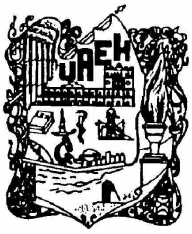

Universidad Autónoma del Estado de Hidalgo

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

PROCESAMIENTO DE AUDIO EN TIEMPO REAL CON EL DSP TMS320C6711

TESIS PARA OBTENER EL TITULO DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

PRESENTA:

EDISAN AGMAR GUERRERO CALVA

ASESOR:

ING. MARÍA DEL CARMEN TAMAYO CIGARROA

Pachuca de Soto Hidalgo, México. 2 de febrero de 2007

Agradecimientos

Como dice en la Biblia dad gracias a Dios en todo. Principalmente le doy gracias a Dios por su gracia, y también le doy gracias porque cada cosa que he logrado a sido por su gran misericordia, porque se que sin su cariño, protección y fuerza hubiera caído vencido frente a todos los problemas que tuve para llegar a este momento. Muchas gracias Dios se que siempre caminaras a mi lado.

Doy gracias a mis Padres, porque siempre soñaron con este momento y me fueron preparando y apoyando para que yo pudiera lograrlo, hoy todos sus desvelos, preocupaciones, regaños y principalmente su amor tendrán una pequeña recompensa. Dios quiera y pueda darles muchas mas alegrías. Muchas gracias se que siempre estarán en mi corazón.

Les doy gracias a mis hermanos, y doy gracias a Dios por haberlos traído a mi vida para que yo aprendiera a amar a alguien más que a mi mismo Muchas gracias se que siempre nos cuidaremos mutuamente.

Les doy gracias a mis familiares y amigos. A todos les quiero decir gracias, porque con cada mirada de cariño, cada abrazo y cada palabra que me dieron, le fueron poniendo color a mi vida. Muchas gracias se que siempre recordare los buenos momentos juntos.

Y por ultimo pero no por eso menos importante, Le doy gracias a mi asesora, la persona que me dio la mano, me dio su tiempo y me ayudo a terminar un sueño. Muchas gracias por todo.

Índice general

Índice de figuras III

1. Introducción 1
- 1.1. Planteamiento del problema . . . 4
- 1.2. Antecedentes . . . 4
- 1.3. Objetivos . . . 5
- 1.4. Contribución de la tesis . . . 5
- 1.5. Estructura de la tesis . . . 6

2. Señales 7
- 2.1. Señal y clasificación de señales . . . 7
- 2.2. Señal de audio . . . 25
- 2.3. Conversión analógica-digital y digital-analógica . . . 26
- 2.4. Tiempo real . . . 28

3. DSP TMS320C6711 31
- 3.1. Hardware . . . 31
- 3.2. Software . . . 32

4. Procesamiento de audio en el DSP 35
- 4.1. Configuración setup . . . 37
- 4.2. Depurando y probando con las herramientas de análisis en tiempo real de DSP/BIOS . . . 42
- 4.3. Controlando el periodo . . . 44
- 4.4. Resultados . . . 48

5. Conclusiones y trabajos futuros 51

Glosario 53

Bibliografía 57

Índice de figuras

2.1. Representación de una imagen por coordenadas espaciales. 8
2.2. Señal en tiempo continuo y tiempo discreto. 9
2.3. Señal de valor continuo y valor discreto. 10
2.4. Señal periódica. 11
2.5. Señal par e impar. 12
2.6. Señal exponencial. 13
2.7. Señal exponencial real (a) creciente y (b) decreciente. 14
2.8. Señales exponenciales complejas imaginarias. 15
2.9. Señales sinusoidales en tiempo continuo. 16
2.10. Señal sinusoidal en tiempo discreto. 18
2.11. Secuencias sinusoidales discretas por diferentes frecuencias. 21
2.12. Impulso unitario. 23
2.13. Escalón unidad. 23
2.14. Rampa unidad. 24
2.15. Señal de audio. 25

3.1. Diagrama de la tarjeta DSP 6711. 32
3.2. Diagrama del programa CCS. 33

4.1. Procesamiento de audio con el DSP. 35
4.2. Diagrama del ejemplo de audio. 36
4.3. Propiedades de audioSWI. 38
4.4. Propiedades de DSS_rxPipe. 39
4.5. Propiedades de DSS_txPipe. 40
4.6. DSS_rxPrime y DSS_txPrime. 41
4.7. Propiedades de HWI_init11. 41
4.8. Propiedades del RTA. 42
4.9. Grafica de ejecución. 43
4.10. Propiedades de loadPRD. 44
4.11. Gráfica de entrada del audio. 48
4.12. Gráfica de procesamiento de audio. 48

Acrónimos

Notación	Significado	Traducción
ADC	Analog to Digital Converter	Convertidor analógico a digital
API	Application Programming Interface	Interfaz programable para la aplicación
CCS	Code Composer Studio	Laboratorio de prácticas de código
CD-ROM	Compact Disk Read only memory	Memoria de sólo lectura en disco compacto
CSL	Chip support Library	Librería de soporte del chip
DAC	Digital to Analog Converter	Convertidor digital a analógico
DRR	Data Receive Register	Registro receptor de datos
DSP	Digital Signal Processing	Procesador digital de señales
DSP/BIOS	DSP Basic Input/Output System	Sistema básico de entrada/ salida del DSP
HWI	Hardware Interrupt	Módulo de interrupciones de hardware
IDL	Idle	Módulo de funciones estáticas
IFR	Interrupt Flag Register	Interruptor de registro de bandera
IMR	Interrupt Mask Register	Interruptor de registro de mascara
I/O	Input/Output	Entrada/Salida
LOG	Log	Módulo de eventos
OCR	Optic Character Reconigtion	Reconocimiento óptico de caracteres
PIP	Pipes	Módulo de canalizaciones
PRD	Periodics	Módulo de funciones periódicas
RTDX	Real-Time Data Exchange	Intercambio de datos en tiempo real
STS	Statistic	Módulo de estadísticas
SWI	Software Interrupts	Módulo de interrupciones de Software
SYS	System	Módulo de dispositivos del sistema
TRC	Trace	Módulo de trazo
TSK	Task	Módulo de tareas

Capítulo 1

Introducción

El procesado digital de señales se ha desarrollado muy rápidamente en los últimos treinta años. Esto es resultado del avance significativo en la tecnología digital de computadoras y en la fabricación de circuitos integrados.

Estos circuitos digitales, que no son muy caros y si relativamente rápidos, han hecho posible construir sistemas digitales altamente sofisticados capaces de llevar a cabo varias funciones y tareas de proceso digital de señales complejas, las cuales resultarían muy difíciles y a veces incluso caras de abordar con circuitos analógicos o con sistemas de procesamiento analógico de señal.

El análisis de sistemas digitales de señales es la solución apropiada para todos los problemas de procesamiento de la señal. El hardware para llevar a cabo dicho análisis digital permite tener operaciones programables, las cuales a través del software se pueden modificar para que realicen diferentes funciones del procesamiento digital.

Algunos de los campos que se han revolucionado con el procesamiento digital de señales aparecen en el siguiente diagrama[3].

Astronomía	*Mejora de imágenes espaciales
	*Compresión de datos
Medicina	*Diagnóstico apartir de imágenes
	*Análisis de electrocardiogramas
	*Almacenamiento y recuperación de datos de imágenes médicas
Actividad Comercial	*Compresión de imágenes y sonido para presentaciones multimedia
	*Efectos especiales
	*Videoconferencias
Telefonía	*Compresion de datos y voz
FuerzasArmadas	*Radar
Industria	*Búsqueda mineral y petrolera
Ciencia	*Adquisición de datos

Quizás una mejor manera de ubicar el área para alguien ajeno a ella, es la de mencionar las aplicaciones y los frutos que ha logrado ésta disciplina en diferentes campos. Mencionaremos cinco contextos en los cuales se pueden encontrar éstos:

Un primer conjunto de aplicaciones lo presenta el problema de diseñar un sistema para procesar señales y predecir su comportamiento futuro. El pronóstico económico presenta un ejemplo común de esta situación, por ejemplo, muchos programas de computadora han sido creados para realizar análisis detallados de los promedios del índice bursátil (y de otras señales económicas) y realizar predicciones en base a la historia de estas señales. Si bien, la mayor parte de estas señales no son totalmente predecibles, es un hecho importante el que su comportamiento futuro si se puede predecir, al menos aproximadamente y dependiendo de la técnica de análisis utilizada en la predicción.

El segundo conjunto de aplicaciones es la restauración de señales que han sido degradadas de alguna manera. Por ejemplo, la restauración de grabaciones de audio antiguas. Otro ejemplo de éste tipo de procesamiento se tiene cuando se quiere depurar una señal de audio que se recibe con ruido de fondo, por ejemplo, en la transmisión de un piloto a la torre de control de tráfico aéreo, la voz del piloto estará contaminada con el ruido de fondo de la cabina del avión, en éste caso se debe diseñar un sistema para eliminar el ruido de fondo y resaltar la voz del piloto.[1]

En el tercer conjunto de aplicaciones muy similar al anterior, es el de procesar señales de manera de *mejorar*, o resaltar alguna característica de ellas. El procesamiento de imágenes provenientes de satélite es un caso típico. Así, además de la restauración que necesariamente se practicará sobre la imagen para compensar errores debido a limitaciones del equipo, efectos atmosféricos y hasta errores en la transmisión, es posible procesar la señal de manera que se realcen características deseadas de la imagen, tales como: cauces de ríos o lagos, regiones cultivadas, entre otras. o bien, se puede realizar la amplificación de una porción deseada de la imagen, o la *traslación* de la imagen infrarroja a luz visible (para visión nocturna), por mencionar algunas.

Un conjunto de aplicaciones que ha tenido un gran desarrollo en los últimos años ha sido el reconocimiento de patrones. Éste se refiere al procesamiento de un conjunto de señales de la misma naturaleza con el fin de clasificarlas o de *identificar*, cada una de ellas dentro de una categorización dada. Así, se puede mencionar en éste campo, el reconocimiento de voz, la clasificación de piezas mecánicas en una línea de producción por un brazo mecánico, el reconocimiento óptico de caracteres (OCR), el reconocimiento de huellas digitales, de firmas, de rostros o de manos, etc[1].

Otra clase importante de aplicaciones es cuando se desea modificar las características de comportamiento de un sistema dado, normalmente a través de la manipulación de señales de entrada específicas, o combinando el sistema dado con otros sistemas. Éste es el campo denominado control automático. Por ejemplo, un área referida normalmente como control de procesos, la cual se refiere al control de plantas químicas. En ésta clase de aplicaciones, un conjunto de sensores miden las señales físicas como temperatura, humedad, concentraciones químicas, etc. dichas señales son procesadas por un sistema

encargado de manipular las señales de control tales como flujo de combustible o agua de enfriamiento, dosificación de sustancias, etc. para regular el proceso químico en marcha.

Es importante mencionar otras aplicaciones que han recibido gran impulso por el desarrollo del procesamiento digital de señales, tal es el campo de las comunicaciones electrónicas como en la modulación de señales, transmisión y recepción en AM y FM, microondas, comunicación por fibra óptica, etc. Y en el campo de la síntesis de señales como: sintetizadores musicales, síntesis de voz, etc.

Las diferentes áreas o campos que han realizado aplicaciones de este tipo, aprovecha cada una de las ventajas que proporciona el procesamiento digital de señales, las cuales se describen a continuación.

Ventajas del procesado digital de señales

Económico: La tecnología de los circuitos integrados permite la fabricación de sistemas digitales potentes, rápidos y baratos. Actualmente, existen sistemas digitales que realizan tareas procesado de señal que antes eran caras y/o difíciles de realizar con sistemas de procesado analógico.

Flexibilidad: Un sistema digital programable permite alterar la funcionalidad del sistema sin necesidad de alterar el hardware. Para ello, simplemente es necesario modificar el software de aplicación. Por el contrario, para reconfigurar un sistema analógico es necesario rediseñar el hardware y después comprobar y verificar su correcto funcionamiento.

Estabilidad y repetibilidad: En los sistemas analógicos, las tolerancias de los componentes hacen difícil al diseñador el control del comportamiento de un sistema. Por ejemplo, una pequeña variación en el valor de una resistencia puede convertir en inestable el sistema. Además, estos componentes suelen ser mucho más sensibles a cambios de las condiciones externas, como por ejemplo la temperatura. A este respecto, los sistemas digitales suelen presentar una mayor estabilidad.

Almacenamiento: Las señales digitales se pueden almacenar de una forma sencilla y sin pérdida alguna de fidelidad de la señal. En el caso de señales analógicas, el almacenamiento resulta más difícil y sobre todo introduce una pérdida de calidad de señal.

Desventajas del procesado digital de señales

El procesado digital también tiene sus limitaciones, las cuales estriban en la velocidad de operación de los conversores analógicos/digital, y la de los procesadores digitales de señal.

El resto del capítulo se divide en cinco secciones, cuyos contenidos son los siguientes. En la Sección 1.1 se especifica el problema a resolver en este trabajo, en la Sección 1.2 se dan los antecedentes que orillaron a plantear el problema, la Sección 1.3 plantea los objetivos a perseguir, basándose en el planteamiento del problema y la conjetura que plantea el mismo. La Sección 1.4 describe la contribución de este trabajo, y finalmente

la Sección 1.5 explica la estructura de la tesis.

1.1. Planteamiento del problema

La señal que se propone estudiar en este trabajo es del tipo unidimensional aperiódicas cuya variable independiente es el tiempo, con la finalidad de que los resultados sean perceptibles al oido del usuario se opta por utilizar la señal de audio. No obstante, se considera en el análisis de este trabajo el uso de señales unidimensionales.

Dada una señal de audio, la cuestión que se plantea se relaciona con el procesamiento adecuado para la recepción, almacenamiento y transmisión segura de la señal. Agregaríamos también la posibilidad de realizar este proceso en tiempo real.

Este proceso se propone resolver a través de un programa, el cual hace uso de objetos por medio de una interfaz para utilizar específicamente el DSP/BIOS del procesador digital de señales o DSP para mayor brevedad y por sus siglas en inglés. Se afirma que el empleo del DSP contribuye en gran medida a resolver el problema de medición en tiempo real.

1.2. Antecedentes

El manejo de señales que proporcione una completa seguridad en su transmisión, actualmente es de gran importancia e interés en varios campos o áreas de aplicación. Prueba de ello, son trabajos tales como:

DESARROLLO DE UN SISTEMA DE ADQUISICIÓN Y TRATAMIENTO DE SEÑALES ELECTROCARDIOGRÁFICAS, 2004 Univ. Tarapacá, y CIRCUITOS ELECTRÓNICOS PARA EL PROCESAMIENTO PARALELO DE SEÑALES ESTOCÁSTICAS, 1998. Universidad de Colima, tiene desventajas porque se usa una computadora para el procesamiento de señales, la cual no tiene la capacidad que posee el DSP y además es más estorbosa.

CIRCUITOS ELECTRÓNICOS PARA EL PROCESAMIENTO PARALELO DE SEÑALES ESTOCÁSTICAS, 1995. Tiene desventajas porque utiliza una compuerta AND para el procesado de señales lo que la hace no sólo mas lento sino con mucha mayor pérdida de información.

Sin embargo, existen diferentes aspectos que no son tocados en los proyectos mencionados en el párrafo previo. Entre ellos podemos citar a los siguientes, la señal de entrada es de manera general una señal unidimensional (datos, información, audio, señales médicas, etc.), la dependencia en todo momento de un equipo de cómputo, y el procesamiento en tiempo real.

Se han realizado algunos intentos en esta dirección, aunque resulta importante mencionar las diferencias con respecto a esta tesis. Se resalta el hecho de programar un DSP para llevar a cabo el procesamiento en tiempo real, y obtener la independencia después de programada la tarjeta DSP de la computadora.

Se opta por realizar la programación del DSP a través del DSP/BIOS API's que enseñan como usar las herramientas de configuración y análisis para la transferencia de la señal.

Una motivación para la elaboración de este trabajo de investigación es la poca cantidad de información, bibliografía así como de programas que sirvan de guía, para la elaboración del procesamiento de audio en tiempo real con la utilización del DSP. Con la finalidad principalmente de que los lectores, y en especial los alumnos del área de electrónica de la UAEH tengan un acercamiento al programador por medio de objetos DSP/BIOS para los DSP's.

1.3. Objetivos

Es posible resumir en un objetivo general y objetivos específicos lo dicho en el planteamiento del problema, en la solución que se propone y en la discusión de los antecedentes.

Objetivo general

Con los conocimientos del procesamiento digital de señales y dada una señal de audio cualquiera, el objetivo de este trabajo de tesis es realizar la transmisión de la señal con buena calidad y en tiempo real.

Lo que conjeturamos en la Sección 1.1 permite listar una serie de objetivos específicos cuya realización llevará al logro del objetivo general. De tal forma que los objetivos específicos son los siguientes:

Objetivos específicos

- Analizar las aplicaciones y operaciones de las señales y sistemas.
- Poner en operación el DSP.
- Conocer y aplicar el DSP/BIOS para la elaboración de programas.
- Implementar el programa en el DSP.
- Adquirir la señal de audio, a través del DSP.

1.4. Contribución de la tesis

La investigación de este trabajo de tesis ha dado diversas aportaciones, las cuales se mencionan a continuación.

Debido a que existe muy poca bibliografía especializada y en español sobre la tarjeta DSP, es importante elaborar un compendio básico de los tipos de señales, para resaltar la importancia y facilitar su compresión y además del uso del DSP.

Se explica de manera detallada como llevar a cabo la programación del DSP, a través del DSP/BIOS, incluyendo la práctica para su implementación en dicho dispositivo.

Todo tiene gran importancia en el área de la ingeniería en electrónica y telecomunicaciones, así como en las diferentes carreras afines con los adelantos de la ciencia y tecnología.

1.5. Estructura de la tesis

La tesis se encuentra estructurada en 4 capítulos.

En el Capítulo 2 se inicia dando la parte introductoria para el estudio del procesamiento digital de señales, desde su definición, clasificación así como las operaciones básicas para el manejo de las mismas.

En el Capítulo 3 se hace una descripción del hardware y el software utilizado para el procesamiento.

En el Capítulo 4 es donde se ve la parte principal de la tesis, que es el desarrollo de un programa para realizar la transmisión de audio en tiempo real por medio del programador con objetos DSP/BIOS.

Por último en el Capítulo 5 se dan las conclusiones del trabajo realizado en la tesis y las propuestas para trabajos futuros con base en este tema.

Capítulo 2

Señales

2.1. Señal y clasificación de señales

El objetivo de este capítulo es presentar una introducción al estudio de señales para el análisis del procesamiento de señales, resumiendo las ideas básicas sobre las señales, introduciendo a diferentes conceptos, representaciones gráficas y matemáticas de las señales. Definiendo señales básicas, tanto continuas como discretas. Éstas incluyen señales exponenciales complejas, señales sinusoidales y funciones al impulso y escalón unitario. Además, se observa el concepto de periodicidad para los dos tipos de señales[3].

Tipos de señales

La materia prima en el procesamiento digital de señales, es la señal. A continuación se describe y clasifica. Una señal se define como una cantidad física que varia con el tiempo, espacio o cualquier otra variable o variables independientes, mediante las cuales se puede transmitir mensajes o información. Existe una gran variedad de señales que son de importancia práctica en la descripción de fenómenos físicos. Algunos ejemplos son, las imágenes, señales telefónicas, magnitudes físicas como: la temperatura, la humedad, la voz, la velocidad del viento, la intensidad de la luz, altura y presión atmosférica.

Señal determinista y aleatoria

Cualquier señal que pueda ser definida por una forma matemática explícita, un conjunto de datos o una regla bien definida se denomina **determinista**. Un ejemplo de este tipo de señales es la ecuación (2.1).

$$s(t) = \frac{1}{2}\cos\left[600t + \frac{1}{3}\right] \tag{2.1}$$

Sin embargo, existen señales que no se pueden describir con un grado de precisión razonable mediante fórmulas matemáticas explícitas, o cuya descripción es demasiado complicada para ser de utilidad práctica, a estas señales se les conoce como **señales aleatorias**. Un ejemplo de este tipo de señales es el ruido.

Señal multidimensional y unidimensional

Matemáticamente, las señales deterministas se representan como funciones de una o más variables independientes. Si una señal sólo depende de una variable, se denomina como una señal **unidimensional**. Por ejemplo, las señales de voz, los electrocardiogramas (ECG) y los encefalogramas (EEG) son señales que llevan información y que varían como funciones de una sola variable independiente, el tiempo[3].

Si la señal depende de M variables, entonces se dice que se tiene una señal M-**dimensional**. Una imagen como la figura (2.1) es un ejemplo de señal que se forma de dos variables independientes. Las dos variables independientes en éste caso son las coordenadas espaciales.

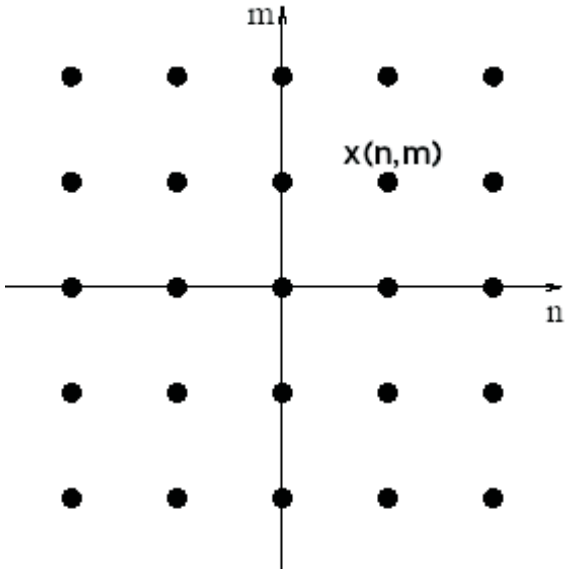

Figura 2.1: Representación de una imagen por coordenadas espaciales.

Señal en tiempo continuo y señal en tiempo discreto

Existen dos tipos básicos de señales: continuas y discretas con respecto al tiempo. En el caso de las señales análogicas la variable independiente es continua , por lo que éstas señales se definen para una sucesión continua de valores de amplitud. Por otra parte,

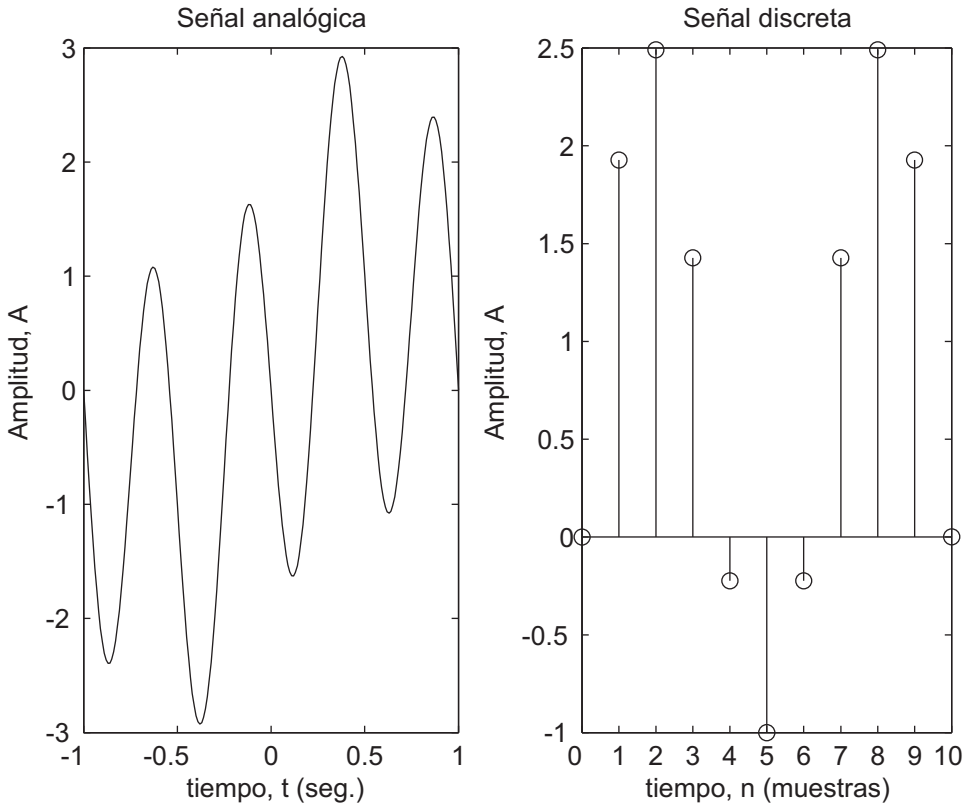

Figura 2.2: Señal en tiempo continuo y tiempo discreto.

en las señales tiempo discreto la variable independiente toma solamente un conjunto discreto de instantes en el tiempo, no importando el valor de su amplitud.

La señal de voz como una función del tiempo, y la presión atmosférica como una función de la altitud son ejemplos de señales discretas[3].

En la figura (2.2) se muestra la diferencia entre una señal continua y una discreta, utilizando como unidad de medida al segundo (seg.) y a las muestras (n) respectivamente.

Señal de valor continuo y señal de valor discreto

El valor de amplitud de una señal, en tiempo continuo o discreto, puede ser continuo o discreto. Si una señal toma en su amplitud todos los valores posibles en un intervalo tanto finito como infinito, se dice que es de valor continuo. Por el contrario, si toma valores de amplitud de un conjunto finito de valores se dice que es de valor discreto, normalmente, éstos valores son equidistantes eso quiere decir que es la misma distancia

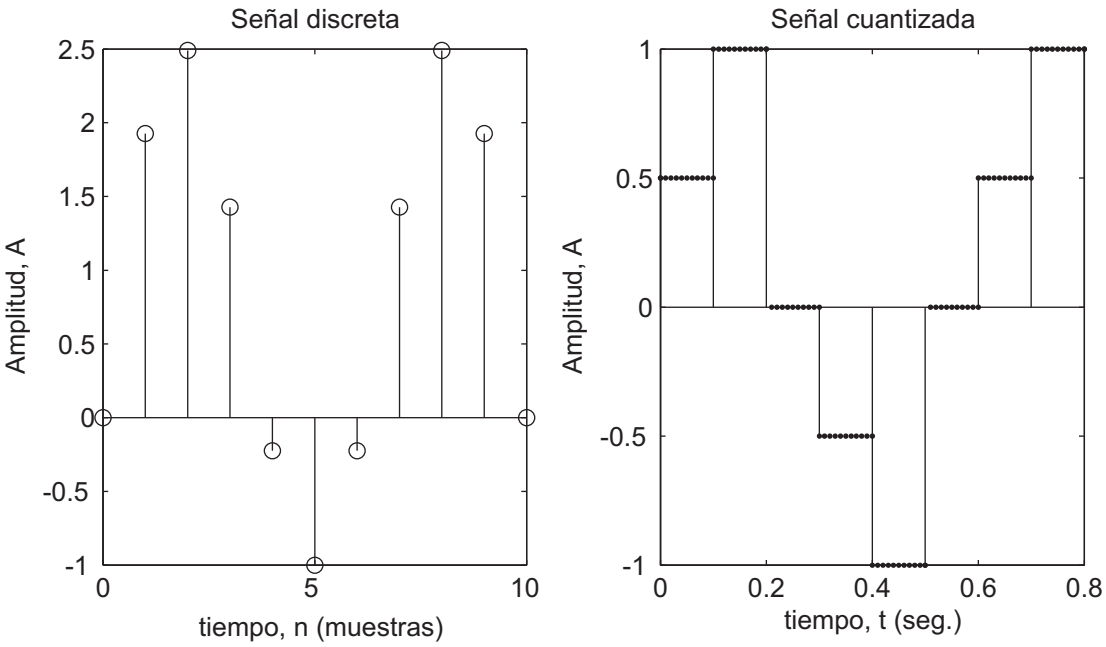

Figura 2.3: Señal de valor continuo y valor discreto.

entre cualquiera de dos muestras; aunque no necesariamente deben ser equidistantes, pero en la práctica se escogen así por conveniencia computacional[3].

Una señal en tiempo discreto, que toma valores en un conjunto discreto se denomina señal **digital**. A una señal en tiempo continuo, con valores continuos se le conoce como señal **analógica**.

La figura (2.3) es un ejemplo de una señal en tiempo discreto de valor continuo y una señal de valor discreto en tiempo continuo.

Para que una señal pueda ser procesada digitalmente tiene que ser en tiempo discreto y tomar valores discretos (es decir una señal digital). Si la señal a procesar es analógica, se convierte a digital muestreándola en el tiempo y obteniendo por tanto una señal en tiempo discreto y posteriormente cuantificando sus valores en un conjunto discreto. El proceso de convertir una señal analógica a discreta, se denomina cuantificación. Es básicamente un proceso de aproximación[3].

Señales periódicas

Un tipo importante de señales es la clase de señales periódicas. Una señal periódica continua $x(t)$ tiene la característica de que hay un valor positivo T llamado *periodo* para el cual

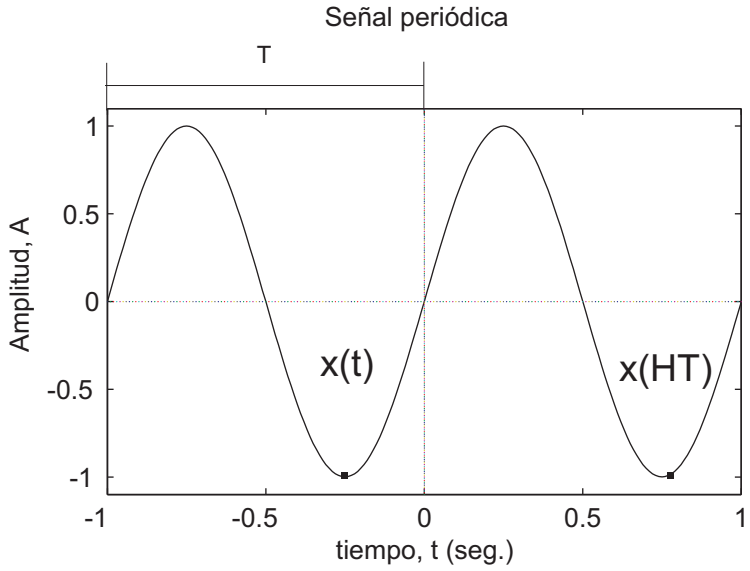

Figura 2.4: Señal periódica.

$$x(t) = x(t + T) \tag{2.2}$$

para todos los valores de t. En otras palabras una señal periódica tiene la propiedad de que no cambia para un corrimiento de tiempo T. En éste caso decimos que $x(t)$ es periódica con periodo T. Las señales periódicas continuas surgen en una gran variedad de contextos.

Por ejemplo, la respuesta natural de sistemas en los cuales se conserva la energía, como los circuitos LC ideales sin disipación de energía resistiva y los sistemas mecánicos ideales sin pérdida por fricción, son señales periódicas figura(2.4).

Señales par e impar

Otro conjunto de propiedades útiles de las señales esta relacionado con la simetría que presentan con la inversión de tiempo. Una señal $x(t)$ o $x[n]$ es conocida como una señal par si es idéntica a su contraparte invertida en el tiempo, es decir, con su reflejo con respecto al origen. Una señal es par si:

$$x(-t) = x(t) \tag{2.3}$$

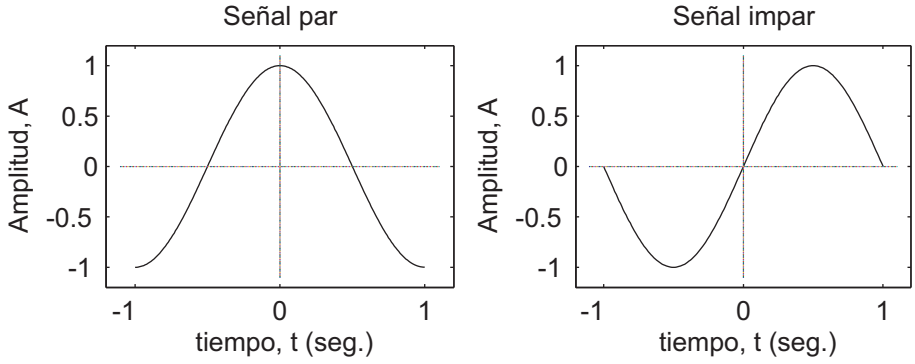

Figura 2.5: Señal par e impar.

mientras que una señal en tiempo discreto es par si

$$x[-n] = x[n] \tag{2.4}$$

a una señal se le considera impar si

$$x(-t) = -x(t) \tag{2.5}$$

$$x[-n] = -x[n] \tag{2.6}$$

una señal impar debe ser necesariamente 0 en $t = 0$ o $n = 0$, ya que las ecuaciones requieren que $x(0) = -x(0)$ y $x[0] = -x[0]$. Si $x(t)$ o $x[n]$ es impar,$x(t) = 0$ o $x[0] = 0$. En la figura (2.5) se representan señales con simetría par e impar.

Una señal arbitraria puede expresarse como la sumas de dos componentes, una de las cuales es par y la otra impar. La componente par de las señal se construye sumando $x(t)$ y $x(-t)$ o $x[n]$ y $x[-n]$ y dividiendo por 2

$$x_e(t) = \frac{1}{2}\{x(t) + x(-t)\} \tag{2.7}$$

$$x_e[n] = \frac{1}{2}\{x[n] + x[-n]\} \tag{2.8}$$

claramente, la componente par $x_e[n]$ de las señales satisface la ecuación de simetría. De forma similar, se forma la componente impar de la señal $x_0[n]$ de acuerdo con la relación

$$x_0(t) = \frac{1}{2}\{x(t) - x(-t)\} \tag{2.9}$$

$$x_0[n] = \frac{1}{2}\{x[n] - x[-n]\} \tag{2.10}$$

Señales exponenciales

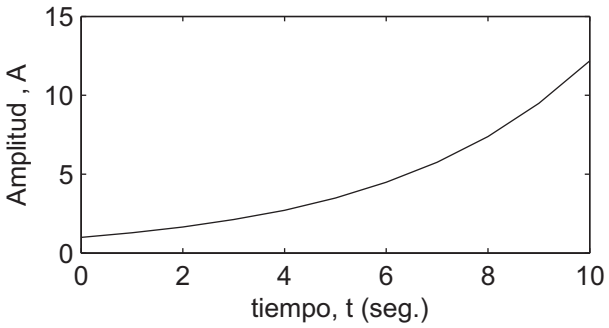

Figura 2.6: Señal exponencial.

Se presenta varias señales básicas continuas y discretas. Éstas señales no sólo ocurren con frecuencia, sino que también sirven como bloques fundamentales a partir de los cuales se puede construir muchas otras señales ejemplo la figura (2.6).

Señal continua exponencial compleja

La señal continua exponencial compleja es de la forma

$$x(t) = Ce^{at} \tag{2.11}$$

donde C y a son, en general, números complejos. Dependiendo de los valores de estos parámetros, la exponencial compleja puede adoptar características diferentes[3].

Señal exponencial real

Como se ilustra en la figura (2.7)(a), si C y a son reales en cuyo caso $x(t)$ se llama exponencial real, básicamente hay dos tipos de comportamiento. Si es positiva, entonces conforme t se incrementa $x(t)$ es una exponencial creciente, una forma que se usa para describir muchos procesos físicos diferentes, incluyendo reacciones en cadena en explosiones atómicas y reacciones químicas complejas. Si a es negativa, entonces $x(t)$ es una exponencial decreciente figura (2.7)(b), una señal que también se utiliza para describir una amplia variedad de fenómenos, entre los que se incluyen los procesos de desintegración radiactiva y las respuestas de circuitos RC y de los sistemas mecánicos amortiguadores.

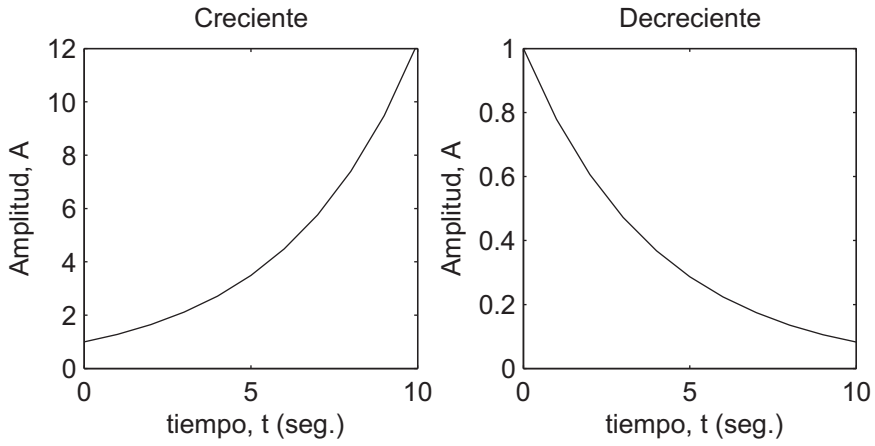

Figura 2.7: Señal exponencial real (a) creciente y (b) decreciente.

Señal exponencial compleja

Una segunda clase de exponenciales complejas como muestra la figura (2.8) de importancia se obtiene considerando el campo puramente imaginario. Específicamente considere que

$$x(t) = e^{jw_0 t} \tag{2.12}$$

Una propiedad importante de esta señal consiste en que es periódica. Para verificar lo anterior, la ecuación (2.12) que $x(t)$ será periódica con periodo T si

$$e^{jw_0 t} = e^{jw_0(t+T)} \tag{2.13}$$

o, puesto que

$$e^{jw_0(t+T)} = e^{jw_0 t} e^{jw_0 T} \tag{2.14}$$

se desprende que, para que sea periódica, debemos tener

$$e^{jw_0 T} = 1 \tag{2.15}$$

Sí $w_0 = 0$, entonces $x(t) = 1$, la cual es periódica para cualquier valor de T. Si $w_0 = 0$, entonces el periodo fundamental T de $x(t)$, es decir el valor positivo más pequeño de T para que la ecuación (2.15) se cumpla es

$$T_0 = \frac{2\pi}{|w_0|} \tag{2.16}$$

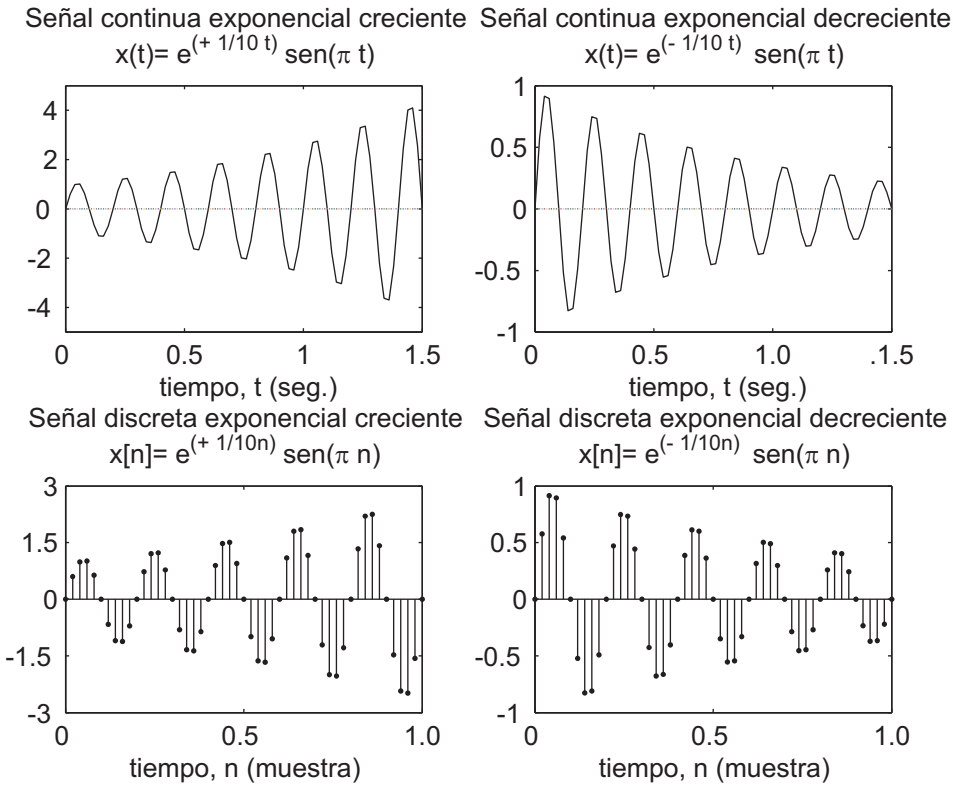

Figura 2.8: Señales exponenciales complejas imaginarias.

de esta forma, las señales e^{jw_0T} y e^{-jw_0T} tienen el mismo periodo fundamental. Las señales sinusoidales y las exponenciales complejas se usan para describir las características de muchos procesos físicos en particular en los sistemas físicos en los cuales se conserva la energía[3].

Señales sinusoidal en tiempo continuo

Una simple oscilación armónica se describe matemáticamente mediante la siguiente señal en tiempo continuo que se muestra en la ecuación (2.17)

$$x_a(t) = A\cos(\Omega t + \theta), \qquad -\infty < t < \infty \tag{2.17}$$

representa a una **señal sinusoidal continua**. El subíndice a usado con $x(t)$ denota una señal analógica como la que se muestra en la figura (2.9). Ésta señal está completamente caracterizada por tres parámetros: A es la **amplitud** de la sinusoidal, Ω es la **frecuencia angular** en radianes por segundo (rad/s), y θ es la **fase** en radianes. En

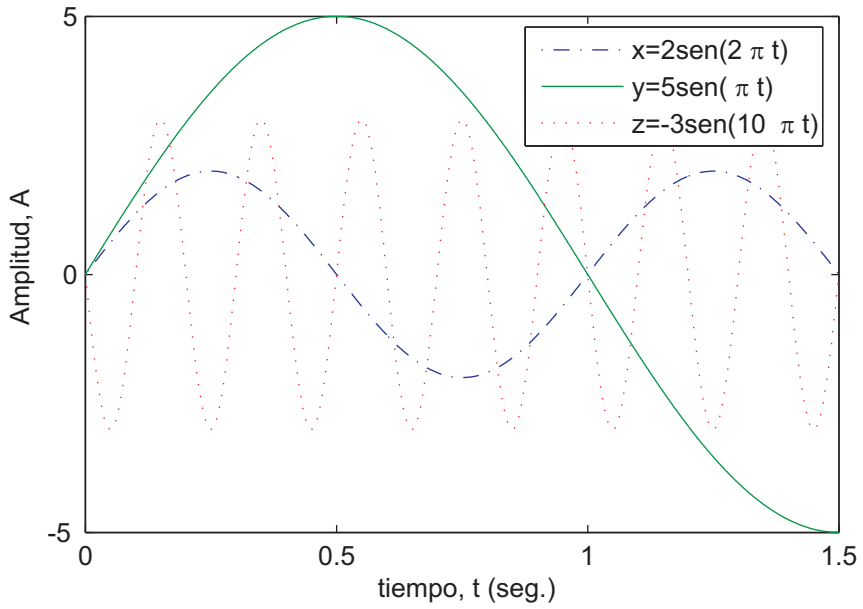

Figura 2.9: Señales sinusoidales en tiempo continuo.

lugar de Ω a menudo se utiliza la frecuencia lineal F, ciclos por segundo o hertz (Hz.) donde:

$$\Omega = 2\pi F \tag{2.18}$$

la ecuación (2.17) puede escribirse en términos de F como

$$x_a(t) = A\cos(2\pi F t + \theta), \qquad -\infty < t < \infty \tag{2.19}$$

La señal analógica sinusoidal en la ecuación (2.17) está caracterizada por las siguientes propiedades:

- **A1.** Para todo valor fijo de la frecuencia F, $x_a(t)$ es periódica. en efecto, puede demostrarse fácilmente, usando trigonometría elemental, que

  $$x_a(t + T) = x_a(t) \tag{2.20}$$

  donde $T = 1/F$ es el periodo fundamental de la señal sinusoidal.

- **A2.** Las señales en tiempo continuo con frecuencias F diferentes, son diferentes.

- **A3.** El aumento de la frecuencia F resulta en un aumento de la tasa de oscilación de la señal, en el sentido que se incluyen más periodos T en un intervalo de tiempo dado.

Como se observa, para $F = 0$, el valor $T = \infty$ es consistente con la relación fundamental $F = 1/T$ debido a la continuidad de la variable temporal t, se puede aumentar la frecuencia F sin limite, con el consiguiente aumento en la tasa de oscilación.

Las propiedades que se han descrito para señales sinusoidales son aplicables a la clase de señales **exponenciales complejas**.

$$x_a(t) = Ae^{j(\Omega t+\theta)} \tag{2.21}$$

Esto es fácil de ver expresando éstas señales en términos de sinusoidales mediante la identidad de Euler

$$e^{\pm j\phi} = \cos(\phi) \pm j\sin(\phi) \tag{2.22}$$

por definición la frecuencia F es una cantidad física inherentemente positiva. Ésto es obvio si interpretamos la frecuencia como el número de ciclos por unidad de tiempo de una señal periódica. Sin embargo, en muchos casos, únicamente por conveniencia matemática, se necesita introducir frecuencias negativas. Para entender ésto, se recorda que la señal sinusoidales en la ecuación (2.17) se expresa como

$$x_a(t) = A\cos(\Omega t + \theta) = \frac{A}{2}\, e^{j(\Omega t+\theta)} + \frac{A}{2}\, e^{-j(\Omega t+\theta)} \tag{2.23}$$

que se deduce de la ecuación (2.22). Debe advertirse que una señal sinusoidal se puede obtener como la suma de dos señales exponenciales complejas de igual amplitud, en ocasiones llamadas fasores, a medida que transcurre el tiempo, los fasores rotan en direcciones opuestas con frecuencias angulares $\pm\Omega$ radianes por segundo. Dado que una frecuencia positiva se corresponde con un movimiento angular uniforme en el sentido de las agujas del reloj[3].

Señales sinusoidal en tiempo discreto

Una **señal sinusoidal en tiempo discreto** se expresa como

$$x[n] = A\cos(\omega n + \theta), \qquad -\infty < n < \infty \tag{2.24}$$

donde n es una variable entera $n \in \mathbb{Z}$, denominada número de muestra, A es la **amplitud** de la sinusoide, ω es la **frecuencia angular** en radianes por muestra (rad/m), y θ es la **fase** en radianes (rad.).

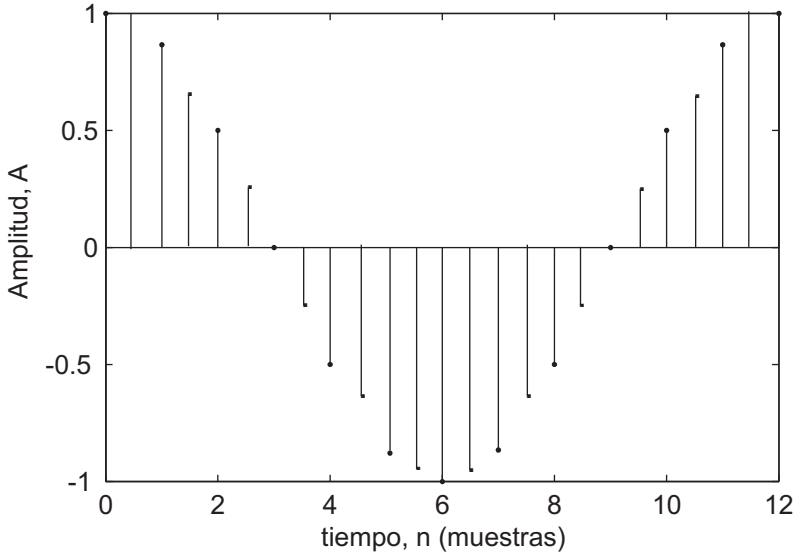

Figura 2.10: Señal sinusoidal en tiempo discreto.

Si, en lugar de ω, se utiliza la variable de frecuencia f definida por

$$\omega = 2\pi f \tag{2.25}$$

la relación en la ecuación (2.24) se convierte en

$$x[n] = A\cos(2\pi f n + \theta), \qquad -\infty < n < \infty \tag{2.26}$$

la frecuencia f tiene las dimensiones de ciclos por muestra. Considerando la sinusoidal en tiempo discreto en la ecuación (2.24), independientemente de la sinusoidal en tiempo continuo dada en la ecuación (2.17). La figura (2.10) muestra una señal sinusoidal de frecuencia $\omega = \pi/6$ radianes por muestra ($f = 1/12$ ciclos por muestra) y fase $\theta = \pi/3$.

En contraste con las sinusoidales en tiempo continuo, las sinusoidales en tiempo discreto están caracterizadas por las siguientes propiedades

- **B1**. Una sinusoidal en tiempo discreto es periódica sólo si su frecuencia f es un número racional, $f \in \mathbb{Q}$.
  Por definición, una señal en tiempo discreto $x[n]$ es periódica con periodo $N (N > 0$ y $N \in \mathbb{Z})$ sí y sólo sí

  $$x[n + N] = x[n] \quad \text{para todo } n \in \mathbb{Z} \tag{2.27}$$

  el valor más pequeño de N para el que se cumple la ecuación (2.27) se denomina *periodo fundamental*. La demostración de la propiedad de periodicidad es simple.

Para que una sinusoidal con frecuencia f_0 sea periódica, debemos tener

$$\cos[2\pi f_0(N+n)+\theta] = \cos[2\pi f_0 n+\theta] \tag{2.28}$$

esta relación es cierta sí y sólo sí existe un $K \in \mathbb{Z}$ tal que

$$2\pi f_0 N = 2K\pi \tag{2.29}$$

o, equivalentemente,

$$f_0 = \frac{K}{N} \tag{2.30}$$

de acuerdo con la ecuación (2.30), una señal sinusoidal en tiempo discreto es periódica sólo si su frecuencia f_0 se puede expresar como cociente de dos enteros esto es, si f_0 es racional. Para determinar el periodo fundamental N de una sinusoidal periódica, se debe expresar su frecuencia como en la ecuación (2.30) y cancelar factores comunes hasta que K y N sean números primos relativos. Entonces el periodo fundamental de la sinusoidal es N. Obsérvese que una pequeña variación en la frecuencia puede originar un gran cambio en el periodo. Por ejemplo, obsérvese que $f_1 = 31/60$ implica que $N_1 = 60$, mientras que $f_2 = 30/60$ implica que $N_2 = 2$.

- **B2**. Las sinusoidales en tiempo discreto cuyas frecuencias están separadas por un múltiplo entero 2π, son idénticas.
  Para comprobarlo, consideramos la sinusoidal $\cos(\omega_0 n+\theta)$. Fácilmente se comprueba que

$$\cos[(\omega_0+2\pi)n+\theta] = \cos(\omega_0 n+2\pi n+\theta) = \cos(\omega_0 n+\theta) \tag{2.31}$$

como resultado, todas las secuencias sinusoidal

$$x_k[n] = A\cos(\omega_k n+\theta), \qquad k = 0,1,2... \tag{2.32}$$

donde

$$\omega_k = \omega_0 + 2k\pi, \qquad -\pi \leq \omega_0 \leq \pi \tag{2.33}$$

son *indistinguibles* (esto es, *idénticas*). Por otro lado, las secuencias de dos sinusoidales cualesquiera de frecuencias en el rango $-\pi \leq \omega \leq \pi$ ó $-\frac{1}{2} \leq f \leq \frac{1}{2}$ son distintas. En consecuencia, las señales sinusoidales en tiempo discreto de frecuencias $|\omega| \leq \pi$ ó $|f| \leq \frac{1}{2}$ son únicas.

La secuencia resultante de una sinusoidal con una frecuencia $|\omega| > \pi$, o $|f| > \frac{1}{2}$, debido a ésta similitud, denominamos a la sinusoidal que tiene la frecuencia $|\omega| > \pi$ un *alias* de la sinusoidal correspondiente de frecuencia $|\omega| < \pi$.

Por ésta razón consideramos las frecuencias en el rango $-\pi \leq \omega \leq \pi$, ó $-\frac{1}{2} \leq f \leq \frac{1}{2}$ como únicas y todas las frecuencias $|\omega| > \pi$, o $|f| > \frac{1}{2}$, como alias.

- **B3**. La mayor tasa de oscilación en una sinusoidal en tiempo discreto se alcanza cuando $w = \pi$(ó $\omega = -\pi$) ó, equivalentemente, $f = \frac{1}{2}$ (ó $f = -\frac{1}{2}$).

  Para ilustrar esta propiedad, investigamos las características de la señal secuencia sinusoidal

$$x[n] = \cos \omega_0 n \tag{2.34}$$

  cuando la frecuencia varía desde 0 a π. Como ejemplo tomamos valores de $\omega_0 = 0, \frac{\pi}{8}, \frac{\pi}{4}, \frac{\pi}{2}, \pi$ correspondientes a $f = 0, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}$, que dan lugar a secuencias periódicas con periodos $N = \infty, 16, 8, 4, 2$, el periodo de la sinusoidal disminuye a medida que la frecuencia aumenta. La tasa de oscilación aumenta cuando aumenta la frecuencia, como se puede observar en la figura (2.11).

  Se puede ver que sucede cuando $\pi \leq \omega_0 \leq 2\pi$, se considera la sinusoidales de frecuencias $\omega_1 = \omega_2$ y $\omega_2 = 2\pi - \omega_0$. Obsérvese que mientras ω_1 varia de $\pi a 2\pi$, ω_2 varía de π a 0. Puede verse fácilmente que

$$\begin{aligned} x_1 &= A\cos\omega_1 n = A\cos\omega_0 n \\ x_1 &= A\cos\omega_1 n = A\cos(2\pi - \omega_0)n \\ &= A\cos\omega_1 n = A\cos\omega_0 n \end{aligned}$$

  Por consiguiente, ω_2 es un alias de ω_1.
  Como en el caso de la señal en tiempo continuo, también se pueden introducir las frecuencias negativas para las señales en tiempo discreto. Con la identidad

$$x[n] = A\cos(\omega n + \theta) = \frac{A}{2}e^{j(\omega n+\theta)} + \frac{A}{2}e^{-j(\omega n+\theta)} \tag{2.35}$$

  dado que las señales sinusoidales en tiempo discreto de frecuencias separadas por un múltiplo entero 2π son idénticas, se deduce que las frecuencias en cualquier intervalo $\omega_1 \leq \omega \leq \omega + 2\pi$ constituyen *todas* las **sinusoidales o exponenciales complejas en tiempo discreto** existentes.

Por tanto, el rango de frecuencias para sinusoidales en tiempo discreto es *finito* con duración 2π.

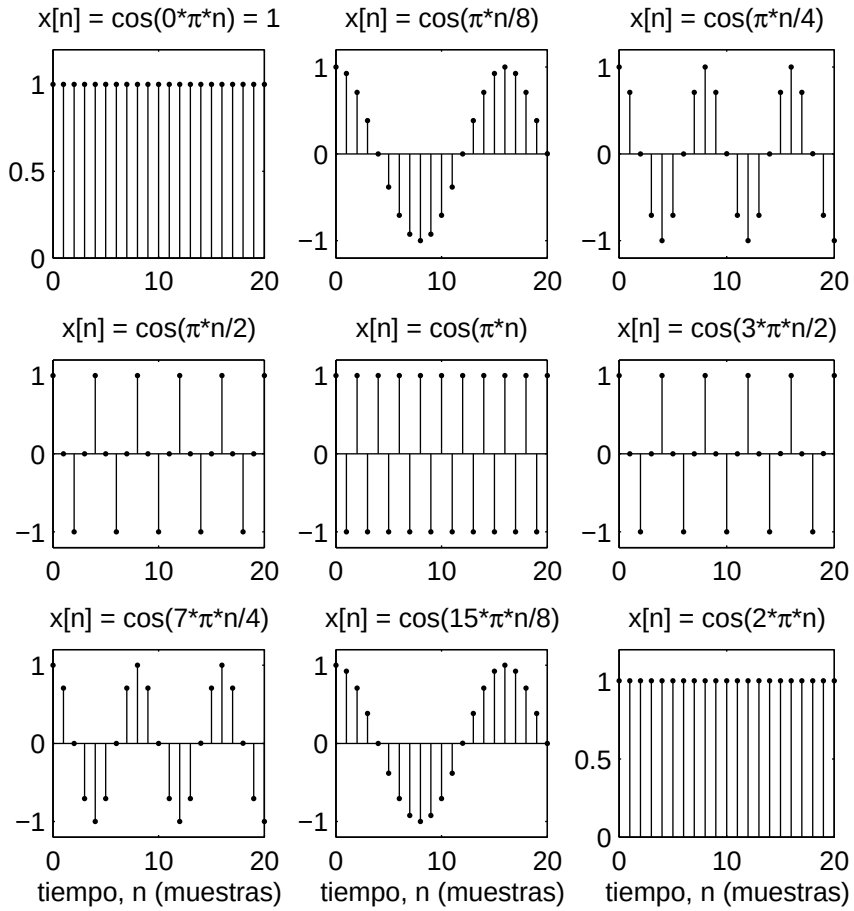

Figura 2.11: Secuencias sinusoidales discretas por diferentes frecuencias.

Habitualmente, se elige el rango $0 \leq \omega \leq 2\pi$ ó $-\pi \leq \omega \leq \pi$ ó($0 \leq f \leq 1$, $-\frac{1}{2} \leq f \leq \frac{1}{2}$), que se denomina el *rango fundamental*. Además de la representación gráfica de una señal en tiempo discreto o secuencia figura (2.11), existen otras alternativas que a menudo son mas convenientes, éstas son

1. Representación funcional, por ejemplo

$$x[n] = \begin{cases} 1, & \text{para } n = 1, 3; \\ 4, & \text{para } n = 2; \\ 0, & \text{en el resto.} \end{cases}$$

2. Representación tabular,

Ejemplo

n	...	-2	-1	0	1	2	3	4	5	...
x[n]	...	0	0	0	1	4	1	0	0	...

3. Representación como secuencia Una señal o secuencia de duración infinita con el origen de tiempo ($n = 0$) indicado por el símbolo $\uparrow$ se representa como

$$x[n] = \left\{ \begin{array}{cccccccccc} ... & 0, & 0, & 1, & 4, & 1, & 0, & 0, & ... \\ & & \uparrow & & & & & & \end{array} \right\}$$

una secuencia $x[n]$, que es cero para $n < 0$, se puede representar como

$$x[n] = \left\{ \begin{array}{ccccccc} 0, & 1, & 4, & 1, & 0, & 0, & ... \\ \uparrow & & & & & & \end{array} \right\}$$

el origen de tiempo de una secuencia $x[n]$, que es cero para $n < 0$, se interpreta como el primer comenzando por la izquierda, punto de secuencia.

Una secuencia de duración finita se puede representar como

$$x[n] \;=\; \left\{ \begin{array}{ccccccc} 3, & -1, & -2, & 5, & 0, & 4, & -1 \\ & & \uparrow & & & & \end{array} \right\} \tag{2.36}$$

mientras que una secuencia de duración finita que satisface la condición $x[n] = 0$ para $n < 0$ se puede representar como

$$x[n] \;=\; \left\{ \begin{array}{cccc} 0, & 1, & 4, & 1 \\ \uparrow & & & \end{array} \right\} \tag{2.37}$$

la señal dada en la ecuación (2.36) está formada por siete muestras o puntos (en el tiempo), de manera que se denomina secuencia de siete puntos. De forma similar, la secuencia dada por la ecuación (2.37) es una secuencia de cuatro puntos[3].

Algunas señales elementales en tiempo discreto

1. El *impulso unitario* se denomina $\delta[n]$ y se define como

$$\delta[n] = \left\{ \begin{array}{ll} 1, & \text{para } \; n = 0 \\ 0, & \text{para } \; n \neq 0 \end{array} \right. .$$

en otras palabras, el impulso unitario es una señal que vale cero siempre excepto para $n = 0$ donde vale uno.

Al contrario que la señal analógica $\delta(t)$, que también se conoce como impulso unitario y vale cero siempre excepto en $t = 0$, donde tiene área unidad, la secuencia de respuesta al impulso es mucho menos complicada matemáticamente. La representación gráfica de $\delta[n]$ se muestra en la figura (2.12)

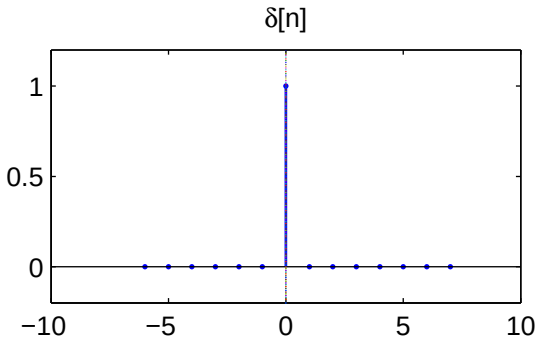

Figura 2.12: Impulso unitario.

2. La señal *escalón unidad* se denota como $u[n]$ y se define como

$$u[n] = \begin{cases} 1, & \text{para } n \geq 0 \\ 0, & \text{para } n < 0 \end{cases}$$

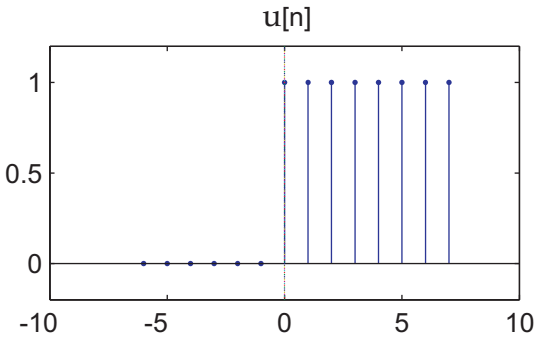

Figura 2.13: Escalón unidad.

la figura (2.13) ilustra la señal escalón unidad.

3. La señal *rampa unidad* se denota como $u_r[n]$ y se define como

$$u[n] \equiv \begin{cases} 1, & \text{para } n \geq 0 \\ 0, & \text{para } n < 0 \end{cases}$$

la figura (2.14) muestra como aumenta en un valor constante la señal[3].

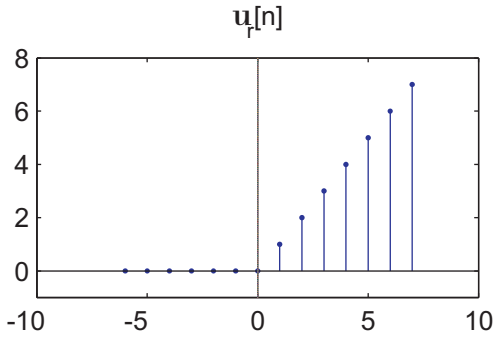

Figura 2.14: Rampa unidad.

Señales de energía finita y potencia media finita

Sea $x(t)$ una señal real. Si $x(t)$ representa la tensión en una resistencia de valor R, se producirá una corriente cuya intensidad será $i(t)/R$. La potencia instantánea de la señal es $Ri^2(t) = x^2(t)/R$, y la energía correspondiente al intervalo infinitesimal dt es $[x^2(t)/R]dt$. En general, no se conoce sí $x(t)$ corresponde a una señal de tensión o corriente, y para normalizar la potencia, asumiremos que la resistencia R es de 1 ohmio. Por tanto, la potencia instantánea correspondiente a la señal $x(t)$ vale $x^2(t)$. La energía E de la señal durante un intervalo temporal de duración $2L$ se define así:

$$E_{2L} = \int_{-L}^{L} |x(t)|^2 dt \tag{2.38}$$

y la energía total de la señal en el intervalo $t(-\infty, \infty)$ es

$$E = \lim_{L\to\infty} \int_{-L}^{L} |x(t)^2| dt \tag{2.39}$$

la potencia media puede definirse entonces así

$$P = \lim_{L\to\infty} [\frac{1}{2L} \int_{-L}^{L} |x(t)^2| dt] \tag{2.40}$$

aunque en el desarrollo de las ecuaciones (2.39) y (2.40) se empleo señales eléctricas, dichas ecuaciones definen respectivamente la energía y la potencia media para cualquier señal arbitraria $x(t)$.

Cuando el limite definido en la ecuación (2.39) existe y es finito ($0 < e < \infty$), se dice que la señal $x(e)$ es de energía finita. Observando la ecuación (2.40) se ve que las señales de energía finita tiene una potencia media cero. Por otra parte, si el limite de

la ecuación (2.40) existe y es finito ($0 < p < \infty$), se dice que $x(p)$ es de potencia media finita. Las señales de potencia media finita tienen energía infinita.

Como se menciona anteriormente, las señales periódicas están definidas para cualquier valor del tiempo entre $-\infty < e < \infty$ y, por lo tanto, tienen energía infinita. En la mayoría de los casos las señales periódicas tienen potencia media finita. Por el contrario, las señales de duración limitada son señales de energía finita[3].

2.2. Señal de audio

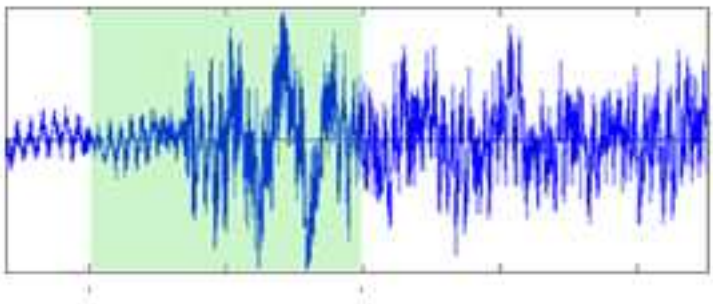

Figura 2.15: Señal de audio.

Una señal de audio que se muestra en la figura (2.15) es una señal electrónica que es una representación eléctrica exacta de una señal sonora; normalmente está acotada en el rango de frecuencias audibles para los seres humanos que se encuentra aproximadamente entre los 20 y los 20 000 Hz. (el equivalente, casi exacto a 10 octavas). Dado que el sonido es una onda de presión se requiere un transductor de presión (un micrófono) que convierte las ondas de presión de aire (ondas sonoras) en señales eléctricas (señales analógicas).

La conversión contraria se realiza mediante un altavoz -también llamado altoparlante en algunos países latinoamericanos, por traducción directa del inglés loudspeaker, que convierte las señales eléctricas en ondas de presión de aire. Un sólo micrófono puede captar adecuadamente todo el rango audible de frecuencias, en cambio para reproducir de manera fidedigna ese mismo rango de frecuencias suelen requerirse dos altavoces (de

agudos y graves) o más.

Una señal de audio se puede caracterizar, por su dinámica (valor de pico, rango dinámico, potencia, relación señal-ruido) o por su espectro de potencia (ancho de banda, frecuencia fundamental, armónicos, distorsión armónica, etc.).

Así, por ejemplo, una señal que represente voz humana (señal vocal) no suele tener información relevante más allá de los 10 kHz, y de hecho en telefonía fija se toman sólo los primeros 4 kHz. Con 2 kHz basta para que la voz sea comprensible, pero no para reconocer al hablante.

2.3. Conversión analógica-digital y digital-analógica

La mayoría de las señales de interés práctico, señales de voz, biológicas, sistemas, radar, sonar y distintos tipos de comunicación, como las señales de audio y video, son analógicas. Para procesar señales analógicas por medios digitales es necesario convertirlas a formato digital, esto es, transformarlas en una secuencia de números de precisión finita. Éste procedimiento se le llama **conversión analógico-digital** (ADC) y los dispositivos correspondientes **conversores** (ADCs). Conceptualmente, se puede ver la conversión ADC como un proceso en tres pasos.

1. **Muestreo**. Ésta es la conversión de una señal en tiempo continuo a una señal en tiempo discreto obtenida tomando *muestras* de la señal en tiempo continuo en instante en tiempo discreto. Así, si $x_a(t)$ es la entrada al muestreador, la salida es $x_a[nT] \equiv x[n]$, donde T se denomina el **intervalo del muestreo**.

2. **Cuantificación**. Ésta es la conversión de una señal en tiempo discreto con valores continuos a una señal en tiempo discreto con valores discretos (señal digital). El valor de cada muestra de la señal se representa mediante un valor seleccionado de un conjunto finito de valores posibles. La diferencia entre la muestra sin cuantificar $x[n]$ y la salida cuantificada $x_q[n]$ se denomina error de cuantificación.

3. **Codificación**. En el proceso de codificación, cada valor discreto $x_q[n]$ se representa mediante secuencia binaria de b bits.

Aunque modelamos el conversor ADC con un muestreador seguido de un cuantificador, en la práctica la conversión ADC se efectúa en un único dispositivo que toma $x_a(t)$ y produce un número codificado en binario. Las operaciones de muestreo siempre tiene lugar antes de la cuantificación.

En muchos casos de interés práctico, por ejemplo en el procesado de voz es deseable, convertir las señales digitales procesadas a forma analógica. Obviamente, no se puede escuchar la secuencia de muestras que presentan la señal de voz o números que corresponden a la señal de TV. El procesamiento de una conversión de una señal digital

en una señal analógica se conoce como **conversión digital analógica** (DAC). Todos los conversores DAC *conectan* los puntos de una señal digital efectuando cierto tipo de interpolación, cuya precisión depende de la calidad del proceso de conversión DAC[2].

Muestreo de señales analógicas

Existen muchas maneras de muestrear una señal. Enfocandose en el muestreo **periódico o uniforme**, éste muestreo es usado más a menudo en la práctica y se describe mediante la relación[1].

$$x[n] = x_a[nT] \qquad -\infty < n < \infty \tag{2.41}$$

donde $x[n]$ es la señal en tiempo discreto obtenida tomando muestras de la señal analógica $x_a(t)$ cada T segundos. El intervalo de tiempo T entre dos muestras sucesivas se denomina **periodo de muestreo o intervalo de muestreo**, y su recíproco $1/T = F_s$ se llama **velocidad de muestreo** (muestras por segundo) o **frecuencia de muestreo** (hertzios). El muestreo establece una relación entre variables t y n de tiempo continuo y tiempo discreto, respectivamente. De hecho, éstas variables se relacionan linealmente a través del periodo de muestreo T o, equivalente, a través de la velocidad de muestreo $F_s = 1/T$, tal que

$$t = nT = \frac{n}{F_s} \tag{2.42}$$

como consecuencias de la ecuación (2.42), existe una relación entre la variable frecuencia F (ó ω) de las señales analógicas y la variable de frecuencia f (ó ω) de las señales en tiempo discreto. Para establecer dicha relación considere una señal analógica de la forma

$$x_a(t) = A\cos(2\pi Ft + \theta) \tag{2.43}$$

que, cuando se muestrea periódicamente a una velocidad de $F_s = 1/T$ muestras por segundo, da lugar a

$$x_a[nT] \equiv x[n] = A\cos(2\pi Ft + \theta) \quad = \quad A\cos\left(\frac{2\pi Fn}{F_s} + \theta\right) \tag{2.44}$$

si se compara la ecuación (2.43) con $x[n] = A\cos(2\pi Ft + \theta)$, observaremos que las variables frecuencia F y f están linealmente relacionadas según

$$f = \frac{F}{F_s} \tag{2.45}$$

o, equivalentemente, según

$$w = \Omega T \tag{2.46}$$

la relación dada en la ecuación (2.45) justifica el nombre **frecuencia normalizada o relativa** que se usa a veces para describir la variable frecuencia f. Como se ve en la ecuación (2.45) se puede usar f para determinar la frecuencia F en hertzios sólo sí la frecuencia de muestreo F_s es conocida. El rango de la variable frecuencia F ó Ω para sinusoidales en tiempo continuo es

$$-\infty < F < \infty \qquad -\infty < \Omega < \infty \tag{2.47}$$

sin embargo, la situación es diferente para sinusoidales en tiempo discreto. De la ecuación (2.45) tenemos que

$$-\frac{1}{2} < f < \frac{1}{2} \qquad -\pi < \Omega < \pi \tag{2.48}$$

de éstas relaciones se desprende que la diferencia fundamental entre señales en tiempo discreto y señales en tiempo continuo es el rango de los valores de las variables frecuencia F y f, ó Ω y ω. El muestreo periódico de una señal en tiempo continuo supone una correspondencia entre un rango de frecuencia infinito correspondiente a la variable F (ó Ω) y un rango de frecuencia finito correspondiente a la variable f (ó ω). Dado que la frecuencia máxima de una señal en tiempo discreto es $\omega = \pi$ o $f = \frac{1}{2}$, los valores máximos de F y Ω para una velocidad de muestreo F_s son

$$F_{max} = \frac{F_s}{2} = \frac{1}{2T} \qquad \Omega_{max} = \pi F_s = \frac{\pi}{T} \tag{2.49}$$

por lo tanto, el muestreo introduce ambigüedad; así, la máxima frecuencia de una señal en tiempo continuo que puede determinarse cuando dicha señal se muestra a una velocidad $F_s = \frac{1}{T}$ es $F_{max} = \frac{F_s}{2}$, ó $\Omega_{max} = \pi F_s$[1].

2.4. Tiempo real

Un sistema de tiempo real es aquel en el que para que las operaciones computacionales estén correctas no depende solo de que la lógica e implementación de los programas computacionales sea correcto, sino también en el tiempo en el que dicha operación entregó su resultado. Si las restricciones de tiempo no son respetadas el sistema se dice que ha fallado[10, 11].

Por lo tanto, es esencial que las restricciones de tiempo en los sistemas sean cumplidas. El garantizar el comportamiento en el tiempo requerido necesita que el sistema sea predecible. Es también deseable que el sistema obtenga un alto grado de utilización a la vez que cumple con los requerimientos de tiempo[10, 11].

Un buen ejemplo es el de un robot que necesita tomar una pieza de una banda sinfín. Si el robot llega tarde, la pieza ya no estará donde debía recogerla. Por lo tanto el trabajo se llevó acabo incorrectamente, aunque el robot haya llegado al lugar adecuado. Si el

robot llega antes de que la pieza llegue, la pieza aun no estará ahí y el robot puede bloquear su paso[10]. En algunas ocasiones podemos ver referencias sobre sistemas de tiempo real cuando solo se quiere decir que el sistema es rápido. Cabe mencionar que *tiempo real* no es sinónimo de rapidez; esto significa que no es la latencia de la respuesta lo que nos enfoca en un sistema de tiempo real (esta latencia a veces esta en el orden de los segundos), el enfoque en tiempo real de la latencia es el asegurarse de que la latencia del sistema es la suficiente para resolver el problema que al cual el sistema está dedicado[10].

Si el tener una falla en el tiempo de latencia de un proceso del sistema lleva como consecuencia un error en el sistema entonces esos procesos se consideran de tiempo real duro. Si el tener una falla en un proceso del sistema no conlleva una falla en el sistema siempre y cuando esta falla este dentro de ciertos límites establecidos (es posible fallar en la latencia una de cada 1000 veces o una de cada 100, o fallar siempre y cuando el error no exceda el 3 % de la latencia) entonces esos procesos se llaman procesos de tiempo real suave. Si el funcionamiento incorrecto del sistema puede llevar a la perdida de vidas o catástrofes similares entonces el sistema de tiempo real es nombrado como sistema de tiempo real de misión crítica[10, 11].

Características de los sistemas de tiempo real

- ## Determinismo

  El determinismo es una cualidad clave en los sistemas de tiempo real. Es la capacidad de determinar con una alta probabilidad, cuanto es el tiempo que se toma una tarea en iniciarse. Esto es importante por que los sistemas de tiempo real necesitan que ciertas tareas se ejecuten antes de que otras puedan iniciar. Esta característica se refiere al tiempo que tarda el sistema antes de responder a una interrupción. Este dato es importante saberlo por que casi todas las peticiones de interrupción se generan por eventos externos al sistema (i.e. por una petición de servicio), así que es importante determinar el tiempo que tardara el sistema en aceptar esta petición de servicio[12].

- ## Respuesta

  La respuesta se enfoca en el tiempo que se tarda una tarea en ejecutarse una vez que la interrupción ha sido atendida. Los aspectos a los que se enfoca son:

  - La cantidad de tiempo que se lleva el iniciar la ejecución de una interrupción.
  - La cantidad de tiempo que se necesita para realizar las tareas que pidió la interrupción.
  - Los Efectos de Interrupciones anidadas.

  Una vez que el resultado del cálculo de determinismo y respuesta son obtenidos. Se convierte en una característica del sistema y un requerimiento para las aplicaciones

que correrán en él. (e.g. Si diseñamos una aplicación en un sistema en el cual el 95 % de las tareas deben terminar en cierto periodo de tiempo entonces es recomendable asegurarse que las tareas ejecutadas de nuestra aplicación no caigan en el 5 % de bajo desempeño)[12].

- **Usuarios controladores**

  En estos sistemas, el usuario (i.e. los procesos que corren en el sistema) tienen un control mucho más amplio del sistema.

  - El proceso es capaz de especificar su prioridad.
  - El proceso es capaz de especificar el manejo de memoria que requiere (que parte estará en caché y que parte en memoria swap y que algoritmos de memoria swap usar).
  - El proceso especifica que derechos tiene sobre el sistema.

  Esto aunque parece anárquico no lo es, debido a que los sistemas de tiempo real usan tipos de procesos que ya incluyen estas características, y usualmente estos tipos de procesos son mencionados como requerimientos. Un ejemplo es el siguiente: Los procesos de mantenimiento no deberán exceder el 3 % de la capacidad del procesador, a menos de que en el momento que sean ejecutados el sistema se encuentre en la ventana de tiempo de menor uso[12].

- **Confiabilidad**

  La confiabilidad en un sistema de tiempo real es otra característica clave. El sistema no debe de ser solamente libre de fallas pero más aun, la calidad del servicio que presta no debe de degradarse más allá de un límite determinado. El sistema debe de seguir en funcionamiento a pesar de catástrofes, o fallas mecánicas. Usualmente una degradación en el servicio en un sistema de tiempo real lleva consecuencias catastróficas[12].

- **Operación a prueba de fallas duras (Fail soft operation)**

  El sistema debe de fallar de manera que cuando ocurra una falla, el sistema preserve la mayor parte de los datos y capacidades del sistema en la máxima medida posible. Que el sistema sea estable, i.e. que si para el sistema es imposible cumplir con todas las tareas sin exceder sus restricciones de tiempo, entonces el sistema cumplirá con las tareas más críticas y de más alta prioridad. Los sistemas de tiempo real y el análisis de sus requerimientos[12].

Capítulo 3

DSP TMS320C6711

3.1. Hardware

El DSP 6711 es un dispositivo de Texas Instrument diseñado de tal manera que es fácil de usar, es práctico porque tiene herramientas de se pueden utilizar todos los diseños de programas para la serie TMS320C6000 con un alto rendimiento y una gran velocidad en procesamiento figura (3.1).

Las aplicaciones del DSP 6711 son: productos profesionales de audio, mezcladores, sintetizadores de audio, audio conferencia y broadcast, biomediciones, medicina, industria, imagen digital, reconocimiento de voz.

El hardware incluye:

- C6711 DSK tarjeta fácil de conectar a la PC a través del cable del puerto paralelo(incluido) a 150 MHz
- SDRAM externa de 16 MB y flash de 128 KB
- TI'S TLC320AD535 16-bit Convertidor de datos
- TI'S TPS56100 Dispositivo de manejo de energía
- Controlador JTAG - Provee fácil emulación y carga
- Tarjeta hija de expansión - Provee sistema extensible de desarrollo
- CE-adaptador universal para dar energía al DSK

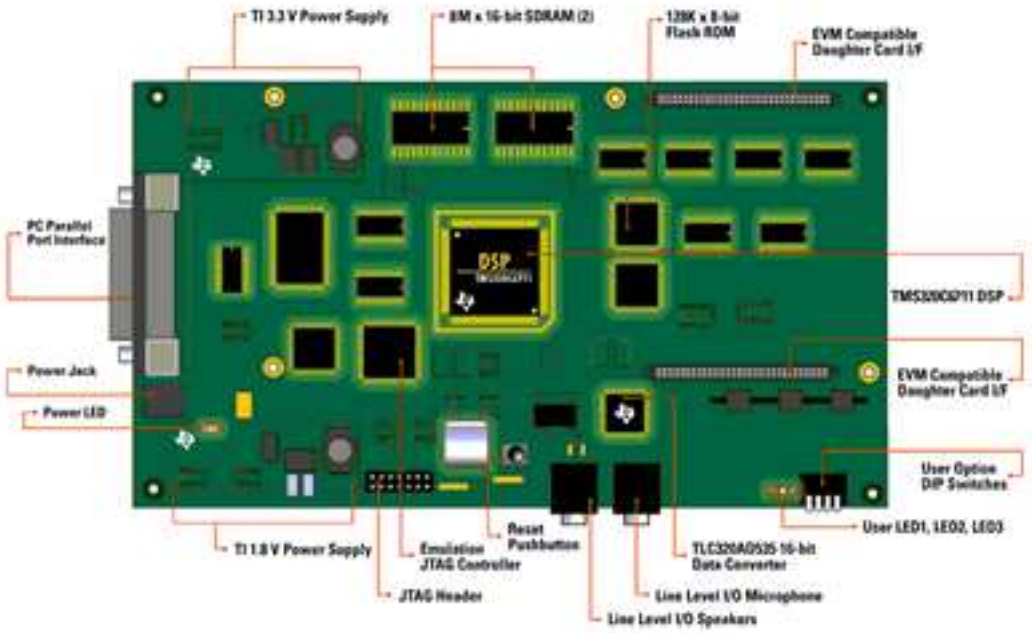

Figura 3.1: Diagrama de la tarjeta DSP 6711.

3.2. Software

Code Composer Studio

El Code Composer Studio es el software para la programación y puesta en acción del DSP 6711. Creado por Texas Instruments proporciona un ambiente donde se puede programar usando el código standard C++, por medio del ensamblador o con el DSP/BIOS que es por medio de objetos preprogramados.

Requerimientos de instalación

Son necesarios estos requerimientos para operar la plataforma DSP e instalar el Code Composer Studio (CCS)[7].

Mínimo:

- Pentium^{TM}-compatible de 233 MHz o más
- 600 MB de espacio libre en el disco duro
- 64 MB de RAM

- Monitor SVGA (800 x 600)
- Internet Explorer (4.0 o anterior) o Netscape Navigator (4.0 o anterior)
- Unidad de disco local para CD-ROM
- Sistema operativo Windows XP

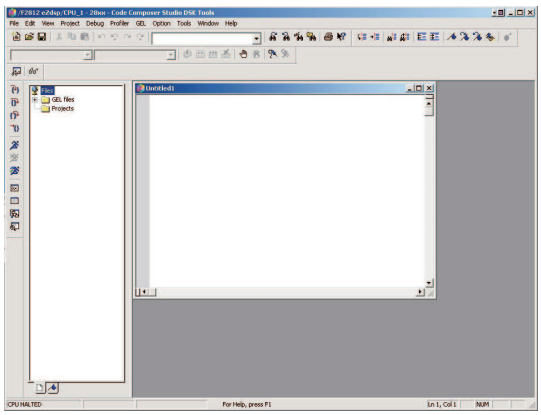

Figura 3.2: Diagrama del programa CCS.

Recomendado:

- 128 MB de RAM
- Monitor de color 16-bits
- Pentium^{TM}-compatible de 500 MHz o mas

Instalación de Code Composer Studio (CCS)

Nota: Para Windows NT y Windows 2000, Se deberá instalar y correr el Code Composer Studio usando los privilegios de administrador.

1. Insertar el CD de instalación dentro del lector de CD-ROM
2. Escoger la opción de instalación del Code Composer Studio
3. Llenar los cuadros de diálogos de instalación para que el programa ejecute

DSP/BIOS

DSP/BIOS es un kernel escalar en tiempo real. Esta diseñado para las aplicaciones que requieren programación en tiempo real y sincronización, comunicación entre el host y la tarjeta o instrumentación en tiempo real. El DSP/BIOS proporciona múltiples tareas con jerarquización, abstracción de hardware, en tiempo real y herramientas de configuración.

El DSP/BIOS es un conjunto de módulos que se enlazan con una aplicación. Esta aplicación incluye sólo aquellas funciones del DSP/BIOS que están referenciados (directa o indirectamente) por el código del programa. La configuración de las herramientas del DSP/BIOS permite optimizar el tamaño del código y la rapidez de ejecución.

El DSP/BIOS se puede usar para instrumentar, probar, trazar y monitorear cualquier aplicación en tiempo real, los programas que usan la configuración del DSP/BIOS tienen la ventaja de instrumentación implícitamente.

Componentes del DSP/BIOS

DSP/BIOS contiene los siguientes componentes:

- **Herramientas para la configuración del DSP/BIOS**

  Esta herramienta permite crear y configurar los objetos usados en los programas. Se puede usar esta herramienta para configurar la memoria, definir prioridades y controlar las interrupciones.

- **Herramientas análisis en tiempo real del DSP/BIOS**

  Esta ventana permite observar la actividad de un programa en tiempo real. Por ejemplo, la ejecución de gráficos, muestra un diagrama de la actividad. La herramienta complementa al ambiente del CCS ya que activa el análisis del programa en tiempo real. Puede monitorear una aplicación del DSP[9].

- **API's del DSP/BIOS**

  Es un conjunto de constantes, tipos, variables y funciones usadas para la programación interactivos con el software (por medio de objetos). Usando API's, se activa la tarjeta para capturar y cargar la información del host a través del Code Composer Studio.

Capítulo 4

Procesamiento de audio en el DSP

Figura 4.1: Procesamiento de audio con el DSP.

En este capítulo se muestra como usar **DSP/BIOS API's** para programar la transferencia de datos entre el hardware (periféricos l/O) y la tarjeta DSP. Los siguientes pasos guiarán a través de *Audio exemplo setup*, en donde se enseña como usar las herramientas de configuración del *DSP/BIOS* y las herramientas de análisis en tiempo real del *DSP/BIOS* para la eliminación de fallos y probar el código del programa.

EL archivo audio, que es necesario para correr el programa fue instalado en C6000 Code Composer Studio y puede ser encontrado en el directorio **C:**\$ti \ examples \ dsk$6711 \ bios \ audio.

La función **audio** esta escrita en C y localizada en el archivo del programa fuente **audio.c** que se puede hallar en el directorio **C:**$\backslash ti \backslash examples \backslash dsk6711 \backslash bios \backslash audio$[5].

En este programa se usan dos conductos (*pipe objects*) para el cambio de datos entre el software de interrupción y el puerto serial que esta conectado al codec. Los conductos son **DSS_rxPipe** y **DSS_txPipe**. La salida de datos desde el codec fluye del puerto serial **ISR**(*serial port Interrupt Service Routine*) a través de *DSS_rxPipe* por el software

de interrupción, donde es copiado por *DSS_txPipe* y regresado por el puerto serial ISR para ser retransmitido hacia afuera a través del codec, como se muestra en la figura (4.2).

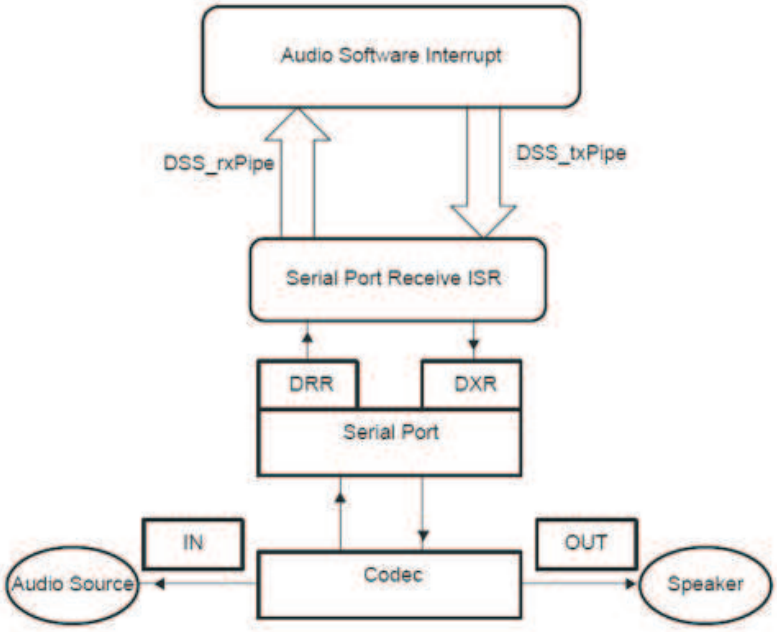

Figura 4.2: Diagrama del ejemplo de audio.

El ISR, a través del puerto serial, recibe nuevas copias de muestras de datos interrumpidas cada 32 bits usando la trama (frame) **DRR** (Data Receive Register) a una trama del conducto *DSS_rxPipe* para ser leído por la función audio.

La función *DSS_init* (encontrada en el archivo dss.c) se encarga de la inicialización del puerto serial y del codec. La función *DSS_init* programa la taza de muestreo del codec y coloca a los bits en IMR(*Interrupt Mask Register*) y IFR(*Interrupt Flag Register*) que activan el interruptor del puerto serial de entrada.

NOTA:

* La función *DSS_init* no habilita los interruptores y deberá ser llamada antes de que los interruptores sean habilitados por el *DSP/BIOS* al regresar al *MAIN*.

* La función *DSS_init* no activa al interruptor vector table para que llame al ISR para la interrupción del puerto serial. Esto es ejecutado y colocado por el HWI (*Hardware Service Routine Manager* de las herramientas de configuración del *DSP/BIOS*)[6].

4.1. Configuración setup

Todo objeto del *DSP/BIOS* esta preconfigurado y vinculado a un programa de imagen ejecutable. Esto se hace a través de las herramientas de configuración del *DSP/BIOS*. Cuando se guarda el archivo de configuración, las herramientas de configuración crean los archivos ensamblador, cabeceras y un archivo comando *settings*. Estos archivos son los que conectan al programa cuando se arma (*building*) el programa de aplicación. Para más información cosulta la Sección **Using the configuration tool** en la guía de usuarios del DSP y/o **Creating a configuration file** en el tutorial de *Code Composer Studio*.

Los siguientes objetos en DSP/BIOS serán inicializados y creados en esta sección

- El software de interrupción **audioSWI** que es corrido por la función **audio**.
- Dos conductos de datos (*pipes*), *DSS_rxPipe* y *DSS_txPipe*, para el cambio de datos entre **audioSWI** y el ISR que se encarga de la interrupción del puerto serial de entrada.
- Conectar la correspondencia ISR para el puerto serial usando HWI. *Hardware interrupt Service Routine Manager*.

Para empezar

Se necesita abrir el proyecto con el programa *Code Composer Studio* y examinar los archivos del programa fuente y librerías usadas en este proyecto.

1. Una vez instalado el programa *Code Composer Studio* en C:\$ti$ crea una carpeta llamada **audio.mak** en la dirección C:\$ti \setminus myprojects$.
2. Copia todos los archivos de C:\$ti \setminus examples \setminus dsk6711 \setminus bios \setminus audio$ a esta nueva carpeta.
3. En el menú inicio de windows, escoge $Programs \rightarrow Code\ Composer\ Studio$.
4. Escoge $Project \rightarrow New$, en *Proyect Name* escribe el nombre *audio* para el nombre del archivo y en *Location* escribe $C : \setminus ti \setminus myprojects \setminus audio.mak$ de la carpeta que creaste y sálvalo.
5. $Click \rightarrow Project \rightarrow Add\ Files\ to\ Project$ en *sources* escoge $*cmd$ y abre **audiocfg**.
6. Escoge $File \rightarrow New \rightarrow DSP/BIOS\ Configuration$.

7. Selecciona el tipo de plataforma para *DSP board* (En este caso dsk6711) y dale aceptar.

8. En el *pop_up menu*, dar click en el + de *Instrumentation* y dar click-derecho sobre *LOG → event Log Manager* y escoge *Insert LOG* del *pop-up menu*. Esto crea *LOG object* llamado *LOGO*.

9. Dar click-derecho sobre el nombre del objecto LOGO y escoge renombrar del *pop_up menu*, cambia el nombre del objeto por **Trace** y cambia el buffer *length property* a **256**.

10. Click-derecho sobre el LOG_system object y escoge properties. Cambia el buffer *length property* a **256**.

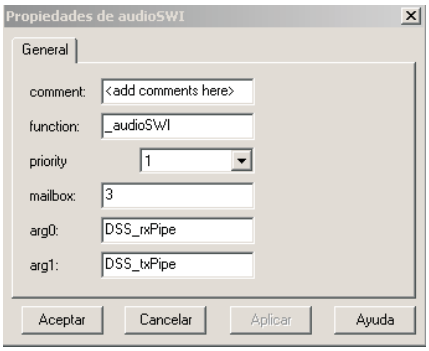

Figura 4.3: Propiedades de audioSWI.

11. En el *pop_up menu*, dar click en el + de *Scheduling* y dar click-derecho sobre el *SWI_Software Interrupt Manager* y escoge *Insert SWI*. Renombrar el nuevo *SWI0 object* **audioSWI**.

12. Dar click-derecho sobre **audioSWI** y seleccionar *Properties* del menu. En el **audioSWI** properties window, como en la figura (4.3), en **function** escribe **_audioSWI**, 3 para el **mailbox**, **DSS_rxPipe** para **arg0**, y **DSS_txPipe** para **arg1**. Dar click **aceptar** para salvar tus cambios.

13. En el *pop_up menu*, dar click en el + de *Input_output* y dar click-derecho sobre la *PIP_Buffered Pipe Manager* y escoge *Insert PIP* dos veces. Renombra el primer *pipe* para **DSS_rxPipe** y al segundo *pipe* **DSS_txPipe**.

14. Dar click-derecho sobre **DSS_rxPipe** y selecciona *Properties* del menu. Mete las si-guientes propiedades para **DSS_rxPipe:** como en la figura (4.4).

Figura 4.4: Propiedades de DSS_rxPipe.

15. Dar click-derecho sobre **DSS_txPipe** y selecciona *Properties* del menu. Mete las si-guientes propiedades para **DSS_txPipe:** figura (4.5). Cuando la trama esta llena en *DSS_rxPipe* se pone la función **notifyReader** que limpiara el segundo bit en el *mailbox* para activar el **audioSWI**. Y cuando una trama vacía esta libre en el *DSS_txPipe*, pone el primer bit en el *mailbox* para que el **audioSWI** sea limpiado. De esta manera, audioSWI es activado sólo cuando hay una trama completa disponible en *DSS_rxPipe* y una trama vacía disponible en *DSS_rxPipe*.

    El **notifyWriter** de *DSS_rxPipe*, **DSS_rxPrime**, es una función de C, puede ser encontrado en dss.c.

    DSS_rxPrime llama al **PIP_alloc** para destinar una trama vacía de **DSS_rxPipe** que será usada por el ISR para escribir los datos recibidos del codec. *DSS_rxPrime* es llamado siempre que una trama vacía esta disponible en *DSS_Pipe* (el ISR fue hecho en la trama anterior). El ISR llama *DSS_rxPrime* después que se llena la trama, vea la figura (4.6.

    El **notifyReader** de *DSS_txPipe*, **DSS_txPrime**, es una función de C que se encuentra en el programa dss.c. *DSS_txPrime* llama **PIP_get** que consigue una trama llena para *DSS_txPipe*. Los datos en esta trama serán transmitido por el ISR al codec. El *DSS_txPrime* es llamado siempre que una trama llena esta

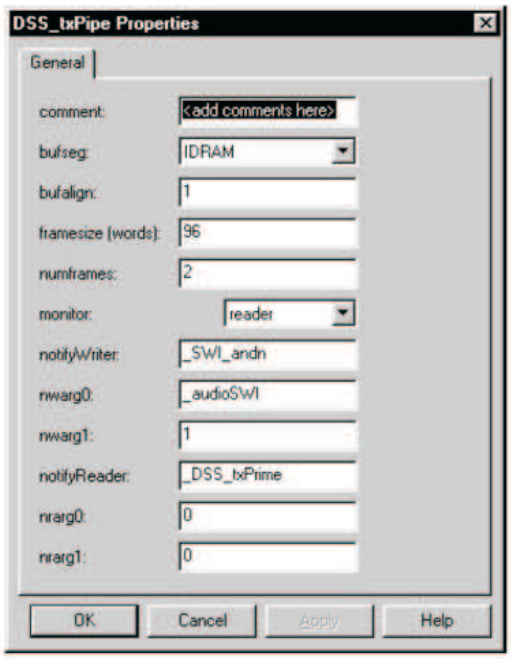

Figura 4.5: Propiedades de DSS_txPipe.

disponible en *DSS_txPipe* y el ISR ha transmitido los datos previos a la trama. El ISR llama al *DSS_Prime* después que se ha transmitido una trama para conseguir la siguiente trama llena, vea la figura (4.6). Ahora se necesita conectar el ISR correspondiente para el puerto serial.

16. Dar click-derecho en (+) a lado de *HWI_Hardware Interrupt Service Routine Manager*. Cada uno de estos objetos corresponden a un interruptor en el TMS320C6X *interrupt vector table*. Dar click_derecho sobre **HWI_INT11** y selecciona *properties*. Escoge la fuente del interruptor que corresponda al *interrupt Multichannel Buffered* del puerto serial de entrada 0 (MCSP_0_Receive), y también cambia la función *_DSS_isr* como sigue: figura (4.7).

    Entrando el *_DSS_isr* en la función de campo, En el *DSP/BIOS* se colocará el interruptor TMS320C6x *vector table* para *_DSS_isr* que maneja el interruptor del puerto serial de entrada.

17. Dar click-derecho sobre *SWI_Software Interrupt Manager Object* y selecciona *Properties* del menu. Selecciona microseconds en el Statistics Unit field. Esto hará que las herramientas de análisis en tiempo-real de *DSP/BIOS* visualicen los datos estáticos para el software de interrupción en microsegundos.

18. Salva el archivo de configuración como **audio.cdb** en $File \rightarrow Save\ as$. Si pregunta

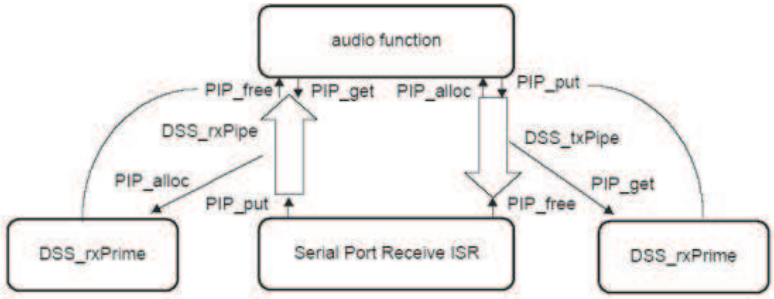

Figura 4.6: DSS_rxPrime y DSS_txPrime.

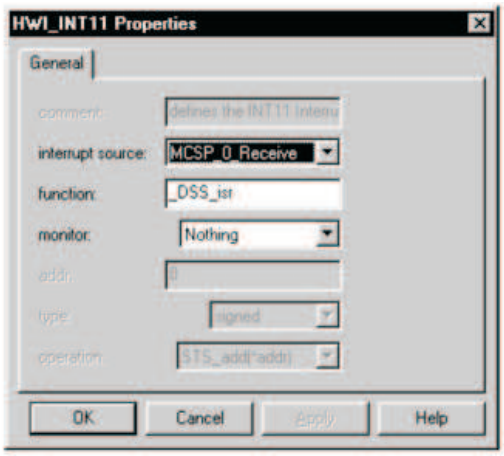

Figura 4.7: Propiedades de HWI_init11.

por reemplazar el original; dar click en aceptar.

19. Escoge *Project* → *Add Files to Project*.

    Selecciona los archivos de configuración (*.cdb). Y selecciona el archivo **audio.cdb** y da click en abrir. Fíjate que el *Project View*(visor del proyecto) ahora contiene audio.cdb en una carpeta llamada *DSP/BIOS Config*. Además, el archivo audiocfg.s62 está ahora listado como un archivo de *Generated Files*.

20. El archivo de salida debe coincidir con el archivo .cdb (audio.out y audio.cdb) Ve a *Project* → *build options* y escoge *Linker*. Verifica el nombre del campo de saluda como **audio.out**[6].

21. Escoger *Project* → y dar un click en *Rebuild All*.

Revisando el programa

Revisa el programa audio en audio.c, Fíjate que el programa audio tiene una trama llena de *DSS_rxPipe* y una vacía de *DSS_txPipe*, y copia el contenido de la trama de entrada a la de salida. La función audio sólo correrá cuando este llena una trama disponible en DSS_rxPipe y una trama vacía disponible en *DSS_txPipe*.

4.2. Depurando y probando con las herramientas de análisis en tiempo real de DSP/BIOS

Para correr este ejemplo necesitaras conectar la entrada del codec a alguna señal acústica. Por ejemplo, si tu computadora tiene un CD-ROM, puedes usarlo para tocar un CD y conectar la salida de los audífonos al puerto de entrada de la tarjeta. La salida de la tarjeta necesita estar conectado a algún aparato de salida, como una bocina.

Una vez que la tarjeta esta conectada a un aparato de salida y de entrada, ahora inicia la entrada de señal (ejecuta play al reproductor).

1. Choose *File* → *Load program*. Selecciona el programa que acabas de compilar, audio.out, dale click en abrir.

2. Escoge *Debug* → *Go Main*. El programa correrá la primer declaración en la función *main*.

3. Escoge *DSP/BIOS* → *RTA Control Panel* como muestra la figura (4.8).

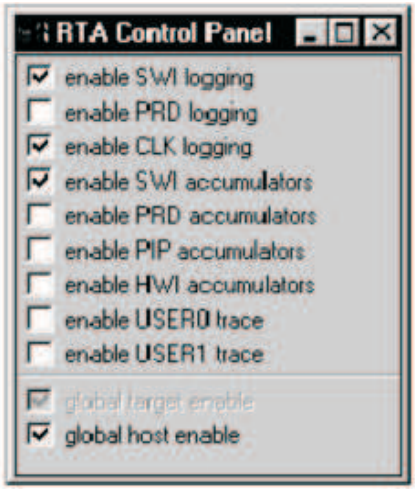

Figura 4.8: Propiedades del RTA.

4. Dar click-derecho sobre el área que contiene las cajas de visualización (*check boxes*) y inhabilita *Allow Docking*, o selecciona *Float* en la ventana Main, para visualizar por separado la ventana de panel de control de RTA. Cambia el tamaño de la ventana para que puedas verlo todo lo que las *check boxes* te muestran.

5. Marcas las cajas que se describen abajo para habilitar SWI y CLK logging, *SWI accumulators* y *globally enable tracing* sobre el host.

6. Escoge $DSP/BIOS \rightarrow CPU\ Load\ Graph$.

7. Escoge $DSP/BIOS \rightarrow Execution\ Graph$. La gráfica de ejecución (*Execution Graph*) aparece con un botón de la ventana del Code Composer Studio. Tal vez quieras cambiar el tamaño o visualizar por separado la ventana.

8. Dar click-derecho sobre RTA Control Panel y escoge Property Page del *pop_up menu*.

9. Verifica que Refresh Rate of Messege Log/Execution Graph sea 1 segundo y dale click en aceptar.

10. Escoge $Debug \rightarrow Run$ o dale click en (Run) en toolbar. Deberás poder oír una señal acústica en la salida de la bocina. La ejecución grafica será similar a esta, observa la figura (4.9):

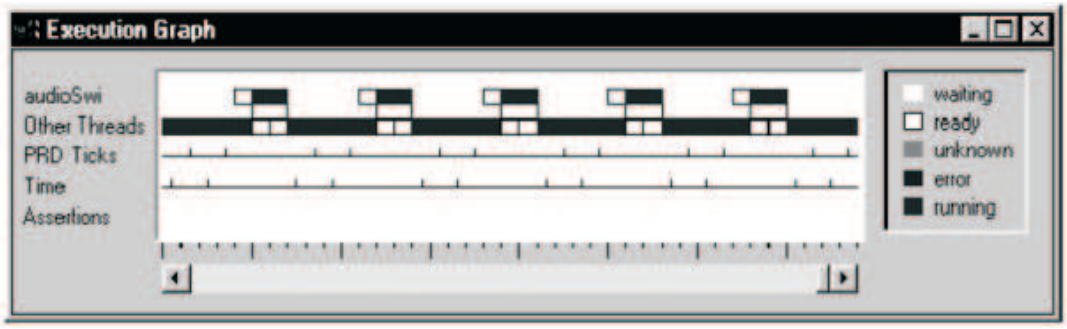

Figura 4.9: Grafica de ejecución.

11. Cuenta las marcas entre los tiempos de **audioSWI** que está corriendo. Debe correr cada 2 milisegundos. Podrás ver que entre dos ejecuciones consecutivas de *audioSWI* hay dos sistemas de reloj (cada uno correspondiente a un milisegundo). La función tiempo correrá cada tiempo que halla una trama llena *DSS_rxPipe* y una vacía *DSS_txPipe*. El ISR esta corriendo cada 1/48000 microsegundos.

4.3. Controlando el periodo

Vamos a sumarle una nueva función para que nuestra aplicación corra periódicamente a intervalos de 8 milisegundos. El código para esta función, llamado **load**, esta en audio.c.

loadVal es una variable global que se inicializa poniéndose en **0.AUDIO_load** es una rutina de ensamblador que simula un proceso que usa las partes altas del ciclo del CPU. Puedes hallar el código fuente para *AUDIO_load en AUDIO_LD.S62.*

Usando las herramientas de análisis en tiempo real, puedes poner cual va a ser la carga del CPU cambiando el valor de la variable global **loadVal**.

1. En Code Coposer Studio, abre **audio.cdb**. click -derecho sobre el *Periodic Function Manager* y selecciona **insert PRD** desde el menu.

2. Renómbralo a **PRD** como **loadPrd**. Dar click-derecho sobre él y selecciona *Properties* desde el menu. Mete las siguientes propiedades para este periodo como muestra la figura (4.10)

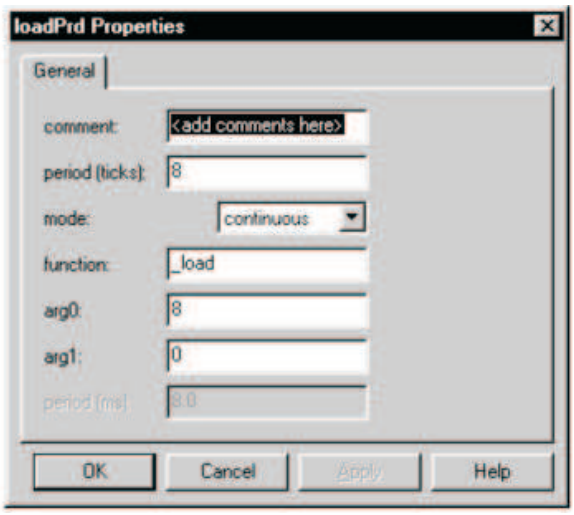

Figura 4.10: Propiedades de loadPRD.

3. Escoge *Project* → *Rebuild All* y *rebuild* audio.out.

Corriendo el programa con la función Periodic

1. Escoge *File* → *Reload Program* en Code Composer Studio para releer audio.out.

2. Escoge *debug* → *Run*.

3. Escoge $View \rightarrow WatchWindow$. En Watch Window, Insert New Expression **loadVal**.

4. En la ventana Trace State, apaga **CLK** logging. Click sobre **PRD** logging.

5. Observa la ventana de Execution Graph. Verás la función loadPRD ejecutada cada 8 ticks de reloj del sistema. En la ventana **STS** para el audio SWI, veras la notificación de que el máximo valor para la señal es por abajo de 2 milisegundos de plazo.

6. En la ventana de Data Memory , dar doble click sobre loadVal y edita la variable a 100. después deberás ver que la carga de la computadora aumenta. La función AUDIO_load simula una carga de la computadora que es 1000*loadVal.

7. Si pusiste el valor mas grande a la variable loadVal, fíjate que la CPU Load graph y la Execution Graph se detuvieron, esto es porque las actualizaciones son hechas con un idle task, que es el que tiene el valor mas bajo de prioridad en la ejecución del programa. Porque el valor mas alto esta siendo usado en el tiempo de procesado, no hay suficiente tiempo para el *control host* en las actualizaciones que han sido hechas. El programa ahora ésta perdiendo mucho de los valores en tiempo real. Poniéndole un valor mas chico en loadVal la comunicación continuara.

8. Observa el dato **STS** para audioSWI. El incremento en la carga de la función *periodo* se incrementará al máximo.

9. Al incrementarse loadVal entrando valores nuevos en la ventana Data Memory. Intenta 150,200,etc. Observa el incremento sobre la carga del CPU y el máximo campo para **audioSWI STS.** Observa como eventualmente en el audioSWI STS sus máximas rectas son mas de 2 milisegundos, y la aplicación comienza a perder tiempo real. Te darás cuenta de que la calidad de la señal acústica en la salida va decayendo en calidad.

10. En la ventana System log, observa como el incremento en el valor loadVal que esta prolongada a través del tiempo es tomada de la variable loadPRD para ejecutarse. (Podrás ver esto el número de ticks de reloj que pasan mientras corres loadPRD).

Incrementando el número de tramas

Un incremento en la carga de loadPRD hace que se pierda tiempo-real, como loadPRD tardo en finalizarse, hizo que audioSWI inicie su ejecución tarde, porque audioSWI tiene que esperar hasta que loadPRD allá terminado. Con sólo 2 tramas, el ISR debe terminar el llenado (o transmisión) de una de las tramas antes de que se active audioSWI, retrasado por loadPRD, que ha de copiado la otra trama y liberarado el conducto. Como

resultado, la interrupción pasa antes de que una nueva trama este disponible, causando la perdida de datos por el ISR. Para solucionar este problema puedes incrementar el número de tramas en cada conducto. Para que este otra trama disponible para el llenado del ISR aunque hay un retraso en audioSWI.

1. Abre **audio.cdb** adentro de Code Composer Studio.
2. Abre el Buffered Pipe Manager, y resalta DSS_rxPipe. Da click-derecho para abrir la ventana Properties y cambiar el número de tramas en **numframes** a 3. Da click en aceptar y salva los cambios.
3. Repite el mismo procedimiento con el DSS_txpipe.
4. Escoge *Project* → *Rebuild All* y guarda los cambios.
5. Reload audio.out y escoge *debug* → *Run*.
6. Escoge *View* → *Memory*, escribe el valor loadVal que tenías escrito antes.
7. Observe la ventana **STS** ¿Está audioSWI llegando a los límites?.
8. Sigue incrementando loadVal. La aplicación tendrá eventualmente el mismo problema. Y audioSWI comenzará a perder tiempo-real.

Corrigiendo las prioridades

Al añadir otro buffer parecerá que se resuelve la situación, mientras loadVal se hizo mas larga, y la aplicación pierde tiempo-real otra vez. ¿Porqué pasa esto?.

Cuando fue llamado audioSWI, no corrió inmediatamente porque tiene que esperar que loadPRD termine. Ahora comienza a tardar más de 2 milisegundos para ejecutarse, eso causa que el audioSWI espere demasiado tiempo, hasta llegar a su limite de tiempo que es 2 milisegundos. Cuando le sumamos un nuevo buffer, incrementamos la cantidad de tiempo que audioSWI se retrasará, y como hay un buffer extra hace que ISR continue llenándose. Sin embargo, este tipo de arreglo no nos ayudará en definitiva. Si loadVal se vuelve lo suficientemente largo, eventualmente audioSWI será retrasado lo suficiente para que afecte al ISR y haga que corra sin tramas disponibles para el llenado.

Para resolver esta situación necesitamos que audioSWI pueda adelantar el periodo del software de interrupción(**PRD_swi**) que llama a loadPRD. Si audioSWI puede adelantar PRE_swi, audioSWI correrá sin importar si esta siendo detenido, a pesar de que loadPRD ha terminado, porque le quitaremos el control del periodo a la CPU adelantando la función periodic. Desde que audioSWI ya no esta retrasado por loadPRD, la carga de loadPRD no hará que la aplicación pierda tiempo-real.

1. Abre **audio.cdb** adentro de Code Composer Studio. Da click sobre Software Interrupt Manager y resáltalo.

2. Sobre el panel derecho, arrastra audioSWI sobre PRE_swi.audioSWI vuélvelo la más alta prioridad del software imterruptor.

3. Escoge *Project* → *RebuildAll* y salva los cambios.

Realiza los pasos de la sección previa y observa como audioSWI ya no pierde tiempo cuando la función loadVal es incrementada. Observa en la ventana Message Log como audioSWI adelanta a loadPRD. Fíjate en la ventana **STS** como para audioSWI el máximo permanece constante a pesar de los cambios de loadVAl.

4.4. Resultados

Resumiendo los tres los pasos que llevaron acabo son los siguientes:

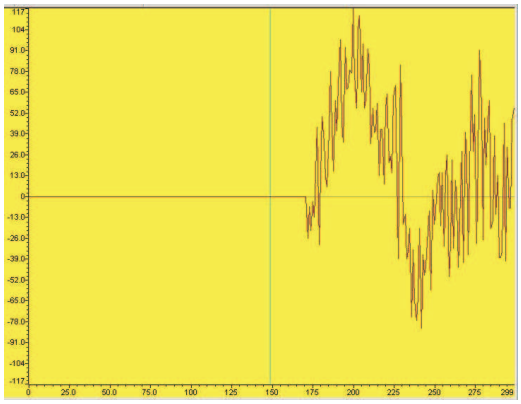

Figura 4.11: Gráfica de entrada del audio.

En primer lugar se conecto la tarjeta DSP a la computadora por el puerto serial, para poder correr el programa Code Composer Studio, el cual posee la aplicación DSP/BIOS API's con la cual se programó la tarjeta DSP para la transferencia de datos entre el hardware (periféricos l/O) y la tarjeta DSP. Se mostró como usar las herramientas de configuración del *DSP/BIOS* y como usar las herramientas de análisis en tiempo real del DSP/BIOS. Se eliminaron fallos y se probó el código del programa.

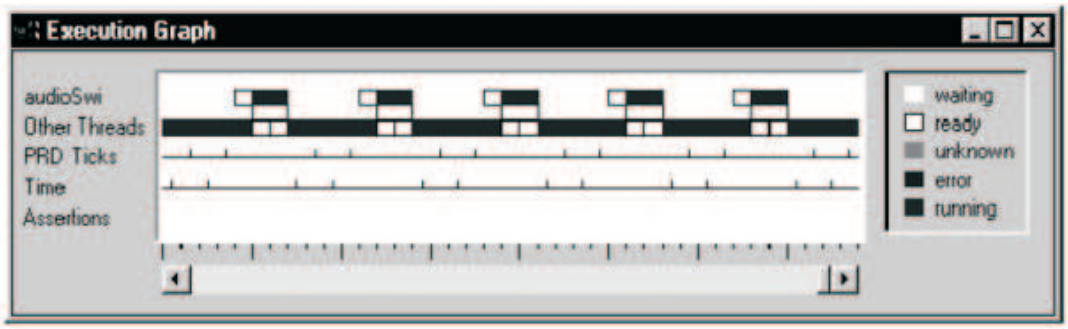

Figura 4.12: Gráfica de procesamiento de audio.

En segundo lugar, ya programado el DSP con la aplicación DSP/BIOS API's, y habiéndose inicializado por medio de la computadora el programa audio para realizar el procesamiento, se conectaron antes de correr el programa los dispositivos de entrada

(el reproductor de audio) y salida (las bocinas) de señal al DSP a través de los puerto I/O localizados a los lados de la tarjeta, Al desarrollarse la practica la señal de entrada figura (4.11) que en este caso fue una canción, fue procesada por el DSP escuchándose el audio con buena calidad a través de las bocinas, figura (4.12).

En tercer lugar se demostró que, una vez programado el DSP a través de la computadora, el procesamiento que realiza el DSP es independiente de la computadora, es decir la tarjeta DSP puede ser desconectada del puerto paralelo de la computadora. Esto es, sin la conexión entre ellas el DSP sigue trabajando, hasta que sea reseteado por medio de un botón que se encuentra en la tarjeta[6].

Capítulo 5

Conclusiones y trabajos futuros

Después de toda la investigación y prácticas que se hicieron para la resolución del problema planteado al principio de esta tesis, se logró hacer los apuntes necesarios para que el lector pueda entender los principios básicos de la señales, señal de audio, conversión analogico-digital y tiempo real. Y que además conozcan el hardware para el procesamiento de señales, especificamente el DSP TMS320C6711 de Texas Instrument, el cual se eligio por tener más velocidad de procesamiento y facilidad de uso y de programación que un Microprocesador, así como, el software Code Composer Studio (CCS) creado para programar este hardware y la forma de manejo de la aplicación DSP/BIOS API's que pertenece al software CCS, y se utiliza en el desarrollo de programas de adquisición de la señal a través del puerto paralelo y que es configurado por medio de objetos que estan preprogramados en lenguaje ensamblador o C++.

Y principalmente se logró procesar una señal de audio en tiempo real con el DSP, con una buena calidad de señal de salida, demostrando que el DSP sólo depende de la computadora para ser programado e inicializado el programa mas no para la adquisición y transmisión de la señal.

En los trabajos futuros se propone desarrollar con el DSP/BIOS API's programas que apliquen diversos algoritmos para el procesamiento digital de cualquier tipo de señales, sean unidimensionales o multidimencionales en tiempo real en el DSP, como ejemplo específico el aplicar un algoritmo que sirva de filtro para mejorar la transmisión de audio en tiempo real, eliminando el mayor ruido posible que tenga la señal.

Glosario

- **Ciclos de reloj**

  Una secuencia de eventos periódicos basados en la entrada de un reloj externo.

- **Codec**

  Compresión/descompresión, dispositivo que codifica en una dirección de transmisión y decodifica en otra.

- **configuration file**

  Un archivo que almacena objetos y propiedades para el empleo con DSP/BIOS y el Chip Support Library (CSL).

  El instrumento de configuración DSP/BIOS y sus archivos de configuración no deberían ser confundidos con otros artículos usados para la configuración dentro del estudio de compositor de código. Estos otros artículos incluyen la configuración de proyecto (típicamente el ajuste o la liberación), la ventana de Control de Configuración RTDX, y el sistema de configuración de sistema en el instrumento de Sistema CCS.

- **ensamblador**

  Software que crea un programa en lenguaje máquina de un archivo fuente que contiene instrucciones de lenguaje ensamblador, directivas y definiciones de macros. El ensamblador sustituye códigos de operaciones absolutas o códigos de localización por direcciones simbólicas.

- **frame**

  La trama es el espacio de 8 palabras en la cáche RAM. Cada paquete de búsqueda en la cáche reside en una sola trama. La cáche actualiza las cargas con el paquete de búsqueda. La cáche contiene 512 tramas.

- **heap**

  Un fondo de direcciones de lo cual la memoria puede ser asignada en la duración de explotación que usa las funciones de MEM. Este fondo de memoria es configurado usando el Instrumento de Configuración DSP/BIOS.

- **idle function**

  Un tipo de thread del DSP/BIOS. Llama a las funciones idle repetidamente cuando DSP/BIOS determina que algún otro thread (no ocioso) está listo a correr. Funciones idle pueden ser añadidas y configuran la utilización del gerente IDL.

- **interrupción**

  Señal enviada por el hardware o software para solicitar la atención del procesador. La señal le indica al procesador que suspenda su operación presente, guarde el estado de la tarea presente, y ejecute un conjunto particular de instrucciones. Las interrupciones se comunican con el sistema operativo y establece un orden de prioridades de las tareas para ejecutarse.

- **Interrupt Service Routine (ISR)**

  Una función que es ejecutada por el target CPU en respuesta a un acontecimiento externo. Usa el módulo HWI para configurar funciones para correr en respuesta a las interrupciones individuales.

- **kernel**

  Es la parte fundamental de un sistema operativo. Es el software responsable de facilitar los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

- **memory section**

  Una parte del archivo ejecutable. Esto es un bloque de código o datos que en última instancia ocupan el espacio contiguo sobre el objetivo. Los ejemplos de secciones de memoria incluyen son .text, .data, y .bios.

  Al contrario un segmento de memoria es una partición de memoria llamada que corresponde a una gama física de memoria sobre el objetivo. Los ejemplos de segmentos de memoria incluyen IPRAM Y IDRAM.

- **memory segment**

  Una memoria llamada partición que corresponde a una gama física de memoria sobre el objetivo. Los ejemplos de segmentos de memoria incluyen IPRAM Y IDRAM.

  El gerente MEM en las plantillas de configuración proveídas de DSP/BIOS contiene un juego de segmentos de memoria. Las propiedades de cada objeto de MEM definen la posición (ubicación) baja y la longitud de la sección. Además, mucha memoria dinámica puede ser configurada en algunas secciones de memoria. Al contrario una sección de memoria es una parte del archivo ejecutable. Los ejemplos de secciones de memoria incluyen son .text, .data, y .bios.

- **module**

  DSP/BIOS es dividido en un juego de módulos. Cada módulo típicamente tiene alguna función API, puede ser configurado en el Instrumento de Configuración DSP/BIOS, y puede ser observado en las herramientas de análisis DSP/BIOS. Dentro de DSP/BIOS, los terminos **module** Y *manager* significan esencialmente lo mismo.

- **object**

  Un caso de la clase definida por un módulo. Por ejemplo, el módulo QUE permite para crear objetos QUE, que puedan ser usados como estructuras de queue.

- **pend**

  Esperar un recurso sea disponible. En DSP/BIOS, sólo los task threads pueden usar el pend sobre un recurso

- **periodic function**

  Una función que corre a intervalos regulares. En DSP/BIOS, estas funciones son creadas y manejadas por el módulo PRD.

- **pipe**

  Una estructura usada en los buffer streams de datos de salida y entrada. En DSP/BIOS, son tubos creados y manejados por el PIP módulo.

- **post**

  Para los task threads, se esta fijando si un recurso se hace disponible. Si un task threads es obstruido mientras espera por el recurso, esto se puede resumir corriéndolo. Ya que un software interrupt, la función de objetos de SWI es puesta a punto para la ejecución cuando el objeto de SWI es fijado por una llamada de API.

- **priority**

  Un valor asignado a un task o el software de interrupción para controlar cuales threads pueden apropiar (acaparar) otros threads.

- **real-time analysis**

  La captura en tiempo real y demostración de datos usados para la temprana detección y diagnóstico de bichos a nivel de sistema.

- **scheduler**

  La parte del kernel responsable de determinar que thread debería correrse después. En DSP/BIOS, maneja la separación de schedulers interrupt (HWI y SWI) y el task (TSK y IDL) threads.

- **software interrupt**

  Un tipo de thread del DSP/BIOS. El Software Interrupt son las funciones de priorizadas que se apropian (acaparan) los task del DSP/BIOS y son apropiadas (acaparadas) por el hardware interrupts y el software interrupts de prioridad más alta. Ellos son creados y manejados por el módulo SWI.

- **stream**

  Un stream es una secuencia continua de datos en tiempo real.

- **task**

  Un tipo de thread del DSP/BIOS. Los tasks son las funciones de prioritizan o que se apropian (acaparan) del idle loop del DSP/BIOS y son apropiadas (acaparadas) por el hardware y el software interrupts. Ellos son creados y manejados por el módulo TSK. Los tasks son el único tipo de thread del DSP/BIOS que puede obstruirse.

- **thread**

  Un término general usado para referirse a cualquier thread de ejecución. Un thread es una schedulable unidad de código. En DSP/BIOS esto incluye el hardware interrupts, el software interrupts, tareas, funciones idle, y funciones periodic.

Bibliografía

[1] Manolakis D. G., Proakis J. G., *Tratamiento Digital de Señales*. Prentice Hall, 1998.

[2] Srinath M. D., Soliman S. S., *Señales y Sistemas continuos y discretos*. Prentice Hall, 1999.

[3] Oppenheim A. V.,Willsky A. S., *Señales y Sistemas*. Prentice Hall, 1997.

[4] *TMS320C6x User´s guide,digital signal processing products*, Texas Instruments, october 1994.

[5] *Texas Instruments Code Composer Studio v.2*

[6] Texas Instruments *An audio example using DSP/BIOS*, november 1999.

[7] Texas Instruments *Code Composer Studio IDE Quit Start*, february 2001.

[8] Texas Instruments *TMS320C6000 CPU and Instruction Set Reference Guide*, octuber 2000.

[9] Texas Instruments *DSP/BIOS Quick Start Reference Guide*, february 2001.

[10] Internet FAQ Archives Online Education, Realtime-Computing: Frequently Asked Questions (FAQs) (version 3.6) http://www.faqs.org/faqs/realtime-computing/faq/

[11] Escuela Técnica Superior de Ingenieros de Telecomunicación E.T.S.E.T., España, Sistemas Operativos Distribuidos y de Tiempo Real, Tema 10 Introducción a los Sistemas de Tiempo Real, Rodríguez Pedro http://www-gti.det.uvigo.es/p̃edro/pub/sodtr/pdf/tema_10.pdf

[12] University at Buffalo, The State University New York, USA, Computer Science and Engeneering Department CSE, Multiprocessor and Real-Time Scheduling, Tema 10, Real-Time System, Bina Ramamurthy http://www.cse.buffalo.edu/faculty/bina/cse421/spring00/lec10/index.htm