b_q igual para q=1 a Q –1 son puestas a cero.

$$y(n) = x(n) - \sum_{p=1}^{P-1} a_p y(n-p)$$ **ecuación 2.10.3**

Para los filtros ARMA la ecuación general aplica. En ambos casos de filtros IIR, una respuesta de un solo curso en la entrada puede proveer una salida de duración infinita dadas ciertas condiciones de los coeficientes. La estabilidad puede ser un problema para los filtros IIR ya que si se escogen coeficientes pobres la salida puede crecer sin control para ciertas entradas.

CAPÍTULO 3

DESARROLLO DE LA TARJETA DE ADQUISICIÓN DE DATOS

3.1 INTERFAZ DE LA COMPUTADORA

3. 1. 1. ENTRADA Y SALIDA PROGRAMADA

Existen tres técnicas posibles para las operaciones de las entradas de salida[6]. En el modo de entrada y salida programada, los datos son intercambiados entre el CPU y el módulo de entrada y salida. El CPU ejecuta un programa que da control directo entre las operaciones de entrada y salida incluyendo el estado del dispositivo, mandando los comandos de lectura y escritura y transfiriendo los datos. Cuando el CPU manda un comando al módulo de entrada y salida debe esperar hasta que la operación es concluida. Si el CPU es más rápido que el módulo de entrada y salida, esto será una pérdida de tiempo del CPU. Con las interrupciones manejadas de entrada y salida el CPU da un comando de entrada y salida, continúa ejecutando otras instrucciones y es interrumpida por el módulo de entrada y salida cuando la última ha completado su trabajo. Con la entrada y salida programada e interrumpida, el CPU es responsable de extraer los datos de la memoria principal para la salida y almacenamiento de datos en la memoria principal para la entrada. La alternativa es conocida como

acceso directo de memoria (DMA); en este modo el módulo de entrada y salida, y la memoria principal intercambian los datos directamente sin que el CPU intervenga.

3. 1. 2 COMANDOS DE ENTRADA Y SALIDA

Para ejecutar una instrucción relacionada con la entrada y salida de datos el CPU asigna una dirección, especificando el módulo de entrada y salida y el dispositivo externo así como un comando de entrada y salida. Existen cuatro tipos de comandos que un módulo puede recibir cuando es direccionado por el CPU. Se clasifican como control, prueba, lectura y escritura.

El comando de control es usado para activar un periférico y decirle qué hacer. Por ejemplo a la unidad de cinta magnética puede dársele la instrucción de tener un movimiento hacia delante o hacia atrás de un registro. Estos comandos están adaptados a un tipo particular de dispositivo periférico.

Un comando de prueba es usado para verificar los diversos estados asociados con los módulos de entrada y salida y sus periféricos. El CPU deseará saber si el periférico de interés está activado o disponible para su uso. También querrá saber si la operación reciente está completada o si algún error ha ocurrido.

Un comando de lectura causa que el módulo de entrada y salida obtenga un grupo de datos de un periférico y los coloque en un registro de datos interno. El CPU puede obtener entonces el conjunto de datos solicitando que el módulo de entrada y salida le coloque en el "bus" de datos. A la inversa, el

comando de escritura causa que el módulo de entrada y salida tome un grupo de datos de "bus" de datos y subsecuentemente lo transfiera al periférico.

En el programa se utilizó la función inpw() en la clase CtesisApp::OnIdle para leer los contadores del SEMIFIFO(WREG y RREG) y los datos de la memoria(PORT_A). Esta función regresa un número de 16 bits sin signo del puerto especificado.

3. 1. 3 ENTRADA Y SALIDA INTERRUMPIDA

El problema con la entrada y salida programada es que el CPU tiene que esperar un tiempo largo para que el módulo de entrada y salida esté listo para la recepción o transmisión de datos. El CPU, mientras que espera, debe de consultar el estado que guarda el módulo de entrada y salida. Como un resultado, el nivel de desempeño del sistema es severamente degradado. Como alternativa es que el CPU asigne un comando de entrada y salida al módulo y vaya a hacer otro trabajo útil. El módulo de entrada y salida entonces interrumpirá al CPU para requerir la implementación del servicio cuando esté lista para intercambiar datos con el CPU. El CPU, entonces ejecutará la transmisión de datos y regresará a su procesamiento anterior.

3. 1. 4 DESVENTAJAS DE LA ENTRADA Y SALIDA PROGRAMADA E INTERRUMPIDA

La entrada y salida interrumpida, aunque más eficiente que la entrada y salida programada, aún requiera de la intervención activa del CPU para transferir datos entre la memoria y el módulo de

entrada y salida y cualquier dato transferido debe pasar a través del CPU. Entonces, ambas formas de entrada y salida sufren de dos desventajas inherentes:

1. El ritmo de transmisión está limitado por la velocidad en que el CPU puede probar y dar servicio a un dispositivo.

2. El CPU está atado en manejar la transmisión de entrada y salida; un número de instrucciones debe de ser ejecutada para cada transmisión.

Existe cierto beneficio en los puntos antes descritos, si se considera la transmisión de un bloque de datos. Usando la entrada y salida programada, el CPU está abocado a la entrada y salida y podrá desplazar los datos a un ritmo muy alto a expensas de no poder efectuar ninguna otra función. La entrada y salida interrumpida libera de cierta manera al CPU a expensas del ritmo de la transmisión de datos.

Cuando un volumen de datos debe de ser movido una técnica mucho más eficiente es requerida: acceso directo de memoria (DMA)

3. 1. 5 ACCESO DIRECTO DE MEMORIA

El DMA requiere un módulo adicional en el sistema de "bus". El módulo DMA es capaz de mimetizar al CPU y, de cierto modo, tomar control sobre el sistema a través del CPU. La técnica funciona del modo siguiente. Cuando el CPU requiere leer o escribir un bloque de datos asigna un comando al módulo DMA, mandando a éste la siguiente información:

1. Si se requiere de lectura o escritura.
2. La dirección del dispositivo de entrada y salida involucrado en el proceso.
3. La posición inicial en la memoria que va a ser leída o escrita.
4. El número de palabras que va a ser leído o escrito.

El CPU entonces continúa con su otro trabajo. Se ha delegado la operación de entrada y salida al módulo DMA el cual tendrá a su cargo dicha función. El módulo DMA transfiere el bloque entero de datos, una palabra en cada tiempo, directamente o de la memoria, sin pasar a través del CPU. Cuando la transmisión se completa, el módulo DMA manda una señal de interrupción al CPU involucrándolo solamente al inicio y final de la transmisión.

El módulo DMA necesita tomar control del "bus" de datos para transferir datos a o de parte de la memoria. Para este propósito el módulo DMA debe usar el "bus" solamente cuando el CPU no lo necesita o debe ser forzado el CPU a suspender temporalmente su operación. La técnica anteriormente descrita es más común y se le llama *robando un ciclo* ya que el módulo DMA en efecto roba un ciclo del "bus".

3. 1. 6 TÉCNICAS DE DECODIFICACIÓN

El método más común para crear una interfase en una computadora, como la PC, es a través del uso de un registro programable digital de entrada o salida[7]. Con los registros digitales de entrada y salida el micro procesador puede escribir datos en un registro, tratando los registros como un puerto de entrada y salida o una dirección de memoria. La salida de estos registros puede ser conectada a un

dispositivo de interfase, como un "relay". Entonces, al escribir datos a un registro de salida, es posible activar o desactivar un "relay". Los registros digitales de entrada son similares pero son utilizados para crear muestras del estado de las señales conectadas a sus entradas. Por ejemplo, si en un programa se requiere determinar si un interruptor está abierto o cerrado, el interruptor puede ser conectado a la entrada de un registro digital y su estado leído y determinado. Un registro digital de entrada puede imaginarse como una dirección de memoria o de un puerto de entrada que está conectado a una dirección individual de bits. Cuando leemos los datos, resulta en un reflejo del estado de las señales. En general, los registros digitales de entrada y salida permiten al micro procesador tener información sobre el mundo exterior y emitir señales de control que causen acciones fuera de la computadora.

3.1.6. 1. CIRCUITO DE DECODIFICACIÓN "BUS BUFFER"

El "bus" de la PC soporta nueve bits de direcciones para decodificar direcciones del puerto de entrada y salida. Los bits A0 a A8 pueden decodificarse para seleccionar 512 direcciones de puerto. El bit A9 es usado como un décimo bit en la decodificación pero su propósito es el indicar si el "bus" del sistema es la causa para la instrucción de entrada o salida. La señal AEN del "bus" debe ser usada también en la decodificación. Su propósito es el de desactivar la decodificación de una dirección de puerto de entrada o salida durante las operaciones en el ciclo DMA. Con relativamente pocas direcciones de entrada y salida, soportadas en el sistema de "bus", existe una gran posibilidad que una decodificación fija quede sobrepuesta con otra tarjeta en el sistema del "bus". Para resolver este problema potencial el circuito está diseñado para que ocho direcciones decodificadas puedan ser puestas en cualquier parte de las 512 direcciones posibles. Esto se logra comparando los valores en los interruptores del "dip" con el valor de dirección presente en el "bus". El circuito comparador entonces

revisa los siete bits de direcciones A3 hasta A9 con los valores de los interruptores y genera una señal de selección para la tarjeta.

En la tarjeta la parte que realiza esta función se encuentra en el módulo PORTDECO de la compuerta lógica programable y en el circuito integrado 74688. Los datos AEN y A9-A3 son comparados con datos prefijados (AEN=0, A9=1, A8=1, A7=0, A6=0, A5=0, A4=0,A3=0) para generar a la salida del circuito comparador 74688 una señal PQ en las direcciones 300h a 307h.

Esta señal PQ en la compuerta programable activa el decodificador 3 a 8 del módulo PORTDECO(Anexo B). Las señales A2, A1 y A0 entran en el decodificador para generar las 8 la señal de activación de las direcciones. Éstas se multiplican por las señales inversas IOR y/o IOW para generar las señales RR0(para la lectura en el puerto 300h para obtener los datos de la memoria), RR2(para la lectura en el puerto 302h para obtener el número de datos escritos por el convertidor analógico a digital) y RR3(para la lectura del puerto 303h para obtener el número de datos leídos por la computadora).

3. 2 ESTRUCTURA DE DATOS EN C++

Los apuntadores y arreglos permiten que los datos sean un sinónimo de arreglos en una lista que puede ser fácilmente accesada por el programa[7]. Los apuntadores también permiten a los arreglos convertirse en funciones eficiente y dinámicamente que son creadas en la memoria. Cuando tipos diferentes de datos que son lógicamente relacionados deben manipularse, el uso de muchos arreglos se convierte en deficiente. Mientras que siempre es necesario procesar los tipos de datos individualmente,

generalmente es deseable mover todos los tipos de datos como una unidad. Los datos en C llamados estructuras permiten que nuevos datos sean definidos como una combinación de cualquier número de tipos estándar de datos en C. Una vez que el tamaño y tipo de datos contenidos en la estructura han sido definidos las estructuras nombradas podrán ser usadas como cualquier otro tipo de datos en C. Los arreglos de estructuras, apuntadores de estructuras y estructuras contienen a su vez otras estructuras que deberán ser definidas.

Una de las desventajas de las estructuras definidas por el usuario es que los operadores estándar en C no trabajan con estructuras nuevas de datos. Aunque, funciones o macros son usualmente creados para manipular las estructuras definidas por el usuario. Como un ejemplo de lo anterior algunas de las funciones y macros requeridos para manipular estructuras de datos complejos flotantes, son empleados.

3.3 ELEMENTOS DE CONTROL

Diferente a otras aplicaciones del procesador como computadoras personales PC y estaciones de trabajo, el controlar y manipular elementos en aplicaciones especializadas de control están inscritas en su aplicación. El usuario del producto está sólo preocupado por la interfase del usuario (como pantallas, teclados y comandos de alto nivel). Raramente el usuario conoce el control especializado adentro.

Es, sin embargo, vital para el diseñador de productos de control especializado el seleccionar el controlador y dispositivos acompañantes más adecuados. Productos de control especializado son encontrados en todos los segmentos del mercado: consumidor, periféricos de PC, telecomunicaciones y en la industria. Muchos de los productos de control especializado deben tener requerimientos

especializados: bajo consumo de energía, bajo costo, tamaño pequeño y un sistema de alto nivel para su integración.

3. 3. 1 MICROCONTROLADORES

Típicamente la mayoría de los sistemas de control especializado son diseñados alrededor de micro controladores que integran memoria programable en el circuito integrado, memoria de datos (RAM) y diversas funciones periféricas, como relojes y comunicación serial[8].

El microcontrolador es una unidad completa e independiente que requiere un mínimo de circuitos exteriores para hacerlo trabajar. Usualmente, todo lo que necesita es una fuente de poder, un oscilador externo como uno de cristal o cerámico y algunos capacitores.

El área de aplicación de los microcontroladores es muy grande y variada, dispersa en varios productos manufacturados incluyendo aviones, automóviles, productos domésticos, equipo de oficina, equipo médico, sistemas de seguridad, dispositivos de telecomunicaciones, juguetes, juegos y aplicaciones industriales como la instrumentación, registro de datos y control de producción, entre otras aplicaciones.

3. 3. 2. LÓGICA PROGRAMABLE

Los dispositivos de lógica programable (PLD) son circuitos integrados que contienen mucha lógica en su paquete[3]. También contienen interruptores internos que están intactos y hacen conexiones entre las puertas adentro del PLD. Estos interruptores pueden abrirse selectivamente por un dispositivo llamado el *programador del PLD* para configurar el circuito integrado que desarrollará las operaciones lógicas requeridas por el ingeniero. Para determinar cómo funciona un PLD, no es suficiente con especificar el número de parte del PLD; el usuario también deberá saber cuáles conexiones han sido abiertas y cuáles permanecen intactas. El abrir conexiones en un PLD es similar a poner un programa en la computadora; en ambos casos se determina cómo el dispositivo se comporta.

Algunos de los PLD'S más comunes son los circuitos integrados de lógica ordenada programable, que contienen puertas AND alimentando puertas OR. Estos circuitos lógicos nos permiten implementar circuitos integrados PAL en sustitución de circuitos discretos como **74x00, 74x08** y así sucesivamente. Los circuitos integrados PAL contienen más lógica que los circuitos discretos y un solo PAL puede desarrollar muchas y diversas funciones lógicas; no está solamente limitado a ser compuerta AND o NAND.

Existen dos tipos de PAL, combinatorios y con registro. Un PAL combinatorio contiene solamente compuertas, mientras que un PAL con registros contiene compuertas y "flip-flops" o registros. Los PAL'S con registros, también llamados PAL, son más complejos. Y se designan con una R en su designación.

La serie FOUNDATION provee un sistema de diseño completo, listo para usarse o para la implementación de dispositivos de lógico programable de Xilinx. El sistema de diseño FOUNDATION ofrece 3 maneras para programar los circuitos integrados:

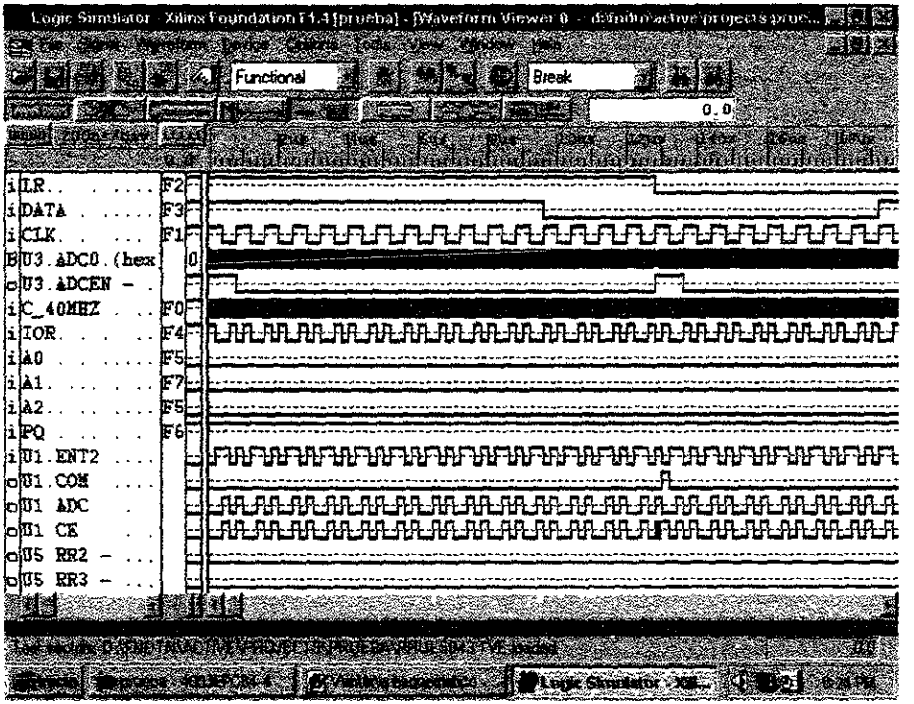
1. A través del Compilador XVHDL.
2. A través del Compilador XABEL (algunas de sus propiedades no son compatibles para FPGA.
3. Por medio del editor esquemático, por medio de dibujos.
4. En el editor de diagramas de estado para circuitos de diagramas de estado.

El editor esquemático es una herramienta de diseño primaria para la entrada de datos. Soporta la creación de esquemáticos jerárquicos para múltiples láminas. Sus principales características son:

- Soporte para múltiples láminas y esquemáticos jerárquicos.
- Integración con el Simulador Lógico que provee una simulación no esquemática.
- Integración con herramientas de diseño primarias para la entrada de datos de tipo no esquemático (Editores HDL y de diagramas de estado), que se proveen para el uso de macros no esquemáticos.
- Soporte de esquemáticos FPGA.
- Importación de esquemáticos Viewlogic.

Por lo sencillo de su empleo se utilizó el editor esquemático. Permitió además, junto con el Simulador Lógico, la verificación de los puntos más importantes y críticos del programa. Esto resultó de gran ayuda ya que a través de la simulación se pudo experimentar con el circuito, lo que permitió optimizar el diseño.

Figura 14. Ventana del simulador de Foundation.



3.3.2.1 ARREGLOS DE COMPUERTAS

Los arreglos de compuertas son una nueva forma de lógica que puede usarse para circuitos más complejos, que requieren un número elevado de compuertas. Son más grandes y más flexibles que los PAL'S y puede manejar muchas más funciones. También requieren más capacidad de entrada y salida y por lo tanto necesitan más "pines" que el DIP puede proveer. Los arreglos de compuertas están empaquetados en PLCC o PGA.

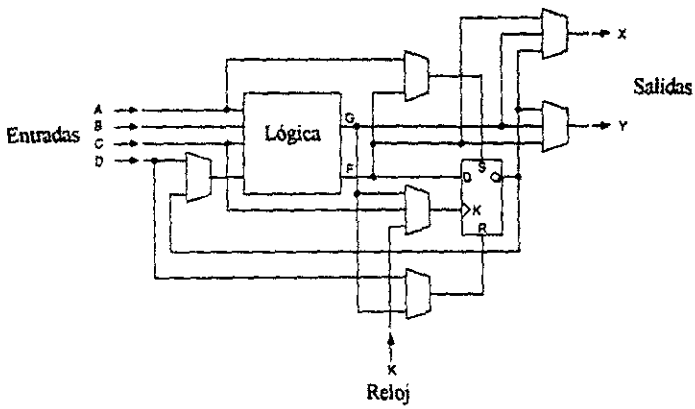
3. 3. 2. 2. ESTRUCTURAS DE LOS ARREGLOS DE COMPUERTAS

Internamente un arreglo de compuertas consiste en tres tipos de circuitos lógicos[3] [9]:

1. Bloques de Lógica configurable (CLB) o Macro Celdas
2. Bloques de Entrada y Salida
3. Matriz de Conexión

Un CLB desarrolla una operación lógica en un conjunto de entradas. El diagrama se muestra a continuación:

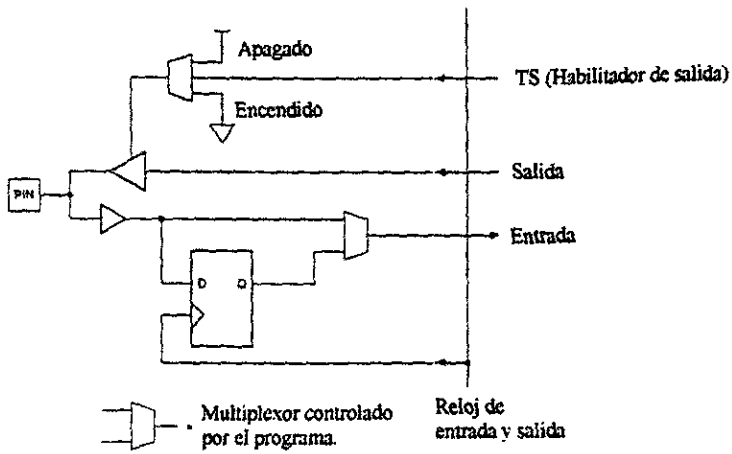
Figura 15. Bloque de lógica configurable.



La caja marcada como COMLOGIC puede desarrollar cualquier función lógica en las cuatro variables de entrada A, B, C, y D o cualquiera de dos funciones lógicas entre variables. La salida puede ser retenida en un FF, que puede ser manejado por la entrada K y SET o RESET Cualquiera de las dos entradas puede ser FG o la salida del FF La opción puede ser llevada a cabo por el usuario.

Un bloque de entrada y salida muy sencillo se muestra a continuación. Éste es conectado a un "pin" de un empaquetado PGA o PLCC. En la salida la señal puede ser de 3-estados y controlado por la señal OUTPUT ENABLE. Bloques más sofisticados de entrada y salida permiten la inversión de la salida o el control de su ritmo de corte.

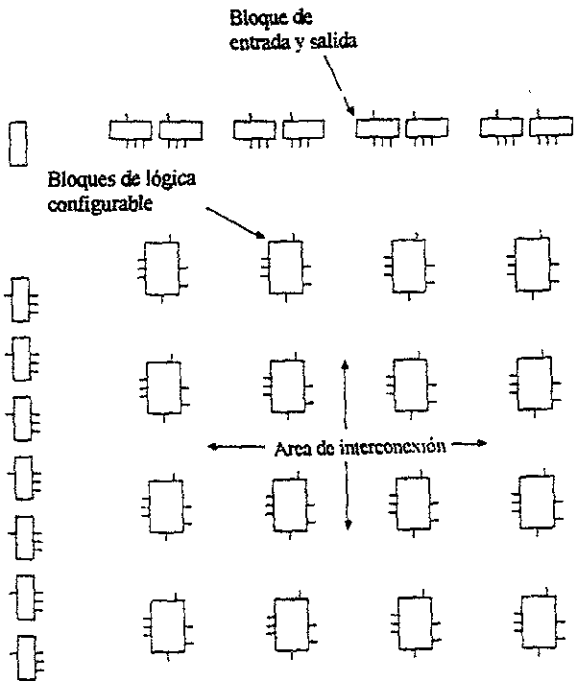
Figura 16. Bloque de entrada y salida



Una señal en su "pin" de entrada puede ser conectada directamente a la línea IN o puede ser cerrada o abierta por un FF. Otra vez esta opción será especificada por el usuario

Existen muchos CLB'S y bloques de entrada y salida en un circuito integrado de arreglo de compuertas. Las interconexiones entre estos bloques están controladas por una matriz de conexión. La estructura de un arreglo de celdas lógicas se muestra a continuación. Los CLB'S se encuentran en el área interna. Los bloques de entrada y salida están alrededor de la periferia donde tienen acceso a los "pines" externos. El área de interconexión es controlada por la matriz de conexión.

Figura 17. Estructura de un arreglo de celdas lógicas.



3. 4 MEMORIAS

Las computadoras y muchos otros circuitos electrónicos requieren la habilidad de retener o recordar grandes cantidades de información[3]. La memoria es un componente esencial de la computadora y es usada para recordar la información en forma de bits binarios. La información es usada para controlar la operación de la computadora

En un microprocesador los datos son transferidos entre él y la memoria a través de un "bus" El "bus" es una serie de cables usados para transmitir datos. Existe solamente un cable por cada bit que se transmite. Los datos son transferidos en paralelo; todos los bits son mandados a través del "bus" al mismo tiempo.

1."Bus" de datos.- el "bus" de datos es bidireccional; los datos fluyen en ambas direcciones. Van de la memoria al microprocesador durante el ciclo de lectura y del microprocesador a la memoria durante el ciclo de escritura.

2."Bus" de direcciones.- este "bus" lleva la dirección de la memoria del microprocesador a la memoria.

3."Bus" de control.- este "bus" lleva líneas que controlan la operación de la memoria del microprocesador a ésta.

La memoria se subdivide en grupos de bits llamados **palabras**. Cada una de la muchas palabras en la memoria se almacena en una dirección particular. Los datos son escritos en la memoria una palabra a la vez y preservados para un uso futuro. Los datos son leídos un tiempo después cuando la información es requerida. Durante el intervalo entre la escritura y la lectura, otras palabras pueden ser almacenadas o leídas en otras direcciones. De esta manera, cada palabra en la memoria tiene dos atributos, su dirección, que la localiza en la memoria, y los datos que se almacenan en dicha dirección.

3. 4. 1 MEMORIAS ESTÁTICAS (SRAM)

El término RAM (Memoria de acceso aleatorio) es usualmente aplicado a las memorias. Hablando literalmente, esto significa que cada palabra en la memoria es igualmente accesible o que el tiempo de acceso a cada palabra es el mismo no importando qué dirección tenga.

Las memorias RAM (escritura y lectura) están divididas en dos categorías, estáticas y dinámicas. Las memorias estáticas SRAM guardan su información en "flip-flops" y son muy fáciles de

usar. Las memorias dinámicas DRAM almacenan su información en cargas eléctricas en capacitores. Pero estas cargas sufren de "fugas" y los capacitores deben de ser recargados periódicamente.

Las memorias SRAM son acomodadas en matrices de n palabras por m bits. Éstas normalmente tienen las siguientes entradas y salidas:

1. m bits de salida.
2. m bits de entrada.
3. n bits para direcciones (para 2^n palabras).
4. Una entrada READ/WRITE.
5. Una entrada ENABLE o CHIP SELECT.

Por las características anteriormente descritas se optó por una memoria de tipo estática Por esta razón y la característica de mínimo retraso(12ns) se decidió utilizar en el diseño una memoria IDT71016. Ésta consta de 44 pines, que deben ir conectados de la siguiente manera:

1. A0 a A15 conectadas a las salidas del módulo SEMIFIFO, DIR[15:0], de la compuerta lógica programable.
2. CS(selector del circuito integrado) a la salida CE de la compuerta lógica programable.
3. WE(habilitador de lectura) a la salida WE de la compuerta lógica programable.
4. OE(habilitador de lectura) a la salida OE de la compuerta lógica programable.
5. BHE(habilitador del byte de más alto valor) y BLE(habilitador de byte de más bajo valor) conectadas a tierra.
6. I/O0 a I/O15(entrada y salida de datos) a las salidas MEMDATA[15:0] de la compuerta lógica programable.

7. V_{CC} a los 5.0V de tren de la computadora.

8. V_{SS} a la tierra del tren de la computadora.

3. 4. 2 MEMORIAS DINÁMICAS (DRAM)

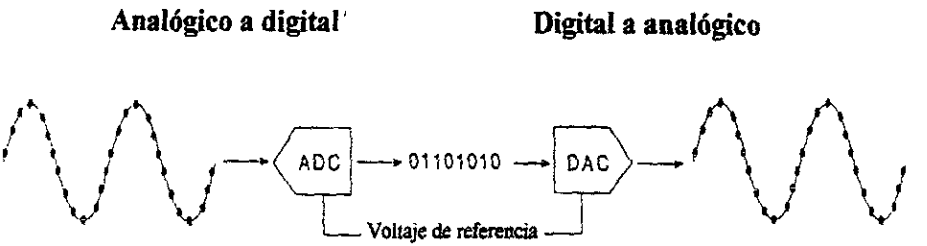
Como se mencionó anteriormente, las memorias dinámicas almacenan su información como cargas en capacitores internos, en vez que en "flip-flops"- Desafortunadamente, los datos guardados en forma de carga en un capacitor se "fugan" mientras que el tiempo transcurre. Consecuentemente, DRAM's deben ser "refrescadas" periódicamente para restaurar la información. Este requerimiento complica el circuito exterior del DRAM.

Aunque son más complicadas que las memorias SRAM dominan muchas aplicaciones que requieren memoria y son usadas ampliamente en las computadoras personales. La razón es que los capacitores requieren menos silicón que los "flip-flops". Por esto contienen hasta cuatro veces más bits.

3. 5 CONVERTIDORES ANALÓGICO-DIGITAL Y DIGITAL-ANALÓGICO

La meta de un controlador analógico a digital es la de generar una palabra digital que representa una magnitud de una señal analógica[4]. La meta de un convertidor digital a analógico es el de generar una señal analógica que representa una palabra digital.

Figura 18. Principales metas de un convertidor.



Los convertidores analógico a digital comparan una entrada (AIN) con un voltaje de referencia(VREF) y regresan un código que es el cociente de AIN y VREF.

Los convertidores digital a analógico desarrollan un voltaje de salida (o corriente) correspondiente a un código de entrada relativo a VREF.

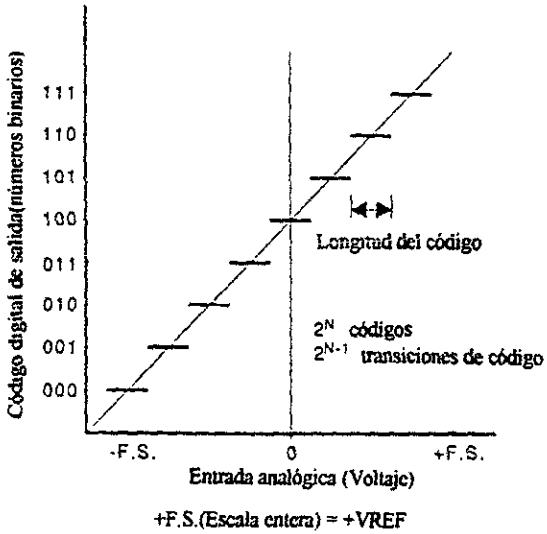
Existen muchas arquitecturas de convertidores analógico a digital; cada una teniendo sus beneficios particulares. A continuación se ilustra una tabla de la resolución típica y la frecuencia de las arquitecturas de los convertidores "flash", aproximación sucesiva y $\Delta\Sigma$.

Los convertidores "flash" sirven para digitalizar las frecuencias altas, aunque lo hace con una menor resolución.

Los convertidores de aproximación sucesiva sirven para un rango intermedio de frecuencias, usualmente usados en aplicaciones "multiplexadas" para digitalizar muchos canales de información a frecuencias más bajas

Los convertidores $\Delta\Sigma$ son convertidores sobre muestreo y se desempeñan muy bien a frecuencias de muestreo efectivas abajo de los 200 kHz.

Figura 19. Convertidor ideal analógico a digital de 3 bits.



3. 5. 1 CONVERTIDORES ANALÓGICO A DIGITAL ΔΣ

Los convertidores analógico a digital ΔΣ son usados en una gran gama de aplicaciones, desde la corriente directa hasta los 500 kHz. A frecuencias bajas la arquitectura ΔΣ nos permite una más alta resolución de conversión.

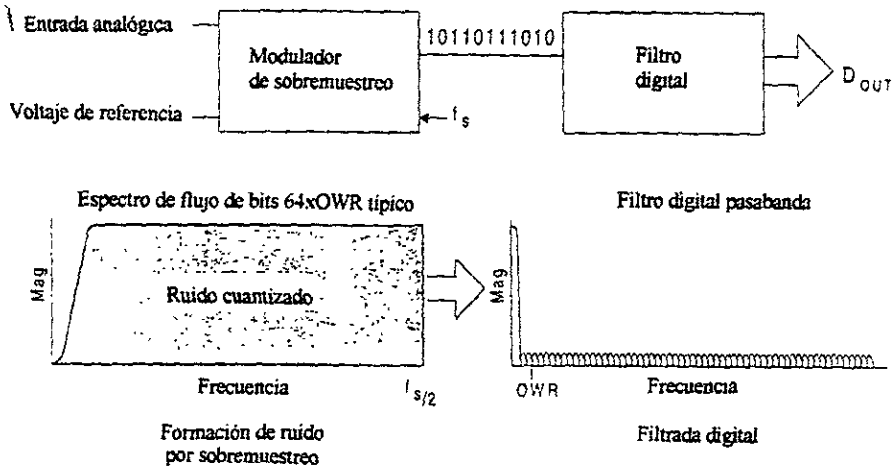
Estos convertidores se usan en sobremuestreo como método fundamental para lograr su alta resolución[4] Los convertidores incluyen un modulador de sobremuestreo seguido de un filtro digital específicamente diseñado para tal fin. El modulador usualmente se compone de un dispositivo de muestreo de un bit, trabajando a frecuencias de muestreo muy altas comparadas con las de la señal que se digitaliza. Son comunes los cocientes de su sobremuestreo de 64, 128 y 256(p.ej. en el convertidor analógico a digital seleccionado se toman 64 muestras para generar una salida).

Pero su sobremuestreo solo no es suficiente. Un dispositivo de sobremuestreo de un bit tiene un error de cuantización muy alto y consecuentemente un ruido de cuantización alto comparado con la señal de entrada. Para lograr altos cocientes de señal a ruido, el modulador de sobremuestreo incluye un filtro regulador de ruido que reduce considerablemente el ruido de cuantización sobre el ancho de banda de la señal de entrada. Esto resulta en un flujo digital de bits del modulador que está espectralmente filtrado.

El flujo de bits es procesado por el filtro digital que está diseñado para tomar ventaja del espectro del ruido. El filtro digital procesa las muestras de un bit removiendo el ruido de cuantización "fuera de banda" y al mismo tiempo aumenta la resolución del convertidor.

El convertidor analógico a digital ΔΣ usa el sobremuestreo, la reducción de ruido y el filtrado digital para lograr una velocidad pequeña y una alta resolución de salida de un dispositivo de muestreo de baja resolución a alta velocidad.

Figura 20. Convetidor analógico a digital ΔΣ



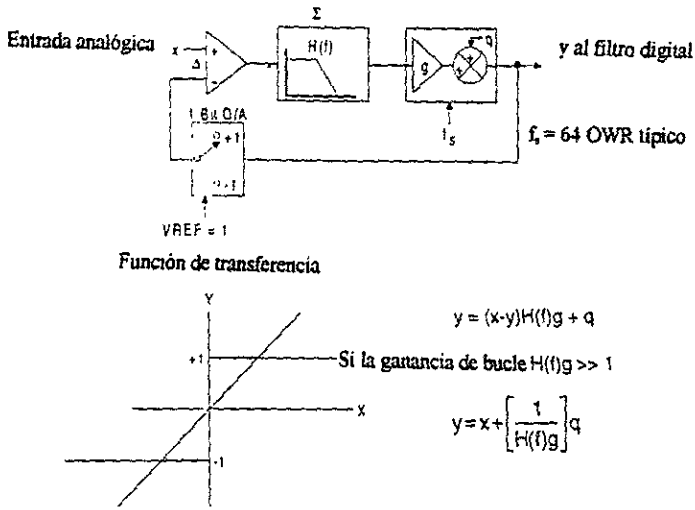
3. 5. 2 MODULADOR ΔΣ

El modulador ΔΣ consiste en un amplificador diferencial que mide la diferencia delta entre la señal de entrada y el bit de retroalimentación del convertidor digital a analógico. La delta es amplificada y aplicada como una entrada a un filtro que consiste en un integrador de muchas etapas. El integrador suma el voltaje de error, de lo anterior deriva Σ (sigma) para el bloque de filtrado. Mientras más grande sea el número de integradores en el filtro de reducción de ruido mejor será la reducción del ruido de cuantización del ancho de banda de interés.

Cada integrador adicional añade 6 dB de mejoría en la señal comparada con el ruido para cada duplicación de la velocidad de muestreo. Esto sucede ya que cada integrador adicional aumenta la caída de filtrado a seis decibeles por incremento de octava en la frecuencia. El número de integradores en el filtro de reducción de ruido indica el orden del modulador ΔΣ.

La salida del comparador resulta en un flujo de un bit, una densidad proporcional de 1 con respecto al cociente de la señal de entrada y el voltaje de referencia de la retroalimentación digital a analógica. Véase que el convertidor digital a analógico está en la señal de retroalimentación en que la salida está tratando de minimizar el voltaje de error a la entrada del amplificador diferencial

Figura 21. Modulador ΔΣ



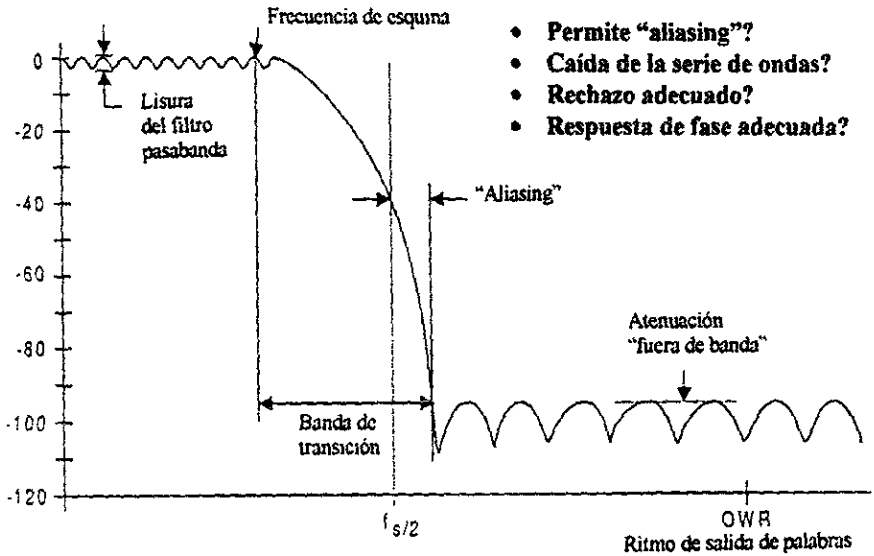
3. 5. 3 ESPECIFICACIONES Y VENTAJAS DEL FILTRO DIGITAL

En un convertidor analógico a digital ΔΣ el filtro digital domina el comportamiento de la señal en el dispositivo[4]. El filtro digital determinará:

1. El ancho de banda de la señal.
2. La lisura del filtro pasabanda.
3. La respuesta de fase a través del filtro pasabanda.
4. El retardo de grupo.
5. Las características de la banda de transición.
6. La atenuación "fuera de banda".

El filtro digital también puede ser diseñado para prevenir el "aliasing" en aplicaciones con entradas transitorias, el filtro digital puede dictar las características del tiempo de establecimiento.

Figura 22. Especificaciones de un filtro.



El filtrado digital ofrece muchas ventajas sobre el filtrado analógico;

1. Menor costo a mayor complejidad.
2. Exactitud reproducible
3. Estabilidad sobre el tiempo y temperatura.
4. Muchas más funciones complejas logradas
5. Puede ser usado para desarrollar funciones de procesamiento digital adicionales.

Ofrece mayor complejidad de funciones de filtrado a menor costo (un filtro análogo necesitará un número similar de polos y ceros para lograr un desempeño comparable). Un filtro analógico de esta complejidad no es realizable. Los filtros digitales ofrecen un desempeño lineal de fase que resulta en un retardo constante de grupo, una muy deseable cualidad para aplicaciones de audio.

Un nivel muy alto de integración soporta muchos dispositivos con las mismas características de filtrado, que permanecen constantes con el tiempo y temperatura.

Con estas características se compararon los convertidores analógico a digital de dos marcas. Se listan a continuación dichos circuitos con sus características:

Componente	**Resolución (bits)**	**Rango dinámico(dB)**	**Alimentación (V)**
CS5336 CS5338 CS5339	16	96	±5
CS5349	16	90	±5
CS5330 CS5331	18	94	+2.7 a 5
CS5389	18	107	±5
CS5390	20	110	±5
PCM1750	18	90	±5
PCM1760	20	108	±5

Debido a que los componentes de la marca CS(Crystal Semiconductor) tienen el doble de costo que los PCM(Burr-Bown), teniendo características similares, se decidió utilizar el componente PCM1760.

Para la conexión del convertidor y filtro digital se utilizaría la sugerida por el fabricante. Aún así existen 4 pines los cuales se deben conectarse de la siguiente manera:

- CLKSEL(Reloj del sistema) conectado en alto("al aire"). Esto debido a que se utilizara un cristal de 11.2896 MHz para el diseño (256 veces la frecuencia de muestreo).
- S/M (Modo amo/esclavo) conectado a tierra (modo amo).
- MODO 1 y MODO2 (formado de salida de datos) conectados en alto("al aire"). Esto ya que se desea una señal de 16 bits.
- LRSC (Reloj L/R) conectado en alto ("al aire").

3.6.1 ELECCIÓN DEL LENGUAJE

El propósito de programar en un lenguaje es el de proveer de una herramienta para que el programador pueda resolver problemas que envuelven algún tipo de información.

Se eligió el lenguaje C por las siguientes razones:

El lenguaje C tiene ventajas significativas para las aplicaciones en el procesamiento digital de señal sobre otros lenguajes como FORTRAN y Pascal. El lenguaje C está construido para encauzar el

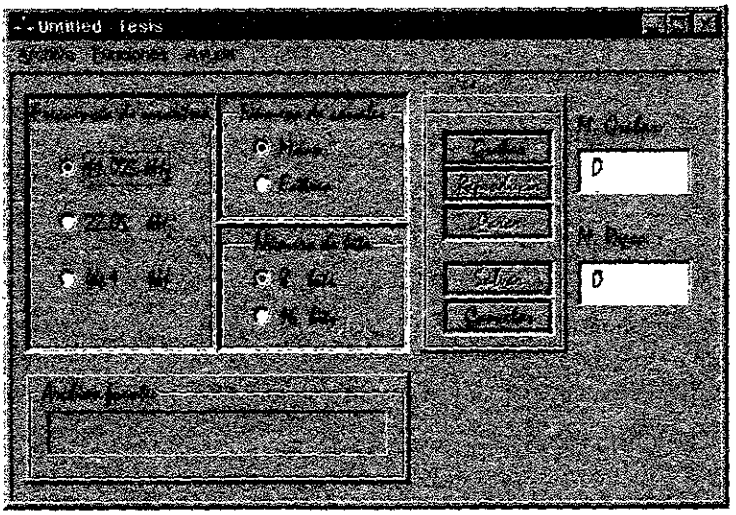
desarrollo de rutinas de librerías que pueden utilizarse como bloques exactamente en la forma requerida por el procesamiento digital de señal.

Otra razón por la que el lenguaje C es una buena opción para el procesamiento digital de señal es la popularidad y uso extendido de este lenguaje. Existen funciones diseñadas específicamente para el audio. Además el código producido por los mejores compiladores es compacto y eficiente.

Dentro del lenguaje C se eligió la versión **Visual C++ 5.0.** La característica más poderosa de este C es un "asistente" llamado **ClassWizard**[10]. Este "asistente" escribe el código de los cuadros de diálogo para uno. Además que se pueden realizar tareas de programación usando el teclado y el ratón para diseñar y escribir visualmente las aplicaciones.

Con la ayuda de este asistente se llevó a cabo la parte gráfica del programa. También se declararon las variables globales, el menú general, y los cuadros de diálogo.

Figura 23. Ventana principal del programa.



3.6.2 INVESTIGACIÓN DEL ENCABEZADO ("HEADER")

Cada archivo WAV consiste en un encabezado ("header") de tamaño fijo y un área de datos variable. El encabezado da el tipo de datos almacenado en el área asignada, el tamaño de elementos y el número de elementos en el área de datos. El área de datos contiene la información prescrita en el encabezado y es almacenada en un formato binario compatible con el formato de datos usado en la máquina.

La investigación de la forma del encabezado se realizó tomando diferentes muestras hechas en el programa **Music Center**. Se realizó también una búsqueda en diferentes libros y programas de los datos que se podrían encontrar. Con esta información se realizó la lectura del encabezado byte por byte. La estructura encontrada fue de la siguiente forma:

1. La palabra "RIFF" compuesta por 4 bytes.
2. La longitud restante del archivo en 4 bytes.
3. La palabra "wavefmt □" compuesta por 14 bytes.
4. El número de canales compuesto por 2 bytes. 1 para mono y 2 para estéreo.
5. El número de muestras por segundo compuesto por 4 bytes.
6. El promedio de datos por segundo compuesto por 4 bytes.
7. El número de bytes que componen cada bloque con un tamaño de 2 bytes.
8. El número de bits por muestra compuesto por 2 bytes.
9. La palabra data compuesta por 4 bytes.
10. La longitud restante de datos compuesto por 4 bytes.

Para desarrollar las rutinas del programa fueron necesarios estos datos.

3.7.1 INVESTIGACIÓN DE LA SEÑAL.

(GRÁFICAS DE LA SEÑAL)

El siguiente paso fue la presentación de los datos, la cual es para poder manejar la información. La investigación se realizó por medio de gráficas. De tal manera se observó que los datos de 8 bits se presentaban de un formato de carácter sin signo, mientras que los de 16 bits lo hacían en forma de entero con signo. Esta investigación es necesaria para poder desarrollar en un futuro los procedimientos del procesamiento digital de señal como el retraso de tiempo.

Se manejó en el programa un buffer de 650 datos, que es la longitud de la pantalla. Se utilizó, además, la función "getc" para obtener los datos de 8 bits, y la función "fread" para los datos de 16 bits. Los datos del encabezado empleados para las gráficas fueron el número de canales y el número de bits por muestra.

El código que se encuentra al inicio crea un objeto de contexto de dispositivo.

```
CClientDC dc(this);
```

El *contexto de dispositivo* (dc) se puede pensar como una pantalla imaginaria que reside en memoria. Una vez que se crea el dc, puede dibujarse en él como si fuera una pantalla normal, pixeles, líneas, formas geométricas, etcétera.

A continuación se declaran los tipos de plumas, que se utilizarán; uno para borrar la pantalla (PantPen) y otro para dibujar los datos(GrafPen).

Después se leen los datos de número de canales y número de bits por muestra, guardándose éstos en las variables "Channels" y "Bits" respectivamente. Esto para dividir en 4 el número de opciones:

1. Archivos con información de 8 bits, un canal. En esta sección se utiliza la función "getc()" para la lectura de datos procedentes del archivo fuente.
2. Archivos con información de 8 bits, dos canales. En esta sección se utiliza también la función "getc()" para la lectura de datos, que se realiza alternadamente para el lado derecho e izquierdo.

3. Archivos con información de 16 bits, un canal. En esta sección se utiliza "fread()" para la lectura de datos.
4. Archivos con información de 16 bits, dos canales.

Después que se escoge la opción a utilizar pasa los siguientes pasos:

1. Se selecciona la pluma para borrar la pantalla

   pOriginalPen = dc.SelectObject(&PantPen);

2. Se borra la pantalla trazando líneas de izquierda a derecha.

   dc.MoveTo(0,(10*i));

   dc.LineTo(650,(10*i));

3. Se selecciona la pluma para graficar.

   POriginalPen = dc.SelectObject(&GrafPen);

4. Se copian los datos del archivo fuente a un arreglo

   info[i] = fgetc(stream);

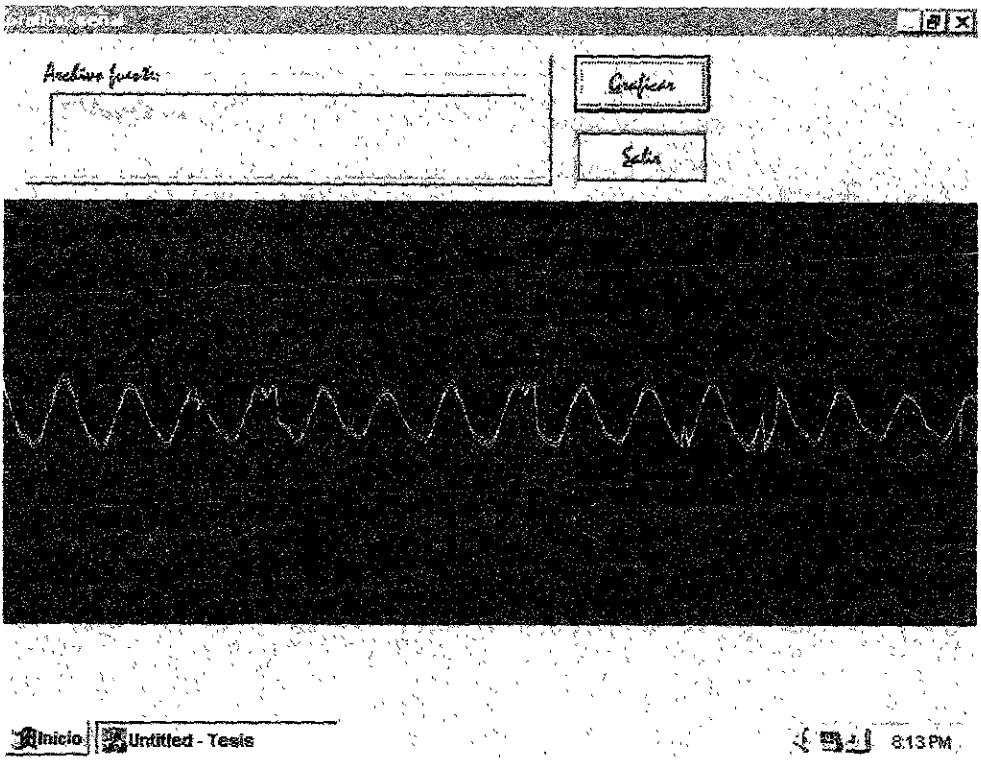
5. y se grafican a continuación.

   value = 350 – info[i];

   dc.Lineto(i, value);

El tamaño de datos dibujados en la pantalla es de 650 por cada lado utilizado. Se repiten estos pasos hasta dibujar todos los datos en la pantalla.

Figura 24. Gráfica de una señal de 16 bits, dos canales.



3.7.2 EJEMPLO DE ALGUNAS APLICACIONES (GENERACIÓN DE LA SEÑAL Y APLICACIÓN DE RETRASO("DELAY") EN UNA SEÑAL)

Ningún programa de televisión o en un CD_ROM interactivo sería completo sin la adición de efectos de sonido. Construyendo sobre la base de operaciones básicas de multiplicación y retraso, operaciones complejas como el "chorus" pueden ser desarrolladas. Estos efectos son utilizados para darles otra dimensión a la música.

De esta forma se decidió desarrollar la operación de retraso como una muestra de lo que podría desarrollarse en el futuro.

Para realizar el retraso de tiempo("delay") se aplicaron los siguientes pasos:

1. Se compone el nombre del archivo de destino.

   m_DiskList.GetText (m_DiskList.GetCurSel(),Selection);

   m_WavDestination = Selection + ":\\" + m_Dir;

2. Apertura de los archivos y toma de la longitud del archivo fuente.

   stream = fopen(m_WavSource,"r+");

   newstream = fopen(m_WavDestination,"w+");

   files = ftell(stream);

3. Lectura del promedio de bytes por segundo.

   result = fread(&nAvgBytesPerSec,sizeof(long),1,stream);

4. Se toma el número de repeticiones de las tareas "task1" y "task2".

5. Se ejecuta la primera tarea con "task1". En esta parte se copia al archivo de destino la información que no lleva retraso en el tiempo("delay").

6. Se ejecuta la segunda tarea con "task2". En esta parte de copia la información que lleva retraso en el tiempo. Primero se guarda en el arreglo "info1" la información sin retraso. Después se guarda en el arreglo "info2" la información que lleva retraso. Estos arreglos se suman y guardan en el arreglo "dest", que se copia posteriormente al archivo de destino. Esto se realiza hasta llegar al fin del archivo fuente.

También se desarrolló un generador de función senoidal para probar la manera en que se almacena la información, así como el buen uso del tipo de datos. Además con esto se demostró que la estructura

del encabezado es la correcta. Éste no se incluyó en el programa final.

Para este generador se realizaron los siguientes pasos:

1. Toma las variables del usuario (m_SampleRate,m_Channels,m_Bits) y calcula otras (BlockAlign y AvgBytesPerSec).
2. Escribe el encabezado con los datos anteriores.
3. Se crea la señal con la función "sin()".

   result = j * 1.0;

   result = sin(coefreq1 * result);

   Para los archivos de dos canales se utiliza para el otro lado la función coseno, "cos()". En las señales de 8 bits se suma al resultado 128 para convertirlo a un valor sin signo.
4. Se guardan los resultados en el archivo de destino con la función "fwrite()".

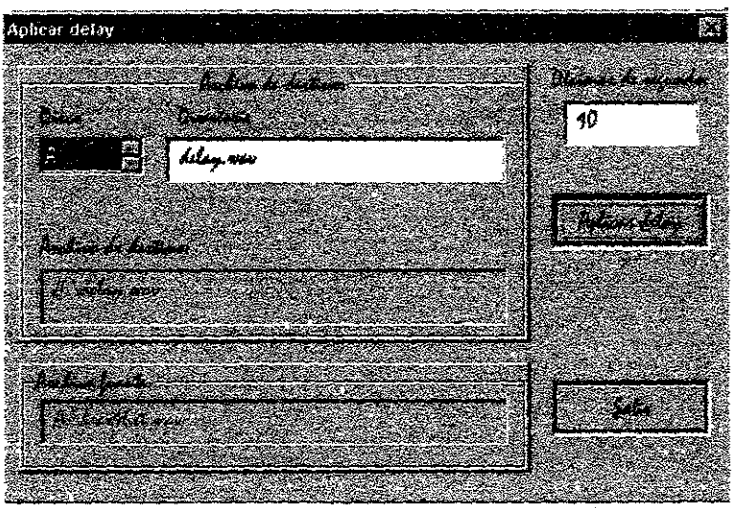


Figura 25. Ventana del cuadro de diálogo "Aplicar delay".

3.7.3 PROGRAMA DE LECTURA DE LA SEÑAL.

(PRUEBAS EN LA COMPUTADORA.

LECTURA DE DATOS Y ALMACENAMIENTO DE DATOS)

Este programa se realizó para observar si era posible obtener los datos en el tiempo correcto. Se desarrollaron programas en los cuales se obtenía el tiempo que transcurría al desarrollar las tareas de lectura y escritura. El programa se desarrolló en **Borland C++ 3.1 para Windows** Los datos obtenidos fueron:

190 ms para la lectura y escritura de 90000 bytes.

380 ms para la lectura y escritura de 180000 bytes.

570 ms para la lectura y escritura de 270000 bytes.

El programa se lleva a cabo con los siguientes pasos.

1. Se toma el tiempo de inicio y se guarda en una variable llamada "start".

   start = time(0);

2. Se realizan las funciones de lectura y escritura.
3. Se toma el tiempo al final y lo guarda en la variable "finish".

   finish = time(0);

4. Se toma la diferencia de tiempo entre "start" y "finish" y lo guarda en la variable "duration". Con esta función se aseguró un registro de tiempo más o menos preciso.

   duration = difftime(finish, start);

Las pruebas desarrolladas tuvieron el propósito de medir cuántos datos se podían guardar en el disco duro. Lo anterior debido a que el tiempo de almacenamiento de datos en el disco duro ocupa al CPU en mayor medida. Como dato importante, no fue necesario medir el tamaño de "buffer" óptimo por la razón de que éste es asignado automáticamente por el lenguaje de programación.

Se observó también, que el tiempo que utiliza la entrada y salida programada no utiliza al CPU en gran medida. No se puede decir lo mismo de la entrada y salida interrumpida que utiliza al CPU en gran medida y con el inconveniente de que ésta no puede durar mucho tiempo. El acceso de datos por DMA complica el diseño de sotfware y hardware por lo que no fue considerada.

Por todo lo anterior, los datos obtenidos apoyaron el acercamiento escogido.

3.8 ELECCIÓN DEL CONVERTIDOR ANALÓGICO A DIGITAL.

La tarjeta necesitaba tener una calidad cercana a las tarjetas que existen en el mercado, para lo que se realizó una comparación utilizando un artículo de la revista "Keyboard" [11], que compara las principales cualidades de diferentes tarjetas de sonido. Por falta de equipo se utilizó los datos de fábrica del convertidor analógico a digital para la tarjeta diseñada en este proyecto.

Por la cantidad de datos involucrados se redujeron a dos características principales:

1. Respuesta de frecuencia.

Al medir la respuesta de frecuencia, vemos lo que se llama "ancho de banda efectivo", o sea, el rango de frecuencias que, dentro de ciertos límites especificados, puede reproducirse con exactitud.

2. La relación de señal a ruido.

La señal de señal a ruido (s/n) de la tarjeta determina cuánto ruido blanco ("hiss") se añade a nuestro sistema.

Las tarjetas involucradas son:

1. Z1 con Z.WAV de Antex.
2. Sound Blaster AWE32 de Creative Labs.
3. Sound Blaster 16 Multi CD de Creative Labs.
4. Sounscape de Ensoniq.
5. Soundman Wave de Logitech.

Dentro de las tarjetas a consideración la mejor respuesta de frecuencia fue la registrada por Ensoniq de -0.10 dB a 20KHz y la relación de señal a ruido fue 80.09 de Antex.

El convertidor analógico digital utilizado, PCM1760[14], para nuestro diseño tiene un desempeño parecido a Ensoniq en cuanto a la respuesta de frecuencia y un desempeño superior al de Antex en cuanto a la relación señal a ruido al ser de 104 dB.

Si los datos del proveedor son correctos el desempeño de este convertidor es mejor que el de la mejor tarjeta que tiene un costo de 560 dólares.

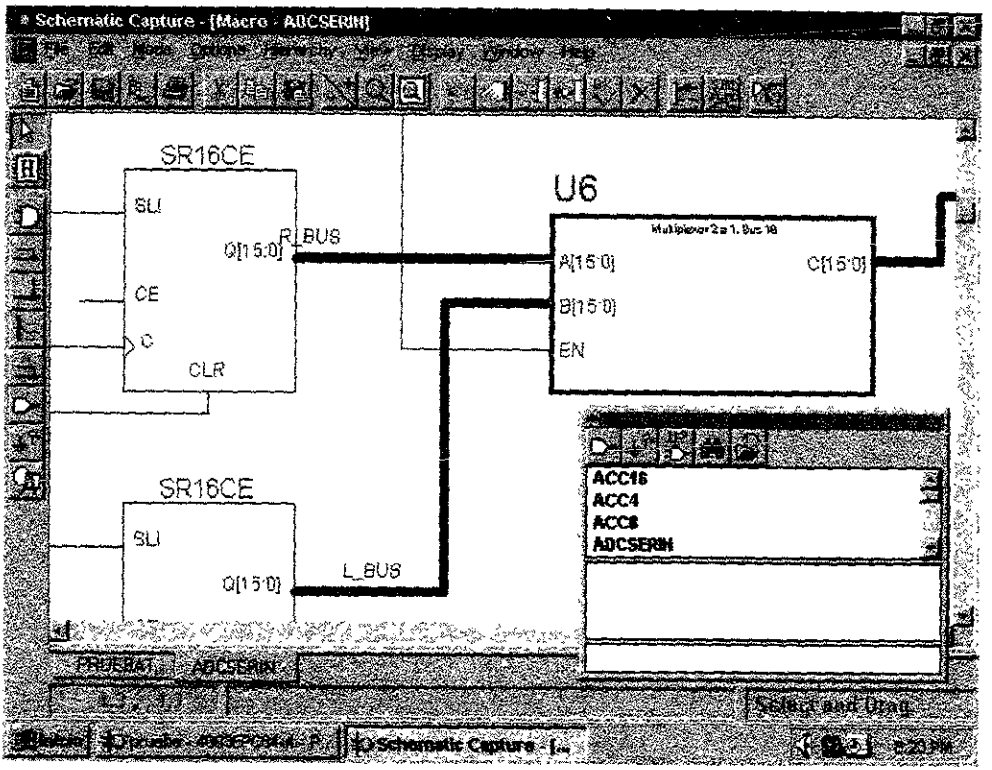
3.9 SELECCIÓN DEL ELEMENTO DE CONTROL.

La elección del elemento de control era algo importante. Éste nos permitiría reducir la complejidad del diseño, además de proporcionar mayor eficiencia en el proceso de adquisición de datos. Lo primero que se pensó fue en utilizar un microcontrolador con un convertidor analógico a digital. Se encontró en la marca Microchip una solución[12]; el PIC16C74 ofrecía esta posibilidad, pero con el inconveniente de necesitar un borrador especial para volverle a programar posteriormente. Esto no resultó y se buscó un microcontrolador más rápido sin tantas complicaciones y un convertidor analógico a digital. Al tener que utilizar un convertidor analógico a digital se buscó uno que fuera enfocado a la aplicación que se necesita, que resultó de 16 bits. Para utilizar microcontroladores de 8 bits para esta aplicación se requieren dos circuitos integrados que trabajen más rápido para manejar los datos. Aunque el PIC podía trabajar a 33MHz necesitaba de 4 ciclos para realizar una operación. La solución, hasta ese momento, fue la de considerar un microcontrolador de Atmel[13], que trabaja a menor velocidad, pero a una instrucción por ciclo. Esto hubiera sido algo lento para desplegar los 2 bytes por cada muestra a la computadora (se necesitaban 2 ciclos por dato, 1 de lectura en el puerto, y 1 para mover los datos a la memoria), ya que el tren de la computadora trabaja a 14 MHz.

El camino era buscar un microcontrolador de 16 bits u otro medio de control. La primera solución era muy costosa por el "kit" de desarrollo; la otra solución resultó ser el uso de compuertas lógicas programables de Xilinx[9]. Éstas trabajan a mayor velocidad y tienen una programación más sencilla, además de que sus empaquetados tienen mayor número de pines.

La programación necesaria era muy rutinaria por lo que su uso no resultaría más complejo, que utilizando microcontroladores. Además el programa utilizado tiene una subdivisión gráfica de simulación que resulta en menos pruebas físicas y la detección de errores con mayor facilidad.

Figura 26. Venta del editor esquemático.



3.9.1 NECESIDADES EN EL ELEMENTO DE CONTROL. CONVERTIDOR ANALÓGICO A DIGITAL (SERIAL A PARALELO). COMPUTADORA (DECODIFICACIÓN Y CICLOS DE LECTURA Y ESCRITURA). MEMORIA (SEMIFIFO)

Para la programación del circuito integrado se decidió dividir las funciones en partes operativas. Esto desarrollado para facilitar el diseño y las pruebas realizadas con el simulador que proporciona el paquete.

Las necesidades de programación resultaron en cuatro funciones principales:

1. La conversión de serial a paralelo del convertidor analógico a digital, para un manejo más sencillo de los datos

   A esta unidad se le dio el nombre de ADCSERIN. Está compuesta por dos arreglos de 16 "flip-flops" tipo D, que convierten los datos de serial a paralelo. Cada arreglo se activa alternadamente, uno para la señal del canal derecho y otro para el canal izquierdo. Éstos después pasan a un multiplexor de 16 pines el cual organiza el paso de la señal a la memoria. También existe un contador de 16 posiciones el cual da un pulso que recibe la unidad ORDEN.

2. El manejo del tren de datos a la computadora.

   A esta unidad se le dio el nombre de PORTDECO. Está compuesto por una unidad decodificadora de 3 a 8. Ésta organiza los pedimentos de información junto con las señales

AEN, IOR y IOW de la computadora. Estas señales entran en la unidad en forma negada por lo que pasan a inversores de señal, para después multiplicarse con las salidas del decodificador. El resultado de esta multiplicación son señales que indican cualquier lectura y escritura de los diferentes puertos.

3. El manejo de la memoria (datos y direcciones).

El manejo de la dirección de la memoria se hace a través de la unidad SEMIFIFO. Esta unidad de desarrollo para abatir costos que de otra manera se hubieran incrementado al utilizar un circuito integrado tipo FIFO de costo cercano a los 100 dólares. Ésta se compone de dos contadores de 16 bits, uno para la lectura de memoria y otra para la escritura. Los pedimentos de lectura y/o escritura de la computadora y el convertidor analógico a digital se desarrolla a través de la unidad ORDEN. Aquí tienen preferencias los pedimentos de la computadora. La unidad genera los pulsos de 20 ns para la activación de la memoria.

4.Manejo de información de la memoria y la computadora.

Para el paso de datos de la memoria y contadores a la computadora se diseñó la unidad ORDATA. Ésta se compone de dos "buffers" de tres estados, uno para el paso de datos a la computadora y el otro para el paso de datos de o hacia la memoria. También se encuentra un arreglo de 16 "flip-flops" tipo D para la retención de la información que va hacia la computadora, ya que los pulsos de la memoria y de ORDEN son de 20ns.

Los planos de las unidades ORDATA, ORDEN, SEMIFIFO, ADSERIN y PORTDECO se incluyen en anexo B

3.10 ELEMENTOS DE LA TARJETA.

Como primer paso se subdividió la tarjeta en 4 partes:

1. Elemento de control.

   Se compone básicamente del CPLD XC95216PQ100-10 y el circuito integrado 74C688. Este último se utilizó para eliminar el número de pines utilizados en el CPLD. El CPLD necesita de un capacitor de 0.1μF conectado de fuente a tierra. También utiliza un oscilador de cristal de 50MHz.

2. Memoria.

   La memoria es una IDT71016 de Integrated Device Technology. En el anexo se puede observar sus cualidades. Se escogió esta memoria ya que tiene un tiempo de acceso muy pequeño (12 a 25ns, dependiendo de la versión), consta de 16 bits y tiene 64000 direcciones. Esto último debido a que se necesitaba por lo menos 44100 datos de capacidad, para el tiempo de lectura y almacenaje de la computadora. Utiliza un capacitor de 0.1μF conectado de fuente a tierra.

3. Convertidor analógico a digital.

   Esta unidad es la que requiere de mayor número de elementos. Los dos principales son el filtro digital dual DF1760 y el convertidor analógico digital PCM1760. Este último requiere de una alimentación externa de 5 V, debido a la potencia que consume. Estos circuitos requieren de un oscilador de cristal de 16.9344MHz o 11.2896 MHz (384f_s y 256 f_s, respectivamente). En la práctica se observó que era más fácil conseguir este último, por lo que en el diseño se utilizará éste.

4. Pines para el tren de la PC.

Los pines de la computadora[15] que se utilizaron son:

a) A2-A9 – Tren de datos Data 0 a Data 7.

b) A22-A31 – Tren de direcciones Address 9 a Address 0.

c) A11 - Habilitador de direcciones AEN.

d) B13 - Pin de escritura entrada y/o salida IOW.

e) B14 - Pin de lectura entrada y/o salida IOR.

f) D2 - Habilitador de 16-bits para entrada y salida.

g) D16 - Pin VDC de 5 voltios.

h) D18 - Pin de tierra.

i) C11-C18 - Tren de datos Data 8 a Data 15.

CONCLUSIONES

En este proyecto se buscó la sencillez. Al buscar ésta se buscaron los elementos adecuados para realizar dicho objetivo. Esto llevó a una investigación más profunda de lo pensado originalmente. Esta investigación nos llevó a circuitos integrados más complejos y a una solución más adaptable a otras aplicaciones.

Por ejemplo, los convertidores y filtros digitales utilizados, nos llevaron a olvidar el uso de filtros analógicos, convirtiendo lo que al principio era un diseño extremadamente complejo de resistencias, capacitores y amplificadores operacionales al uso de dos chips que llevan a la tarjeta a una fidelidad alta. De otra manera hubiera necesitado desarrollar los cálculos para el filtro analógico, así como, de soportar las inexactitudes inherentes a los componentes de este tipo debido a las tolerancias de fabricación y el cambio de sus propiedades debido a las variaciones en temperatura.

Ya que las muestras debían de ser periódicas se utilizó un circuito parecido al FIFO(primero adentro, primero afuera). Este circuito se implementó con una memoria estática y un control de entrada y salida de datos, ya que los circuitos FIFO tienen un costo bastante elevado.

El uso de las compuertas lógicas programables hizo olvidar el uso de la familia de 7400`s, así como nos llevó a aprender el paquete FOUNDATION, a base de esquemáticos y su simulación para generar archivos JEDEC. Esta herramienta redujo el número de elementos en el diseño, así como ayudó a simular hasta cierto grado el circuito final.

Desgraciadamente el circuito no pudo ser implementado ya que resultó que se necesitaba un CPLD de más capacidad sólo disponible en montaje superficial. Para soldar dichos empaquetados se necesita una soldadora especial con boquillas disponibles para el tipo de empaquetado y pasta especial para soldar ya que la de uso común tiende a corroer los pines.

En este proyecto se vio que se necesita un desarrollo posterior para hacerlo más competitivo. La integración de efectos como el "flanger", "wah_wah", "delay", entre otros haría más competitivo al producto. Así también las modificaciones que fueran necesitando los diferentes usuarios de este producto ayudarían en gran medida a este propósito. También otro desarrollo posterior es del modificar el diseño de la tarjeta para incorporar FPGAs, en vez de CPLDs. Lo anterior no se llevó a cabo por no contar con un equipo para programar las memorias que necesitan este tipo de circuitos integrados. Esta modificación, junto con el desarrollo de un filtro digital con la herramienta DSP LOGIBLOX para el programa FOUNDATION reduciría el costo de esta tarjeta, y permitiría incorporar más convertidores A/D, con el consecuente aumento de canales.

BIBLIOGRAFÍA

Para la elaboración de este trabajo se utilizaron los siguientes textos como referencia:

[1] PROAKIS, John G.

Digital Signal Processing

Prentice-Hall, EUA, 1996, Tercera Edición.

[2] POHLMANN, Ken C.

Principles of Digital Audio

McGraw-Hill, EUA, 1992, Tercera Edición.

[3] GREENFIELD, Joseph D.

Practical Digital Design Using Ics

Prentice-Hall, EUA, 1994, Tercera Edición.

[4] Crystal World Tour, Application Seminar

Crystal Semiconductor Corporation, 1995

[5] EMBREE, Paul M.

C language algorithms for digital signal processing

Prentice-Hall, EUA, 1991

[6] STALLINGS, William

Computer Organization and Architecture, Design for Performance

Prentice-Hall, EUA, 1996, Cuarta Edición

[7] EGGEBRECHT, Lewis C.

Interfacing to the IBM Personal Computer

Howard W. Sams & Co, EUA, 1983

[8] McDOWELL, Philip

Choosing and Using 4 Bit Microcontrollers

Butterworth-Heinemann, EUA, 1994

[9] Xilinx AppLINX CD-ROM

Xilinx, Inc. 1998, Sexta Revisión

[10] GUREWICH, Ori

Teach Yourself Visual C++ 5 in 21 Days

Prentice-Hall, EUA, 1997, Cuarta Edición

[11] BARANS, Michael

Keyboard

Miller Freeman Publications, EUA, Octubre 1997, pp. 34-53.

[12] 1996 Microchip Technical Library

Microchip Technology Inc., 1996

[13] Configurable Logic Microcontroller Nonvolatile Memory Atmel Products

Atmel Corporation, 1997

[14] Burr-Brown 1997 CD-ROM Catalog software and database

CADindex, 1997

[15] ROSCH, Winn L.

The Winn Rosch Hardware Bible

Brady Publishing, EUA, 1989

```
/ CopyDlg.cpp : implementation file
/

include "stdafx.h"
include "Tesis.h"
include "CopyDlg.h"

//////////////////////////
/ MI CÓDIGO EMPIEZA AQUÍ
//////////////////////////

include <stdio.h>
define BUFFER 4056

//////////////////////////
/ MI CÓDIGO TERMINA AQUÍ
//////////////////////////

ifdef _DEBUG
define new DEBUG_NEW
undef THIS_FILE
tatic char THIS_FILE[] = __FILE__;
endif

////////////////////////////////////////////////////////////////////////////
/ CCopyDlg dialog


CopyDlg::CCopyDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CCopyDlg::IDD, pParent)

   //{{AFX_DATA_INIT(CCopyDlg)
   m_WavDestination = _T("");
   m_Dir = _T("");
   m_WavSource = _T("");
   //}}AFX_DATA_INIT


oid CCopyDlg::DoDataExchange(CDataExchange* pDX)

   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CCopyDlg)
   DDX_Control(pDX, IDC_DISK_LIST, m_DiskList);
   DDX_Text(pDX, IDC_DESTINATION_EDIT, m_WavDestination);
   DDX_Text(pDX, IDC_DIR_EDIT, m_Dir);
   DDX_Text(pDX, IDC_SOURCE_EDIT, m_WavSource);
   //}}AFX_DATA_MAP


GIN_MESSAGE_MAP(CCopyDlg, CDialog)
   //{{AFX_MSG_MAP(CCopyDlg)
   //}}AFX_MSG_MAP
D_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
 CCopyDlg message handlers

OL CCopyDlg::OnInitDialog()

   CDialog::OnInitDialog();

   // TODO: Add extra initialization here

   //////////////////////////
   // MI CÓDIGO EMPIEZA AQUÍ
   //////////////////////////

   UpdateData(FALSE);

   m_DiskList.AddString("A");
   m_DiskList.AddString("B");
```

```
    m_DiskList.AddString("C");
    m_DiskList.AddString("D");
    m_DiskList.AddString("E");

    m_DiskList.SelectString(0,"A");

    UpdateData(FALSE);

    ////////////////////////
    // MI CÓDIGO TERMINA AQUÍ
    ////////////////////////

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CCopyDlg::OnOK()
{
    // TODO: Add extra validation here
    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////
    // Declara los apuntadores para los archivos
    FILE *stream;
    FILE *newstream;

    int numclosed, result;
    unsigned int i, j;
    char info[BUFFER], dest[BUFFER];
    unsigned long dividend,quotient,files;

    // Actualiza los datos del cuadro de dialogo.
    UpdateData(TRUE);

    CString Selection;

    // Compone el nombre del archivo
    m_DiskList.GetText( m_DiskList.GetCurSel(),
                        Selection);
    m_WavDestination = Selection + ":\\" + m_Dir;
    UpdateData(FALSE);

    // Abre los archivos
    stream      = fopen(m_WavSource,"r+");
    newstream   = fopen(m_WavDestination,"w+");

    // Longitud del archivo
    result  = fseek(stream,0L,SEEK_END);
    files   = ftell(stream);
    result  = fseek(stream,0L,SEEK_SET);
    result  = fseek(newstream,0L,SEEK_SET);

    // Toma de operadores
    dividend=files/BUFFER;
    quotient=files-dividend*BUFFER;

    // Lee byte por byte de la fuente hasta
    // que llega al final del archivo.
    for(i=0;i<dividend;i++)
    {
        result = fread(info,sizeof(char),BUFFER,stream);
        for(j=0;j<BUFFER;j++)
        {
            dest[j]=info[j];
        }
        result = fwrite(dest,sizeof(char),BUFFER,newstream);
    }
    // Copia cociente
    result = fread(info,sizeof(char),quotient,stream);
    for(j=0;j<quotient;j++)
    {
        dest[j]=info[j];
    }
```

```
// DelayDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Tesis.h"
#include "DelayDlg.h"

//////////////////////////
// MI CÓDIGO EMPIEZA AQUÍ
//////////////////////////

#define BUFFER 4096
#define DELAY 1100

//////////////////////////
// MI CÓDIGO TERMINA AQUÍ
//////////////////////////

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDelayDlg dialog


CDelayDlg::CDelayDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CDelayDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CDelayDlg)
    m_Delay = 0;
    m_WavSource = _T("");
    m_WavDestination = _T("");
    m_Dir = _T("");
    //}}AFX_DATA_INIT
}


void CDelayDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDelayDlg)
    DDX_Control(pDX, IDC_DISK_LIST, m_DiskList);
    DDX_Text(pDX, IDC_DELAY_EDIT, m_Delay);
    DDV_MinMaxInt(pDX, m_Delay, 0, 200);
    DDX_Text(pDX, IDC_SOURCE_EDIT, m_WavSource);
    DDX_Text(pDX, IDC_DESTINATION_EDIT, m_WavDestination);
    DDX_Text(pDX, IDC_DIR_EDIT, m_Dir);
    //}}AFX_DATA_MAP



EGIN_MESSAGE_MAP(CDelayDlg, CDialog)
    //{{AFX_MSG_MAP(CDelayDlg)
    //}}AFX_MSG_MAP
ND_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
/ CDelayDlg message handlers

oid CDelayDlg::OnOK()

    // TODO: Add extra validation here
    //////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    //////////////////////////
    // Declara los apuntadores para los archivos
    FILE *stream;
    FILE *newstream;

    int numclosed, result;
```

```
// Auxiliares para las rutinas
unsigned int i, j, k;

// Arreglos de datos
unsigned char info1[DELAY], info2[DELAY], dest[DELAY];
unsigned long nAvgBytesPerSec;
unsigned long dividend,quotient,files;
unsigned position1, position2, task1, task2;

char check[20];
char header[50];

// Actualiza los datos del cuadro de dialogo.
UpdateData(TRUE);

CString Selection;

// Compone el nombre del archivo
m_DiskList.GetText( m_DiskList.GetCurSel(),
                    Selection);
m_WavDestination = Selection + ":\\" + m_Dir;
UpdateData(FALSE);

// Abre los archivos
stream      = fopen(m_WavSource,"r+");
newstream   = fopen(m_WavDestination,"w+");

// Longitud del archivo
result  = fseek(stream,0L,SEEK_END);
files   = ftell(stream);
result  = fseek(stream,28L,SEEK_SET);
result  = fseek(newstream,0L,SEEK_SET);

// Promedio de bytes por segundo
result = fread(&nAvgBytesPerSec,sizeof(long),1,stream);
result = fseek(stream,0L,SEEK_SET);
_ltoa(nAvgBytesPerSec,check,10);
MessageBox(check);

// Toma de operadores
dividend=files/DELAY;
quotient=files-dividend*DELAY;
task1 = m_Delay*nAvgBytesPerSec/11025;
task2 = dividend-task1;

// Preparación
position1 = 0;
position2 = 0;
for(k=0;k<task1;k++)
{
    result = fseek(stream,position1,SEEK_SET);
    result = fread(info1,sizeof(char),DELAY,stream);
    result = fwrite(info1,sizeof(char),DELAY,newstream);
    position1 = position1 + DELAY;
}

// Lee byte por byte de la fuente hasta
// que llega al final del archivo.
for(i=0;i<task2;i++)
{
    // Lee información original
    result = fseek(stream, position1,SEEK_SET);
    result = fread(info1,sizeof(char),DELAY,stream);
    position1 = position1 + DELAY;

    // Lee el retraso
    result = fseek(stream, position2,SEEK_SET);
    result = fread(info2,sizeof(char),DELAY,stream);
    position2 = position2 + DELAY;

    // Se suma la información
    for(j=0;j<DELAY;j++)
```

```
        {
            dest[j]=info1[j]+info2[j]-127;
            if (dest[j]>255)
                dest[j]=250;
            if (dest[j]<0)
                dest[j]=5;
        }

        // Escribe la información en el archivo de destino
        result = fwrite(dest,sizeof(char),DELAY,newstream);
    }
    // Copia cociente
    result = fread(info1,sizeof(char),quotient,stream);
    for(j=0;j<quotient;j++)
    {
        dest[j]=info1[j];
    }
    result = fwrite(dest,sizeof(char),quotient,newstream);

    // escribe header
    result = fseek(stream,0L,SEEK_SET);
    result = fread(header,sizeof(char),50,stream);
    result = fseek(newstream,0L,SEEK_SET);
    result = fwrite(header,sizeof(char),50,newstream);

    // Cierra los archivos
    numclosed = fclose(stream);
    numclosed = fclose(newstream);

    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////
    // CDialog::OnOK();
}

BOOL CDelayDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here
    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////

    UpdateData(FALSE);

    m_DiskList.AddString("A");
    m_DiskList.AddString("B");
    m_DiskList.AddString("C");
    m_DiskList.AddString("D");
    m_DiskList.AddString("E");

    m_DiskList.SelectString(0,"A");

    UpdateData(FALSE);

    ////////////////////////
    // MI CÓDIGO TERMINA AQUÍ
    ////////////////////////

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Tesis.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const

    CFrameWnd::AssertValid();


void CMainFrame::Dump(CDumpContext& dc) const

    CFrameWnd::Dump(dc);


#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

```
// SourceDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Tesis.h"
#include "SourceDlg.h"

//////////////////////////
// MI CÓDIGO EMPIEZA AQUÍ
//////////////////////////

#include <stdio.h>

//////////////////////////
// MI CÓDIGO TERMINA AQUÍ
//////////////////////////

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSourceDlg dialog


CSourceDlg::CSourceDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSourceDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSourceDlg)
    m_WavSource = _T("");
    m_Dir = _T("");
    //}}AFX_DATA_INIT
}


void CSourceDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSourceDlg)
    DDX_Control(pDX, IDC_DISK_LIST, m_DiskList);
    DDX_Text(pDX, IDC_WAVSOURCE_EDIT, m_WavSource);
    DDX_Text(pDX, IDC_DIR_EDIT, m_Dir);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSourceDlg, CDialog)
    //{{AFX_MSG_MAP(CSourceDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSourceDlg message handlers

BOOL CSourceDlg::OnInitDialog()

    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    //////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    //////////////////////////

    m_DiskList.AddString("A");
    m_DiskList.AddString("B");
    m_DiskList.AddString("C");
    m_DiskList.AddString("D");
    m_DiskList.AddString("E");

    m_DiskList.SelectString(0,"A");
```

```
    UpdateData(FALSE);

    ////////////////////////
    // MI CÓDIGO TERMINA AQUÍ
    ////////////////////////

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CSourceDlg::OnOK()
{
    // TODO: Add extra validation here
    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////
    FILE *stream;
    int iResults;
    int numclosed;

    UpdateData(TRUE);
    CString Selection;
    m_DiskList.GetText( m_DiskList.GetCurSel(),
                        Selection);

    m_WavSource = Selection + ":\\" + m_Dir;
    UpdateData(FALSE);

    if ((stream = fopen(m_WavSource,"r+"))==NULL)
    {
    iResults=
        MessageBox( "¿ Desea crearlo ?",
                    "El archivo no existe",
                    MB_YESNO + MB_ICONEXCLAMATION);

        if (iResults==IDYES)
        {
            stream= fopen(m_WavSource,"w+");
            numclosed =fclose(stream);
        }
    }


    ////////////////////////
    // MI CÓDIGO TERMINA AQUÍ
    ////////////////////////
    // CDialog::OnOK();
}
```

```
// Tesis.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Tesis.h"

#include "MainFrm.h"
#include "TesisDoc.h"
#include "TesisView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////
// MI CÓDIGO EMPIEZA AQUÍ
//////////////////////////

#include <conio.h>

#define PORT_A 0x300
#define PORT_B 0x301
#define PORT_C 0x302
#define PORT_D 0x303
#define BUFFER 4096

FILE *stream;
int i, num;
unsigned short  aux[BUFFER];
short int       info[BUFFER];
unsigned long   position;

//////////////////////////
// MI CÓDIGO TERMINA AQUÍ
//////////////////////////

/////////////////////////////////////////////////////////////////////////////
// CTesisApp

BEGIN_MESSAGE_MAP(CTesisApp, CWinApp)
    //{{AFX_MSG_MAP(CTesisApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTesisApp construction

CTesisApp::CTesisApp()

    // TODO: add construction code here,
    // Place all significant initialization in InitInstance


/////////////////////////////////////////////////////////////////////////////
// The one and only CTesisApp object

CTesisApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CTesisApp initialization

BOOL CTesisApp::InitInstance()

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
```

```
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();          // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();    // Call this when linking to MFC statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings();  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CTesisDoc),
        RUNTIME_CLASS(CMainFrame),       // main SDI frame window
        RUNTIME_CLASS(CTesisView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;


////////////////////////////////////////////////////////////////////////////
/ CAboutDlg dialog used for App About

lass CAboutDlg : public CDialog

ublic:
    CAboutDlg();

' Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

  Implementation
 rotected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()


 outDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)

    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
```

```
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CTesisApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CTesisApp commands

BOOL CTesisApp::OnIdle(LONG lCount)
{
    // TODO: Add your specialized code here and/or call the base class
    ////////////////////////////
    // MI CÓDIGO COMIENZA AQUÍ
    ////////////////////////////

    // define variables
    unsigned short WREG, RREG;

    // lee entrada de puertos
    /*WREG = _inpw(PORT_C);
    RREG = _inpw(PORT_D);*/
    WREG = 10000;
    RREG = 1000;

    // Llama a la función CWinApp::OnIdle() de la clase base
    // para terminar sus tareas administrativas.
    CWinApp::OnIdle(lCount);

    // Obtiene un apuntador hacia la plantilla de documento.
    POSITION pos = GetFirstDocTemplatePosition();
    CDocTemplate* pDocTemplate =GetNextDocTemplate(pos);

    // Obtiene un apuntador hacia el documento.
    pos = pDocTemplate->GetFirstDocPosition();
    CDocument* pDoc = pDocTemplate->GetNextDoc(pos);

    // Obtiene un apuntador hacia la vista.
    pos =pDoc->GetFirstViewPosition();
    CTesisView* pView =(CTesisView*) pDoc->GetNextView(pos);

    // Declara e inicializa dos variables estáticas:
    //PrevTimeTask1 y PrevTimeTask2.
    static DWORD PrevTimeTask1 = 0;
    static DWORD PrevTimeTask2 = 0;

    // Obtiene el tiempo actual.
    DWORD CurrentTime = GetTickCount();

    // Actualiza las variables de os controles.
    pView->UpdateData(TRUE);

    // Si han transcurrido más de 50 milisegundos desde que
    // se ejecutó la tarea 1 y está activada la casilla de verificación.
    // Enable Task1, realiza la tarea 1.
    if (CurrentTime > PrevTimeTask1+50 &&
        pView->m_EnableRecordCheck )
```

```
    {

        pView->m_Task1Edit = pView->m_Task1Edit+1;
        pView->UpdateData(FALSE);
        PrevTimeTask1 = CurrentTime;

        while (((WREG>RREG)&&(WREG-RREG>4096))
            || ((RREG>WREG)&&(RREG-WREG<61440)))
        {
            position = (pView->m_Task1Edit)*BUFFER*2;
            for (i=0;i<4096;i++)
            {
                aux[i] = _inpw (PORT_A);
                info[i] = aux[i]-32768;
            }
            num = fseek(stream, position, SEEK_SET);
            num = fwrite(stream,sizeof(short int),BUFFER,stream);

            WREG=RREG;
        }
    }

    // Si han transcurrido más de 50 milisegundos desde que
    // se ejecutó la tarea 2 y está activada la casilla de verificación.
    // Enable Task2, realiza la tarea 2.
    if (CurrentTime > PrevTimeTask2+50 &&
        pView->m_EnablePlayCheck )
    {
        if ((pView->m_Task2Edit)<(pView->m_Task1Edit))
            pView->m_Task2Edit = pView->m_Task2Edit+1;
        pView->UpdateData(FALSE);
        PrevTimeTask2 = CurrentTime;
    }

    // Regresa TRUE para que OnIdle() vuelva a ser llamada.
    return TRUE;

    ////////////////////////
    // MI CÓDIGO TERMINA AQUÍ
    ////////////////////////
    //return CWinApp::OnIdle(lCount);
}

BOOL CTesisApp::InitApplication()
{
    // TODO: Add your specialized code here and/or call the base class

    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////

    stream = fopen("C:\\tesis.aux","w+");

    ////////////////////////
    // MI CÓDIGO TERMINA AQUÍ
    ////////////////////////

    return CWinApp::InitApplication();
}
```

```
// TesisView.cpp : implementation of the CTesisView class
//

#include "stdafx.h"
#include "Tesis.h"

#include "TesisDoc.h"
#include "TesisView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////
// MI CÓDIGO EMPIEZA AQUÍ
//////////////////////////

#define BUFFER 4096

//////////////////////////
// MI CÓDIGO TERMINA AQUÍ
//////////////////////////

/////////////////////////////////////////////////////////////////////////////
// CTesisView

IMPLEMENT_DYNCREATE(CTesisView, CFormView)

BEGIN_MESSAGE_MAP(CTesisView, CFormView)
    //{{AFX_MSG_MAP(CTesisView)
    ON_COMMAND(ID_FILE_SOURCE, OnFileSource)
    ON_COMMAND(ID_FUNCTION_WCOPY, OnFunctionWcopy)
    ON_COMMAND(ID_FUNCTION_GRAPH, OnFunctionGraph)
    ON_COMMAND(ID_FUNCTION_DELAY, OnFunctionDelay)
    ON_BN_CLICKED(IDC_STOP_BUTTON, OnStopButton)
    ON_BN_CLICKED(IDC_ENABLE_RECORD_CHECK, OnEnableRecordCheck)
    ON_BN_CLICKED(IDC_ENABLE_PLAY_CHECK, OnEnablePlayCheck)
    ON_BN_CLICKED(IDC_CANCEL_BUTTON, OnCancelButton)
    ON_BN_CLICKED(IDC_SAVE_BUTTON, OnSaveButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTesisView construction/destruction

CTesisView::CTesisView()
    : CFormView(CTesisView::IDD)
{
    //{{AFX_DATA_INIT(CTesisView)
    m_SampleRate = -1;
    m_Bits = -1;
    m_EnablePlayCheck = FALSE;
    m_EnableRecordCheck = FALSE;
    m_Channels = -1;
    m_Task1Edit = 0;
    m_Task2Edit = 0;
    m_WavSource = _T("");
    //}}AFX_DATA_INIT
    // TODO: add construction code here


esisView::~CTesisView()


id CTesisView::DoDataExchange(CDataExchange* pDX)

    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTesisView)
    DDX_Radio(pDX, IDC_11KHZ_RADIO, m_SampleRate);
```

```
    DDX_Radio(pDX, IDC_8BITS_RADIO, m_Bits);
    DDX_Check(pDX, IDC_ENABLE_PLAY_CHECK, m_EnablePlayCheck);
    DDX_Check(pDX, IDC_ENABLE_RECORD_CHECK, m_EnableRecordCheck);
    DDX_Radio(pDX, IDC_MONO_RADIO, m_Channels);
    DDX_Text(pDX, IDC_EDIT1, m_Task1Edit);
    DDX_Text(pDX, IDC_EDIT2, m_Task2Edit);
    DDX_Text(pDX, IDC_SOURCE_EDIT, m_WavSource);
    //}}AFX_DATA_MAP
}

BOOL CTesisView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFormView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CTesisView diagnostics

#ifdef _DEBUG
void CTesisView::AssertValid() const
{
    CFormView::AssertValid();
}

void CTesisView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CTesisDoc* CTesisView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTesisDoc)));
    return (CTesisDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTesisView message handlers

void CTesisView::OnFileSource()
{
    // TODO: Add your command handler code here
    ////////////////////////////
    // MI CÓDICO EMPIEZA AQUÍ
    ////////////////////////////

    m_Sourcedlg.DoModal();
    m_WavSource = m_Sourcedlg.m_WavSource;
    UpdateData(FALSE);

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////


void CTesisView::OnFunctionWcopy()

    // TODO: Add your command handler code here
    ////////////////////////////
    // MI CÓDICO EMPIEZA AQUÍ
    ////////////////////////////

    m_Copydlg.m_WavSource = m_Sourcedlg.m_WavSource;
    m_Copydlg.DoModal();

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////
```

```
void CTesisView::OnFunctionGraph()
{
    // TODO: Add your command handler code here
    ////////////////////////////
    // MI CÓDICO EMPIEZA AQUÍ
    ////////////////////////////

    m_Graphdlg.m_WavSource = m_Sourcedlg.m_WavSource;
    m_Graphdlg.DoModal();

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////
}

void CTesisView::OnFunctionDelay()
{
    // TODO: Add your command handler code here
    ////////////////////////////
    // MI CÓDICO EMPIEZA AQUÍ
    ////////////////////////////

    m_Delaydlg.m_WavSource = m_Sourcedlg.m_WavSource;
    m_Delaydlg.DoModal();

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////
}

void CTesisView::OnStopButton()
{
    // TODO: Add your control notification handler code here
    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////
    int num;

    m_EnablePlayCheck = FALSE;
    m_EnableRecordCheck = FALSE;
    m_Task2Edit=0;
    UpdateData(FALSE);

    // Habilita los botones de selección

    GetDlgItem(IDC_11KHZ_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_22KHZ_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_44KHZ_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_MONO_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_STEREO_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_8BITS_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_16BITS_RADIO)->EnableWindow(TRUE);

    // Habilita los botones grabar y reproducir

    GetDlgItem(IDC_ENABLE_RECORD_CHECK)->EnableWindow(TRUE);
    GetDlgItem(IDC_ENABLE_PLAY_CHECK)->EnableWindow(TRUE);
    GetDlgItem(IDC_SAVE_BUTTON)->EnableWindow(TRUE);

    num = fcloseall();
    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////




id CTesisView::OnInitialUpdate()

    CFormView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class
```

```
    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////

    m_EnablePlayCheck = FALSE;
    m_EnableRecordCheck = FALSE;
    m_SampleRate = 0;
    m_Channels = 0;
    m_Bits = 0;
    UpdateData(FALSE);


    GetDlgItem(IDC_ENABLE_RECORD_CHECK)->EnableWindow(TRUE);
    GetDlgItem(IDC_ENABLE_PLAY_CHECK)->EnableWindow(FALSE);
    GetDlgItem(IDC_STOP_BUTTON)->EnableWindow(FALSE);
    GetDlgItem(IDC_SAVE_BUTTON)->EnableWindow(FALSE);

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////

}

void CTesisView::OnEnableRecordCheck()
{
    // TODO: Add your control notification handler code here
    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////

    // Inhabilita los botones de selección

    GetDlgItem(IDC_11KHZ_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_22KHZ_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_44KHZ_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_MONO_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_STEREO_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_8BITS_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_16BITS_RADIO)->EnableWindow(FALSE);

    // Inhabilita el botón de reproducir

    GetDlgItem(IDC_ENABLE_PLAY_CHECK)->EnableWindow(FALSE);

    // Habilita el botón parar
    GetDlgItem(IDC_STOP_BUTTON)->EnableWindow(TRUE);

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////
}

void CTesisView::OnEnablePlayCheck()

    // TODO: Add your control notification handler code here

    ////////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////////

    // Inhabilita los botones de selección

    GetDlgItem(IDC_11KHZ_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_22KHZ_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_44KHZ_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_MONO_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_STEREO_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_8BITS_RADIO)->EnableWindow(FALSE);
    GetDlgItem(IDC_16BITS_RADIO)->EnableWindow(FALSE);

    // Inhabilita el botón de reproducir
```

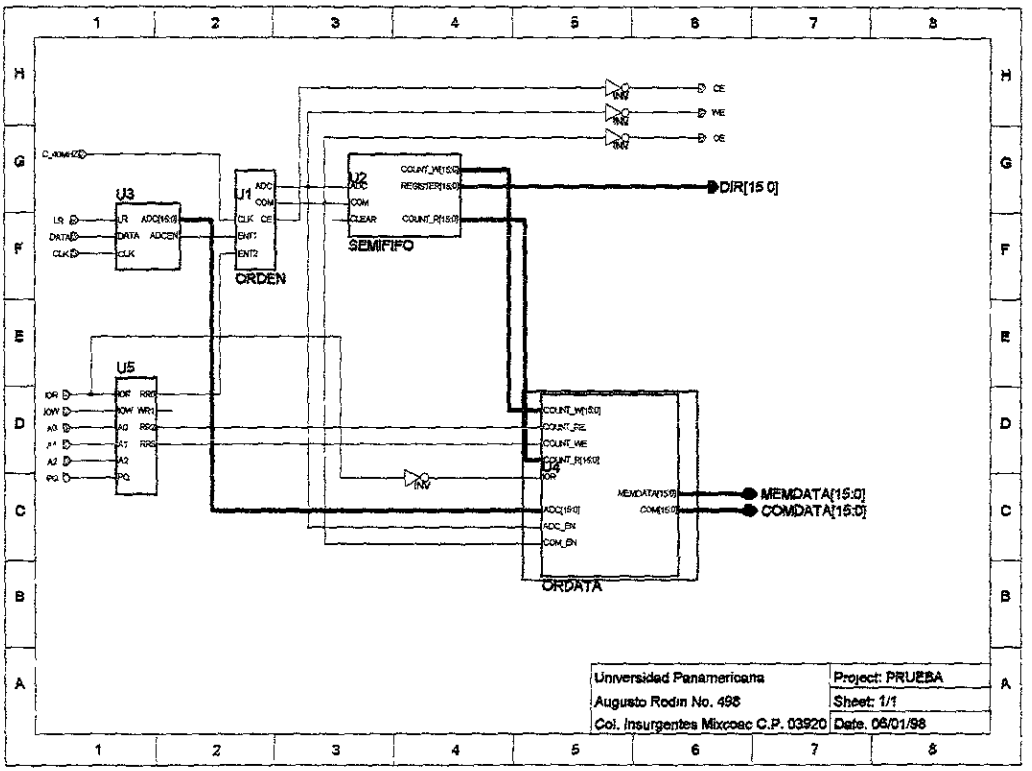
```
    GetDlgItem(IDC_ENABLE_RECORD_CHECK)->EnableWindow(FALSE);

    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////
}

void CTesisView::OnCancelButton()
{
    // TODO: Add your control notification handler code here

    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////

    m_EnablePlayCheck = FALSE;
    m_EnableRecordCheck = FALSE;
    m_Task1Edit=0;
    m_Task2Edit=0;
    UpdateData(FALSE);

    // Habilita los botones de selección

    GetDlgItem(IDC_11KHZ_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_22KHZ_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_44KHZ_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_MONO_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_STEREO_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_8BITS_RADIO)->EnableWindow(TRUE);
    GetDlgItem(IDC_16BITS_RADIO)->EnableWindow(TRUE);

    // Habilita los botones grabar y reproducir

    GetDlgItem(IDC_ENABLE_RECORD_CHECK)->EnableWindow(TRUE);
    GetDlgItem(IDC_ENABLE_PLAY_CHECK)->EnableWindow(FALSE);
    GetDlgItem(IDC_SAVE_BUTTON)->EnableWindow(FALSE);

    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////
}

void CTesisView::OnSaveButton()
{
    // TODO: Add your control notification handler code here
    ////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    ////////////////////////

    // declaración de variables
    FILE *stream;
    FILE *newstream;

    int i;
    unsigned long position,filelong, dividend, j;
    char info[BUFFER];

    // Variables del encabezado
    unsigned short  Channels = 2;
    unsigned int    SamplesPerSec = 44100;
    unsigned int    AvgBytesPerSec = 176400;
    unsigned short  BlockAlign = 4;
    unsigned short  Bits = 16;
    unsigned int    lenght1;
    unsigned int    lenght2;

    // Constantes del encabezado
    const char head1[4] = {82,73,70,70};
    const char head2[14]= {87,65,86,69,102,109,116,
                           32,16,0,0,0,1,0};
    const char head3[4] = {100,97,116,97};

    m_EnablePlayCheck = FALSE;
```

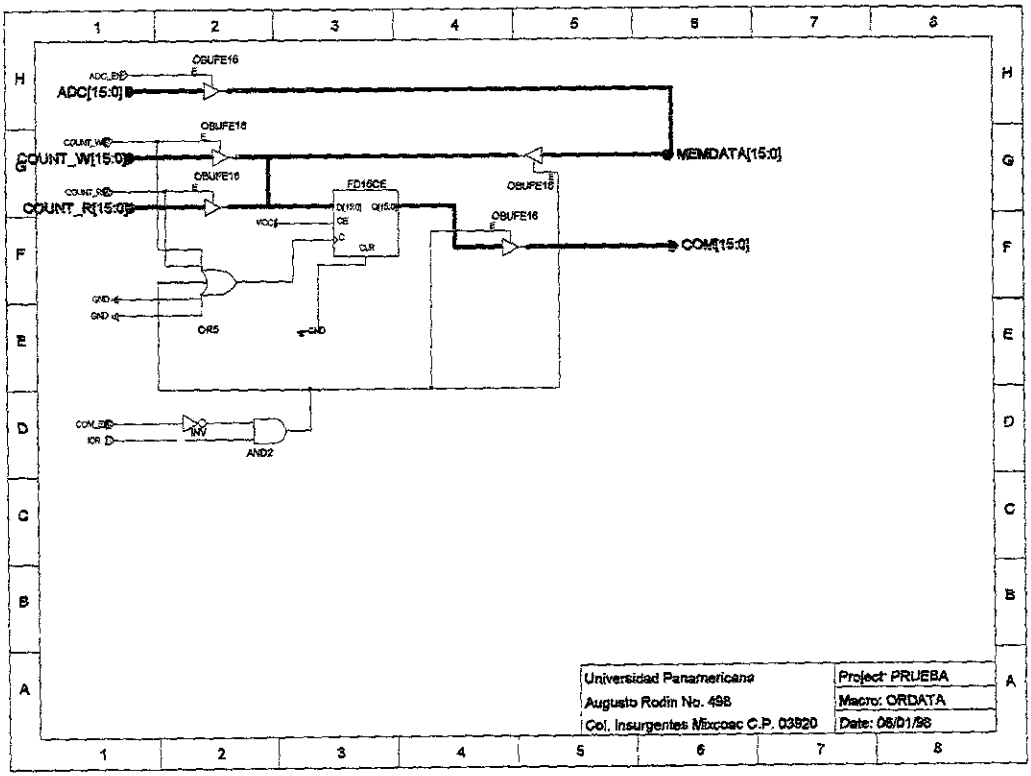
```
m_EnableRecordCheck = FALSE;
UpdateData(FALSE);

// Habilita los botones de selección

GetDlgItem(IDC_11KHZ_RADIO)->EnableWindow(TRUE);
GetDlgItem(IDC_22KHZ_RADIO)->EnableWindow(TRUE);
GetDlgItem(IDC_44KHZ_RADIO)->EnableWindow(TRUE);
GetDlgItem(IDC_MONO_RADIO)->EnableWindow(TRUE);
GetDlgItem(IDC_STEREO_RADIO)->EnableWindow(TRUE);
GetDlgItem(IDC_8BITS_RADIO)->EnableWindow(TRUE);
GetDlgItem(IDC_16BITS_RADIO)->EnableWindow(TRUE);

// Habilita los botones grabar y reproducir

GetDlgItem(IDC_ENABLE_RECORD_CHECK)->EnableWindow(TRUE);
GetDlgItem(IDC_ENABLE_PLAY_CHECK)->EnableWindow(TRUE);
GetDlgItem(IDC_SAVE_BUTTON)->EnableWindow(TRUE);

// Apertura de los archivos

stream = fopen( "c:\\tesis.aux","r+");
newstream = fopen( m_Sourcedlg.m_WavSource ,"w+");

// Longitud del archivo fuente
i = fseek(stream, 0L, SEEK_END);
filelong = ftell(stream);
dividend = filelong/BUFFER;

// Inicialización
i = fseek(stream, 0L, SEEK_SET);
position = 0;

// Cálculo de las variables del encabezado
if (m_SampleRate == 0)
    SamplesPerSec = 11025;
if (m_SampleRate == 1)
    SamplesPerSec = 22050;
if (m_SampleRate == 2)
    SamplesPerSec = 44100;
if (m_Channels == 0)
    Channels = 1;
if (m_Channels == 1)
    Channels = 2;
if (m_Bits == 0)
    Bits = 8;
if (m_Bits == 1)
    Bits = 16;
BlockAlign = Bits*Channels/8;
AvgBytesPerSec = SamplesPerSec*BlockAlign;


// Copia del archivo
for(j=0;j<dividend;j++)
{
    i = fseek (stream,position,SEEK_SET);
    i = fseek (newstream,position,SEEK_SET);
    i = fread (info,sizeof(char),1,stream);
    i = fwrite (info,sizeof(char),1,newstream);
    position = position + BUFFER;
}

// Cálculo de las longitudes
lenght1 = filelong -8;
lenght2 = filelong -44;

// Escritura del encabezado
i = fseek(newstream,0L,SEEK_SET);
i = fwrite(head1,sizeof(char),4,newstream);
i = fwrite(&lenght1,sizeof(int),1,newstream);
i = fwrite(head2,sizeof(char),14,newstream);
i = fwrite(&Channels,sizeof(short),1,newstream);
i = fwrite(&SamplesPerSec,sizeof(int),1,newstream);
```

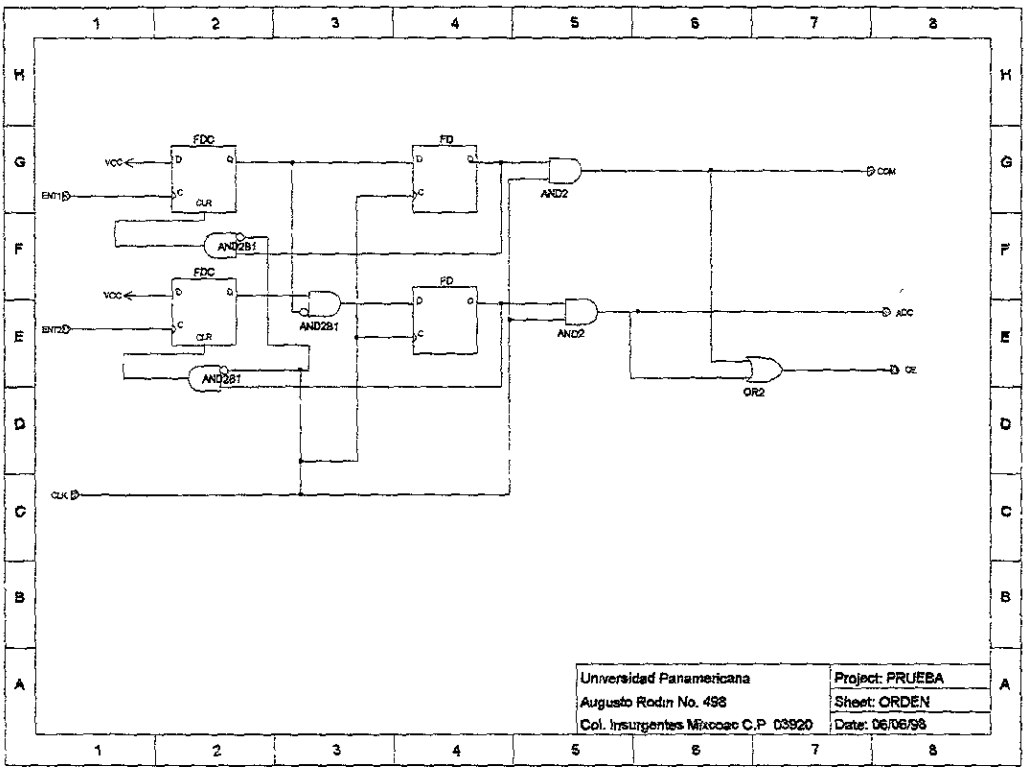
```
i = fwrite(&AvgBytesPerSec,sizeof(int),1,newstream);
i = fwrite(&BlockAlign,sizeof(short),1,newstream);
i = fwrite(&Bits,sizeof(short),1,newstream);
i = fwrite(head3,sizeof(char),4,newstream);
i = fwrite(&lenght2,sizeof(int),1,newstream);

// Cierra los archivos
dividend = fcloseall();

//////////////////////////
// MI CÓDIGO EMPIEZA AQUÍ
//////////////////////////
}
```

```
// Tesis.h : main header file for the TESIS application
//

#if !defined(AFX_TESIS_H__1A44F965_0D48_11D2_91D0_EFA7B18B6843__INCLUDED_)
#define AFX_TESIS_H__1A44F965_0D48_11D2_91D0_EFA7B18B6843__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols
////////////////////////
// MI CÓDICO EMPIEZA AQUÍ
////////////////////////

#include "SourceDlg.h"
#include "CopyDlg.h"
#include "GraphDlg.h"
#include "DelayDlg.h"

////////////////////////
// MI CÓDIGO EMPIEZA AQUÍ
////////////////////////

/////////////////////////////////////////////////////////////////////////////
// CTesisApp:
// See Tesis.cpp for the implementation of this class
//

class CTesisApp : public CWinApp
{
public:
    CTesisApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTesisApp)
    public:
    virtual BOOL InitInstance();
    virtual BOOL OnIdle(LONG lCount);
    virtual BOOL InitApplication();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CTesisApp)
    afx_msg void OnAppAbout();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


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

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_TESIS_H__1A44F965_0D48_11D2_91D0_EFA7B18B6843__INCLUDED_)
```

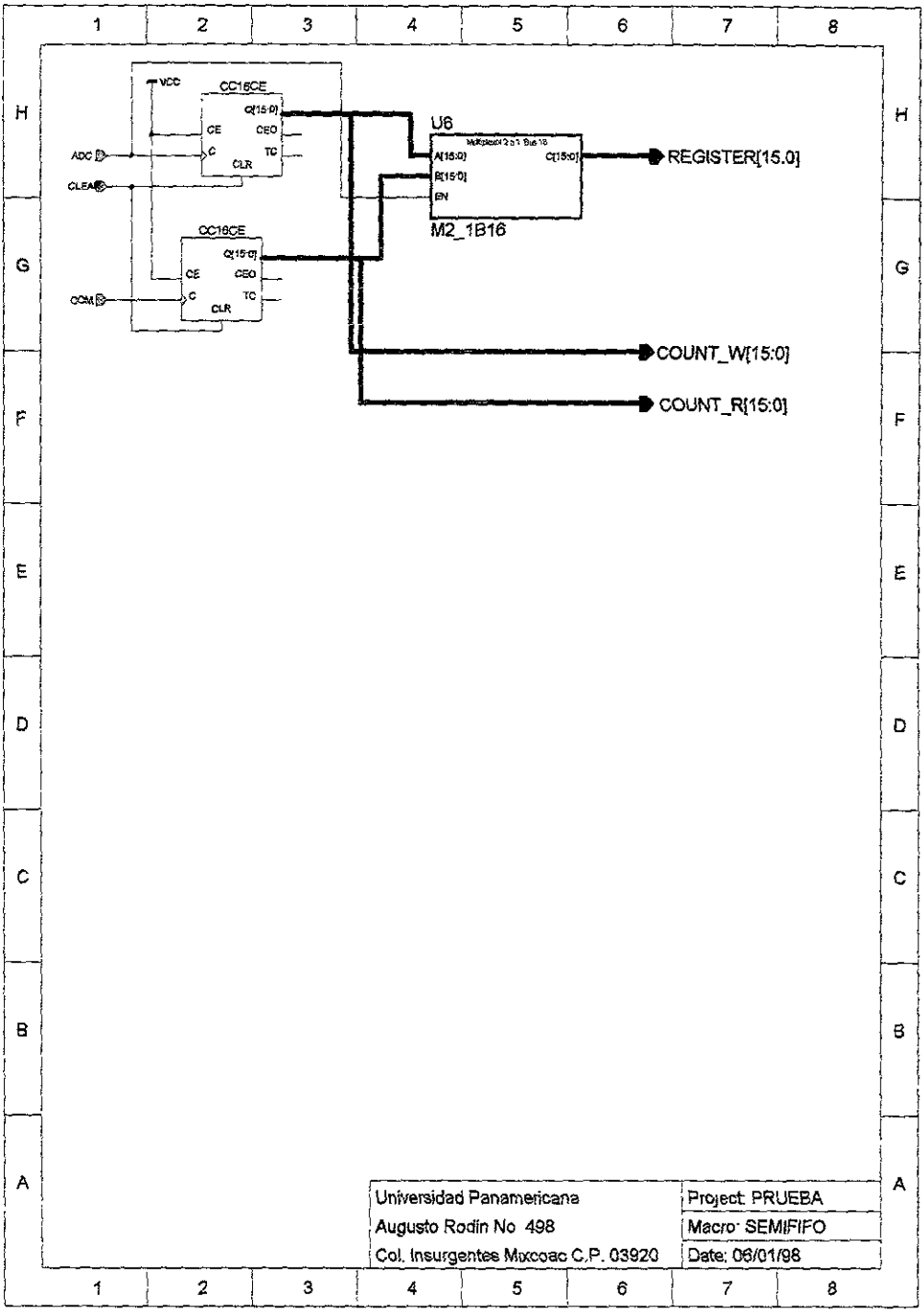
```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__1A44F969_0D48_11D2_91D0_EFA7B18B6843__INCLUDED_)
#define AFX_MAINFRM_H__1A44F969_0D48_11D2_91D0_EFA7B18B6843__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
    //////////////////////////
    // MI CÓDICO EMPIEZA AQUÍ
    //////////////////////////

    CSourceDlg  m_Sourcedlg;
    CCopyDlg    m_Copydlg;
    CGraphDlg   m_Graphdlg;
    CDelayDlg   m_Delaydlg;

    //////////////////////////
    // MI CÓDIGO EMPIEZA AQUÍ
    //////////////////////////

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

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

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_MAINFRM_H__1A44F969_0D48_11D2_91D0_EFA7B18B6843__INCLUDED_)
```

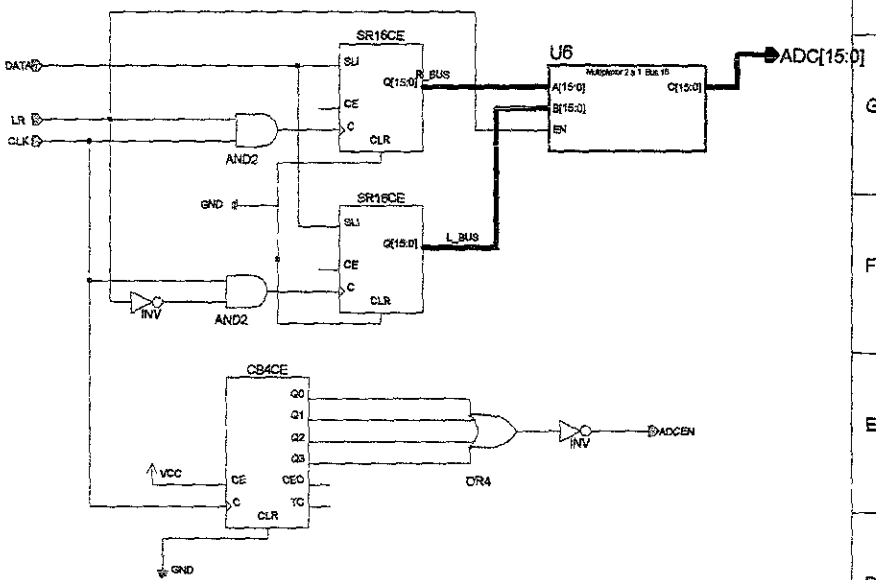

INV
CE
WE
OE
U3
LR
DATA
CLK
ADC[15:0]
ADCEN
U1
ADC
COM
CLK
CE
ENT1
ENT2
ORDEN
U2
ADC
COM
CLEAR
COUNT_W[15:0]
REGISTER[15:0]
COUNT_R[15:0]
SEMIFIFO
DIR[15:0]
U5
IOR
IOW
A0
A1
A2
RR
WR1
RS2
RS1
RS0
U4
COUNT_W[15:0]
COUNT_RE
COUNT_WE
COUNT_R[15:0]
IOR
MEMDATA[15:0]
COM[15:0]
ADC[15:0]
ADC_EN
COM_EN
ORDATA
MEMDATA[15:0]
COMDATA[15:0]
Universidad Panamericana
Augusto Rodin No. 498
Col. Insurgentes Mixcoac C.P. 03920
Project: PRUEBA
Sheet: 1/1
Date: 06/01/98


OBUFE16
ADC_EN
ADC[15:0]
COUNT_W
COUNT_W[15:0]
COUNT_R
COUNT_R[15:0]
FD16CE
D[15:0]
Q[15:0]
CE
C
CLR
VCC
MEMDATA[15:0]
COM[15:0]
GND
OR5
INV
IOR
AND2
Universidad Panamericana
Augusto Rodin No. 498
Col. Insurgentes Mixcoac C.P. 03920
Project: PRUEBA
Macro: ORDATA
Date: 06/01/98



Universidad Panamericana	Project: PRUEBA
Augusto Rodin No. 498	Sheet: ORDEN
Col. Insurgentes Mixcoac C.P. 03920	Date: 06/06/98


1
2
3
4
5
6
7
8
H
G
F
E
D
C
B
A
VCC
CC16CE
Q[15:0]
CE
CEO
ADC
C
TC
CLR
CLEAR
CC16CE
Q[15:0]
CE
CEO
CCM
C
TC
CLR
U6
A[15:0]
C[15:0]
B[15:0]
SEL
M2_1B16
REGISTER[15:0]
COUNT_W[15:0]
COUNT_R[15:0]
Universidad Panamericana
Augusto Rodin No 498
Col. Insurgentes Mixcoac C.P. 03920
Project: PRUEBA
Macro: SEMIFIFO
Date: 06/01/98



Universidad Panamericana	Project: PRUEBA
Augusto Rodin No. 498	Macro: ADCSERIN
Col. Insurgentes Mixcoac C.P. 03920	Date: 06/01/98

XC9500 In-System Programmable (ISP) CPLD-Family

The XC9500™ In-System Programmable (ISP) CPLD family takes complex programmable logic devices to new heights of high-performance, feature-richness, and flexibility. This family delivers industry-leading speeds, while giving you the flexibility of an enhanced customer proven pin-locking architecture along with extensive IEEE Std. 1149.1 JTAG boundary scan support. It features six devices ranging from 36 to 288 macrocells with a wide variety of package combinations that both minimize board space and maintain package footprints as designs grow or shrink. All I/O pins allow direct interfacing to both 3 and 5 volt systems, while the latest in compact, easy-to-use BGA packaging gives you access to as many as 192 signals.

Family Highlights

Highest density

- 5 ns pin-to-pin logic delays on all pins
- f_{CNT} to 125 MHz

Competitively Priced

- Wide range of package styles

Wide Density Range

- 36 to 288 macrocells (800 to 6,400 gates)
- 34 to 192 I/Os

Customer Proven Pin-Locking Architecture

- Flexible 36V18 function block
  - 90 product terms drive any or all of 18 macrocells
  - Global and product term clocks, output enables, set and reset
  - Power selection for each macrocell

Designed for True 5 Volt In-System Programmability

- Industry standard JTAG interface
- Endurance of 10,000 program/erase cycles
- Device programming over the full voltage and temperature range
- 20 years data retention

Programmable Outputs

- 24 mA drive on all pins
- 3.3 V or 5 volt I/O capability
- Slew rate control on each output

Industry-Leading JTAG Boundary Scan Support

- Extended boundary scan functions
  - On-chip logic testing
  - High impedance I/O isolation
  - ID and USER Codes
- Design security
  - Write Only (no readback) function

Seamless Integration of ATE and Third-Party JTAG Programming and Test Tools

- Automatic test system programming support
  - HP3070
  - Genrad
  - Teradyne
- Extensive third-party tool support
  - JTAG Technologies
  - Corelis Incorporated
  - ASSET Intertech

Easy-to-Use Design Tools: Alliance and Foundation Series Software Support

- Windows NT, Windows 95, SunOS, Solaris, HPUX
- Standards-based design flows
- Open third-party EDA systems integration with the Alliance Series Software
- Complete, stand-alone solutions with the Foundation Series Software]

Flexible Pin-Locking Architecture

The XC9500 devices, in conjunction with our fitter software, gives you the maximum in routability and flexibility while maintaining high performance. The architecture is feature

rich, including individual p-term output enables, three global clocks, and more p-terms per output than any other CPLD. The proven ability of the architecture to adapt to design changes while maintaining pin assignments (pin-locking) has been demonstrated in countless real-world customer designs since the introduction of the XC9500 family. This assured pin-locking means you can take full advantage of in-system-programmability and you can easily change at any time, even in the field.

Full IEEE 1149.1 JTAG Development and Debugging Support

The JTAG capability of the XC9500 family is the most comprehensive of any CPLD on the market. It features the standard support including BYPASS, SAMPLE/PRELOAD, and EXTEST. Additional boundary scan instructions, not found in any other CPLD, such as INTEST (for device functional test), HIGHZ (for bypass), and USERCODE (for program tracking), allow you the maximum debugging capability.

The XC9500 family is supported by a wide variety of industry standard third-party development and debugging tools including Corellis, JTAG Technologies, and Asset Intertech. These tools allow you to develop boundary scan test vectors to interactively analyze, test, and debug system failures. The family is also supported on all major ATE platforms including Teradyne, Hewlett Packard, and Genrad.

Leadership FastFLASH™ Technology

Xilinx is the innovator in Flash technology designed specifically for high-performance CPLDs. The superior pin-locking capability of the XC9500 family is made possible by a dense internal switch matrix with over three times the routing switches of any other ISP CPLD. FastFLASH cells that are 1/3 the size of EEPROM cells bring you the benefits of high-density and high-performance along with the added benefit of 10,000 program/erase cycles to assure complete product life cycle support. The rapid growth and rapid development of memory-based flash technologies assures you of lower product costs along with multiple, competitive, state-of-the-art fabrication sources.

Powerful Software Tools

Advanced fitter algorithms and powerful "auto-interactive" features allow you to choose between pushbutton and manual design flows, and standards-based design-flows including EDIF, SDF, VHDL (VITAL), and Verilog. Both the Alliance Series and Foundation Series software include the JTAG software and the JTAG cable required for ISP JTAG programming of all XC9500 products.

XC9500 Product Overview

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
Macrocells	36	72	108	144	216	288
Usable Gates	800	1,600	2,400	3,200	4,800	6,400
t_{PD} (ns)	5.0	7.5	7.5	7.5	10.0	15.0
Registers	36	72	108	144	216	288
Maximum User I/Os	34	72	108	133	166	192
Packages	44VQ					
	44PC	44PC				
		84PC	84PC			
		100TQ	100TQ	100TQ		
		100PQ	100PQ	100PQ		
			160PQ	160PQ	160PQ	

XILINX®

XC9500 In-System Programmable CPLD Family

January 16, 1998 (Version 2.1)

Product Information

Features

- High-performance
  - 5 ns pin-to-pin logic delays on all pins
  - f_{CNT} to 125 MHz
- Large density range
  - 36 to 288 macrocells with 800 to 6,400 usable gates
- 5 V in-system programmable
  - Endurance of 10,000 program/erase cycles
  - Program/erase over full commercial voltage and temperature range
- Enhanced pin-locking architecture
- Flexible 36V18 Function Block
  - 90 product terms drive any or all of 18 macrocells within Function Block
  - Global and product term clocks, output enables, set and reset signals
- Extensive IEEE Std 1149.1 boundary-scan (JTAG) support
- Programmable power reduction mode in each macrocell
- Slew rate control on individual outputs
- User programmable ground pin capability
- Extended pattern security features for design protection
- High-drive 24 mA outputs
- 3.3 V or 5 V I/O capability
- PCI compliant (-5, -7, -10 speed grades)
- Advanced CMOS 5V FastFLASH technology
- Supports parallel programming of multiple XC9500 devices

Family Overview

The XC9500 CPLD family provides advanced in-system programming and test capabilities for high performance, general purpose logic integration. All devices are in-system programmable for a minimum of 10,000 program/erase cycles. Extensive IEEE 1149.1 (JTAG) boundary-scan support is also included on all family members.

As shown in Table 1, logic density of the XC9500 devices ranges from 800 to over 6,400 usable gates with 36 to 288 registers, respectively. Multiple package options and associated I/O capacity are shown in Table 2. The XC9500 family is fully pin-compatible allowing easy design migration across multiple density options in a given package footprint.

The XC9500 architectural features address the requirements of in-system programmability. Enhanced pin-locking capability avoids costly board rework. An expanded JTAG instruction set allows version control of programming patterns and in-system debugging. In-system programming throughout the full device operating range and a minimum of 10,000 program/erase cycles provide worry-free reconfigurations and system field upgrades.

Advanced system features include output slew rate control and user-programmable ground pins to help reduce system noise. I/Os may be configured for 3.3 V or 5 V operation. All outputs provide 24 mA drive.

Architecture Description

Each XC9500 device is a subsystem consisting of multiple Function Blocks (FBs) and I/O Blocks (IOBs) fully interconnected by the FastCONNECT switch matrix. The IOB provides buffering for device inputs and outputs. Each FB provides programmable logic capability with 36 inputs and 18 outputs. The FastCONNECT switch matrix connects all FB outputs and input signals to the FB inputs. For each FB, 12 to 18 outputs (depending on package pin-count) and associated output enable signals drive directly to the IOBs. See Figure 1.

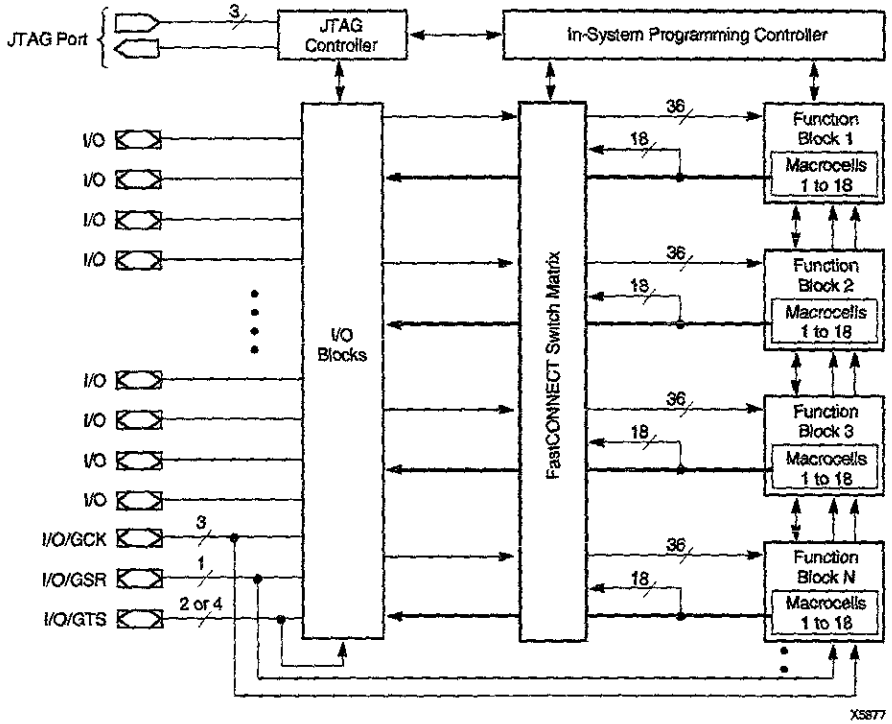


Figure 1: XC9500 Architecture

Note: Function Block outputs (indicated by the bold line) drive the I/O Blocks directly.

Table 1: XC9500 Device Family

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
Macrocells	36	72	108	144	216	288
Usable Gates	800	1,600	2,400	3,200	4,800	6,400
Registers	36	72	108	144	216	288
t_{PD} (ns)	5	7.5	7.5	7.5	10	15
t_{SU} (ns)	4.5	5.5	5.5	5.5	6.5	8.0
t_{CO} (ns)	4.5	5.5	5.5	5.5	6.5	8.0
f_{CNT} (MHz)	100	125	125	125	111	95
f_{SYSTEM} (MHz)	100	83	83	83	67	56

Note: f_{CNT} = Operating frequency for 16-bit counters
f_{SYSTEM} = Internal operating frequency for general purpose system designs spanning multiple FBs.

Table 2: Available Packages and Device I/O Pins (not including dedicated JTAG pins)

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
44-Pin VQFP	34					
44-Pin PLCC	34	34				
84-Pin PLCC		69	69			
100-Pin TQFP		72	81	81		
100-Pin PQFP		72	81	81		
160-Pin PQFP			108	133	133	
208-Pin HQFP					166	168
352-Pin BGA					166	192

Function Block

Each Function Block, as shown in Figure 2, is comprised of 18 independent macrocells, each capable of implementing a combinatorial or registered function. The FB also receives global clock, output enable, and set/reset signals. The FB generates 18 outputs that drive the FastCONNECT switch matrix. These 18 outputs and their corresponding output enable signals also drive the IOB.

Logic within the FB is implemented using a sum-of-products representation. Thirty-six inputs provide 72 true and complement signals into the programmable AND-array to form 90 product terms. Any number of these product terms, up to the 90 available, can be allocated to each macrocell by the product term allocator.

Each FB (except for the XC9536) supports local feedback paths that allow any number of FB outputs to drive into its own programmable AND-array without going outside the FB. These paths are used for creating very fast counters and state machines where all state registers are within the same FB.

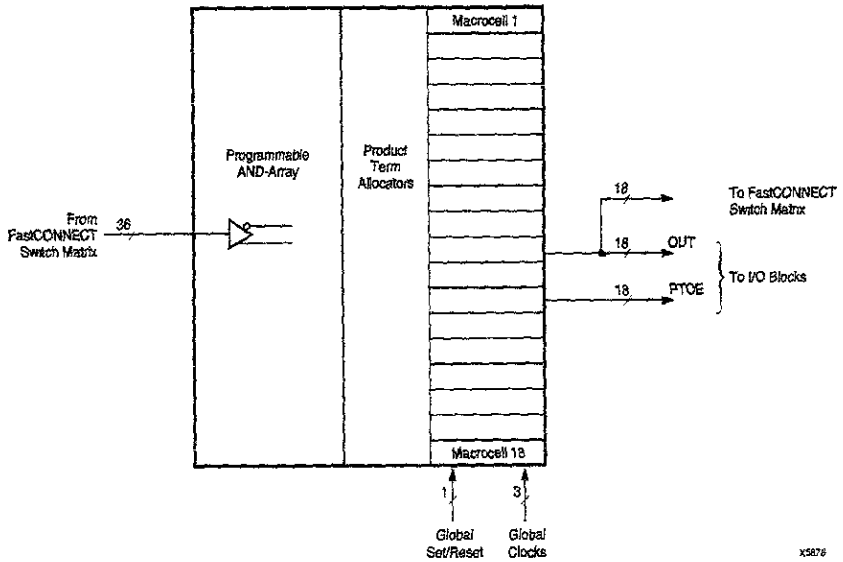


Figure 2: XC9500 Function Block

Macrocell

Each XC9500 macrocell may be individually configured for a combinatorial or registered function. The macrocell and associated FB logic is shown in Figure 3.

Five direct product terms from the AND-array are available for use as primary data inputs (to the OR and XOR gates) to implement combinatorial functions, or as control inputs including clock, set/reset, and output enable. The product term allocator associated with each macrocell selects how the five direct terms are used.

The macrocell register can be configured as a D-type or T-type flip-flop, or it may be bypassed for combinatorial operation. Each register supports both asynchronous set and reset operations. During power-up, all user registers are initialized to the user-defined preload state (default to 0 if unspecified).

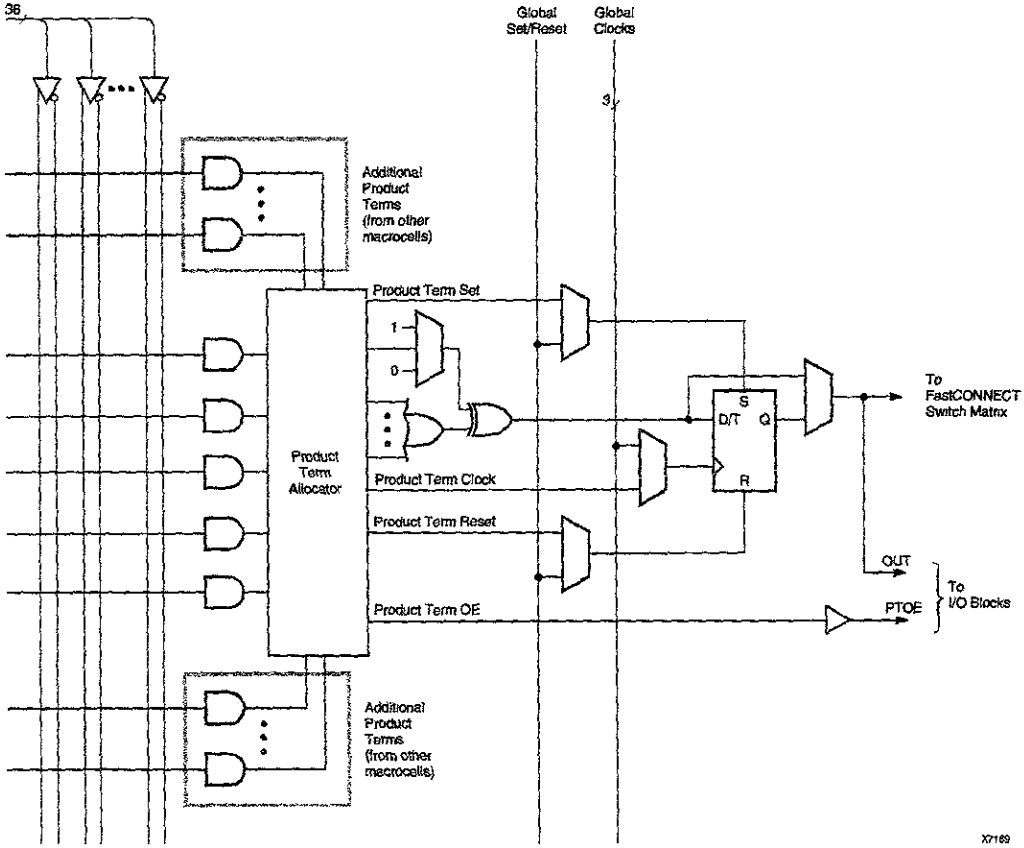


Figure 3: XC9500 Macrocell Within Function Block

All global control signals are available to each individual macrocell, including clock, set/reset, and output enable signals. As shown in Figure 4, the macrocell register clock originates from either of three global clocks or a product term clock. Both true and complement polarities of a GCK pin can be used within the device. A GSR input is also provided to allow user registers to be set to a user-defined state.

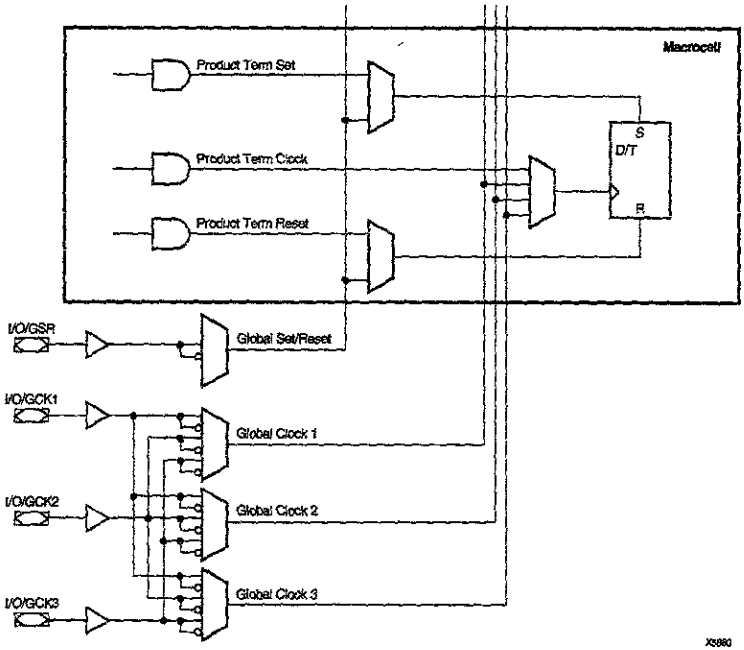


Figure 4: Macrocell Clock and Set/Reset Capability

Product Term Allocator

The product term allocator controls how the five direct product terms are assigned to each macrocell. For example, all five direct terms can drive the OR function as shown in Figure 5.

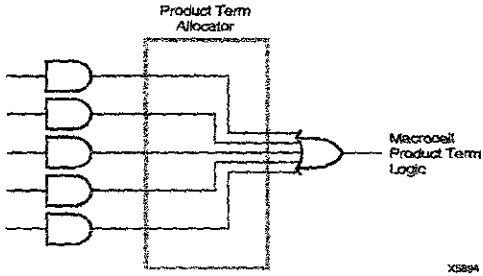


Figure 5: Macrocell Logic Using Direct Product Term

The product term allocator can re-assign other product terms within the FB to increase the logic capacity of a macrocell beyond five direct terms. Any macrocell requiring additional product terms can access uncommitted product terms in other macrocells within the FB. Up to 15 product terms can be available to a single macrocell with only a small incremental delay of t_{PTA}, as shown in Figure 6.

Note that the incremental delay affects only the product terms in other macrocells. The timing of the direct product terms is not changed.

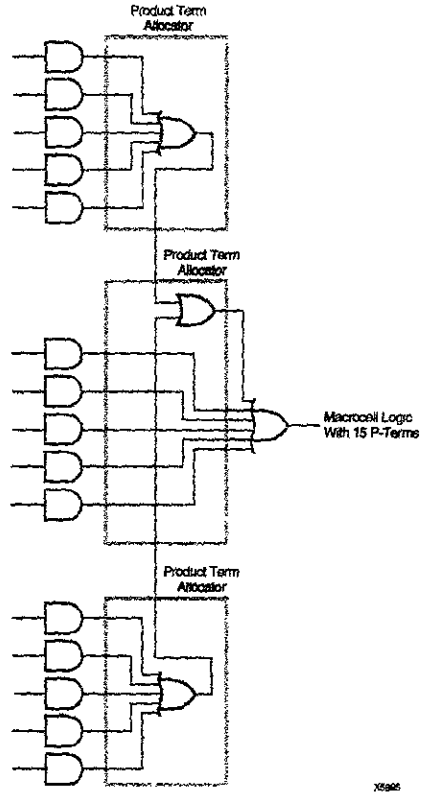


Figure 6: Product Term Allocation With 15 Product Terms

The product term allocator can re-assign product terms from any macrocell within the FB by combining partial sums of products over several macrocells, as shown in Figure 7.

In this example, the incremental delay is only $2*t_{PTA}$. All 90 product terms are available to any macrocell, with a maximum incremental delay of $8*t_{PTA}$.

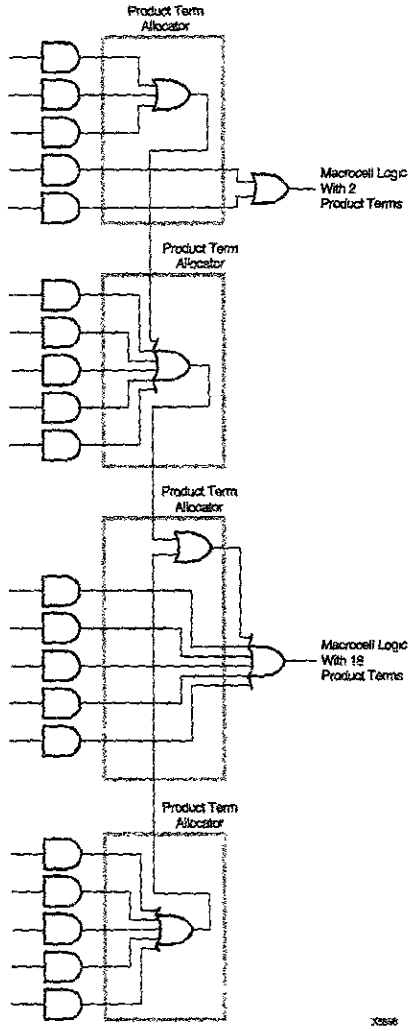


Figure 7: Product Term Allocation Over Several Macrocells

The internal logic of the product term allocator is shown in Figure 8.

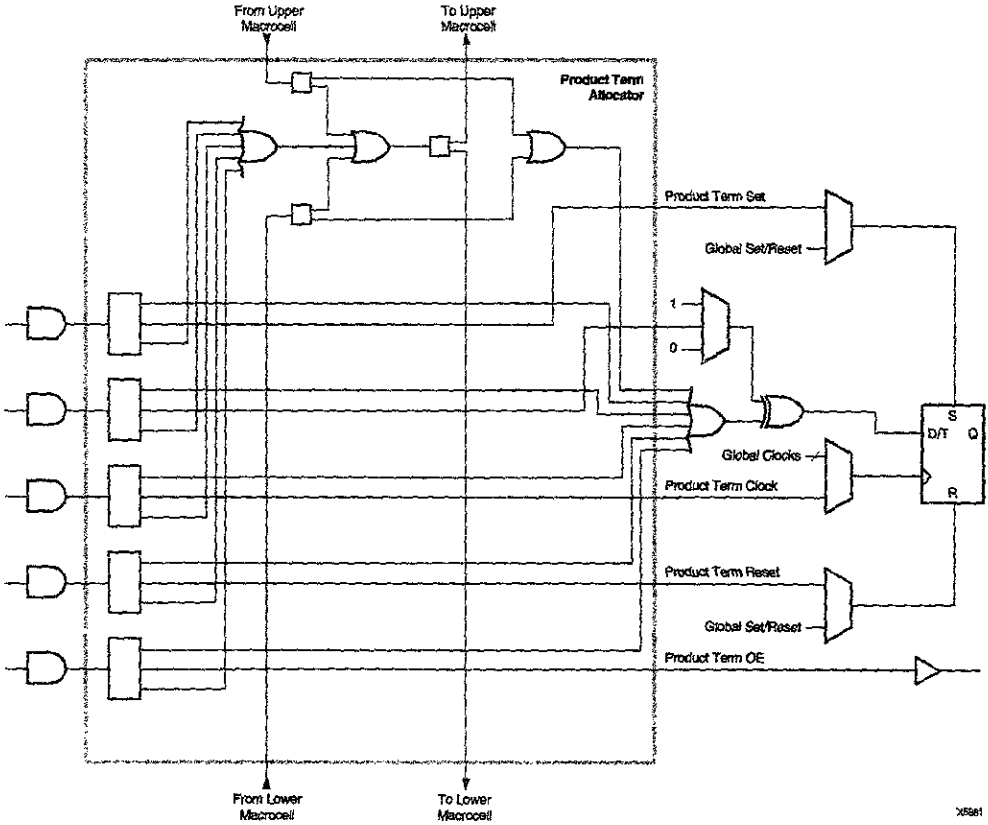


Figure 8: Product Term Allocator Logic

FastCONNECT Switch Matrix

The FastCONNECT switch matrix connects signals to the FB inputs, as shown in Figure 9. All IOB outputs (corresponding to user pin inputs) and all FB outputs drive the FastCONNECT matrix. Any of these (up to a FB fan-in limit of 36) may be selected, through user programming, to drive each FB with a uniform delay.

The FastCONNECT switch matrix is capable of combining multiple internal connections into a single wired-AND output before driving the destination FB. This provides additional logic capability and increases the effective logic fan-in of the destination FB without any additional timing delay. This capability is available for internal connections originating from FB outputs only. It is automatically invoked by the development software where applicable.

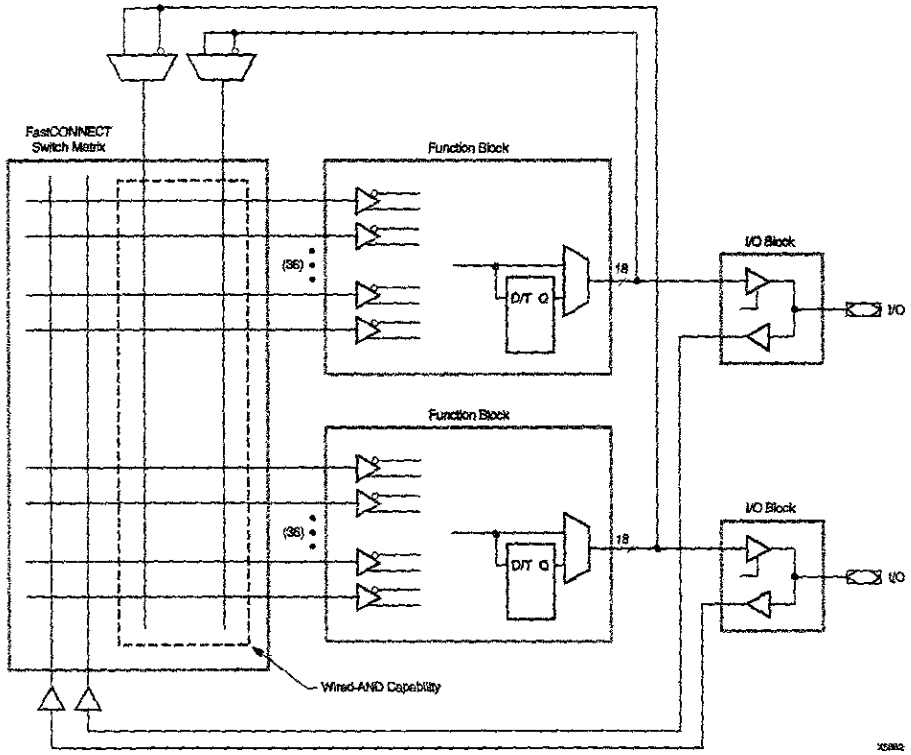


Figure 9: FastCONNECT Switch Matrix

I/O Block

The I/O Block (IOB) interfaces between the internal logic and the device user I/O pins. Each IOB includes an input buffer, output driver, output enable selection multiplexer, and user programmable ground control. See Figure 10 for details.

The input buffer is compatible with standard 5 V CMOS, 5 V TTL and 3.3 V signal levels. The input buffer uses the internal 5 V voltage supply (V_{CCINT}) to ensure that the input thresholds are constant and do not vary with the V_{CCIO} voltage.

The output enable may be generated from one of four options: a product term signal from the macrocell, any of the global OE signals, always "1", or always "0". There are two global output enables for devices with up to 144 macrocells, and four global output enables for devices with 180 or more macrocells. Both polarities of any of the global 3-state control (GTS) pins may be used within the device.

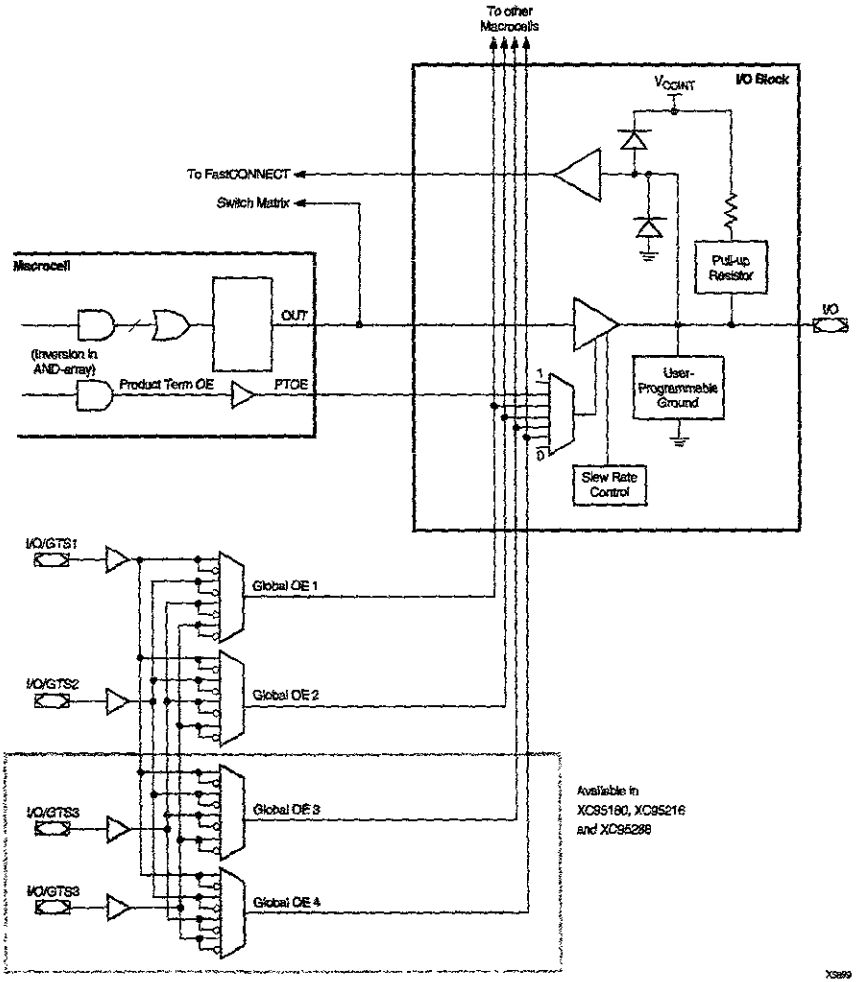


Figure 10: I/O Block and Output Enable Capability

Each output has independent slew rate control. Output edge rates may be slowed down to reduce system noise (with an additional time delay of t_{SLEW}) through programming. See Figure 11.

Each IOB provides user programmable ground pin capability. This allows device I/O pins to be configured as additional ground pins. By tying strategically located programmable ground pins to the external ground connection, system noise generated from large numbers of simultaneous switching outputs may be reduced.

A control pull-up resistor (typically 10K ohms) is attached to each device I/O pin to prevent them from floating when the device is not in normal user operation. This resistor is active during device programming mode and system power-up. It is also activated for an erased device. The resistor is deactivated during normal operation.

The output driver is capable of supplying 24 mA output drive. All output drivers in the device may be configured for either 5 V TTL levels or 3.3 V levels by connecting the device output voltage supply (V_{CCIO}) to a 5 V or 3.3 V voltage supply. Figure 12 shows how the XC9500 device can be used in 5 V only and mixed 3.3 V/5 V systems.

Pin-Locking Capability

The capability to lock the user defined pin assignments during design changes depends on the ability of the architecture to adapt to unexpected changes. The XC9500 devices have architectural features that enhance the ability to accept design changes while maintaining the same pinout.

The XC9500 architecture provides maximum routing within the FastCONNECT switch matrix, and incorporates a flexible Function Block that allows block-wide allocation of available product terms. This provides a high level of confidence of maintaining both input and output pin assignments for unexpected design changes.

For extensive design changes requiring higher logic capacity than is available in the initially chosen device, the new design may be able to fit into a larger pin-compatible device using the same pin assignments. The same board may be used with a higher density device without the expense of board rework.

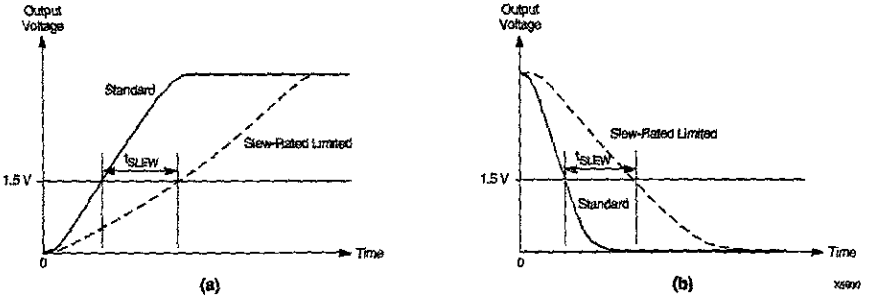


Figure 11: Output Slew-Rate Control For (a) Rising and (b) Falling Outputs

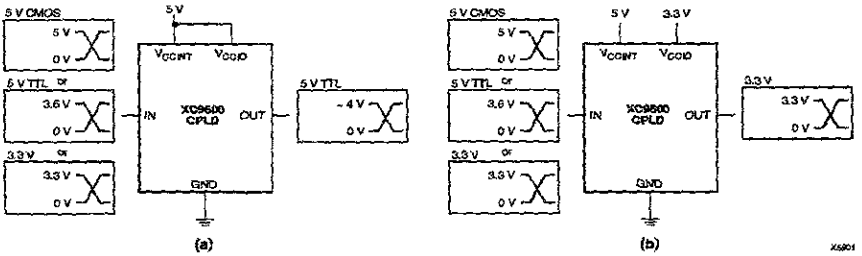


Figure 12: XC9500 Devices in (a) 5 V Systems and (b) Mixed 3.3 V/5 V Systems

In-System Programming

XC9500 devices are programmed in-system via a standard 4-pin JTAG protocol, as shown in Figure 13. In-system programming offers quick and efficient design iterations and eliminates package handling. The Xilinx development system provides the programming data sequence using a Xilinx download cable, a third-party JTAG development system, JTAG-compatible board tester, or a simple microprocessor interface that emulates the JTAG instruction sequence.

All I/Os are 3-stated and pulled high by the IOB resistors during in-system programming. If a particular signal must remain low during this time, then a pulldown resistor may be added to the pin.

External Programming

XC9500 devices can also be programmed by the Xilinx HW130 device programmer as well as third-party programmers. This provides the added flexibility of using pre-programmed devices during manufacturing, with an in-system programmable option for future enhancements.

Endurance

All XC9500 CPLDs provide a minimum endurance level of 10,000 in-system program/erase cycles. Each device meets all functional, performance, and data retention specifications within this endurance limit.

IEEE 1149.1 Boundary-Scan (JTAG)

XC9500 devices fully support IEEE 1149.1 boundary-scan (JTAG). EXTEST, SAMPLE/PRELOAD, BYPASS, USERCODE, INTEST, IDCODE, and HIGHZ instructions are supported in each device. For ISP operations, five additional instructions are added; the ISPEN, FERASE, FPGM, FVFY, and ISPEX instructions are fully compliant extensions of the 1149.1 instruction set.

The TMS and TCK pins have dedicated pull-up resistors as specified by the IEEE 1149.1 standard.

Boundary Scan Description Language (BSDL) files for the XC9500 are included in the development system and are available on the Xilinx FTP site.

Design Security

XC9500 devices incorporate advanced data security features which fully protect the programming data against unauthorized reading or inadvertent device erasure/reprogramming. Table 3 shows the four different security settings available.

The read security bits can be set by the user to prevent the internal programming pattern from being read or copied. Erasing the entire device is the only way to reset the read security bit.

The write security bits provide added protection against accidental device erasure or reprogramming when the JTAG pins are subject to noise, such as during system power-up. Once set, the write-protection may be deactivated when the device needs to be reprogrammed with a valid pattern.

Table 3: Data Security Options

		Read Security: Default	Read Security: Set
Write Security: Default		Read Allowed Program/Erase Allowed	Read Inhibited Program/Erase Allowed
Write Security: Set		Read Allowed Program/Erase Inhibited	Read Inhibited Program/Erase Inhibited

X5906

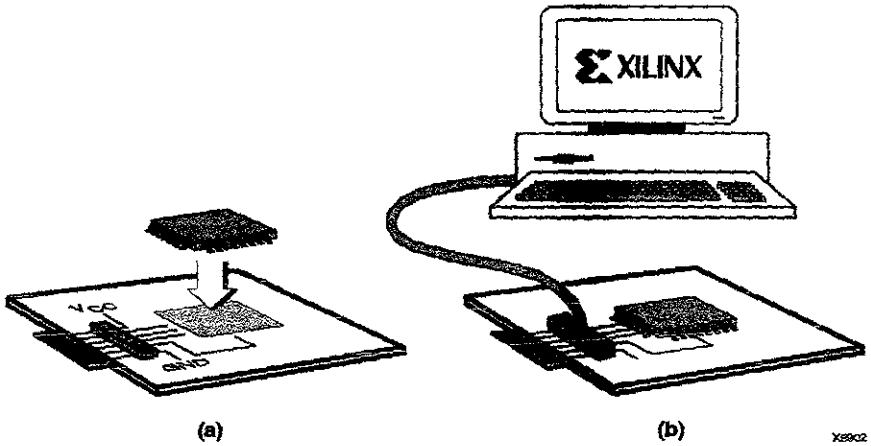


Figure 13: In-System Programming Operation (a) Solder Device to PCB and (b) Program Using Download Cable

Low Power Mode

All XC9500 devices offer a low-power mode for individual macrocells or across all macrocells. This feature allows the device power to be significantly reduced.

Each individual macrocell may be programmed in low-power mode by the user. Performance-critical parts of the application can remain in standard power mode, while other parts of the application may be programmed for low-power operation to reduce the overall power dissipation. Macrocells programmed for low-power mode incur additional delay (t_{LP}) in pin-to-pin combinatorial delay as well as register setup time. Product term clock to output and product term output enable delays are unaffected by the macrocell power-setting.

Timing Model

The uniformity of the XC9500 architecture allows a simplified timing model for the entire device. The basic timing model, shown in Figure 14, is valid for macrocell functions that use the direct product terms only, with standard power setting, and standard slew rate setting. Table 4 shows how each of the key timing parameters is affected by the product term allocator (if needed), low-power setting, and slew-limited setting.

The product term allocation time depends on the logic span of the macrocell function, which is defined as one less than the maximum number of allocators in the product term path. If only direct product terms are used, then the logic span is 0. The example in Figure 6 shows that up to 15 product terms are available with a span of 1. In the case of Figure 7, the 18 product term function has a span of 2.

Detailed timing information may be derived from the full timing model shown in Figure 15. The values and explanations for each parameter are given in the individual device data sheets.

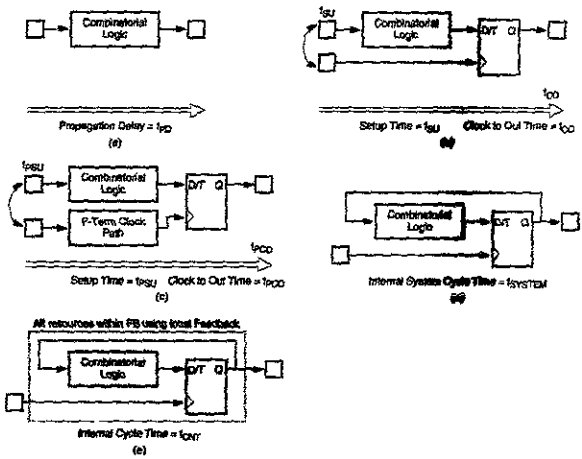


Figure 14: Basic Timing Model

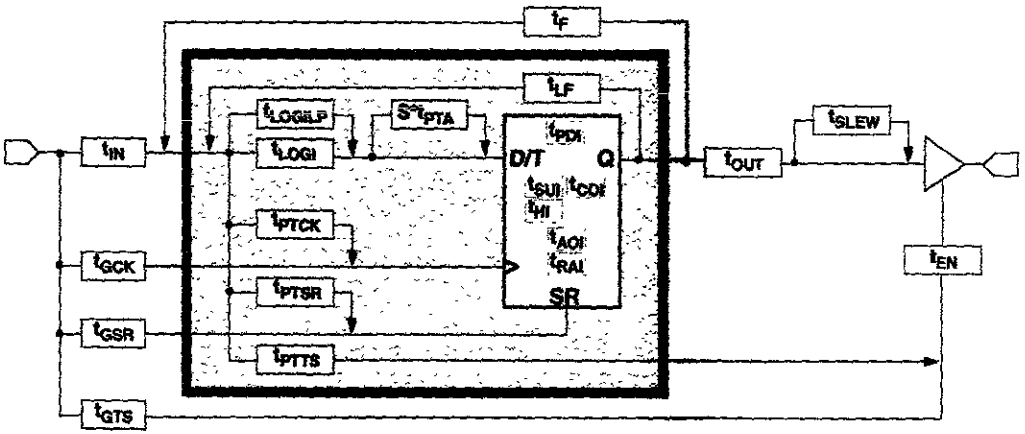


Figure 15: Detailed Timing Model

Power-Up Characteristics

The XC9500 devices are well behaved under all operating conditions. During power-up each XC9500 device employs internal circuitry which keeps the device in the quiescent state until the V_{CCINT} supply voltage is at a safe level (approximately 3.8 V). During this time, all device pins and JTAG pins are disabled and all device outputs are disabled with the IOB pull-up resistors (~ 10K ohms) enabled, as shown in Table 5. When the supply voltage reaches a safe level, all user registers become initialized (typically within 100 µs for 9536 - 95144, 200 µs for 95216 and 300 µs for 95288), and the device is immediately available for operation, as shown in Figure 16.

If the device is in the erased state (before any user pattern is programmed), the device outputs remain disabled with the IOB pull-up resistors enabled. The JTAG pins are enabled to allow the device to be programmed at any time.

If the device is programmed, the device inputs and outputs take on their configured states for normal operation. The JTAG pins are enabled to allow device erasure or boundary-scan tests at any time.

Development System Support

The XC9500 CPLD family is fully supported by the development systems available from Xilinx and the Xilinx Alliance Program vendors.

The designer can create the design using ABEL, schematics, equations, VHDL, or Verilog in a variety of software front-end tools. The development system can be used to implement the design and generate a JEDEC bitmap which can be used to program the XC9500 device. Each development system includes JTAG download software that can be used to program the devices via the standard JTAG interface and a download cable.

FastFLASH Technology

An advanced CMOS Flash process is used to fabricate all XC9500 devices. Specifically developed for Xilinx in-system programmable CPLDs, the FastFLASH process provides high performance logic capability, fast programming times, and endurance of 10,000 program/erase cycles.

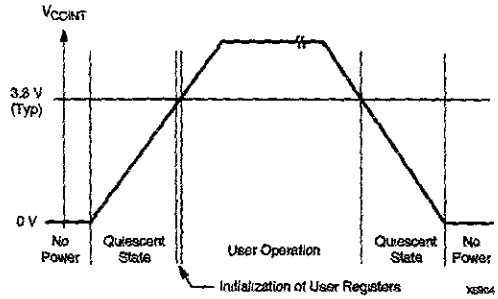


Figure 16: Device Behavior During Power-up

Table 4: Timing Model Parameters

Description	Parameter	Product Term Allocator[1]	Macrocell Low-Power Setting	Output Slew-Limited Setting
Propagation Delay	t_{PD}	+ t_{PTA} * S	+ t_{LP}	+ t_{SLEW}
Global Clock Setup Time	t_{SU}	+ t_{PTA} * S	+ t_{LP}	–
Global Clock-to-output	t_{CO}	–	–	+ t_{SLEW}
Product Term Clock Setup Time	t_{PSU}	+ t_{PTA} * S	+ t_{LP}	–
Product Term Clock-to-output	t_{PCO}	–	–	+ t_{SLEW}
Internal System Cycle Period	t_{SYSTEM}	+ t_{PTA} * S	+ t_{LP}	–

Note: 1. S = the logic span of the function, as defined in the text.

Table 5: XC9500 Device Characteristics

Device Circuitry	Quiescent State	Erased Device Operation	Valid User Operation
IOB Pull-up Resistors	Enabled	Enabled	Disabled
Device Outputs	Disabled	Disabled	As Configured
Device Inputs and Clocks	Disabled	Disabled	As Configured
Function Block	Disabled	Disabled	As Configured
JTAG Controller	Disabled	Enabled	Enabled

BURR-BROWN®

PCM1760P/U
DF1760P/U

Multi-Bit Enhanced Noise Shaping 20-Bit ANALOG-TO-DIGITAL CONVERSION SYSTEM

FEATURES

- DUAL 20-BIT MONOLITHIC MODULATOR (PCM1760) AND MONOLITHIC DECIMATING DIGITAL FILTER (DF1760)
- HIGH PERFORMANCE:
  THD+N: –92dB typ, –90dB max
  Dynamic Range: 108dB typ
  SNR: 108dB min, 110dB typ
  Channel Separation: 98dB typ, 94dB min
- 64X OVERSAMPLING
- CO-PHASE CONVERSION
- RUNS ON 256fs OR 384fs SYSTEM CLOCK
- VERSATILE INTERFACE CAPABILITY:
  16-, 20-Bit Output
  MSB First or LSB First Format
- OPTIONAL FUNCTIONS:
  Offset Error Calibration
  Overflow Detection
  Power Down Mode (DF1760)
- RUNS ON ±5V SUPPLIES (PCM1760) AND 5V SUPPLY (DF1760)
- COMPACT 28-PIN PACKAGES:
  28-Pin DIP and SOIC

DESCRIPTION

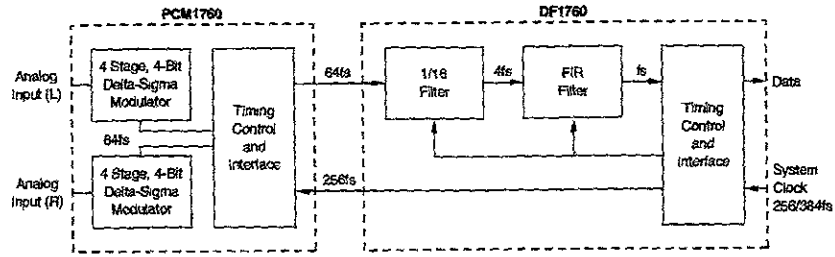
The PCM1760 and DF1760 combine for a low-cost, high-performance dual 20-bit, 48kHz sampling analog-to-digital conversion system which is specifically designed for dynamic applications.

The PCM1760/DF1760 pair form a 4-bit, 4th order, 64X oversampling analog-to-digital converter.

The PCM1760 is a delta-sigma modulator that uses a 4-bit quantizer within the modulation loop to achieve very high dynamic range.

The DF1760 is a high-performance decimating digital filter. The DF1760 accepts 4-bit 64fs data from the PCM1760 and decimates to 20-bit 1fs data.

The FIR filter of the DF1760 has pass-band ripple of less than ±0.001dB and greater than 100dB of the reject band attenuation



International Airport Industrial Park • Mailing Address: PO Box 11400 • Tucson, AZ 85734 • Street Address: 6730 S. Tucson Blvd. • Tucson, AZ 85706
Tel: (520) 746-1111 • Twx: 910-952-1111 • Cable: BBRCORP • Telex: 066-6491 • FAX: (520) 889-1510 • Immediate Product Info: (800) 548-6132

PDS-1174C

Printed in U.S.A. July, 1994

SPECIFICATIONS

ELECTRICAL

At T_A = +25°C, $\pm V_{CC}$, $\pm V_{dd}$ = +5V, $+V_{DD}$ = +5V, f_S = 48kHz and ext. components = ±2% unless otherwise noted.

PARAMETER	CONDITIONS	PCM1760/DF1760 MIN	PCM1760/DF1760 TYP	PCM1760/DF1760 MAX	UNITS
RESOLUTION		20			Bits
ANALOG INPUT					
Input Range	R_{IN}1 = 2.2kΩ		±2.5		Vp-p
Input Impedance	R_{IN}1 = 2.2kΩ		R_{IN}1		Ω
SAMPLING FREQUENCY					
Cover Range of fs	Integrator Constants: Application(1)	30	48	50	kHz
ACCURACY					
Gain Error			±0.5	±1.0	dB
Gain Mismatch				±0.5	dB
Bipolar Zero Error	V_{IN} = 0 at 20s After Power-On			±0.4	% FSR(2)
Gain Drift	0°C to +70°C		±100		ppm/s/°C
Bipolar Zero Drift	0°C to +70°C		±20		ppm/s/°C
DYNAMIC CHARACTERISTICS(4)					
THD+N/(0dBFS) P, U	f_{IN} = 1kHz		-92	-90	dB
P-L, U-L			-90	-88	dB
THD+N/(-20dBFS) P, U	f_{IN} = 1kHz		-76	-70	dB
P-L, U-L			-76	-70	dB
THD+N/(-60dBFS) P, U	f_{IN} = 1kHz		-44	-42	dB
P-L, U-L			-44	-42	dB
Dynamic Range P, U	f_{IN} = 1kHz, V_{IN} = -60dBFS, A Filter	104	108		dB
P-L, U-L		104	108		dB
SNR P, U	V_{IN} = 0, A Filter	108	110		dB
P-L, U-L		106	110		dB
Frequency Response	f_{IN} = 20kHz		±0.1		dB
Channel Separation	f_{IN} = 1kHz, A Filter	94	98		dB
DIGITAL FILTER					
Over Sample Rate			64		fs
Ripple In Band	0 - 0.4535fs			±0.0001	dB
Stopband Attenuation -1	0.5465fs - 63.4535fs	-94			dB
Stopband Attenuation -2	0.5465fs - 3.4535fs	-100			dB
LOGIC INPUTS AND OUTPUTS					
Logic Family Input			TTL Level Compatible CMOS		
Frequency (System Clock 1)	256fs		12.288		MHz
Frequency (System Clock 2)	384fs		18.432		MHz
Duty Cycle (System Clock 1)	256fs	40	50	60	%
Duty Cycle (System Clock 2)	384fs	45	50	55	%
Data Clock Input		32	48	64	fs
Logic Family Output			CMOS		
Data Clock Output			64		fs
Data Coding			Two's Complement		
Data Bit Length		16	20		Bits
Data Format			Selectable		
Output Data Delay	fs = 48kHz		1.5		ms
POWER SUPPLY REQUIREMENTS					
Supply Voltage					
$\pm V_{CC}$	PCM1760	±4.75	±5.0	±5.25	V
$\pm V_{dd}$	PCM1760	±4.75	±5.0	±5.25	V
$+V_{DD}$	DF1760	4.75	5.0	5.25	V
Supply Current					
$+I_{CC}$	PCM1760		24	36	mA
$-I_{CC}$	PCM1760		-30	-45	mA
$+I_{dd}$	PCM1760		12	18	mA
$-I_{dd}$	PCM1760		-8	-12	mA
$+I_{DD}$ -1	DF1760, Normal Mode		40	55	mA
$+I_{DD}$ -2	DF1760, Power-Down Mode		4	6.6	mA
Power Consumption	PCM1760		370	500	mW
	DF1760, Normal Mode		200	275	mW
	DF1760, Power-Down Mode		20	33	mW
TEMPERATURE RANGE					
Operating	PCM1760/DF1760	0	+25	+70	°C
Storage	PCM1760/DF1760	-50		+125	°C

NOTES: (1) Integrator Constants are determined by the external components shown in the block diagram. (2) FSR means Full Scale Range, digital output code is from 90000H to 70000H, FSR = 5.0V. (3) Use 20-bit DAC, 20kHz LPF, 400Hz HPF, average response. (4) Average response using a 20-bit reconstruction DAC with 20kHz low-pass filter and 400Hz high-pass filter.

ABSOLUTE MAXIMUM RATINGS—PCM1760

Supply Voltage	±6V
Voltage Mismatch	0.1V
Analog Input	$+V_{CC}$
Digital Input	$+V_{DD}$ +0.3V
	GND −0.3V
Power Dissipation/P	580mW
Power Dissipation/U	550mW
Lead Temperature/P (soldering, 10s)	260°C
Lead Temperature/U (soldering, 10s)	235°C
Operating Temperature	0°C to +70°C
Storage Temperature	−50°C to +125°C

ABSOLUTE MAXIMUM RATINGS—DF1760

Supply Voltage	7.0V
Voltage Mismatch	0.1V
Digital Input	$+V_{DD}$ +0.5V
	V_{SS} −0.5V
Input Current	±20mA
Power Dissipation/P	460mW
Power Dissipation/U	440mW
Lead Temperature/P (soldering, 10s)	260°C
Lead Temperature/U (soldering, 10s, reflow)	235°C
Operating Temperature	0°C to +70°C
Storage Temperature	−50°C to +125°C

ORDERING INFORMATION

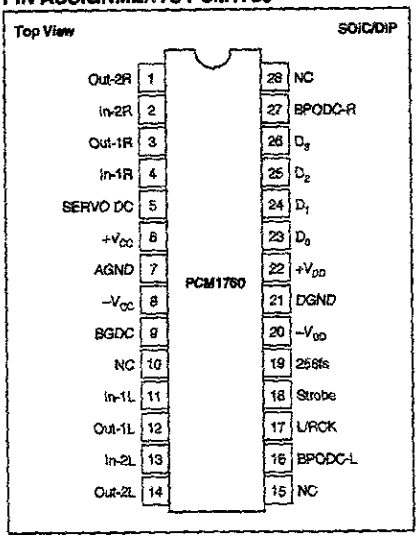
MODEL	PACKAGE	THD +N (fs)	SNR
PCM1760P	PDIP	−90dB	108dB
PCM1760U	SOIC	−90dB	108dB
PCM1760P-L	PDIP	−88dB	106dB
PCM1760U-L	SOIC	−88dB	106dB
DF1760P	PDIP	NA	NA
DF1760U	SOIC	NA	NA

PACKAGE INFORMATION

MODEL	PACKAGE	PACKAGE DRAWING NUMBER[1]
PCM1760P	28-Pin PDIP	800
PCM1760U	28-Pin SOIC	804
PCM1760P-L	28-Pin PDIP	800
PCM1760U-L	28-Pin SOIC	804
DF1760P	28-Pin PDIP	801
DF1760U	28-Pin SOIC	805

NOTE: (1) For detailed drawing and dimension table, please see end of data sheet, or Appendix D of Burr-Brown IC Data Book.

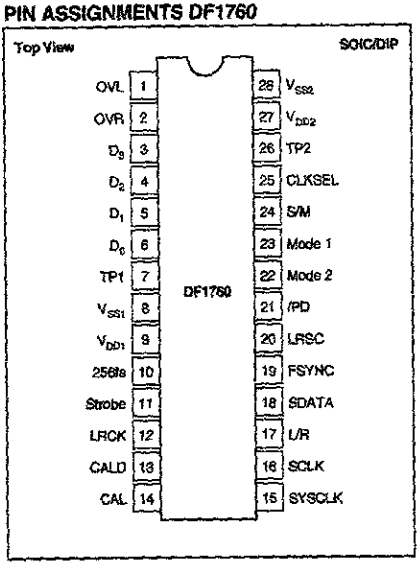
PIN ASSIGNMENTS PCM1760



PIN	I/O[1]	NAME	DESCRIPTION
1	O	Out-2R	Right Channel Second Integrator Output
2	I	In-2R	Right Channel Second Integrator Input
3	O	Out-1R	Right Channel First Integrator Output
4	I	In-1R	Right Channel First Integrator Input
5	–	SERVO DC	Servo Amp Decoupling Capacitor
6	–	$+V_{CC}$	+5V Analog Supply Voltage
7	–	AGND	Analog Common
8	–	$-V_{CC}$	–5V Analog Supply Voltage
9	–	BGDC	Band Gap Reference Decoupling Capacitor
10	–	NC	No Connection
11	I	In-1L	Left Channel First Integrator Input
12	O	Out-1L	Left Channel First Integrator Output
13	I	In-2L	Left Channel Second Integrator Input
14	O	Out-2L	Left Channel Second Integrator Output
15	–	NC	No Connection
16	–	BPODC-L	Left Channel Bipolar Offset Decoupling Capacitor
17	O	L/RCK	LR Clock Output (64fs)
18	O	Strobe	Data Strobe Output (128fs)
19	I	256fs	256fs Clock Input
20	–	$-V_{DD}$	–5V Digital Supply Voltage
21	–	DGND	Digital Common
22	–	$+V_{DD}$	+5V Digital Supply Voltage
23	O	D_0	D_0 Data Output (LSB)
24	O	D_1	D_1 Data Output
25	O	D_2	D_2 Data Output
26	O	D_3	D_3 Data Output (MSB)
27	–	BPODC-R	Right Channel Bipolar Offset Decoupling Capacitor
28	–	NC	No Connection

NOTE: (1) O = Output terminal; I = Input terminal.

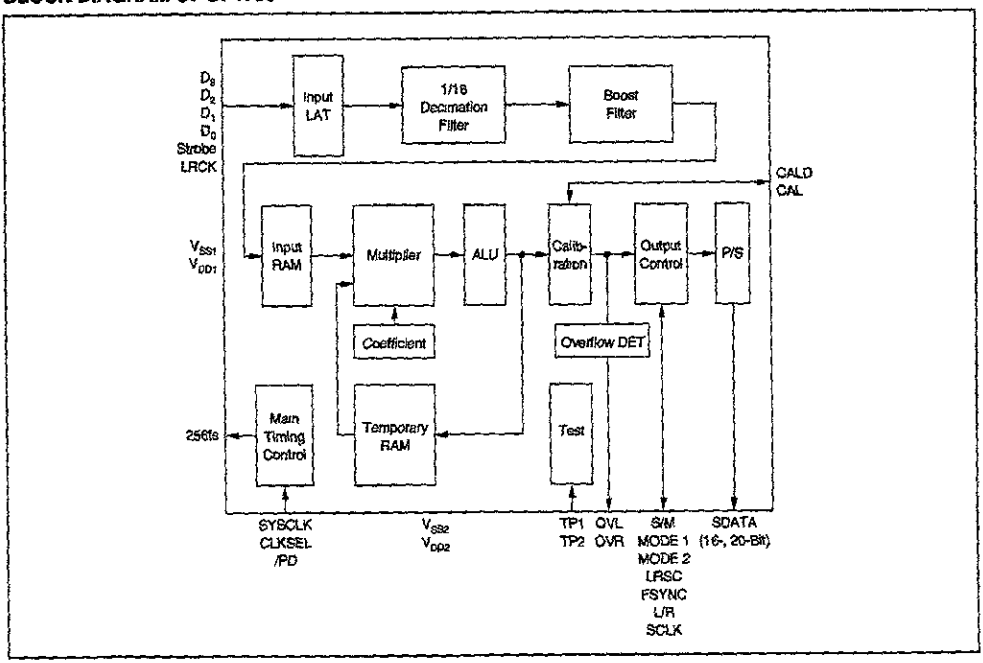
PIN ASSIGNMENTS DF1760



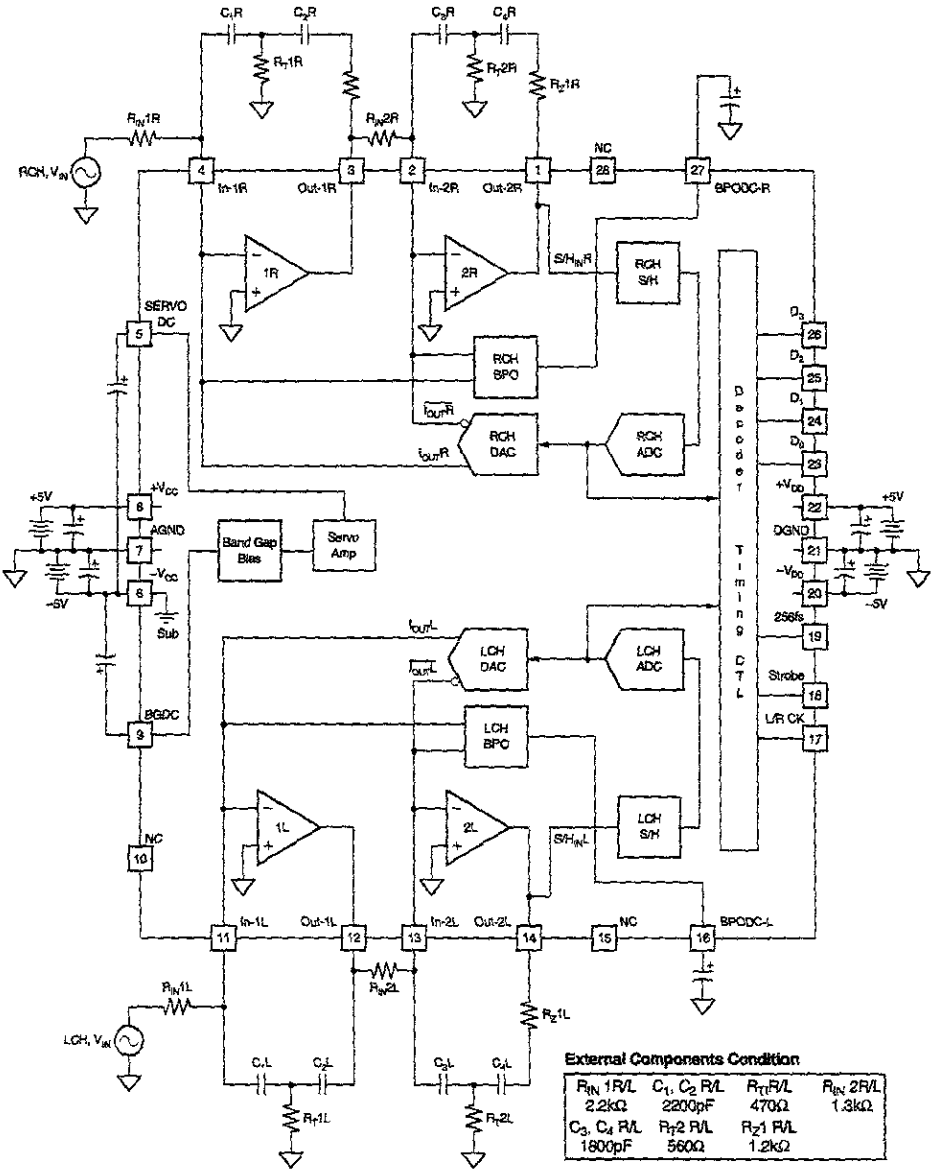
PIN	I/O[1]	NAME	DESCRIPTION
1	O	OVL	Left Channel Overflow Output (Active High)
2	O	OVR	Right Channel Overflow Output (Active High)
3	I	D_3	D3 Data Input (MSB)
4	I	D_2	D2 Data Input
5	I	D_1	D1 Data Input
6	I	D_0	D0 Data Input (LSB)
7	–	TP1	Test Pin (No Connection)
8	–	V_{SS1}	Common Channel 1
9	–	V_{DD1}	+5V Channel 1
10	O	256fs	256fs Clock Output
11	I	Strobe	Data Strobe Clock Input (128fs)
12	I	LRCK	LR Clock Input
13	I↑	CALD	Calibration Function Enable (Active Low)
14	O	CAL	Calibration Output (High During Calibration)
15	I	SYSCLK	System Clock Input (256fs or 384fs)
16	I↑/O	BCLK	Data Clock
17	I↑/O	L/R	LR Channel Phase Clock
18	O	SDATA	Serial Data Output (1fs)
19	I↑/O	FSYNC	Frame Clock (2fs)
20	I↑	LRSC	Phase Control of LR Channel Phase Clock
21	I↑	/PD	Power Down Mode Enable Input (Active Low)
22	I↑	Mode2	Output Format Selection Input 2
23	I↑	Mode1	Output Format Selection Input 1
24	I↑	S/M	Slave/Master Mode Selection Input (High Makes Slave Mode
25	I↑	CLKSEL	System Clock Selection Input (High Makes 256fs)
26	–	TP2	Test Pin (No Connection)
27	–	V_{DD2}	+5V Channel 2
28	–	V_{SS2}	Common Channel 2

NOTE. (1) O = Output terminal; I = Input terminal.

BLOCK DIAGRAM OF DF1760

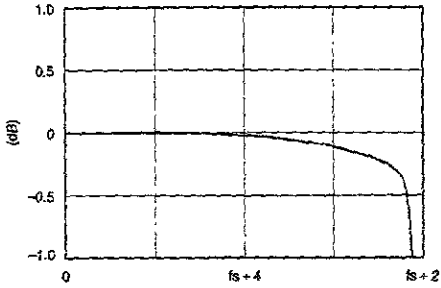


BLOCK DIAGRAM OF PCM1760

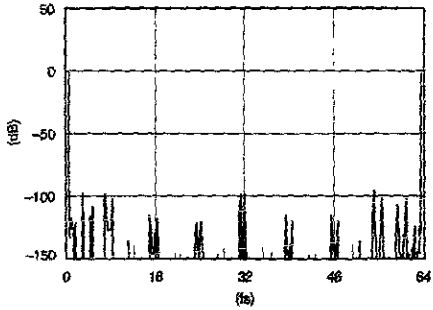


TYPICAL PERFORMANCE CURVES

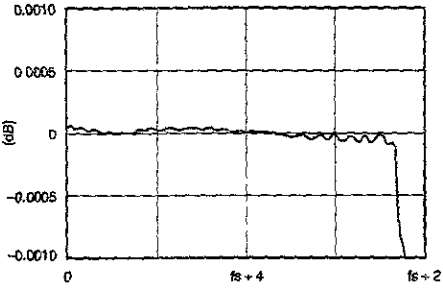
OVERALL PASS-BAND CHARACTERISTICS OF THE DF1760



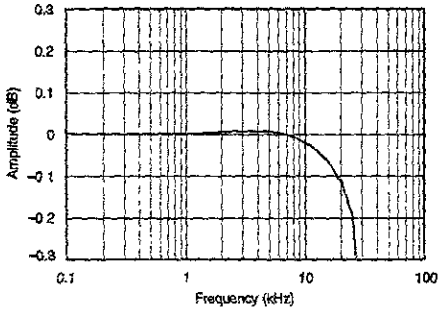
OVERALL CHARACTERISTICS OF THE DF1760



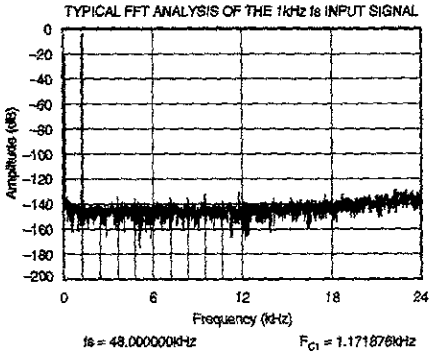
PASS-BAND CHARACTERISTICS OF THE FIR PORTION OF THE DF1760



TOTAL PASS-BAND FREQUENCY RESPONSE, COMBINATION OF PCM1760 AND DF1760

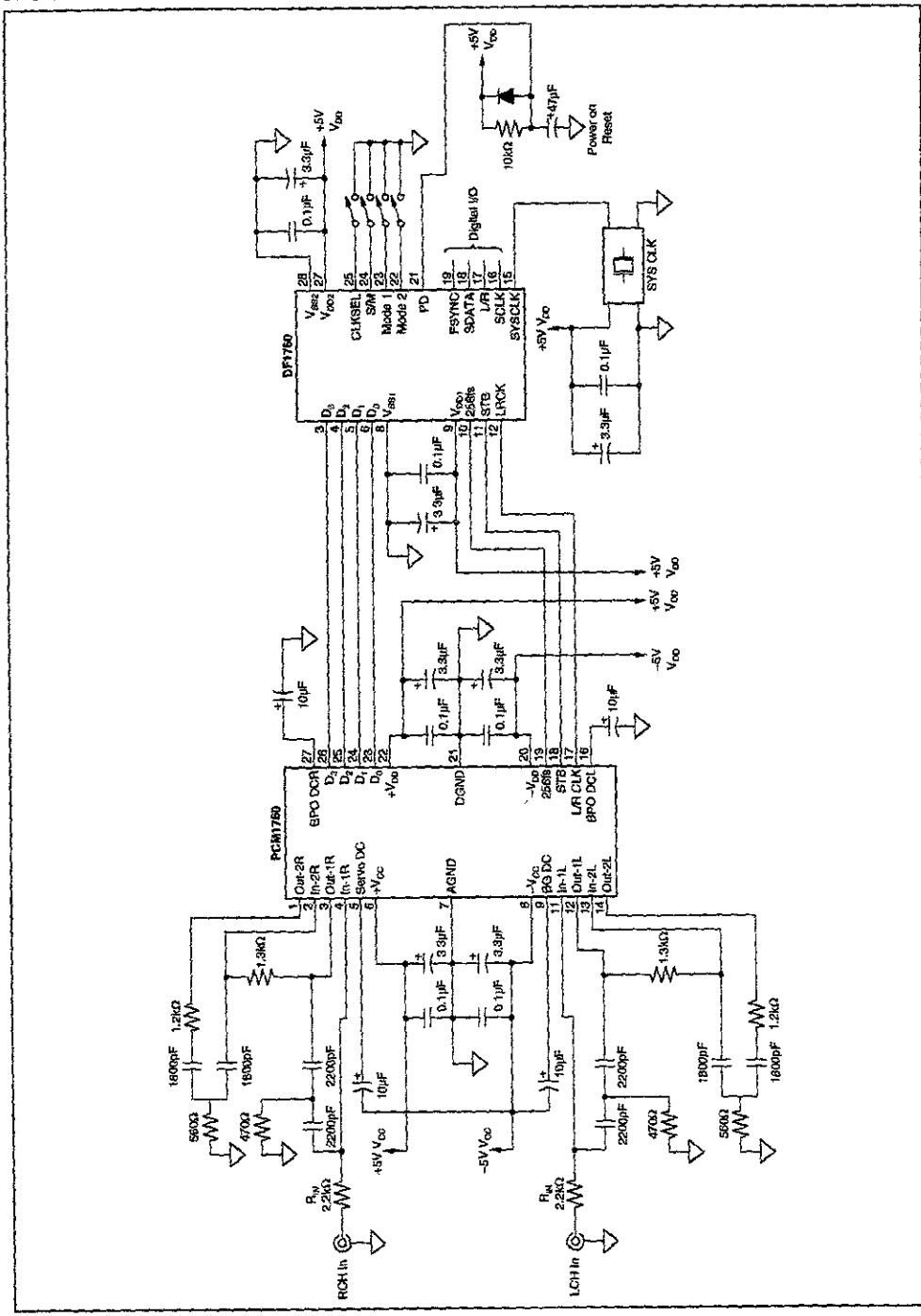


TYPICAL FFT ANALYSIS OF THE 1kHz fs INPUT SIGNAL



fs = 48.000000kHz F_{C1} = 1.171876kHz

BASIC CONNECTION DIAGRAM OF PCM1760 AND DF1760



FUNCTIONS OF THE DIGITAL FILTER

SYSTEM CLOCK

The DF1760 can accept a system clock of either 256fs or 384fs. If a 384fs system clock is used, the DF1760 divides by 2/3 to create the 256fs system clock required for the PCM1760. The system clock is applied to pin 15 (SYSCLK input). The actual clock selection is done by setting pin 25 (CLKSEL input) "high" for 256fs clock and "LOW" for 384fs clock.

The detailed timing requirements for the system clock are shown in Figure 3c.

CLKSEL	SYSCLK
H	256fs
L	384fs

MASTER/SLAVE MODE

The DF1760 can be used in both the master mode and slave mode. In the master mode, the DF1760 outputs L/R (left/right channel phase clock), SCLK (data clock) and FSYNC (frame clock 2fs) signals. In the slave mode, the DF1760 accepts L/R, SCLK and FSYNC signals. The mode selection is done by taking pin 24 (S/M INPUT) "HIGH" for slave mode and "LOW" for master mode.

S/M	MODE
H	Slave
L	Master

OUTPUT DATA FORMAT

The serial output data has four possible formats. The selection of the formats can be done by the Mode 1 and Mode 2 inputs.

MODE 1	MODE 2	FORMATS
H	H	MSB First, 16 Bits, Falling Edge
L	H	MSB First, 20 Bits, Falling Edge
H	L	MSB First, 20 Bits, Rising Edge
L	L	LSB First, 20 Bits, Falling Edge

LR CHANNEL PHASE CLOCK

The status of the LR channel phase clock can be set by the LRSC input.

LRSC	L/R CLOCK AND CHANNEL	
H	H = LCH,	L = RCH
L	L = LCH,	H = RCH

OVERFLOW DETECTION

When a near-to-clipping input condition is detected, OVL output (Pin 1), or OVR output (Pin 2), becomes "HIGH" for a duration of 4096/fs (about 85ms) depending upon on the channel detected.

The OVL and OVR output return to "LOW" after 4096/fs duration automatically.

OFFSET CALIBRATION MODE

The offset error is calibrated by storing the digital data when the input is zero in registers and subtracting it from the future data with actual signal input.

CALD	CALIBRATION
H	Disable
L	Enable

To enable the calibration mode, set the CALD input (Pin 13) "LOW". The calibration mode is disabled by setting the CALD input (Pin 13) "HIGH". The calibration cycle is initiated by setting the /PD input (Pin 21) "LOW" for more than 2 system clock periods and then setting it "HIGH". During the calibration cycle, the CAL output (Pin 14) becomes "HIGH", all the serial data is forced to "LOW", and the L/R (Pin 17), SCLK (Pin 16) and FSYNC (Pin 19) pins become input terminals after the completion of the calibration cycle. The CAL output is "LOW".

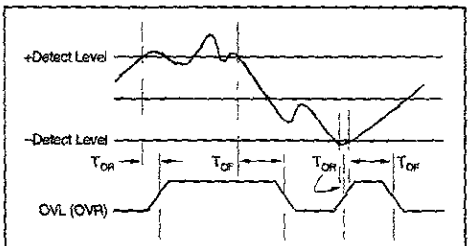
POWER DOWN MODE/RESET

The /PD input (Pin 21) has two functions. First, it should be set at "HIGH" after application or restoration of power (V_{SS} and/or V_{DD}) to accomplish the power-on/mode reset function. The detail timing requirements for this function are shown in Figure 3f. Second, the DF1760 is placed in the power down mode by setting the /PD input (Pin 21) "LOW". Set the /PD input (Pin 21) "HIGH" for normal operation mode.

/PD	OPERATION
H	Normal
L	Power Down

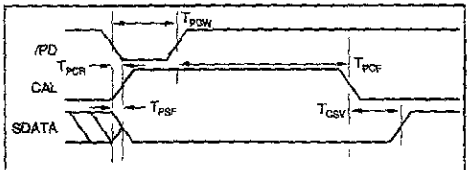
The power dissipation of the DF1760 in the power down mode is about 1/10 of the normal operation mode. During the power down mode, the L/R, SCLK, and FSYNC pins become input pins and all the serial data is forced "LOW". The 256fs output is enabled even in the power down mode.

The detailed timing of the power down mode operation and the offset calibration is shown in Figure 3b.



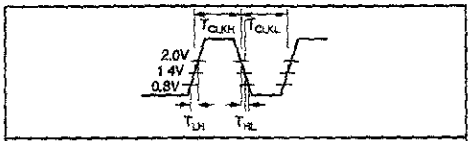
DESCRIPTION	NAME	MIN	TYP	MAX	UNITS
Delay from Overflow Detection to OVL (OVR) Output	T_{OR}	–	–	0	ns
OVL (OVR) Output Pulse Width	T_{OF}	–	4096	–	1/fs

FIGURE 3a. DF1760 Overflow Detection.



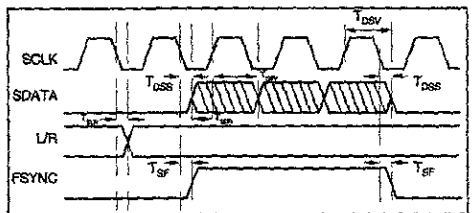
DESCRIPTION	NAME	MIN	TYP	MAX	UNITS
Pulse Width of /PD Input	T_{PDW}	2	–	–	1/Fclk
Delay from /PD Input to CAL Output	T_{PCR}	–	–	6	1/Fclk
Calibration Cycle Duration	T_{PDF}	–	4096	–	1/fs
Delay from /PD Input to S_{DATA} L	T_{PSF}	–	–	6	1/Fclk
Delay from Completion of Calibration to SDATA Valid	T_{CSV}	–	1	–	1/fs

FIGURE 3b. DF1760 Power Down and Offset Calibration.



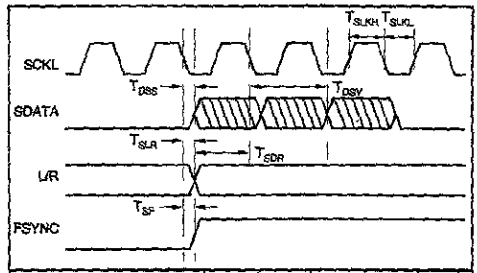
SYSTEM CLOCK: 256fs					
DESCRIPTION	NAME	MIN	TYP	MAX	UNITS
Low Level Duration	T_{CLKL}	31	–	–	ns
High Level Duration	T_{CLKH}	31	–	–	ns
SYSTEM CLOCK: 384fs					
DESCRIPTION	**NAME**	**MIN**	**TYP**	**MAX**	**UNITS**
Low Level Duration	T_{CLKL}	24	–	–	ns
High Level Duration	T_{CLKH}	24	–	–	ns
Rise Time	T_{LH}	–	–	6	ns
Fall Time	T_{HL}	–	–	6	ns

FIGURE 3c. System Clock Timing Requirements of DF1760.



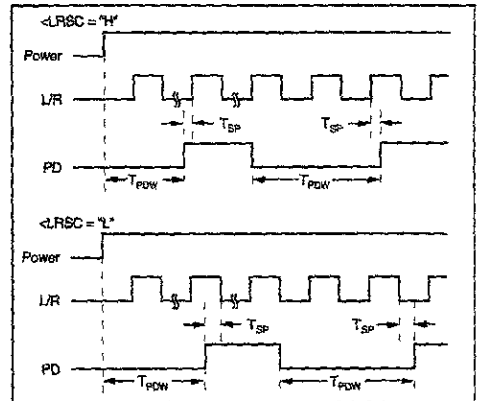
DESCRIPTION	NAME	MIN	TYP	MAX	UNITS
SCLK Frequency	F_{SLK}	–	64fs	–	
SCLK Frequency Duty Cycle		–	50	–	%
FSYNC Frequency	F_{SYNC}	–	2fs	–	
FSYNC Frequency Duty Cycle		–	50	–	%
Delay from SCLK to L/R Edge	T_{SLR}	–20	–	50	ns
Delay from Falling Edge of SCLK to SDATA Valid	T_{DSS}	–	–	50	ns
Delay from SCLK to FSYNC Edge	T_{SF}	–20	–	50	ns
Delay from Rising Edge of SCLK to SDATA Valid	T_{SDR}	100	–	–	ns
Delay from SDATA Valid to Rising Edge of SCLK	T_{DSV}	100	–	–	ns

FIGURE 3d. Output Timing of Master Mode, DF1760.



DESCRIPTION	NAME	MIN	TYP	MAX	UNITS
SCLK Frequency	F_{SLK}	32fs	48fs	64fs	–
Low Duration of FSCLK	T_{SLKL}	100	–	–	ns
High Duration of FSCLK	T_{SLKH}	100	–	–	ns
Delay from SCLK to L/R Edge	T_{SLR}	–70	–	70	ns
Delay from Falling Edge of SCLK to SDATA Valid	T_{DSS}	–	–	50	ns
Delay from SCLK to FSYNC Edge	T_{SF}	–70	–	0	ns
Delay from Rising Edge of SCLK to SDATA Valid	T_{DSV}	100	–	–	ns
Delay from SDATA Valid to Rising Edge of SCLK	T_{SDR}	100	–	–	ns

FIGURE 3e. Timing of Slave Mode, DF1760.



DESCRIPTION	NAME	MIN	TYP	MAX	UNITS[1]	APPLIES TO MODE
Power on to PD ↑	T_{PDW}	2			1/fs	Master/Slave
PD ↑ to L/R ↑ (LRSC = "H")	T_{SP}	–1		+1	1/Fclk	Slave
PD ↑ to L/R ↓ (LRSC = "L")	T_{SP}	–1		+1	1/Fclk	Slave

NOTE: (1) fs: sampling rate, Fclk: system clock frequency

FIGURE 3f. Power On and Mode Reset Timing.

THEORY OF OPERATION

MULTI-BIT ENHANCED NOISE SHAPING

A block diagram of a typical 1-bit delta-sigma modulator is shown in Figure 4.

In Figure 4, the quantizer consists of a single bit which has two possible states, either "0" or "1". The input signal is sampled at a much higher sample rate than the nyquist sampling frequency. The quantizer output data stream is digitally filtered for higher resolution nyquist data. The theoretical SNR is determined by the number of the order of the integrator and the oversampling rate.

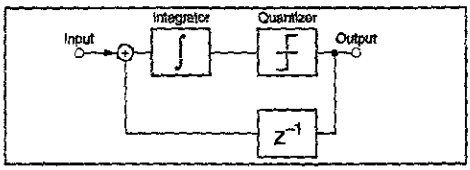


FIGURE 4. Single Stage 1-Bit Delta-Sigma.

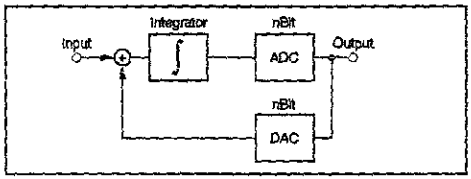


FIGURE 5. Single Stage Multi-bit Delta-Sigma.

There is a practical limit to increasing the numbers of order of the integrator due to an inherent oscillation in the modulator. There is also a limit to increasing the sample rate due to the increase in jitter sensitivity associated with high clock frequencies.

The PCM1760 utilizes a four-bit quantizer instead of the conventional one-bit method. The quantizing noise of a four-bit quantizer is 1/16 of the one-bit version. Using the four-bit quantizer allows for a lesser order number of the integrator and a lower oversampling rate to achieve similar performance to that of a more complex one-bit system.

A block diagram of the PCM1760 modulator is shown in Figure 6. The PCM1760 is a fourth-order integrator that samples at 64x oversampling, and samples left and right channel input signal simultaneously.

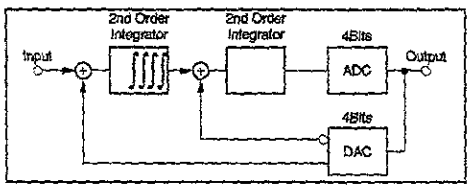


FIGURE 6. Multi-bit Enhanced Noise Shaping.

The DF1760 accepts the four-bit 64fs noise shaped data stream from the PCM1760 and decimates to 1/16 with an initial filter, and then decimates to 1fs 20-bit data using a 4x oversampling filter.

The PCM1760 and DF1760 combination achieves a dynamic range of 108dB and SNR of 110dB even with a single-ended input.

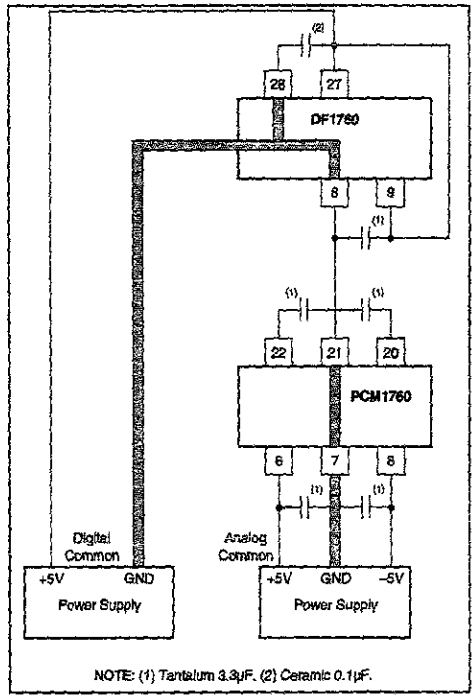


FIGURE 7. Recommended Power Supply Connection and Decoupling.

LAYOUT PRECAUTIONS

Analog common and digital common of the PCM1760 are not connected internally. These should be connected together with the common of the DF1760 as close to the unit as possible, preferably to a large ground plane under the PCM1760.

The use of a separate +5V supply is recommended for the PCM1760 and DF1760, and to connect the common at one point as described above. Low impedance analog and digital commons returns are essential for better performance.

The power supplies should be bypassed with tantalum capacitors as close as possible to the units. See Figure 7 for recommended common connections and power supplies bypassing.

OUTPUT TONE ELIMINATION

When the sampling frequency (fs) is between 40kHz and 50 kHz and the L/R relative offset voltage (ΔVs) is less than or equal to 0.05% of full scale range, the PCM1760 may output a tone similar to an idle tone. This tone is very low and its frequency depends on the input L/R relative offset voltage, ΔVs. This tone never occurs when the sampling frequency (fs) is 32kHz.

To avoid this tone, the offset voltage should be summed using an amplifier, buffer, active low pass filter, etc., to cause the input L/R relative offset voltage (ΔVs) to be greater than 0.05% of full scale range.

It is recommended that:

(A) Sum offset at both L/R channels
Lch: V_{IL} = –20mV ±10%
Rch: V_{IR} = +10mV ±10%

(B) Sum offset at L channel
Lch: V_{IL} = –30mV ±10%
Rch: V_{IR} = ±1mV (by a precircuit)

When FSR = 5V (±2.5V).

Figure 8 shows an application circuit for summing the offset at both L/R channels.

Alternately, Figure 9 shows an application circuit for use when fs = 48kHz which changes the external integrator circuit of the PCM1760.

MODULATOR COMPONENTS AND SAMPLING FREQUENCY

The PCM1760/DF1760 are capable to 30kHz to 50kHz fs sampling frequency by condition with external components value which are shown in Basic Connection Diagram.

The characteristics of the modulator's integrator can be set by external components. The values in the block diagram on page five are recommended for optimized performance. Low leakage, low voltage coefficient capacitors are recommended for integration capacitors.

The tolerance of external components should be better than ±2%.

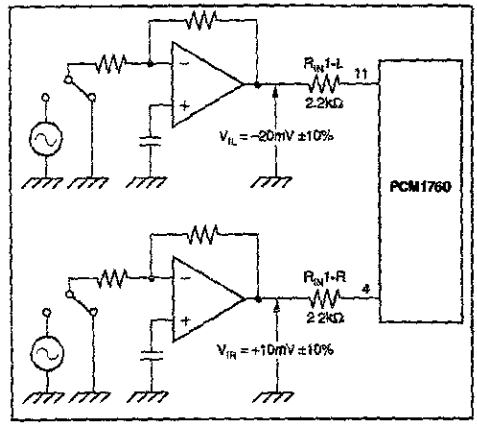


FIGURE 8. Application Example to Eliminate the Tone (offset voltage implementation for both channels).

OFFSET ERROR CALIBRATION

The offset voltage of the PCM1760 and the input stage of the system can be compensated by using the calibration mode of the DF1760. Offset calibration is shown in Figure 10. An optional analog switch is driven by a CAL output of the DF1760. The PD input of the DF1760 is used to initiate the calibration cycle.

ANALOG INPUT AND DIGITAL OUTPUT

Ideal output digital code range for 20-bit resolution is from 80000H (–Full Scale) to 7FFFFH (+Full Scale).

The DF1760, combined with 70000H (±FSR) of the PCM1760, produces a digital output code range at ±FSR input of 90000H (–FSR).

The relationship between analog input and digital output is shown in Table I.

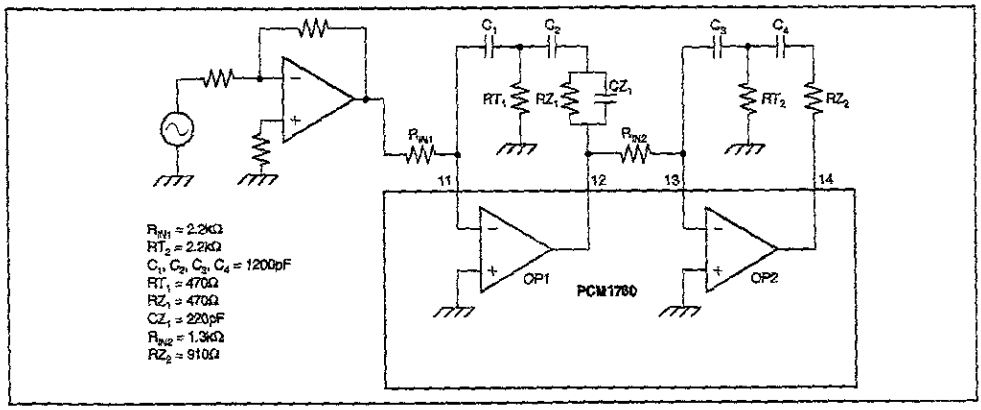


FIGURE 9. Application Example to Eliminate the Tone (alternative modulator's integrator circuit. Only for fs = 48kHz).

ANALOG INPUT	CONDITION	DIGITAL OUTPUT
+2.55V	+Max Input	72000H
+2.50V to +2.55V	Overflow	70000H to 72000H[2]
+2.50V	+FSR	70000H
0V	BPZ (Ideal)	00000H[1]
−2.50V	−FSR	90000H
−2.83V to −2.85V	Overflow	82FFFH to 82000H[2]
−2.85V	−Max Input	82000H

NOTES: (1) Incase of BPZ Error = 0. (2) Overflow detection level is over 70000H or under 82FFFH of digital output code.

TABLE I. Output Codes.

POWER SUPPLY SEQUENCING

The PCM1760 requires $\pm V_{CC}$ and $\pm V_{DD}$ power supplies. To avoid any possibility of latch-up, the $\pm V_{CC}$ and $\pm V_{DD}$ power should all be applied simultaneously or the $+V_{CC}$ and $+V_{DD}$ applied first followed by $-V_{CC}$ and $-V_{DD}$.

POWER-ON RESET AND MODE RESET

The timing requirements for POWER-ON RESET and MODE RESET are shown in Figure 3f. The DF1760 requires POWER-ON RESET when power is applied or restored. MODE RESET is required when any of the following has been changed: system clock, master/slave mode, output data format, L/R clock, calibration after POWER-ON in slave mode.

This reset should be done by holding the /PD input (pin 21) low for more than 2/fs. Suggested reset circuits are given in Figures 11, 12 and 13.

CLOCK INPUT

After power is applied to the DF1760, the system clock should be provided continuously. The DF1760 employs a dynamic logic architecture.

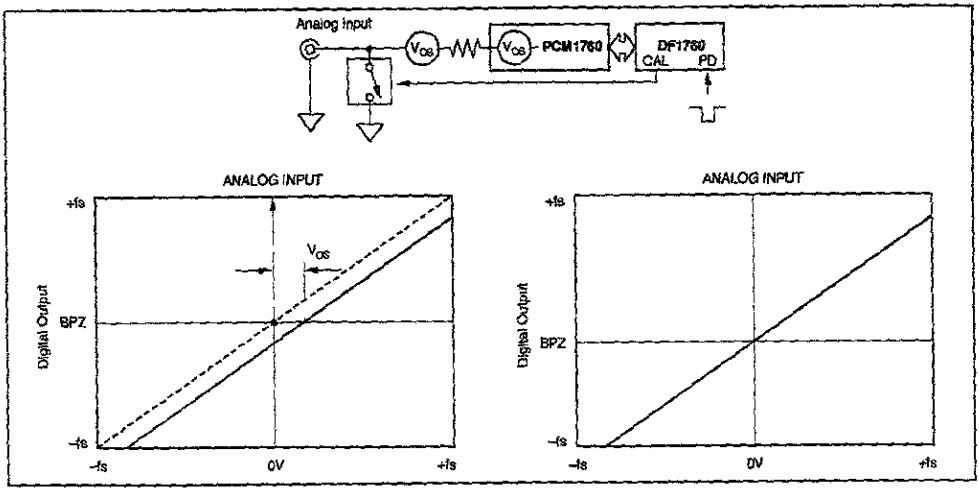


FIGURE 10. Illustration of Offset Calibration.

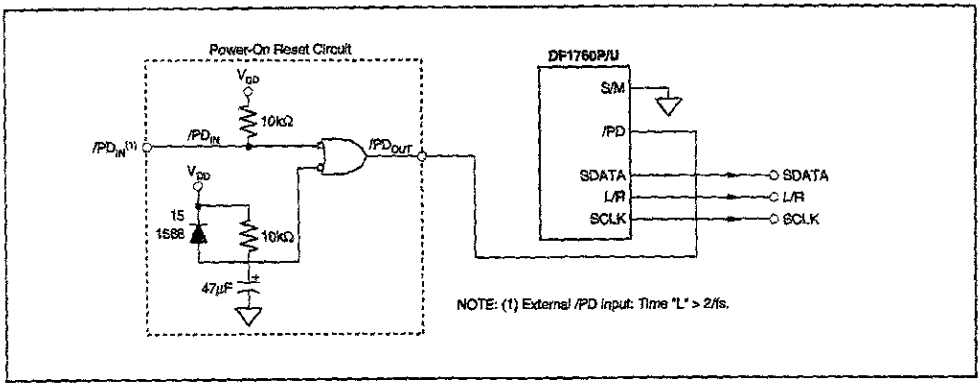


FIGURE 11. Master Mode Reset Circuit.

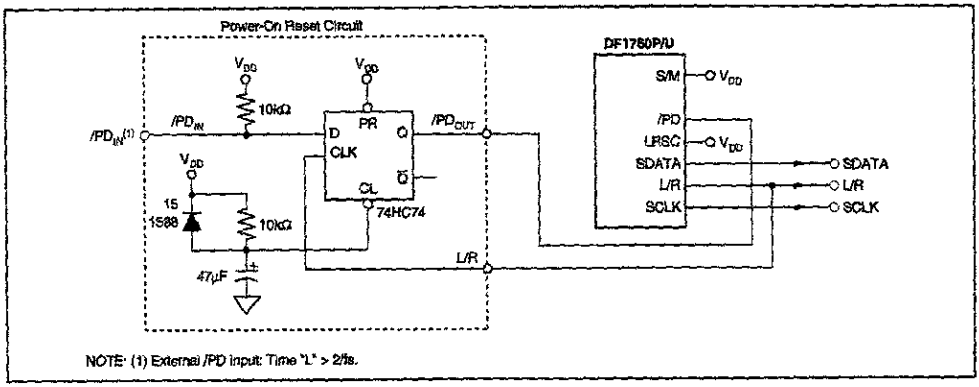


FIGURE 12. Slave Mode Reset Circuit, (LRSC = H).

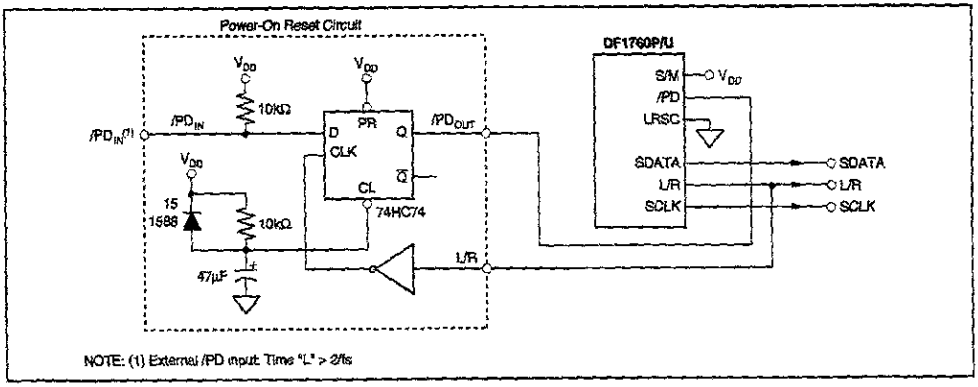


FIGURE 13. Slave Mode Reset Circuit, (LRSC = L).

TIMING CHARACTERISTICS

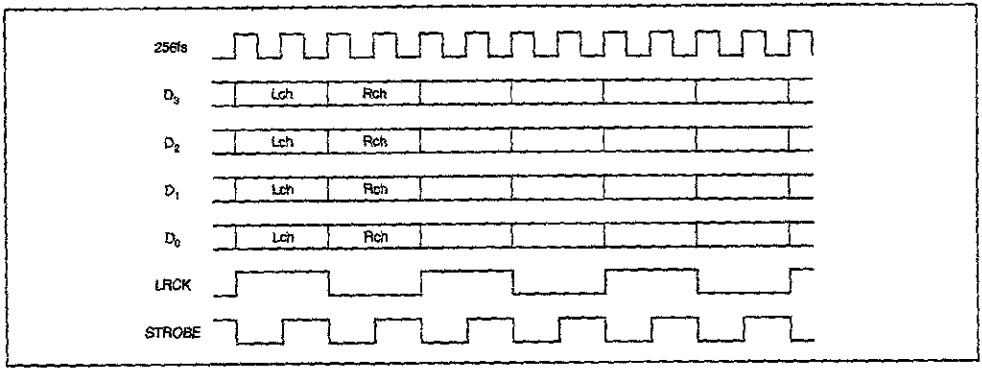


FIGURE 14. Input and Output Format of the DF1760 and PCM1760.

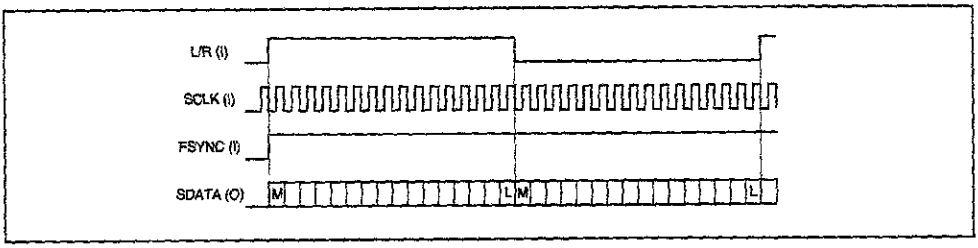


FIGURE 15a. Slave Mode and SCLK = 32fs. (Output format of the DF1760).

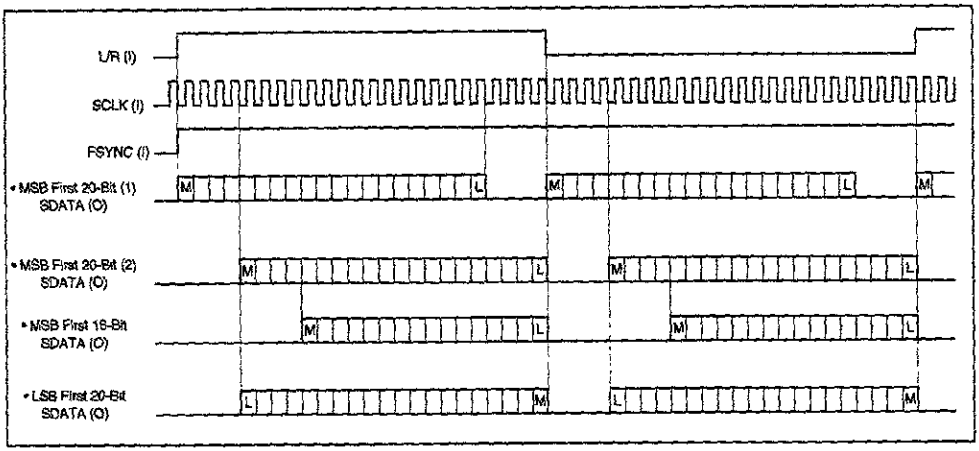


FIGURE 15b. Slave Mode and SCLK = 48fs.

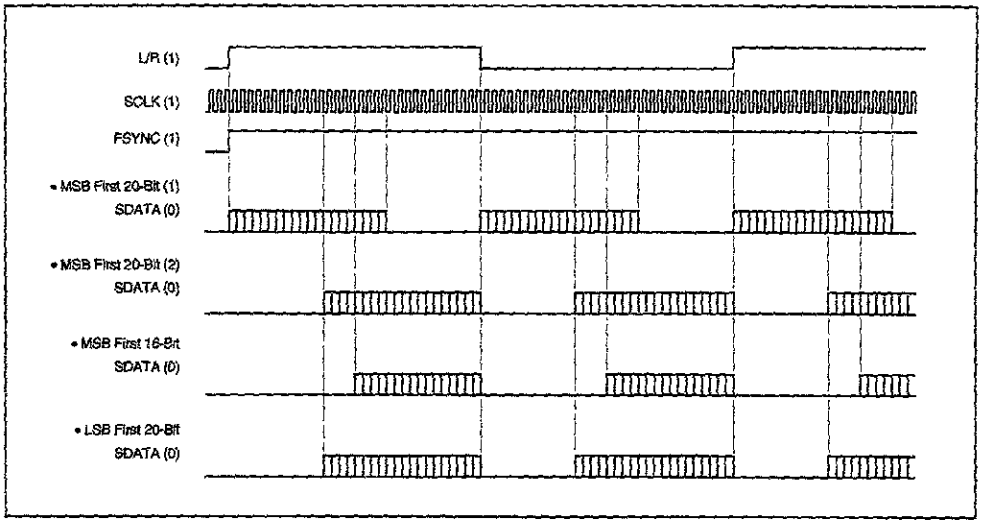


FIGURE 15c. Slave Mode and SCLK = 64fs.

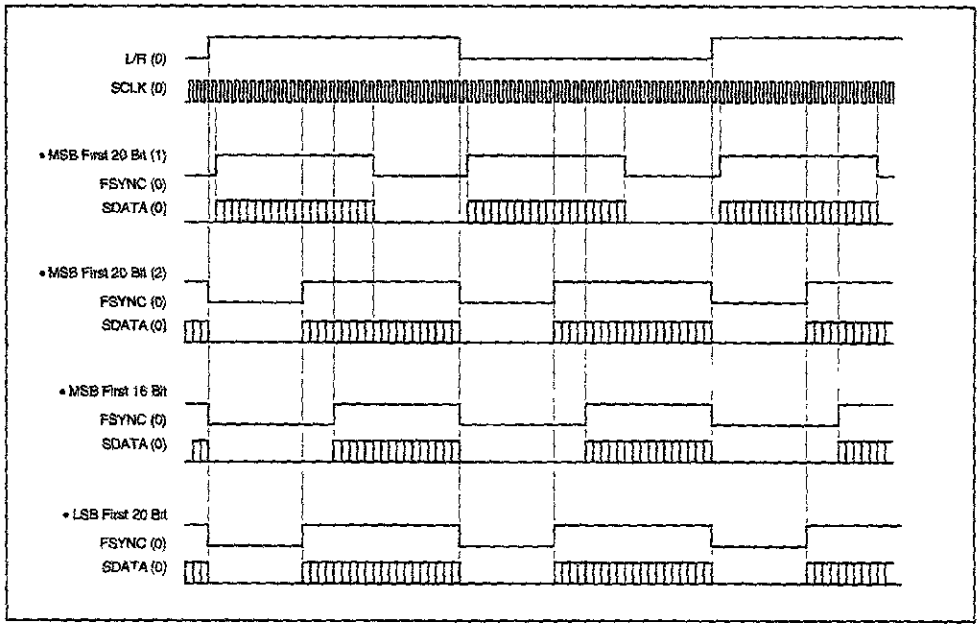


FIGURE 15d. Master Mode.

Integrated Device Technology, Inc.

CMOS STATIC RAM 1 MEG (64K x 16-BIT)

PRELIMINARY IDT71016

FEATURES:

- 64K x 16 advanced high-speed CMOS Static RAM
- Equal access and cycle times
  - — Commercial: 12/15/20/25ns
- One Chip Select plus one Output Enable pin
- Bidirectional data inputs and outputs directly TTL-compatible
- Low power consumption via chip deselect
- Upper and Lower Byte Enable Pins
- Available in 44-pin Plastic SOJ package and 44-pin TSOP package

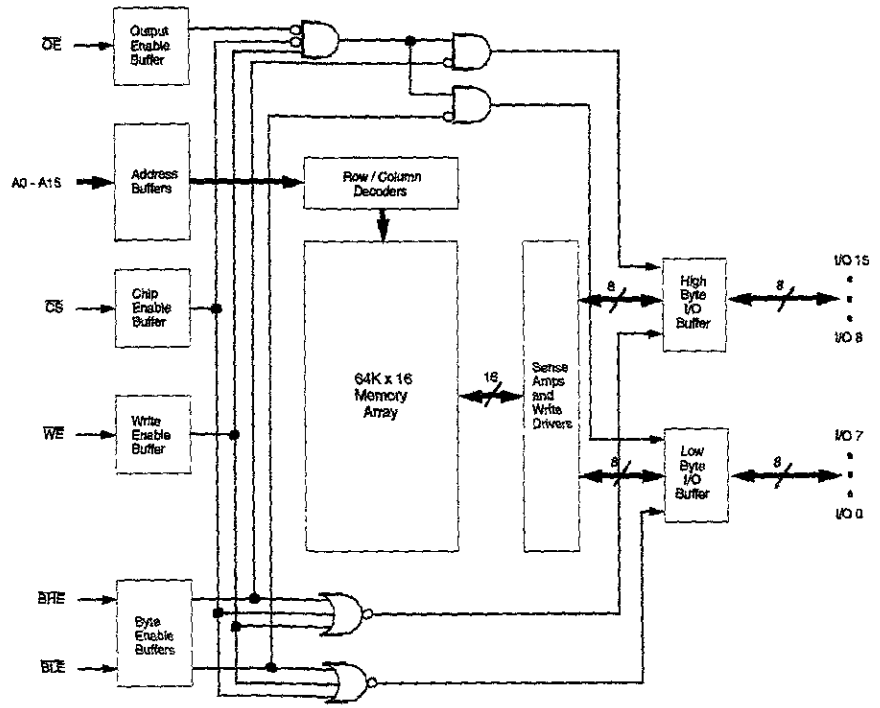
DESCRIPTION:

The IDT71016 is a 1,048,576-bit high-speed Static RAM organized as 64K x 16. It is fabricated using IDT's high-performance, high-reliability CMOS technology. This state-of-the-art technology, combined with innovative circuit design techniques, provides a cost-effective solution for high-speed memory needs.

The IDT71016 has an output enable pin which operates as fast as 7ns, with address access times as fast as 12ns. All bidirectional inputs and outputs of the IDT71016 are TTL-compatible and operation is from a single 5V supply. Fully static asynchronous circuitry is used, requiring no clocks or refresh for operation.

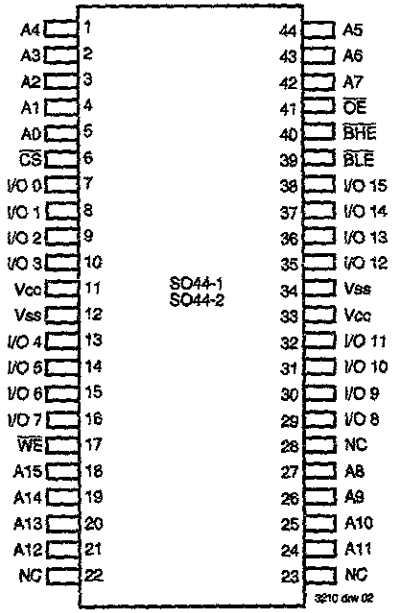
The IDT71016 is packaged in a JEDEC standard 44-pin Plastic SOJ and 44-pin TSOP Type II.

FUNCTIONAL BLOCK DIAGRAM



3210 drw 01

PIN CONFIGURATIONS



SOJ/TSOP
TOP VIEW

PIN DESCRIPTIONS

$A_0 - A_{15}$	Address Inputs	Input
$\overline{CS}$	Chip Select	Input
$\overline{WE}$	Write Enable	Input
$\overline{OE}$	Output Enable	Input
$\overline{BHE}$	High Byte Enable	Input
$\overline{BLE}$	Low Byte Enable	Input
$I/O_0 - I/O_{15}$	Data Input/Output	I/O
V_{CC}	5.0V Power	Pwr
V_{SS}	Ground	Gnd

3210 tbl 01

TRUTH TABLE[1]

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	$\overline{BLE}$	$\overline{BHE}$	I/O_0-I/O_7	I/O_8-I/O_{15}	Function
H	X	X	X	X	High-Z	High-Z	Deselected - Standby
L	L	H	L	H	$DATA_{OUT}$	High-Z	Low Byte Read
L	L	H	H	L	High-Z	$DATA_{OUT}$	High Byte Read
L	L	H	L	L	$DATA_{OUT}$	$DATA_{OUT}$	Word Read
L	X	L	L	L	$DATA_{IN}$	$DATA_{IN}$	Word Write
L	X	L	L	H	$DATA_{IN}$	High-Z	Low Byte Write
L	X	L	H	L	High-Z	$DATA_{IN}$	High Byte Write
L	H	H	X	X	High-Z	High-Z	Outputs Disabled
L	X	X	H	H	High-Z	High-Z	Outputs Disabled

3210 tbl 02

NOTE:
1. H = V_{IH}, L = V_{IL}, X = Don't care.

RECOMMENDED DC OPERATING CONDITIONS

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{CC}	Supply Voltage	4.5	5.0	5.5	V
GND	Supply Voltage	0	0	0	V
V_{IH}	Input High Voltage	2.2	—	V_{CC}+0.5	V
V_{IL}	Input Low Voltage	–0.5[1]	—	0.8	V

NOTE: 3210 tbl 03

1. V_{IL} (min.) = –1.5V for pulse width less than tRC/2, once per cycle.

ABSOLUTE MAXIMUM RATINGS[1]

Symbol	Rating	Com'l.	Unit
V_{TERM}[2]	Terminal Voltage with Respect to GND	–0.5 to +7.0	V
T_A	OperatingTemperature	0 to +70	°C
T_{BIAS}	Temperature Under Bias	–55 to +125	°C
T_{STG}	Storage Temperature	–55 to +125	°C
P_T	Power Dissipation	1.25	W
I_{OUT}	DC Output Current	50	mA

NOTES: 3210 tbl 04

1. Stresses greater than those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
2. V_{TERM} must not exceed V_{CC} + 0.5V.

CAPACITANCE

(T_A = +25°C, f = 1.0MHz, SOJ package)

Symbol	Parameter[1]	Conditions	Max.	Unit
C_{IN}	Input Capacitance	V_{IN} = 3dV	6	pF
$C_{I/O}$	I/O Capacitance	V_{OUT} = 3dV	7	pF

NOTE: 3210 tbl 05

1. This parameter is guaranteed by device characterization, but not production tested.

DC ELECTRICAL CHARACTERISTICS

V_{CC} = 5.0V ± 10%, Commercial Temperature Range

Symbol	Parameter	Test Condition	IDT71016		Unit
			Min.	Max.	
$\|I_{LI}\|$	Input Leakage Current	V_{CC} = Max., V_{IN} = GND to V_{CC}	—	5	µA
$\|I_{LO}\|$	Output Leakage Current	V_{CC} = Max., $\overline{CS}$ = V_{IH}, V_{OUT} = GND to V_{CC}	—	5	µA
V_{OL}	Output Low Voltage	I_{OL} = 8mA, V_{CC} = Min.	—	0.4	V
V_{OH}	Output High Voltage	I_{OH} = –4mA, V_{CC} = Min.	2.4	—	V

3210 tbl 06

DC ELECTRICAL CHARACTERISTICS[1]

(V_{CC} = 5.0V ± 10%, V_{LC} = 0.2V, V_{HC} = V_{CC}–0.2V)

Symbol	Parameter	71016S12[3]		71016S15		71016S20		71016S25		Unit
		Com'l.	Mil.	Com'l.	Mil.	Com'l.	Mil.	Com'l.	Mil.	
I_{CC}	Dynamic Operating Current $\overline{CS} \le V_{IL}$, Outputs Open, V_{CC} = Max., f = f_{MAX}[2]	210	—	180	—	170	—	160	—	mA
I_{SB}	Standby Power Supply Current (TTL Level) $\overline{CS} \ge V_{IH}$, Outputs Open, V_{CC} = Max., f = f_{MAX}[2]	60	—	45	—	40	—	35	—	mA
I_{SB1}	Standby Power Supply Current (CMOS Level) $\overline{CS} \ge V_{HC}$, Outputs Open, V_{CC} = Max., f = 0[2] $V_{IN} \le V_{LC}$ or $V_{IN} \ge V_{HC}$	10	—	10	—	10	—	10	—	mA

NOTES: 3210 tbl 07

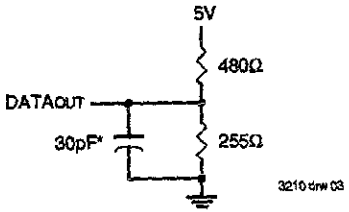
1. All values are maximum guaranteed values.
2. f_{MAX} = 1/t_{RC} (all address inputs are cycling at f_{MAX}); f = 0 means no address input lines are changing.
3. 12ns specification is preliminary.

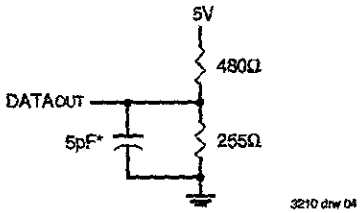
AC TEST CONDITIONS

Input Pulse Levels	GND to 3.0V
Input Rise/Fall Times	1.5ns
Input Timing Reference Levels	1.5V
Output Reference Levels	1.5V
AC Test Load	See Figures 1, 2, and 3

3210 tbl 07

AC TEST LOADS





*Including jig and scope capacitance.

Figure 1. AC Test Load

Figure 2. AC Test Load
(for t_{OLZ}, t_{OLZ}, t_{CHZ}, t_{OHZ}, t_{OW}, and t_{WHZ})

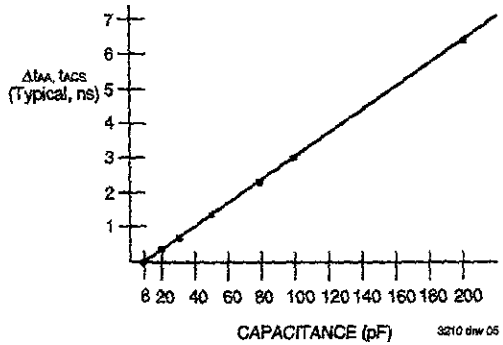


Figure 3. Output Capacitive Derating

AC ELECTRICAL CHARACTERISTICS (Vcc = 5.0V ± 10%, Commercial Temperature Range)

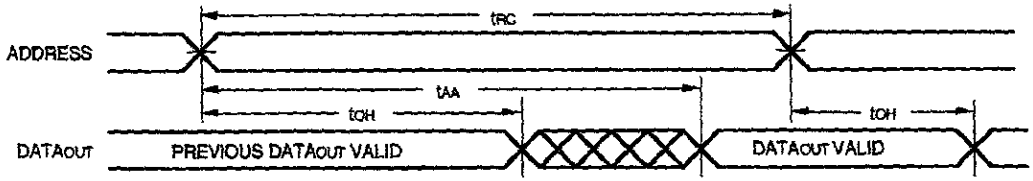
Symbol	Parameter	71016S12[2]		71016S15		71016S20		71016S25		Units
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
Read Cycle										
t_{RC}	Read Cycle Time	12	—	15	—	20	—	25	—	ns
t_{AA}	Address Access Time	—	12	—	15	—	20	—	25	ns
t_{ACS}	Chip Select Access Time	—	12	—	15	—	20	—	25	ns
t_{CLZ}[1]	Chip Select Low to Output in Low-Z	4	—	5	—	5	—	5	—	ns
t_{CHZ}[1]	Chip Select High to Output in High-Z	—	6	—	6	—	8	—	8	ns
t_{OE}	Output Enable Low to Output Valid	—	7	—	8	—	10	—	12	ns
t_{OLZ}[1]	Output Enable Low to Output in Low-Z	0	—	0	—	0	—	0	—	ns
t_{OHZ}[1]	Output Enable High to Output in High-Z	—	6	—	6	—	8	—	8	ns
t_{OH}	Output Hold from Address Change	4	—	4	—	5	—	5	—	ns
t_{BE}	Byte Enable Low to Output Valid	—	7	—	8	—	10	—	12	ns
t_{BLZ}[1]	Byte Enable Low to Output in Low-Z	0	—	0	—	0	—	0	—	ns
t_{BHZ}[1]	Byte Enable High to Output in High-Z	—	6	—	6	—	8	—	8	ns
Write Cycle										
t_{WC}	Write Cycle Time	12	—	15	—	20	—	25	—	ns
t_{AW}	Address Valid to End of Write	9	—	10	—	12	—	14	—	ns
t_{CW}	Chip Select Low to End of Write	9	—	10	—	12	—	14	—	ns
t_{BW}	Byte Enable Low to End of Write	9	—	10	—	12	—	14	—	ns
t_{AS}	Address Set-up Time	0	—	0	—	0	—	0	—	ns
t_{WR}	Address Hold from End of Write	0	—	0	—	0	—	0	—	ns
t_{WP}	Write Pulse Width	9	—	10	—	12	—	14	—	ns
t_{DW}	Data Valid to End of Write	7	—	8	—	10	—	10	—	ns
t_{DH}	Data Hold Time	0	—	0	—	0	—	0	—	ns
t_{OW}[1]	Write Enable High to Output in Low-Z	1	—	1	—	1	—	1	—	ns
t_{WHZ}[1]	Write Enable Low to Output in High-Z	—	6	—	6	—	8	—	8	ns

NOTE:

3210 tbl 08

1. This parameter is guaranteed with the AC Load (Figure 2) by device characterization, but is not production tested.
2. 12ns specification is preliminary.

TIMING WAVEFORM OF READ CYCLE NO. 1[1,2,3]

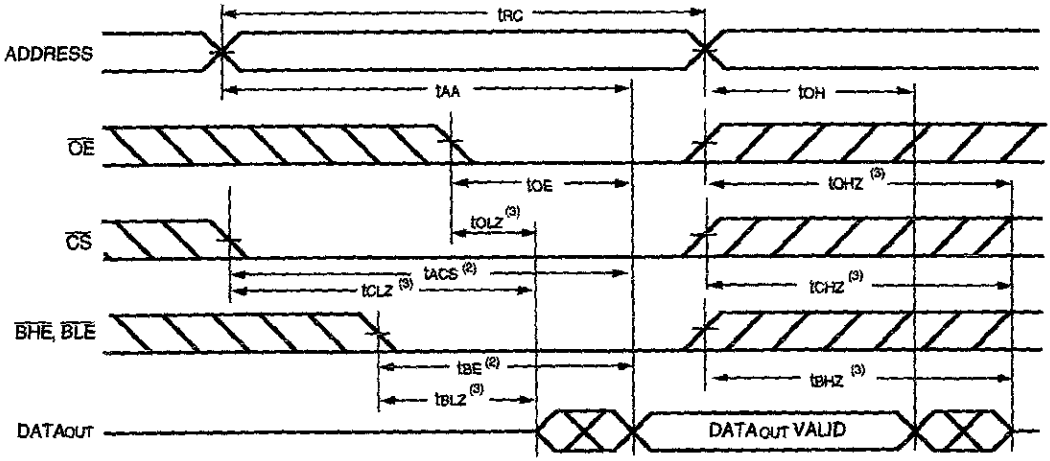


3210 drw 06

NOTES:

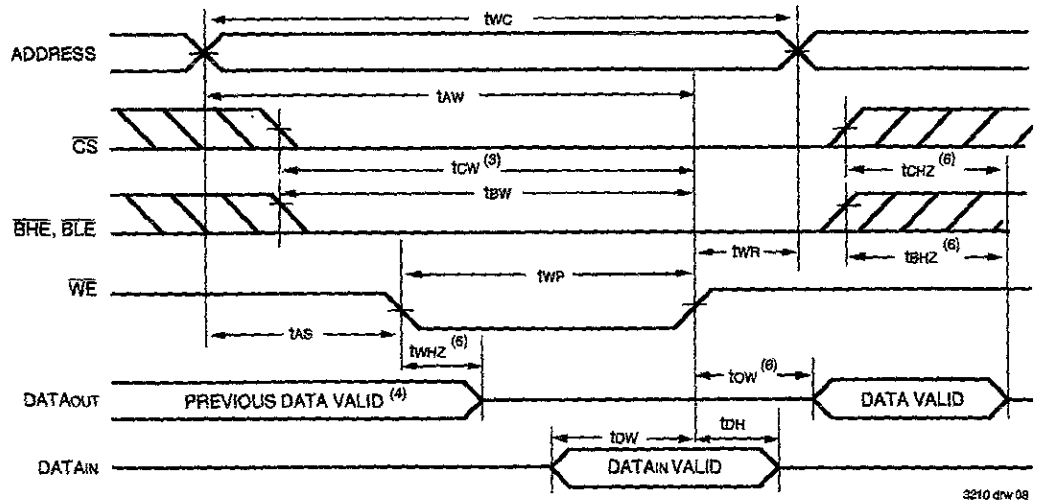
1. $\overline{WE}$ is HIGH for Read Cycle.
2. Device is continuously selected, $\overline{CS}$ is LOW.
3. $\overline{OE}$, $\overline{BHE}$, and $\overline{BLE}$ are LOW.

TIMING WAVEFORM OF READ CYCLE NO. 2(1)



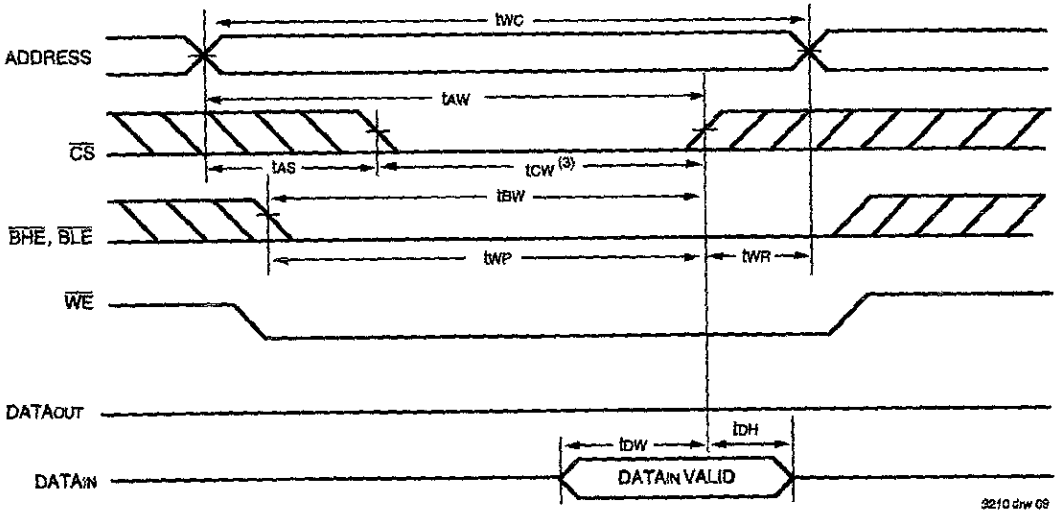
NOTES:
1. $\overline{WE}$ is HIGH for Read Cycle.
2. Address must be valid prior to or coincident with the later of $\overline{CS}$, $\overline{BHE}$, or $\overline{BLE}$ transition LOW; otherwise t_{AA} is the limiting parameter.
3. Transition is measured ±200mV from steady state.

TIMING WAVEFORM OF WRITE CYCLE NO. 1 ($\overline{WE}$ CONTROLLED TIMING)(1,2,3,5)

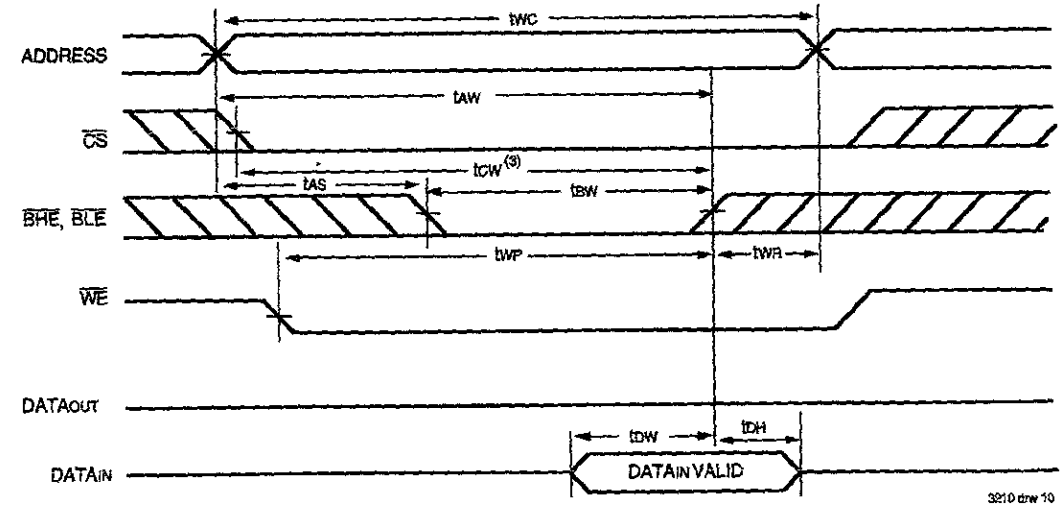


NOTES:
1. $\overline{WE}$, $\overline{BHE}$ and $\overline{BLE}$, or $\overline{CS}$ must be HIGH during all address transitions.
2. A write occurs during the overlap of a LOW $\overline{CS}$, LOW $\overline{BHE}$ or $\overline{BLE}$, and a LOW $\overline{WE}$.
3. $\overline{OE}$ is continuously HIGH. If during a $\overline{WE}$ controlled write cycle $\overline{OE}$ is LOW, t_{WP} must be greater than or equal to $t_{WHZ} + t_{DW}$ to allow the I/O drivers to turn off and data to be placed on the bus for the required t_{DW}. If $\overline{OE}$ is HIGH during a $\overline{WE}$ controlled write cycle, this requirement does not apply and the minimum write pulse is as short as the specified t_{WP}.
4. During this period, I/O pins are in the output state, and input signals must not be applied.
5. If the $\overline{CS}$ LOW or $\overline{BHE}$ and $\overline{BLE}$ LOW transition occurs simultaneously with or after the $\overline{WE}$ LOW transition, the outputs remain in a high-impedance state.
6. Transition is measured ±200mV from steady state.

TIMING WAVEFORM OF WRITE CYCLE NO. 2 ($\overline{CS}$ CONTROLLED TIMING)[1,2,5]



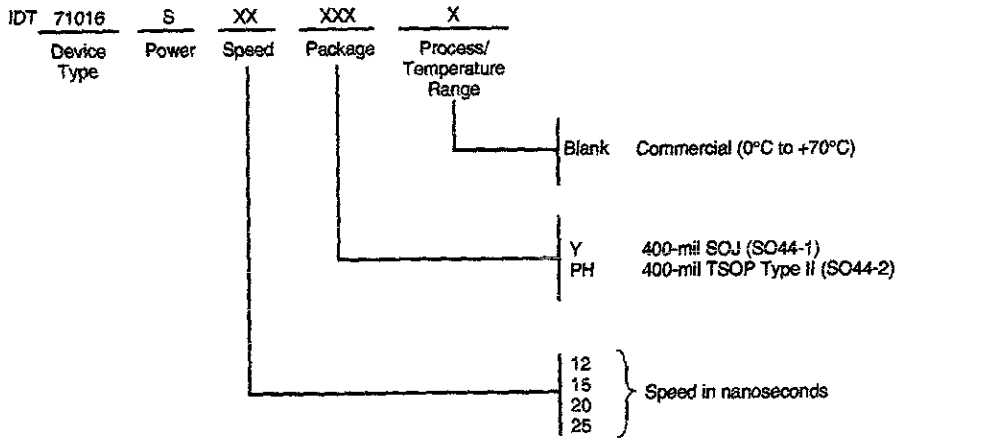
TIMING WAVEFORM OF WRITE CYCLE NO. 3 ($\overline{BHE}$, $\overline{BLE}$ CONTROLLED TIMING)[1,2,5]



NOTES:

1. $\overline{WE}$, $\overline{BHE}$ and $\overline{BLE}$, or $\overline{CS}$ must be HIGH during all address transitions.
2. A write occurs during the overlap of a LOW $\overline{CS}$, LOW $\overline{BHE}$ or $\overline{BLE}$, and a LOW $\overline{WE}$.
3. $\overline{OE}$ is continuously HIGH. If during a $\overline{WE}$ controlled write cycle $\overline{OE}$ is LOW, t_{WP} must be greater than or equal to $t_{WHZ} + t_{DW}$ to allow the I/O drivers to turn off and data to be placed on the bus for the required t_{DW}. If $\overline{OE}$ is HIGH during a $\overline{WE}$ controlled write cycle, this requirement does not apply and the minimum write pulse is as short as the specified t_{WP}.
4. During this period, I/O pins are in the output state, and input signals must not be applied.
5. If the $\overline{CS}$ LOW or $\overline{BHE}$ and $\overline{BLE}$ LOW transition occurs simultaneously with or after the $\overline{WE}$ LOW transition, the outputs remain in a high-impedance state.
6. Transition is measured ±200mV from steady state.

ORDERING INFORMATION



3210 drw 11