ESTUDIO DE LOS PROCESADORES DIGITALES DE SEÑALES PARA EL DESARROLLO DE APLICACIONES EN TIEMPO REAL

REINALDO RODRIGUEZ VILLALOBOS
KENNY FAJARDO SUAREZ

DIRECTORA
SONIA HELENA CONTRERAS ORTIZ

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA ELECTRICA Y ELECTRONICA
CARTAGENA DE INDIAS D. T. Y C.
2006

ESTUDIO DE LOS PROCESADORES DIGITALES DE SEÑALES PARA EL DESARROLLO DE APLICACIONES EN TIEMPO REAL

REINALDO RODRIGUEZ VILLALOBOS
KENNY FAJARDO SUAREZ

Trabajo de monografía presentado como requisito para optar al titulo de Ingeniero Electricista

DIRECTORA
SONIA HELENA CONTRERAS ORTIZ
MAGISTER EN POTENCIA ELÉCTRICA

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA ELECTRICA Y ELECTRONICA
CARTAGENA DE INDIAS D. T. Y C.
2006

Articulo 105

La Universidad Tecnológica de Bolívar se reserva el derecho de propiedad de los trabajos de grado aprobados y no pueden ser explotados comercialmente sin autorización.

Nota de aceptación

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del Jurado

_____

Firma del Jurado

Cartagena, Enero de 2006

Cartagena D. T. Y C., Enero de 2006

Señores

COMITÉ DE EVALUACIÓN DE PROYECTOS
Programa de Ingeniería Eléctrica y Electrónica

UNIVERSIDAD TECNOLOGICA DE BOLIVAR
La ciudad

Respetados señores:

Con toda atención nos dirigimos a ustedes con el fin de presentarles a su consideración, estudio y aprobación la monografía titulada ESTUDIO DE LOS PROCESADORES DIGITALES DE SEÑALES PARA EL DESARROLLO DE APLICACIONES EN TIEMPO REAL como requisito parcial para optar al titulo de Ingeniero Eléctricista

Atentamente

_____ _____

REINALDO RODRIGUEZ V. KENNY FAJARDO SUAREZ

Cartagena D. T. Y C., Enero de 2006

Señores

COMITÉ DE EVALUACIÓN DE PROYECTOS
Programa de Ingeniería Eléctrica y Electrónica
UNIVERSIDAD TECNOLOGICA DE BOLIVAR
La ciudad

Cordial saludo:

A través de la presente me permito entregar la monografía titulada ESTUDIO DE LOS PROCESADORES DIGITALES DE SEÑALES PARA EL DESARROLLO DE APLICACIONES EN TIEMPO REAL para su estudio y evaluación la cual fue realizada por los estudiantes REINALDO RODRIGUEZ VILLALOBOS y KENNY FAJARDO SUAREZ, de la cual acepto ser su director.

Atentamente,

_____

SONIA HELENA CONTRERAS ORTIZ
Magíster en Potencia Eléctrica

AUTORIZACIÓN

Yo REINALDO RODRIGUEZ VILLALOBOS, identificado con la cedula de ciudadanía numero 73.201.297 de Cartagena, autorizo a la universidad tecnológica de Bolívar, para hacer uso de mi trabajo de monografía y publicarlo en el catalogo on-line de la biblioteca

_____

REINALDO RODRIGUEZ VILLALOBOS

AUTORIZACIÓN

Yo KENNY FAJARDO SUAREZ, identificado con la cedula de ciudadanía numero 73.203.108 de Cartagena, autorizo a la universidad tecnológica de Bolívar, para hacer uso de mi trabajo de monografía y publicarlo en el catalogo on-line de la biblioteca.

_____

KENNY FAJARDO SUAREZ

GLOSARIO

ADC: Analog-Digital Converter. Conversor Análogo Digital

DAC: Digital Analog Converter. Conversor Digital Análogo

DSP: Digital Signal Processor . Procesador digital de señal.

CPU: Central Process Unit. Unidad Central de Proceso

ALU: Arithmetic/Logic Unit. Unidad aritmético-lógica. Unidad de ejecución de un procesador responsable de la aritmética (suma, resta, desplazamiento) y de la lógica (AND, OR, XOR).

DMA: Direct Memory Access. Técnica de acceso directo a memoria

CISC: Complex Instruction Set Computer. Computadores de conjunto de instrucciones complejo.

RISC: Reduced Instruction Set Computer. Computadores de conjunto de instrucciones reducido.

VLIW: Very long instruction word. Formato muy largo de palabra de instrucción.

SIMD: Single Instruction Multiple Data. Instrucción Simple, Datos Múltiples.

MAC: Multiply-accumulate per second. Número de multiplicaciones y acumulaciones por segundo.

MIPS: Million instructions Integer per second. Millón de instrucciones Enteras por segundo.

MFLOPS: Million floating-point operations per second) Millón de operaciones en punto flotante por segundo.

FFT: Fast Fourier Transform. Transformada de Fourier rápida. Método computacional para estima el espectro en frecuencia de una señal.

FIR: Finite impulse response. Respuesta al impulso finita. Categoría de filtros digitales.

IIR: Infinite impulse response. Respuesta al impulso infinita. . Categoría de filtros digitales.

GPP: General Purpose Processor. Procesador de Propósito General

PLL: Phase-Locked Loop. Circuito de enganche de fase.

ASIC: Application Specific Integrated Circuits. Circuitos Integrados de Aplicación Específica.

FPGA: Field Programmable Gate Arrays. Los Arreglos de compuertas programables en campo

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

Pág.

INTRODUCCIÓN

El procesamiento digital de señales es la ciencia de la ingeniería que se dedica al análisis de señales del mundo real, como el audio, la voz, video, entre otras, utilizando técnicas matemáticas para mejorar, modificar y extraer información de esa señales. Muchos cambios se han obtenido a partir del procesamiento digital de señales en diferentes campos: en comunicaciones, medicina, radar y sonar, reproducción de música de alta calidad, entre otros.

Los procesadores digitales de señales o DSPs (sigla en inglés de *Digital Signal Processor*) son microprocesadores específicamente diseñados para el procesamiento digital de señales. Algunas de sus características básicas como el formato aritmético, la velocidad, la organización de la memoria o la arquitectura interna hacen que sean o no adecuados para una aplicación en particular.

Dado el auge que tiene el uso de los DSPs en gran cantidad de aplicaciones industriales, a nivel de investigación y de electrónica de consumo, por medio de este trabajo se pretende realizar un estudio general de las características, arquitecturas, criterios de selección, diferencias con otros tipos de procesadores que algunas veces se emplean en tareas de procesamiento de señales y en fin, todo lo correspondiente a la implementación de los DSPs en el desarrollo de aplicaciones en tiempo real.

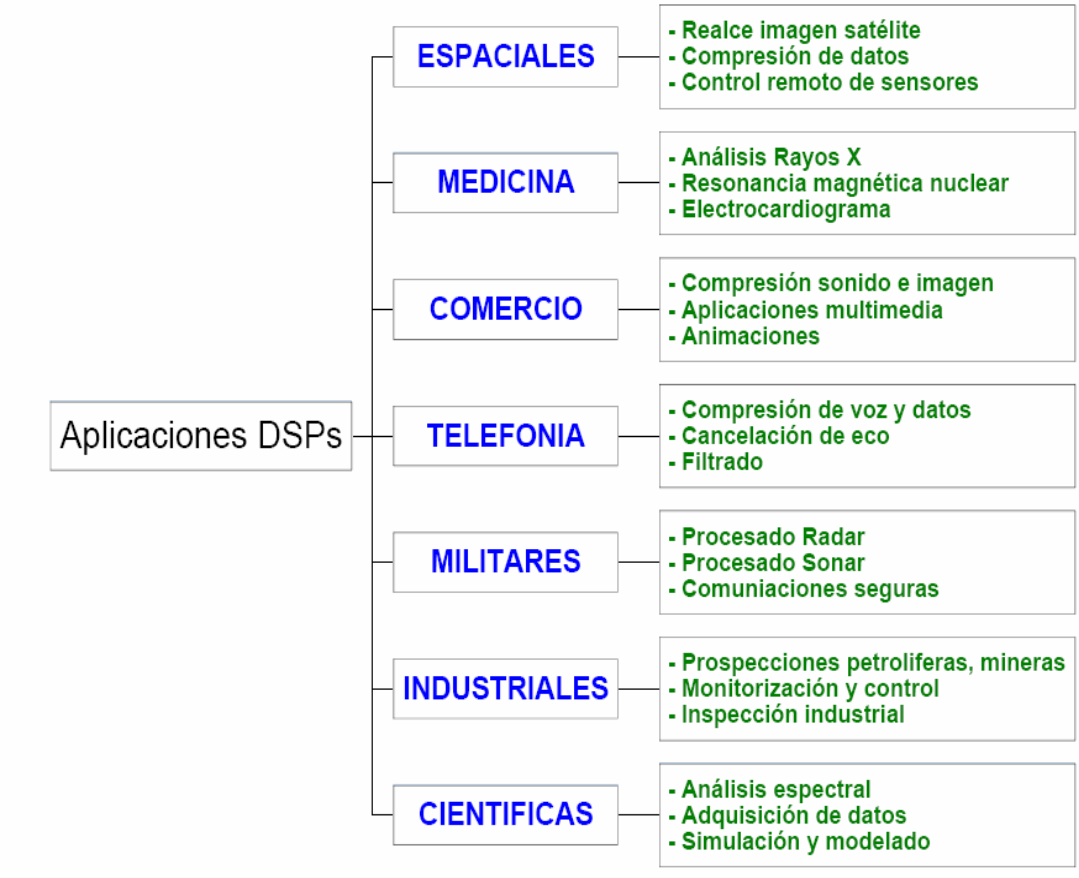
1. EVOLUCIÓN DEL PROCESAMIENTO DIGITAL DE SEÑALES

El procesamiento digital de señales (DSP, sigla en ingles de Digital Signal Processing) es una de las tecnologías más poderosas de la ciencia e ingeniería en el siglo XXI. Se distingue de otras áreas por el tipo único de datos que utiliza: ***señales.*** Estas señales se originan como datos sensoriales del mundo real: vibraciones sísmicas, imágenes, ondas acústicas, entre otras. El procesamiento digital de señales es la matemática, los algoritmos, y las técnicas usadas para manipular estas señales después de que se hayan convertido en forma digital. Esto incluye una variedad amplia de aplicaciones, por ejemplo: perfeccionamiento de imágenes, reconocimiento y generación de la voz, compresión de los datos, radar, sonar, entre otras. Las raíces de DSP datan de los años 60 y 70 cuando surgieron los primeros computadores digitales. Las computadoras eran costosas durante esta era, por ende el procesamiento digital de señales fue limitado solamente a algunos usos críticos:

Radar y sonar: Cuando la seguridad nacional estaba en riesgo.
Exploración de petróleo: Donde se invertían grandes cantidades de dinero.
Exploración del espacio: Por el uso complejo de los datos.
Proyección de imágenes médicas: Debido a que se podían salvar vidas.

La revolución de los computadores personales en los años 80 y 90 hizo que el procesamiento digital de señales explorara nuevas aplicaciones. Después de emplearse en la industria militar y gubernamental, su uso se extendió hacia el mercado comercial. Es entonces cuando aparecieron productos al alcance del público tales como: teléfonos móviles, reproductores de discos compactos, y correo de voz electrónico. La figura 1.1 muestra algunas de las aplicaciones.

Figura 1.1. Aplicaciones del procesamiento digital de señales



Después de los años 80, el procesamiento digital de señales se incluyó como curso complementario en ingeniería eléctrica y electrónica. Una década más tarde, formaba parte del núcleo básico del plan de estudios de la mayoría de estudiantes en el mundo. Hoy el procesamiento digital de señales es una necesidad básica para los científicos e ingenieros en muchos campos[1].

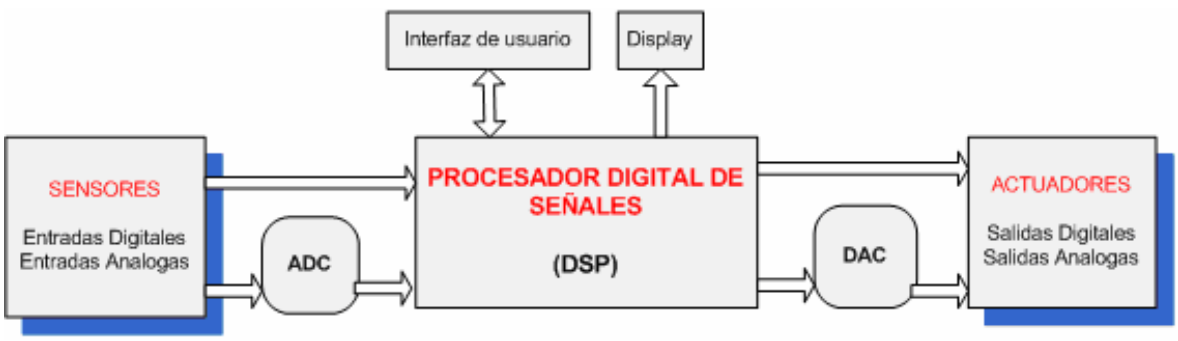
[1]SMITH, Steven W. The scientist and engineer's guide to digital signal processing. 2 ed.California.

1.1 Definición de un Procesador Digital de Señales (DSPs)

Un Procesador Digital de Señales o DSP es un procesador cuyo hardware, software, y conjunto de instrucciones están optimizados para aplicaciones que requieren de operaciones numéricas a altas velocidades, esenciales en el procesamiento de datos digitales que representan señales analógicas en tiempo real. En otras palabras un DSP toma señales del mundo real como la voz, video, temperatura, presión o posición, que después de haber sido digitalizadas las procesa a altas velocidades y en tiempo real.

La figura 1.2 muestra un sistema de Sistema de Procesamiento Digital de Señales típico, que consta de un Procesador Digital de Señales (DSP) y otro hardware externo como lo son los Convertidores Análogos-Digitales (ADC), que convierten señales continuas en discretas (formato digital de 1's y 0's) y los Convertidores Digitales–Análogos (DAC), que convierten la señal de forma discreta a continua. Aunque las señales del mundo real pueden ser procesadas en su forma continua o analógica, procesar señales digitalizadas brinda ventajas en velocidad y precisión.

Figura 1.2 Sistema de Procesamiento Digital de Señales



Para ilustrar este concepto, la figura 1.3 muestra cómo un DSP es usado en un reproductor de MP3. En la etapa de grabado de la voz (recording) la señal se recibe por un micrófono. Esta señal analógica es convertida a digital por un ADC y luego se pasa al DSP. El DSP realiza la codificación del sonido en formato de MP3 y guarda el archivo en memoria. En la etapa de reproducción del sonido (playback), el archivo es tomado de la memoria, decodificado por el DSP y después convertido en forma análoga usando un DAC, para ser escuchado en el sistema de altavoces. En un ejemplo más complejo, el DSP podría desempeñar otras funciones tales como un control de volumen, ecualizador o una interfaz de usuario.

Figura 1.3. DSP como reproductor de MP3

Cabe resaltar que los ingenieros de hardware usan el término DSP para referirse aun Procesador Digital de Señales, pero los diseñadores de software usan el término DSP para referirse al Procesamiento Digital de Señales.

1.3 Historia de los Procesadores Digitales de Señales

En 1978, INTEL lanzó el 2920 como un "Procesador analógico de señales". Este tenía un chip ADC/DAC con un procesador de señales interno, pero no poseía un hardware multiplicador y por esto el 2920 no tuvo éxito en el mercado.

En 1979, AMI lanza el S2811, fue diseñado como un microprocesador periférico y tenía que ser inicializado por el computador central. El S2811, al igual que el 2920 no tuvo gran éxito en el mercado. En el mismo año, BELL LABS introduce el primer chip Procesador Digital de Señales (DSP), The Mac 4 Microprocessor. Luego en 1980 fueron presentados en el IEEE International Solid State Circuits Conference los primeros DSP completos: el PD7710 de NEC y el DSP1 de AT&T, ambos procesadores fueron inspirados en las investigaciones de PSTN Telecomunicaciones. En ese mismo año NEC comenzó la producción del PD7710, la primera producción de DSP completos en el mundo.

El primer DSP producido por Texas Instruments (TI), el TMS32010 presentado en 1983 probó ser un suceso mayor, y TI es hoy en día el líder en el mercado de DSPs de propósito general. Otro diseño exitoso diseñado fue el Motorota 56000, pero no fue tan exitoso como los modelos siguientes y hoy por hoy aún no son líderes en el mercado.

En 1999, Improv Systems lanzó el JAZZ DSP, el primer DSP en el mundo de arquitectura VLIW (por sus siglas en ingles Very Long Instruction Word, es decir que la arquitectura de la CPU lee un grupo de instrucciones y las ejecuta al mismo tiempo) completamente configurable que era el objetivo del mercado de sistemas embebidos o integrados. Al ser el JAZZ DSP un procesador VLIW, este provee alto desempeño cuando se ejecutan operaciones en paralelo.

2. ARQUITECTURA DE MEMORIA DE UN DSP

2.1 Organización del sistema de memorias

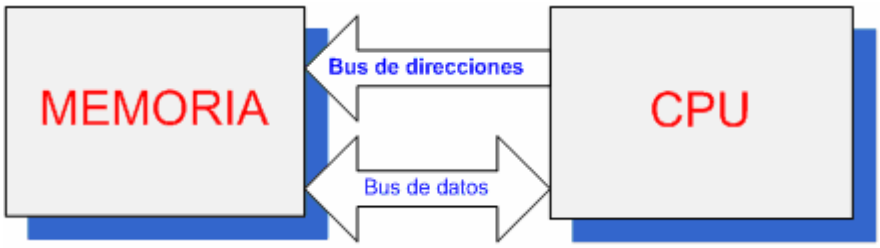
Uno de los mayores problemas al ejecutar algoritmos de procesamiento digital de señales es la transferencia de información desde y hacia la memoria de forma rápida. Esta información puede constar de datos, como muestras de una señal de entrada y coeficientes de un filtro digital, o las instrucciones de programa. Por ejemplo, supongamos que se necesitan multiplicar dos números que están en una localidad de la memoria, para hacer esto se deben traer tres valores binarios de la memoria, los números que se multiplicarán, más la instrucción de programa que describe qué hacer.

2.1.1 Arquitectura de Von Neumann

La figura 2.1 muestra la organización de la memoria en un microprocesador tradicional. Esta arquitectura se denomina **Von Neumann**, en honor al matemático estadounidense Jhon Von Neumann (1903-1957). Como se observa en la figura, la arquitectura posee una sola memoria y un solo bus para transferir datos hacia y desde la unidad central de procesamiento (CPU)[2].

[2] Ibid., p. 20

Figura 2.1 Arquitectura Von Neumann



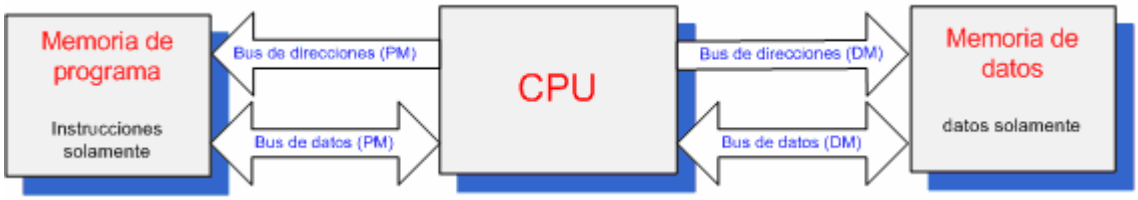
La presencia de un solo bus conlleva a que los datos no puedan ser cargados mientras se lee una instrucción de programa, disminuyendo la velocidad de ejecución del procesador. Por eso, multiplicar dos números requiere por lo menos tres ciclos de reloj, es decir un ciclo para transferir cada uno de los tres números sobre el bus, de la memoria a la CPU. Esta arquitectura se usa cuando se ejecutan todas las tareas requeridas de forma serial. De hecho, la mayoría de las computadoras de hoy utilizan este diseño.

2.1.2 Arquitectura Harvard

La figura 2.2 muestra lo que se conoce como la **Arquitectura Harvard**. Este nombre se debe al trabajo realizado en la Universidad de Harvard en los años 40's bajo la dirección de Howard Aiken (1900-1973). Esta arquitectura emplea dos memorias separadas, una para datos y otra para las instrucciones de programa, y se acceden con buses separados cada una.

Ya que los buses operan independientemente, las instrucciones de programa y los datos pueden ser cargados al mismo tiempo, es decir, se pueden realizar dos accesos a memoria durante cualquier ciclo de instrucción, mejorando la velocidad que tenía el diseño de Von Neuman. La mayoría de los DSPs de hoy en día utilizan esta arquitectura.

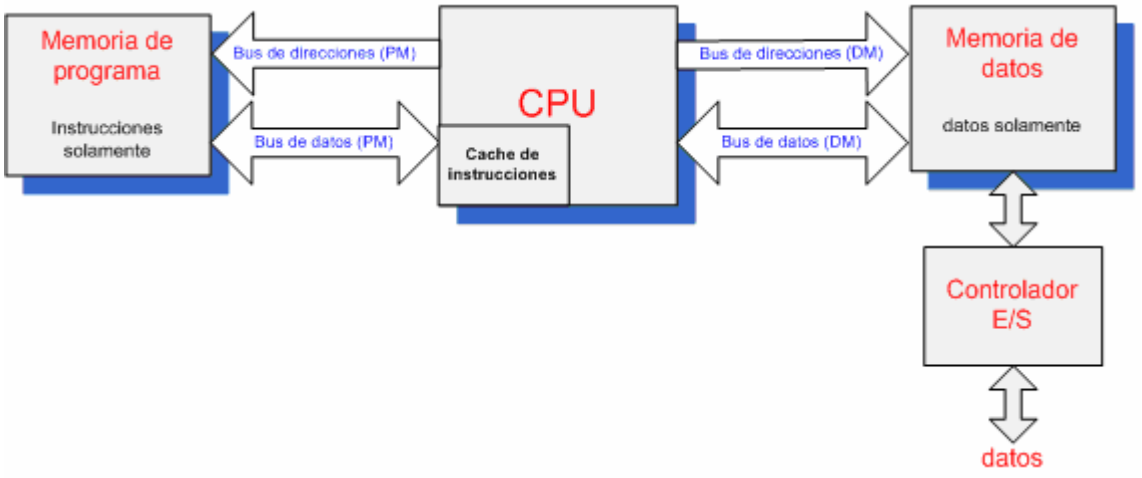
Figura 2.2 Arquitectura Harvard



2.1.3 Arquitectura Súper Harvard

La figura 2.3 ilustra el siguiente nivel de sofisticación. **La Arquitectura Súper Harvard**. Este término fue creado por Analog Devices para describir la operación interna de sus nuevas familias de DSPs, la ADSP-2106X y la ADSP-211XX. Estos son denominados **SHARC** DSP, contracción de **S**uper **H**arvard **ARC**hitecture. La idea era adicionarle características a la arquitectura Harvard para mejorar el desempeño. Esta arquitectura incluye una memoria cache de instrucciones (Instruction Cache) y un controlador de entrada/salida (E/S).

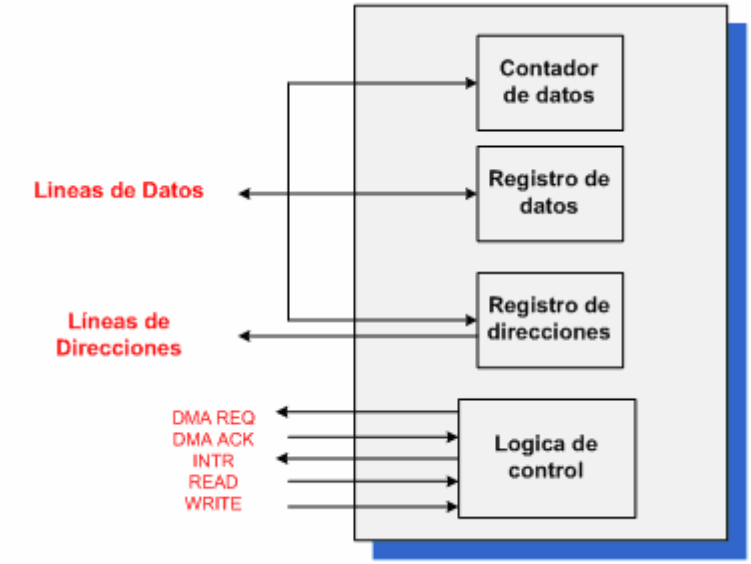
Figura 2.3 Arquitectura Súper Harvard



La memoria cache de instrucciones mejora las prestaciones de la arquitectura harvard. El problema que presenta la arquitectura harvard es que el bus de la memoria de datos esta más ocupado que el bus de la memoria de programa. Por ejemplo, cuando dos números son multiplicados, dos valores binarios (los números) deben pasar por el bus de la memoria de datos, mientras que solo un valor binario (la instrucción del programa) pasa por el bus de la memoria de programa. Por otra parte los algoritmos de procesamiento digital de señales gastan la mayor parte de su tiempo de ejecución en ciclos (loops), esto significa que el mismo grupo de instrucciones de programa pasarán continuamente de la memoria de programa a la CPU. La arquitectura Super Harvard aprovecha esta situación incluyendo una cache de instrucciones en la CPU, que es una pequeña memoria que contiene las instrucciones más recientes del programa. En la primera iteración de un ciclo, las instrucciones deben pasar a través del bus de la memoria de programa. Sin embargo, en las sucesivas ejecuciones del ciclo, las instrucciones pueden ser cargadas desde la cache.

El controlador E/S, por lo general, emplea la técnica de acceso directo a memoria (DMA - Direct Memory Access) que permite la transferencia de datos desde la memoria principal hacia un dispositivo periférico o viceversa, sin la intervención de la CPU. La figura 2.4 muestra un diagrama de un modulo DMA.

Figura 2.4. Diagrama de bloques típico de un modulo DMA



Cuando el procesador desea leer o escribir un bloque de datos, envía una orden al modulo DMA, incluyendo la siguiente información[3]:

- Si se va a leer o escribir, se utiliza la línea de control de lectura o escritura entre el procesador y el modulo DMA.

[3] STALLINGS, William. Organización y arquitectura de computadores. 5 ed. Madrid: Prentice- Hall, 2000.

- La dirección del dispositivo de E/S en cuestión, indicada a través de la línea de datos.
- La posición inicial de memoria a partir de donde se lee o se escribe, indicada a través de la línea de datos y almacenada por el modulo de DMA en su registro de direcciones.
- El número de palabras a leer o escribir, también indicado a través de la línea de datos y almacenado en el registro de cuenta de datos.

Después de esto el procesador continua con otro trabajo delegando la operación de E/S al modulo DMA, que se encargará de ello. El modulo DMA transfiere el bloque completo de datos, palabra a palabra, directamente desde o hacia la memoria, sin que pase a través del procesador. Cuando la transferencia ha terminado, el módulo de DMA envía una señal de interrupción al procesador. Así el procesador solo interviene al comienzo y al final de la transferencia[4].

El controlador DMA se usa para mejorar el desempeño de los dispositivos de entrada y salida, y se puede implementar dentro del chip del DSP o por medio de un hardware externo. Además algunos controladores DMA pueden manejar múltiples transferencias DMA en paralelo. A estos controladores se dice que tienen múltiples canales, donde cada uno puede manejar una transferencia y tienen su propio conjunto de registros de control, y cada canal puede ser usado para transferencias memoria-memoria o memoria-periferico.

[4] Ibid., p.28

2.2 Memorias de accesos múltiples

La arquitectura harvard puede llevar a cabo múltiples accesos a memoria por ciclo de instrucción, dado que utiliza dos memorias independientes conectadas a la CPU con su respectivo bus. Mientras que gran número de DSPs utilizan esta ventaja, existen otras maneras de alcanzar múltiples accesos a memoria por ciclo de instrucción, incrementando la velocidad de ejecución. Estas son las memorias rápidas y las memorias multi-puerto[5].

2.2.1 Memorias rápidas

Algunos procesadores utilizan memorias rápidas, porque soportan accesos secuénciales múltiples por ciclo de instrucción sobre el mismo conjunto de buses, y permiten completar un acceso en la mitad de un ciclo de instrucción. Esto significa que dos accesos independientes a una sola memoria se pueden completar en secuencia. Las memorias rápidas pueden ser combinadas con la arquitectura harvard, permitiendo mejores prestaciones que las obtenidas utilizando cualquiera de las dos técnicas independientemente. Por ejemplo, al considerar una arquitectura harvard con dos memorias rápidas, cada memoria permite ejecutar dos accesos secuénciales por ciclo de instrucción, Las dos memorias juntas permiten ejecutar cuatro accesos a memoria por ciclo de instrucción.

[5] LAPSLEY, Phil; BIER, Jeff; SHOHAM, Amit and LEE, Edward. Dsp processor fundamentals: architectures and features". California: Berkeley Design Technology, Inc. 1994-1996

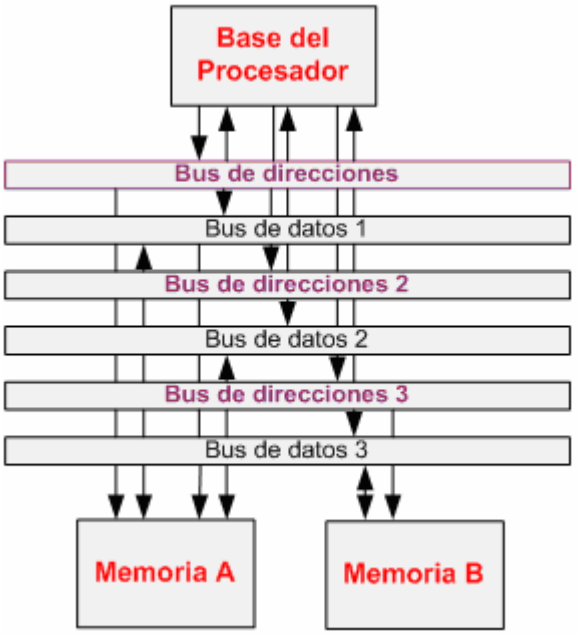
En general, si los accesos a memoria no pueden ser divididos de esta forma, por ejemplo, tres accesos son hechos a una memoria, el procesador automáticamente alarga la ejecución de la instrucción para permitir el tiempo necesario que los tres accesos secuénciales a memoria emplean en completarse. Así no hay riesgo que accesos a memoria de este tipo causen resultados erróneos; Esto simplemente causa que el programa corra lentamente.

2.2.2 Memorias Multi-Puertos

Las memorias multi-puertos es otra técnica para incrementar la capacidad de accesos a memoria. Este tipo de memorias permiten que múltiples accesos independientes a memoria se ejecuten en paralelo sobre varios conjuntos de buses de datos y direcciones. El tipo de memoria multi-puerto más común es la de puerto dual, la cual permite dos accesos simultáneos. Sin embargo, memoria de puertos triples y cuádruples también son usadas. Las memorias multi-puertos colocan los datos entre múltiples memorias independientes para alcanzar un máximo desempeño.

Algunos DSPs modifican la arquitectura harvard con el uso de memorias multi-puerto. Por ejemplo la arquitectura de memoria de la figura 2.5, incluye una memoria de programa de un solo puerto (B) con una memoria de datos de puerto dual (A). Este arreglo es utilizado en la familia de procesadores DSP561xx de Motorola.

Figura 2.5. Arquitectura Harvard con una memoria de datos de puerto dual (A) y una memoria de programa de un solo puerto (B)



El uso de memorias rápidas y de memorias multi-puertos está limitado por tamaño del chip, debido a los requerimientos en desempeño y capacidad que imponen las entradas y salidas que deben estar dentro del chip. En el caso de las memorias rápidas, sacar la memoria (o parte de esta) fuera del chip implica que significativos retardos adicionales sean introducidos entre la CPU y la memoria. A menos que la velocidad de ejecución del procesador sea relativamente baja, estos retardos no permitirán obtener dos o más accesos secuénciales a memoria por ciclo de instrucción.

En el caso de memorias multi-puertos, mover toda o parte de la memoria fuera del chip implica que múltiples buses de direcciones y de datos deben ser llevados fuera del chip. Esto trae como consecuencia que el chip necesitaría muchos pines de entrada/salida, lo que conduce a un chip más largo y más costoso.

2.3 Segmentación (“Pipelining”)

Pipelining es una técnica para incrementar el desempeño de un procesador, que consiste en dividir una secuencia de operaciones en otras más sencillas de ejecutar, que en lo posible se ejecutan cada una de ellas en paralelo. En consecuencia se reduce el tiempo total requerido para completar el conjunto de operaciones. Casi todos los DSP del mercado incorporan el uso del Pipelining en mayor o menor medida. Para ilustrar como el Pipelining mejora el desempeño de un procesador, se considera un hipotético procesador que utiliza etapas o unidades de ejecución separadas para el desarrollo de una única instrucción. Estas son[6]:

1. Obtener la instrucción de la memoria
2. Decodificar la instrucción
3. Leer o escribir un operando de la memoria
4. Ejecutar la instrucción.

La tabla 2.1 muestra la temporización de varias instrucciones ejecutadas de forma secuencial.

[6] SALAZAR, Jordi. Procesadores digitales de señal (dsp), Arquitecturas y Criterios de selección. Universidad Politécnica de Cataluña. Faculta de Ingeniería electrónica. Centro de sistemas y sensores electrónicos.

Tabla 2.1 Ejecución de las instrucciones sin pipeline. I1 e I2 representan la instrucción 1 y 2 respectivamente

	CICLO DE RELOJ							
	1	2	3	4	5	6	7	8
Obtención Instrucción	I1				I2			
Decodificación		I1				I2		
Lectura/ Escritura operando			I1				I2	
Ejecución				I1				I2

Si se supone que cada unidad de ejecución tarda 20ns en ejecutar su parte de la instrucción, entonces el procesador ejecuta una instrucción cada 80ns. Sin embargo, también se observa que el hardware asociado a cada etapa de ejecución está inactivo el 75% del tiempo. Esto ocurre porque el procesador no empieza a ejecutar una nueva parte de la instrucción hasta que finaliza la ejecución de la instrucción en curso.

Un procesador que implementara la Pipelining obtendría una nueva instrucción inmediatamente después de haber obtenido la anterior. De forma similar, cada instrucción sería decodificada después de haber terminado la decodificación de la instrucción anterior. Esto se ilustra a continuación:

Tabla 2.2 Procesador que utiliza la técnica del Pipelining

	CICLO DE RELOJ (INSTRUCCIÓN)							
	1	2	3	4	5	6	7	8
Obtención Instrucción	I1	I2	I3	I4	I5	I6	I7	I8
Decodificación		I1	I2	I3	I4	I5	I6	I7
Lectura/ Escritura operando			I1	I2	I3	I4	I5	I6
Ejecución				I1	I2	I3	I4	I5

Las unidades de ejecución trabajan en paralelo, mientras una obtiene el código de una instrucción, otra está decodificando la anterior y así sucesivamente. En consecuencia, una vez que la "pipeline" está llena, cada 20 ns se ejecuta una instrucción, lo cual representa un factor de mejora de prestaciones de cuatro respecto a un procesador que no incorpore dicha técnica.
Aunque la mayoría de los DSP utilizan Pipelining, el número de etapas varía de un procesador a otro. En general, cuanto mayor sea el número de etapas menor tiempo tardará el procesador en ejecutar una instrucción.

En el ejemplo anterior se ha supuesto un procesador con una eficiencia en el uso de la "pipeline" del 100%. En realidad, esto no siempre ocurre así. La eficiencia se ve disminuida por varias causas, entre las cuales se encuentra el hecho de que un procesador necesite dos ciclos para escribir en memoria, se obtenga el código de una instrucción de salto de programa o bien la petición de una interrupción. La tabla 2.3 muestra lo que pasa cuando una instrucción de salto llega a la "pipeline".

Tabla 2.3 Efecto en la pipeline ante la llegada de una instrucción de salto.

	CICLO DE RELOJ (INSTRUCCIÓN)							
	1	2	3	4	5	6	7	8
Obtención Instrucción	Salto	I2	-	-	N1	N2	N3	N4
Decodificación		Salto	-	-	-	N1	N2	N3
Lectura/ Escritura operando			Salto	-	-	-	N1	N2
Ejecución				Salto	NOP	NOP	NOP	N1

En el momento en que el procesador detecta la llegada de una instrucción de salto en la decodificación del segundo ciclo de reloj, la “pipeline” se vacía y detiene la obtención de nuevas instrucciones. Esto provoca que la instrucción de salto se ejecute en cuatro ciclos. Posteriormente, el procesador comienza la obtención de las instrucciones (N1-N4) a partir de la dirección de salto y del quinto ciclo de reloj. A causa de este tipo de situaciones, casi todos los DSP incorporan algún tipo de mejora en el uso de la segmentación con el propósito de reducir su posible ineficiencia temporal[7].

[7] Ibid., p. 33.

2.4 Tipos de Arquitecturas

La CPU atendiendo al tipo de instrucciones que utilizan pueden clasificarse en:

2.4.1 CISC Complex Instruction Set Computer

CISC: (Complex Instruction Set Computer) Computadores de conjunto de instrucciones complejo, que disponen de un repertorio de instrucciones elevado (más de 80), algunas de ellas muy sofisticadas y potentes, pero que como contrapartida requieren muchos ciclos de máquina para ejecutar las instrucciones complejas.

2.4.2 RISC: Reduced Instruction Set Computer

RISC: (Reduced Instruction Set Computer) Computadores de conjunto de instrucciones reducido, en los que el repertorio de instrucciones es muy reducido, las instrucciones son muy simples y suelen ejecutarse en un ciclo máquina. Además los RISC deben tener una estructura segmentada (pipeline) y ejecutar todas las instrucciones a la misma velocidad.

2.4.3 VLIW: Very Long Instruction Word

Hablar de DSP obliga a hacer referencia a las nuevas arquitecturas VLIW (Very Long Instruction Word) que están siendo adoptadas por los DSP de muy altas prestaciones. Las Arquitecturas VLIW son distintas de las tradicionales RISC y CISC implementadas en el mercado actual de microprocesadores.

Especificar múltiples operaciones por instrucción crea una arquitectura de palabra de instrucción muy larga o VLIW. Una implementación VLIW tiene capacidades muy parecidas a las de un procesador superescalar (distribuir y ejecutar más de una instrucción a la vez), con una importante excepción: el hardware VLIW no es responsable de descubrir oportunidades de ejecutar múltiples operaciones a la vez. En la implementación con VLIW la palabra de instrucción ya codifica operaciones a la vez. Esta codificación lleva a reducir enormemente la complejidad del hardware comparado a una implementación superescalar de alto nivel RISC o CISC.

La gran ventaja de VLIW es que una implementación con un mayor paralelismo es más fácil y más barata que la equivalente RISC o CISC. VLIW es una forma más sencilla de construir procesadores superescalares[8].

[8] www.semiconductors.philips.com/ acrobat/other/vliw-wp.pdf

2.4.3.1 Comparación de las arquitecturas: CISC, RISC y VLIW

La siguiente tabla resume las diferencias.

Tabla 2.4 Comparación de las arquitecturas: CISC, RISC y VLIW

CARACTERÍSTICA DE LA ARQUITECTURA	CISC	RISC	VLIW
Tamaño de instrucción	Varios	Tamaño único, normalmente 32 bits	Tamaño único
Semántica de instrucción	Va de sencilla a Compleja; muchas operaciones dependientes posibles por instrucción	Casi siempre una operación sencilla	muchas operaciones sencillas e independientes
Registros	Pocos y a veces especializados	Muchos de uso general	Muchos de uso general
Referencias a memoria	Ligadas a operaciones en distintos tipos de instrucciones	No están ligadas a operaciones (arquitectura de carga y almacenamiento)	No están ligadas a operaciones (arquitectura de carga y almacenamiento)
Objetivo del diseño hardware	Explotar las implementaciones microcodificadas	Explotar implementaciones con una segmentación y sin microcódigo	explotar implementaciones con segmentación múltiple, sin microcódigo

Las instrucciones CISC varían en tamaño, a veces especifica una secuencia de operaciones, y puede requerir algoritmos de decodificación en serie (como consecuencia el proceso conlleva más tiempo). Los CISC tienen por lo general pocos registros de los cuales algunos son de uso específico, lo que restringe su uso. Las referencias a memoria son típicamente combinadas con otras operaciones (como agregar memoria a un registro). Las instrucciones CISC son diseñadas para aprovecharse del microcódigo.

Las instrucciones RISC especifican operaciones simples, tienen un tamaño constante y son fáciles y rápidas de decodificar. Las arquitecturas RISC tienen un número relativamente grande de registros de uso general. Las instrucciones pueden acceder a memoria sólo a través de operaciones de carga/almacenamiento de registros a memoria. El conjunto de instrucciones RISC no necesita microcódigo y esta diseñado para simplificar el pipelining.

Las instrucciones VLIW son parecidas a las del RISC pero más largas para poder especificar varias operaciones independientes. Una instrucción VLIW puede ser pensada como varias instrucciones RISC juntas.

2.4.3.2 Comparación de implementaciones: CISC, RISC y VLIW

Las diferencias entre las arquitecturas CISC, RISC y VLIW radican en sus respectivas implementaciones. Los diseños RISC y CISC de alto desempeño se denominan implementaciones superescalares.

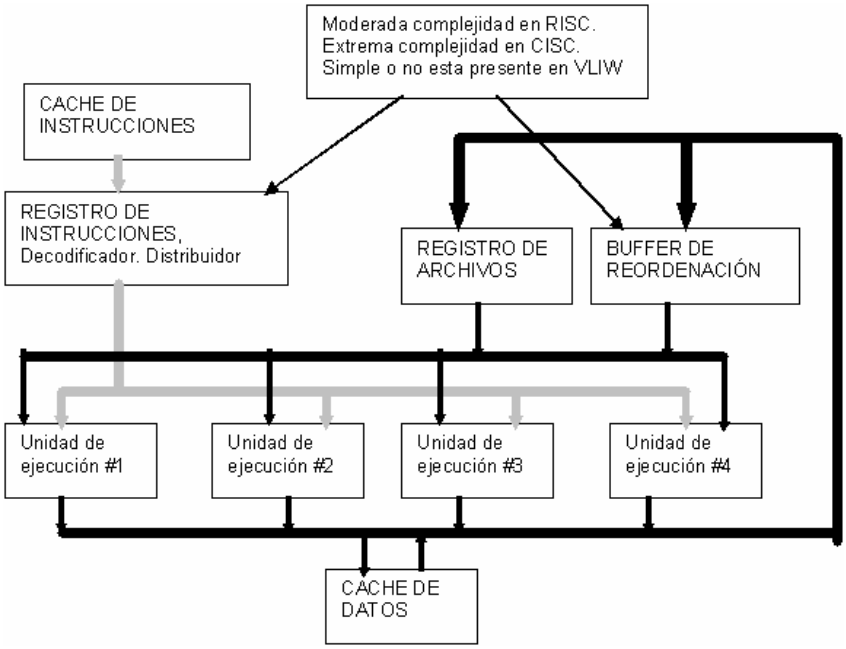
La mayor parte de las arquitecturas CISC, fueron diseñadas con la idea de que una implementación tomara una instrucción, la ejecutara completamente, y luego se moviera a la próxima instrucción. Este modelo de ejecución se asumió de naturaleza serial.

Las arquitecturas RISC fueron diseñadas con un modelo de ejecución segmentado. En este modelo se toma una instrucción, se segmenta, y luego se mueve a la próxima instrucción antes de que la instrucción previa haya completado todo el proceso de ejecución.

Para que una arquitectura CISC o RISC alcance niveles más altos de desempeño que el suministrado por una sola segmentación, una implementación superescalar debe ser construida. La naturaleza de una implementación superescalar es que esta toma, ejecuta y completa más de una instrucción CISC o RISC por ciclo. Algunas de las más recientes arquitecturas RISC han sido diseñadas con implementaciones superescalares, por ejemplo DEC Alpha e IBMPOWER, del cual se deriva PowerPC. No obstante, las implementaciones superescalares RISC y CISC comparten complejidades fundamentales: la necesidad que tiene el hardware de descubrir y explotar el paralelismo de instrucciones. La figura 2.6 muestra el diagrama de bloques de alto nivel de una implementación superescalar de un procesador RISC o CISC. La implementación consiste o un conjunto de unidades de ejecución, que pueden ser Unidades Aritmético-Lógicas (ALU) enteras o de punto flotante, unidades de carga/almacenamiento o unidades de salto, entre otras, a las cuales se les suministran operaciones desde un distribuidor de instrucciones y de operándos desde un registro de datos[9].

[9] Ibid., p. 38.

Figura 2.6. Diagrama de bloques de una implementación superescalar RISC o CISC



Las unidades de ejecución tienen registros temporales para las operaciones que se han emitido pero no se han ejecutado. El distribuidor de instrucciones examina la ventana de instrucciones contenidas en un registro y decide cual de las instrucciones pueden ser distribuidas a las unidades de ejecución, tratando de distribuir tantas instrucciones a la vez como se posible, para conseguir un mayor paralelismo de instrucciones.

Es teóricamente simple pero costosa una implementación con muchas unidades de ejecución y un potente distribuidor de instrucciones. La razón tiene que ver más respecto al hardware que al software. Los compiladores para los procesadores superescalares RISC y CISC simples producen un código con el objetivo de minimizar su tamaño y el tiempo de ejecución. Por otro lado las implementaciones superescalares de alto desempeño, el objetivo de minimizar el tamaño del código

limita el desempeño que esta puede alcanzar, dado que en un código minimizado resultan saltos condicionales frecuentes, cada seis instrucciones. En teoría el procesador debe esperar hasta que se resuelva el salto antes de que comience a buscar paralelismo por la trayectoria del salto. Para evitar esto las implementaciones superescalares de alto desempeño, realizan predicciones de saltos. Con esto el procesador realiza una suposición del siguiente salto y empieza a buscar paralelismo por la trayectoria predicha. El proceso de distribuir y ejecutar instrucciones desde la trayectoria predicha se denomina ejecución predictiva .

Desafortunadamente, la predicción de saltos no es 100% segura. Sin embargo con la ejecución predictiva, es necesario ser capaz de cancelar los efectos de instrucciones ejecutadas predictivamente en el caso de no predecir un salto. Algunas implementaciones, como Pentium superescalar de Pentium simplemente impiden a instrucciones entre la trayectoria predicha modificar algún estado del procesador visible, pero para sacar provecho de la ejecución predictiva, es necesario permitir que se ejecuten completamente las instrucciones entre la trayectoria predicha. Para cancelar todos los efectos de la ejecución predictiva, se emplea una estructura de hardware llamada buffer de reordenación. Este registro se adjunta al registro de datos que mantiene la pista de todos los resultados producidos por las instrucciones que se han ejecutado recientemente o que han sido distribuidas a las unidades de ejecución pero que aún no se han completado. Este buffer guarda los resultados de las instrucciones ejecutadas predictivamente. Cuando un salto condicional se resuelve, lo resultados de las instrucciones ejecutadas predictivamente pueden ser leídos desde el buffer (cuando no se predice un salto) o escritos desde el buffer al registro de datos (cuando se predice un salto correctamente)[10].

[10] Ibid., p. 38.

2.4.3.2.1 Ventajas de la implementación de VLIW

Una implementación VLIW consigue el mismo efecto de una implementación superescalar como RISC o CISC, y lo realiza sin las dos partes más complejas de un diseño superescalar de alto rendimiento.

Debido a que las instrucciones VLIW especifican explícitamente paralelismo no es necesario tener hardware decodificador y distribuidor que intente reconstruir el paralelismo a partir de un flujo de instrucciones serie. En vez de tener hardware intentando descubrir paralelismo, los procesadores VLIW se apoyan en un compilador que genera código VLIW. Para empezar, el compilador tiene la habilidad de mirar ventanas de instrucciones mucho más largas que un hardware. Para un procesador superescalar, una ventana muy larga implica una lógica mayor y por lo tanto una mayor área del chip. En cierto punto, puede que no haya suficiente de ninguna de las dos cosas y que el tamaño de la ventana esté restringido. Peor todavía, antes de que un simple límite en la cantidad de hardware sea alcanzado, la complejidad podría afectar a la velocidad de la lógica, de este modo el tamaño de la ventana se limita para evitar reducir la velocidad de reloj del chip. Las ventanas de ejecución de software pueden ser arbitrariamente grandes. Así, buscar el paralelismo en una ventana de software es probable que conduzca a mejores resultados.

En segundo lugar, el compilador tiene conocimiento del código fuente del programa. El código fuente suele contener información acerca del comportamiento del programa que puede ser utilizada para expresar un alto paralelismo a nivel del conjunto de instrucciones. Una técnica potente llamada compilación Trace-Driven puede ser usada para mejorar notablemente la calidad de código producido por el compilador.

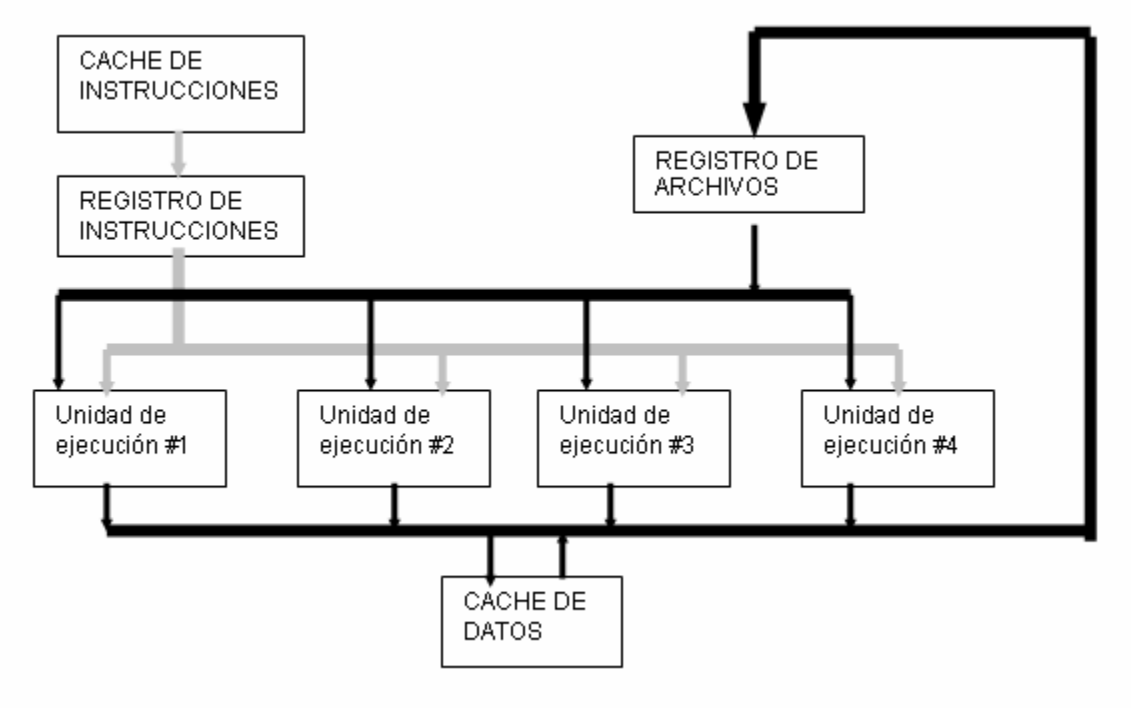
Esta técnica, primero produce un programa VLIW que no llega a ser el óptimo pero funciona. El programa tiene rutinas integradas que toman notas del comportamiento de este. Estas notas, por ejemplo pueden indicar cuales saltos se han tomado y con qué frecuencia, son usadas por el compilador en una segunda compilación para producir el código de acuerdo a las notas precisas del comportamiento del programa.

Por último, con suficientes registros es posible simular las funciones del buffer de reordenación de la implementación superescalar. El propósito de un buffer de reordenación es permitir a un procesador superescalar ejecutar predictivamente las instrucciones y después descartar rápidamente los resultados predichos si fuera necesario. Con suficientes registros una maquina VLIW puede situar los resultados de instrucciones ejecutadas predictivamente en registros temporales.

El compilador sabe cuantas instrucciones serán ejecutadas predictivamente, por lo que simplemente utiliza registros temporales a lo largo de la trayectoria predicha e ignora los valores en los registros que serian tomados si el salto condicional estuviera mal predicho. La siguiente figura nos muestra un ejemplo de implementación genérica del hardware VLIW, sin el complejo buffer de reordenación ni hardware decodificador y despachador[11].

[11] Ibid., p. 38.

Figura 2.7. Implementación genérica del hardware VLIW



2.4.3.3 Complejidad en Software Vs. Hardware

Mientras una arquitectura VLIW reduce la complejidad de hardware comparado con la implementación superescalar, requiere un compilador mucho más complejo. Extraer el máximo rendimiento de las implementaciones superescalares RISC o CISC requiere sofisticadas técnicas de compilación, mientras que el nivel de compilación de la arquitectura VLIW es mucho mayor.

VLIW simplemente transporta la complejidad de hardware a software. Afortunadamente este intercambio tiene un beneficio: la complejidad sólo se paga una vez cuando el compilador es escrito en vez de cada vez que el chip se fabrica. Entre los posibles beneficios se destacan un tamaño de chip reducido, que dejan a los fabricantes mayores ganancias y menores precios para los consumidores de esta clase de procesadores.

La complejidad siempre es más fácil de combatir en software que en hardware. Así el chip puede reducir su precio al ser diseñado, se diseña más rápido y ofrece mayor facilidad para depurar errores. También las mejoras al compilador pueden hacerse después de que el chip sea fabricado. Las mejoras a los procesadores superescalares requieren cambios en el microprocesador, lo que produce un incremento en costos de diseño.

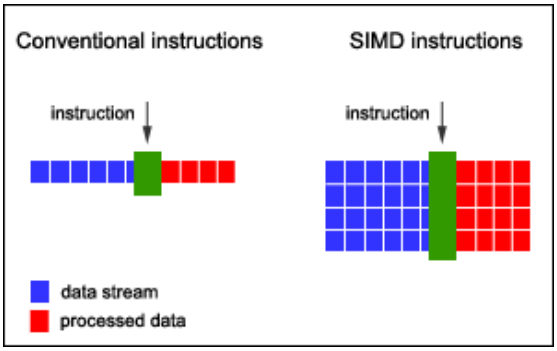
2.5 SIMD Single Instruction Multiple Data

SIMD que traduce Instrucción Simple, Datos Múltiples, no es precisamente una arquitectura sino mas bien una técnica arquitectónica que mejora el desempeño de algunos algoritmos y puede ser usada en cualquiera de las arquitecturas descritas con anterioridad. SIMD se basa en los conceptos de paralelismo de datos y del procesamiento vectorial.

El paralelismo de datos ocurre cuando existe la necesidad de aplicar la misma instrucción a una gran cantidad de datos. Por ejemplo, en un juego 3D podría necesitar incrementar el brillo de cada píxel en la pantalla. En esta clase de situaciones SIMD puede mejorar significativamente el desempeño.

El procesamiento vectorial agrupa los datos y los procesa como una sola unidad en vez de componentes individuales. Un procesador que soporte SIMD y con un registro de 128 bits puede procesar cuatro conjuntos de datos de 32 bits por medio del procesamiento vectorial, figura 2.8. Esto es similar a un profesor que da una instrucción a toda la clase en ves de repetir la misma instrucción a cada estudiante.

Figura 2.8. Ejecución de instrucciones SIMD



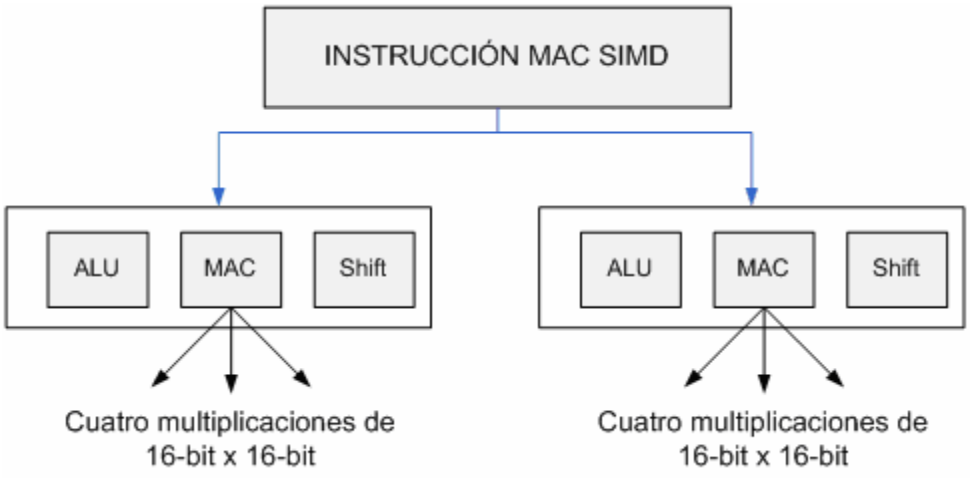
En conclusión, la técnica SIMD permite al procesador ejecutar con una instrucción la misma operación simultáneamente usando conjuntos de datos diferentes. Por ejemplo, una instrucción de multiplicación SIMD puede realizar dos o más multiplicaciones en paralelo con diferentes conjuntos de operándos de entrada en un ciclo de reloj. Esta técnica puede aumentar enormemente la tasa de cómputo de algunas operaciones vectoriales que son utilizadas frecuentemente en aplicaciones multimedia y procesamiento de señales.

Existen DSPs que soportan operaciones SIMD, pero el hardware sobre el que se basan varía ampliamente. Analog devices, por ejemplo, modificó la arquitectura básica de los DSPs de punto flotante, los ADSP-2106x, agregando un segundo conjunto de unidades de ejecución que son el duplicado exacto del conjunto original. La nueva arquitectura se denomina la ADSP-2116x que incluye una unidad MAC, una ALU y un registro desplazador (shifter) y cada uno tienen su propio conjunto de registros. Esta arquitectura puede emitir una instrucción y ejecutarla en paralelo en ambos conjuntos de unidades de ejecución usando diferentes datos y duplicando de esta forma, el desempeño en algunos algoritmos[12].

[12] EYRE, Jennifer and BIER, Jeff. The evolution of dsp processors. Berkeley Design Technology, Inc.

En contraste, en lugar de tener múltiples conjuntos de las mismas unidades de ejecución, algunos DSP pueden dividir sus unidades de ejecución (ejemplo las ALUs o las unidades MAC) en múltiples sub unidades que procesan operándos mas pequeños. Estos procesadores tratan los operándos en registros largos (ej: 32 bits) como múltiples operándos pequeños (ej: dos operándos de 16 bits o cuatro de 8 bits). Quizás la aplicación más importante de las capacidades de SIMD se encuentre en los procesadores TigerSHARC de Analog Devices. TigerSHARC posee una arquitectura VLIW y combina dos tipos de SIMD: una instrucción puede controlar la ejecución de sus dos conjuntos de unidades de ejecución, y esta instrucción puede especificar una operación de unidad de ejecución dividida (e.g., split-ALU or split-MAC) que será ejecutada en cada conjunto. Utilizando esta capacidad jerárquica de SIMD, Tiger-SHARC puede ejecutar ocho multiplicaciones de 16 bits, figura 2.9[13].

Figura 2.9. Capacidad jerárquica de SIMD en Tiger-SHARC. Posee seis unidades de ejecución agrupadas en dos grupos de tres.



[13] Ibid., p. 48

Hacer efectivo el uso de las capacidades de SIMD puede requerir un esfuerzo importante por parte del programador. Los programadores a menudo deben organizar los datos en memoria para que el procesamiento SIMD se desarrolle a velocidades altas (ej: organizar los datos para que puedan ser recuperados en grupos de cuatro operandos al tiempo) y también deben reorganizar los algoritmos para hacer máximo uso de la capacidades del procesador.

SIMD es efectivo en algoritmos que procesan datos en paralelo; para algoritmos que son inherentemente seriales (por ejemplo algoritmos que usan el resultado de una operación como entrada de la próxima operación), SIMD no se usa[14].

[14] Choosing a DSP Processor. Berkeley Design Technology, Inc. 1996-2000

3. CRITERIOS DE SELECCIÓN DE UN DSP

Los DSPs se utilizan en gran variedad de aplicaciones, desde sistemas radar y sonar hasta la electrónica de consumo. Naturalmente, ningún procesador satisface todas las necesidades de todas o la mayoría de aplicaciones. Por lo tanto, la primera tarea para el diseñador al elegir un DSP es ponderar la importancia relativa de las prestaciones, coste, integración, facilidad de desarrollo, consumo y otros factores que una determinada aplicación necesita.

A continuación se presentan algunas características de los DSPs que se deben tener en cuenta a la hora de su elección para una determinada aplicación.

3.1 Formato de datos

Existen dos formatos usados para almacenar y manipular datos en un DSP:

- Punto Fijo (Fixed Point), es decir, utiliza números enteros
- Punto Flotante (Floating Point), es decir, utiliza números reales

3.1.1 DSP de Punto Fijo

Un DSP de punto fijo típicamente representa cada número con un mínimo de 16 bits; Pero en algunos casos los representa con un mínimo de 24 bits. En los DSPs de punto fijo los enteros sin signo se pueden representar en el rango de 0 a 2^{16} definiendo valores hasta de 65535; por otra parte los enteros con signo pueden ser representados en valores desde -32768 hasta +327677.

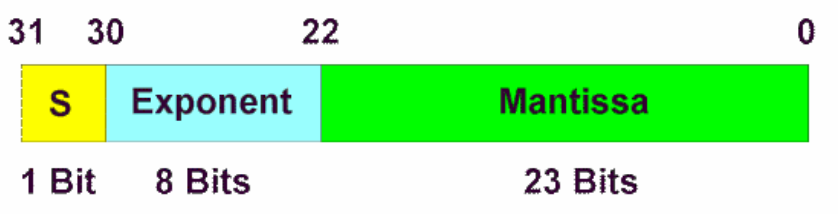
Las fracciones sin signo toman cualquier valor entre 0 y 1, y las fracciones con signo toman cualquier valor entre – 1 (FFFF) y 1.

Las velocidades de procesamiento de un sistema en punto fijo es especificada en Millones de Operaciones Enteras Por Segundo, MIPS (Million Integer Operations Per Second).

3.1.2 DSP de Punto Flotante

Un DSP de punto flotante representa cada número con un mínimo de 32 bits, denominado precisión simple o en algunos casos con un mínimo de 64 bits, denominado doble precisión. Este formato matemático es definido en el estándar IEEE 754.1985. En la figura se muestra el formato típico de 32 bits de un DSP de punto flotante que almacena valores numéricos usando 23 bits de mantisa y 8 bits de exponente, más un bit del signo.

Figura 3.1. Formato de un DSP de punto flotante



El número en punto flotante F es igual: $F = (-1)^{S} * M * 2^{(E-127)}$

De donde:

S = 0, si el número es positivo y S = 1, si el número es negativo.

E = es un número entre 0 y 255 representando al exponente de 8 bits.

M = es la mantisa formada de 23 bits de una fracción binaria (Por ejemplo la fracción binaria 1.0101 significa 1+0/2+1/4+0/8+1/16).

Un DSP de punto flotante es más versátil que un DSP de punto fijo. Un DSP de punto flotante puede desarrollar operaciones en punto fijo o en punto flotante, pero se limitan solo a operaciones en punto flotante.

Las velocidades de procesamiento de un sistema en punto flotante es especificada en Millones de Operaciones de Punto Flotante Por Segundo, MFLOPS (Million Floating Point Operations Per Second).

El formato de punto flotante es más flexible que el formato de punto fijo. Con un DSP de punto flotante, los diseñadores de sistemas tienen acceso a un rango dinámico más amplio (mayor relación entre los números más grandes y más pequeños que pueden ser representados). En consecuencia, los DSPs de punto flotante son generalmente más fáciles de programar que los de punto fijo, dado que el programador no debe preocuparse por el margen dinámico ni por la precisión.

Por el contrario, en los DSPs de punto fijo el programador a menudo debe escalar las señales en varias etapas de sus programas para asegurar una adecuada precisión numérica, por el limitado margen dinámico que un DSP de punto fijo posee. Los DSPs de punto flotante son más costosos y consumen más potencia que los DSPs de punto fijo. El mayor costo y consumo de potencia es resultado de una mayor complejidad circuital que se traduce en un mayor tamaño del chip. Por lo general, las aplicaciones con un gran volumen de unidades y/o bajo consumo utilizan los DSP de punto fijo al ser la prioridad en este tipo aplicaciones el bajo costo.

3.2 Ancho de palabra

Los DSPs de punto flotante utilizan una palabra de datos de 32 bits, mientras que en los DSPs de punto fijo el ancho de la palabra de datos más común es de 16 bits. El tamaño de la palabra de datos tiene un gran impacto en el coste, ya que influye notablemente en el tamaño del chip y en el número de pines del encapsulado, así como en el tamaño de la memoria externa de los dispositivos conectados al DSP. Por lo tanto, los diseñadores intentan utilizar el circuito integrado con el menor tamaño de palabra que la aplicación pueda tolerar.

De la misma forma que ocurre con la elección entre punto fijo y punto flotante, existe un compromiso entre tamaño de palabra y complejidad. Una aplicación que requiera 24 bits puede ser desarrollada por un DSP de 16 bits a costa de un aumento de complejidad en el software. Por ejemplo, con un DSP de punto fijo de 16 bits, un programador puede realizar operaciones con aritmética de doble precisión y 32 bits combinando las instrucciones adecuadas.

Naturalmente, la doble precisión será mucho más lenta que la precisión simple. Si el tamaño de la aplicación puede desarrollarse en precisión simple, pero la aplicación necesita mayor precisión en pequeñas partes del código, podría tener sentido emplear la doble precisión únicamente en aquellas partes del programa que lo necesiten. Pero si la mayor parte de la aplicación requiere más precisión, entonces un DSP con un tamaño de palabra mayor sería la opción adecuada.

Cabe resaltar que la mayoría de los DSP utilizan un ancho de la palabra de instrucción igual al ancho de la palabra de datos, pero no todos lo hacen. Por ejemplo, la familia ADSP-21xx de Analog Devices, usa una palabra de datos de 16 bits y una palabra de instrucción de 24 bits.

3.3 Velocidad

La medida clave para saber si un DSP es o no apropiado para una aplicación es la velocidad de ejecución de instrucciones. La velocidad de proceso depende de cuanto el dispositivo pueda procesar o desarrollar en un ciclo de reloj. Existen varias formas para medir la velocidad de un procesador. El parámetro más usual es el tiempo de ciclo de instrucción: tiempo necesario para ejecutar la instrucción más rápida del procesador. Su inverso dividido por un millón y multiplicado por el número de instrucciones ejecutadas por ciclo da lugar a la velocidad máxima de ejecución de instrucciones del procesador en millones de instrucciones por segundo o MIPS[15].

[15] SALAZAR, Op. cit., p 33.

Un problema que se presenta cuando se compararan los tiempos de ejecución de instrucciones de varios procesadores es que la cantidad de trabajo realizado por una instrucción varía significativamente de un procesador a otro, aquellos que usan arquitecturas VLIW. Estos procesadores usan instrucciones muy sencillas que desarrollan menos trabajo que las instrucciones típicas de procesadores convencionales produciendo una comparación desigual. Ademas las comparaciones hechas en base a MIPS suelen ser medidas imprecisas del desempeño de un procesador. Por ejemplo, algunos DSPs disponen de desplazadores combinatorios ("barrel shifters") que permiten hacer desplazamientos de múltiples bits de datos con sólo una instrucción, mientras que otros DSPs requieren que el dato sea desplazado con repetidas instrucciones de desplazamiento de un solo bit. De forma similar, algunos DSP permiten el movimiento de datos en paralelo (carga simultánea de datos mientras se ejecuta una instrucción) que no están relacionados con la instrucción que la ALU está ejecutando, pero otros DSP sólo soportan movimientos en paralelo que estén relacionados con las instrucciones que esté ejecutando la ALU.

Una solución a estos problemas consiste en implementar una operación básica (en vez de una instrucción) y utilizarla como referencia al comparar distintos DSPs. La operación que suele tomarse como referencia es la MAC (Multiplicaciones y Acumulaciones) que multiplica dos números (generalmente un coeficiente de peso y una muestra de entrada) y le suma el resultado a otro valor previamente guardado(A = A[N] + B[N]*C[N], donde N son las muestras).

Cuando esta operación se realiza continuamente, el proceso se denomina convolución, que es necesaria para la implementación de todo tipo de filtros. Desafortunadamente, los tiempos de ejecución de la MAC por lo general proporcionan poca información para poder diferenciar entre distintos DSPs, ya que en la mayoría de ellos esta instrucción se ejecuta en un solo ciclo de instrucción.

Además, los tiempos de ejecución de la MAC no suele reflejar las prestaciones de otro tipo importante de operaciones como los bucles que están presentes en todas las aplicaciones.

Un enfoque más general consiste en definir un conjunto algoritmos o funciones, como un filtro FIR o IIR, e implementarlo en distintos DSPs y de esta forma ver cuál de ellos proporciona mejores prestaciones. Sin embargo, la implementación y posterior análisis de estos algoritmos para distintos DSPs puede resultar una tarea ardua. En este sentido, una buena referencia pueden ser las pruebas que efectúa la Berkeley Design Technology Inc, pionera en utilizar distintas porciones de algoritmos y funciones para medir las prestaciones de los diferentes DSP.

Cabe resaltar dos notas finales acerca de las velocidades de los procesadores: Primero, hay que ser cuidadosos al comparar las velocidades de los procesadores en términos de Millones de operaciones por segundo (MOPS) o Millones de operaciones en punto flotante por segundo (MFLOPS), dado que diferentes fabricantes tienen diferentes ideas de lo que constituye una operación. Por ejemplo muchos procesadores de punto flotante afirman tener tasas de MFLOPS de dos veces la tasa de MIPS, porque estos son capaces de ejecutar un multiplicación en punto flotante en paralelo con una suma en punto flotante.

Segundo, hay que ser cuidadosos al comparar los ciclos de reloj de un procesador. La entrada de reloj de un DSP puede ser de la misma de frecuencia de la tasa de instrucción del procesador, o pueden ser dos o cuatro veces mayores a esta, dependiendo del procesador. Además, los chips de los DSPs poseen circuitos de enganche de fase (PLL: Phase-Locked Loop), que permiten el uso de bajas frecuencias de reloj externas para generar una alta frecuencia de reloj en el chip[16].

[16] SALAZAR, Op. cit., p 33.

3.4 Organización de la memoria

La organización de la memoria de un DSP puede tener un gran impacto en sus prestaciones.

Una ejecución rápida de la instrucción MAC requiere que la lectura en memoria de la palabra de instrucción y de sus dos palabras de datos se haga a una tasa efectiva de una vez cada ciclo de instrucción. Existe una variedad de formas de hacerlo, utilizando memorias multi-puerto para permitir múltiples accesos a memoria en un ciclo de instrucción, mediante memorias de datos e instrucciones separadas (arquitectura Harvard), y memorias caches que permiten que las instrucciones sean leídas desde la cache en vez de la memoria, liberando el acceso a la memoria para la obtención de otros datos.

Otro punto importante a tener en cuenta es la cantidad de memoria que soporta el DSP, interna y externamente. Atendiendo a las características de la aplicación, la mayoría de los DSP de coma fija poseen memorias internas, en el propio chip, de tamaño entre 1M y 256K palabras, y un bus externo de direcciones pequeño. Así por ejemplo, la mayoría de los DSP de coma fija de Analog Devices, Motorola y Texas Instruments tienen buses de direcciones de 16 bits o menos, lo que limita la cantidad de memoria externa de acceso directo. Por el contrario, la mayoría de los DSP de coma flotante proporcionan poca o ninguna memoria interna, pero se caracterizan por tener buses de direcciones externos de gran tamaño, para soportar una gran cantidad de memoria externa.

Como la mayoría de las características de los DSPs, la mejor combinación de organización de memoria, tamaño y número de buses externo es fuertemente dependiente de la aplicación.

3.5 Soporte técnico y facilidad de desarrollo

En el momento de elegir un DSP u otro será necesario conocer completamente los requisitos de procesamiento del sistema.

El DSP que finalmente se elija deberá disponer de un amplio conjunto de herramientas de desarrollo que permitan llevar a cabo un diseño tan simple como sea posible. Algunos requerimientos básicos son:

- Documentación de diseño detallada
- Herramientas de desarrollo de código en ensamblador y/o en lenguaje de alto nivel
- Herramientas para la prueba de la funcionalidad del diseño
- Notas de aplicación u otro tipo de ayuda al diseño

El objetivo será seleccionar el DSP que permita terminar el proyecto en el tiempo previsto y que la solución alcanzada sea la que presente la mejor relación coste-eficiencia. En aplicaciones de gran volumen de producción, esto probablemente signifique que el DSP escogido será el más barato que pueda realizar la aplicación. Para aplicaciones con un volumen bajo-medio existirá el compromiso entre el coste de las herramientas de desarrollo y el coste y eficiencia del DSP. En cambio, para aplicaciones con un volumen bajo de producción tendrá más sentido utilizar un DSP que facilite el diseño o que tenga las herramientas de desarrollo más baratas.

Cabe la posibilidad que la elección del DSP sea un proceso iterativo. En otras palabras, puede no haberse escogido el dispositivo correcto. Podría ser que aparecieran problemas imprevistos en la fase de desarrollo y prueba del código o incluso que se encontrara que un DSP más barato y menos potente pudiera ser el elegido. Comúnmente, las especificaciones del diseño alterarán y forzarán a replantear la solución escogida. Los dos primeros casos pueden evitarse haciendo más minuciosa la búsqueda del DSP que más se adecue a la aplicación en particular. Algunas veces vale la pena la compra de herramientas de desarrollo tales como los simuladores software para algunos DSP y ejercitar el código antes de comprometerse a un solo DSP.

3.6 Consumo

El uso cada vez más extendido de los DSP en aplicaciones portátiles como la telefonía celular y reproductores de audio hace que el consumo de energía sea un factor importante al momento elegir un DSP. Por eso los fabricantes de DSP están reduciendo los voltajes de alimentación e incorporan prestaciones para la gestión de energía. Las características disponibles en algunos DSPs son:

- Operación con voltajes reducidos: Muchos fabricantes ofrecen DSPs que utilizan tensiones de trabajo de 3.3, 2.5 o 1.8V. Esto procesadores consumen menos energía que aquellos que trabajan a 5v de alimentación a la misma velocidad de reloj.
- Modos Sleep o IDLE (Modos de bajo consumo): En este modo de operación se deshabilita el reloj del DSPI, reduciendo el consumo de energía. Las interrupciones sacan al procesador del modo sleep.

- Divisores programables de reloj: algunos DSPs permiten variar la frecuencia de reloj por software, para usar la mínima velocidad de reloj requerida para una aplicación específica.

3.7 Costo

Generalmente el coste del DSP es el principal parámetro en todos aquellos productos que se van a fabricar en grandes volúmenes. En tales aplicaciones, el diseñador intenta utilizar el DSP con coste inferior y que satisfaga las necesidades de la aplicación aún cuando ese dispositivo pueda ser considerado poco flexible y más difícil de programar que otros DSP más caros.

Entre las familias de DSP, el más barato será aquel que tenga menos características funcionales, menos memoria interna y probablemente menos prestaciones que otro más caro. Sin embargo, una diferencia clave en el precio está en el encapsulado. Los encapsulados PQFP (Plastic Quad Flat Pack) y TQFP (Thin Quad Flat Pack) son más baratos que los PGA (Pin Grid Array)[17].

En el Anexo A se presenta una tabla con los criterios básicos de selección de un DSP.

[17] SALAZAR, Op. cit., p. 33.

4. DIFERENCIAS ENTRE LOS DSPS Y LOS PROCESADORES DE PROPÓSITO GENERAL

Los computadores se utilizan ampliamente en dos áreas: (1) la Manipulación de datos, como el procesamiento de palabras y el manejo de bases de datos y (2) cálculos matemáticos, usados en la ciencia, ingeniería y procesamiento digital de señales. Sin embargo la mayoría de los computadores no están optimizados para desempeñar ambas funciones de manera eficiente. En aplicaciones informáticas como el procesamiento de palabras, los datos deben ser guardados, comparados, movidos, etc., y el tiempo de ejecución de una instrucción particular no es crítico. Por otra parte, en las aplicaciones de procesamiento digital de señales se requiere que las operaciones matemáticas se ejecuten rápidamente, y el tiempo de ejecución de una instrucción específica debe ser conocido y predecible. Por lo tanto para alcanzar esto el hardware y el software deben ser extremadamente eficientes.

La diferencia esencial entre un DSP y un Procesador de Propósito General o GPP (por su sigla General Purpose Proccessor) es que un DSP posee características diseñadas para soportar tareas de alto rendimiento, como la ejecución de operaciones numéricas a altas velocidades y en tiempo real. En contraste, los GPPs están diseñados para un amplio rango de funciones de propósito general y normalmente ejecutan grandes bloques de software, como son los sistemas operativos como UNIX, sin ejecutar tareas en tiempo real, esto en el caso de los microprocesadores, o están diseñados para aplicaciones orientadas al control, como manejo de interrupciones, instrumentación y control de eventos externos, esto en el caso de los microcontroladores.

Cabe anotar que al diferenciar los DSPs de los GPPs se debe tener en cuenta si son de alto desempeño (high-end) que se refiere a aquellas arquitecturas que utilizan técnicas para mejorar el paralelismo de instrucciones y tienen velocidades de reloj más altas o si son bajo desempeño (low-end) que se refiere a las arquitecturas viejas.

Cuando se hace referencia al conjunto de instrucciones se dice que un DSP de bajo desempeño utiliza un conjunto especializado de instrucciones complejas, que pueden realizar múltiples operaciones y poseen pobre ortogonalidad; un conjunto de instrucciones se dice que es ortogonal si una instrucción puede utilizar cualquier registro en cualquier modo de direccionamiento, por ejemplo:

mac x0,y0,a x:(r0)+,x0 y:(r4)+,y0

En este caso la instrucción MAC siempre utiliza el mismo conjunto de registros. Por otra parte un GPP de bajo desempeño utiliza instrucciones de propósito general, que realizan una operación por instrucción y poseen buena ortogonalidad, por ejemplo:

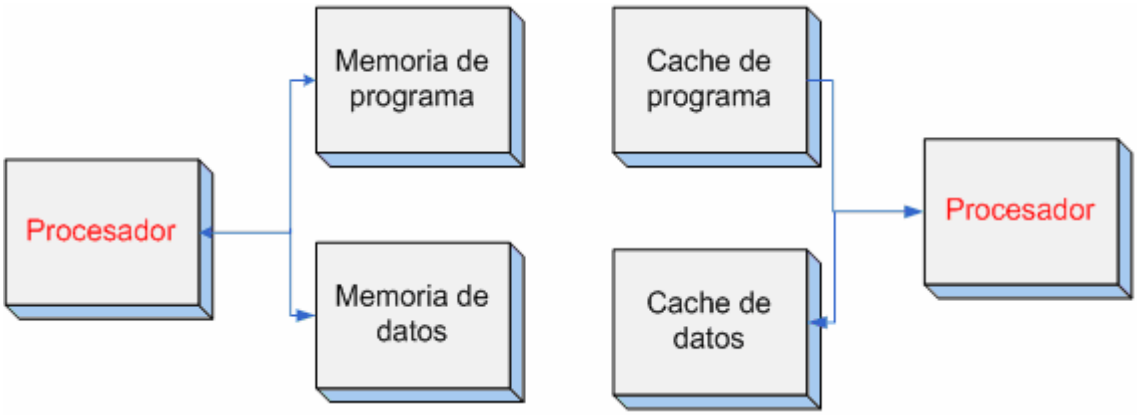
mov (r0),r2 ; mov (r1),r3; inc r0; inc r1

En los DSPs y GPPs de alto desempeño la ortogonalidad de las instrucciones es de moderada a excelente, con la diferencia que las instrucciones de los DSPs son complejas mientras que las de los GPPs son más simples.

En la organización del sistema de memorias los DSPs de bajo rendimiento por lo general utilizan la arquitectura Harvard, la cual permite de 2-4 accesos a memoria por ciclo, además estos DSPs no utilizan cache. Por otro lado los GPPs de bajo rendimiento utilizan la arquitectura Von Neuman, la cual permite típicamente un acceso a la memoria por ciclo y generalmente utilizan cache.

Los DSP de alto rendimiento también utilizan la arquitectura Harvard, asimismo tiene acceso por ciclo desde 1 a 8 instrucciones, estos algunas veces utilizan cache. Los GPPs de alto desempeño también utilizan la arquitectura Harvard y tiene un acceso por ciclo de 1 a 4 instrucciones y usualmente utilizan cache.

Figura 4.1. DSP de alto desempeño vs GPP de alto desempeño



En el tipo de arquitectura los DSPs de alto rendimiento utilizan VLIW que permiten ejecutar hasta 8 instrucciones/ciclo, mientras que los GPPs utilizan arquitecturas superescalares que ejecutan hasta 4 instrucciones/ciclo.

Los DSPs de alto y bajo desempeño utilizan unidades dedicadas para generar direcciones, además utilizan modos de direccionamiento especializados como: auto incremento, Modulo (circular), Reserva de bit (para el calculo de la FFT). Por otra parte los GPPs de alto y bajo desempeño generalmente no usan unidades dedicadas para generar direcciones y utilizan modos de direccionamiento de propósito general.

En términos de compatibilidad y disponibilidad los DSPs de alto y bajo desempeño poseen mas que todo arquitecturas propietarias (una arquitectura para un solo fabricante), además, en el mejor de los casos, hay limitada compatibilidad entre generaciones sucesivas, mientras que los GPPs de alto y bajo desempeño utilizan arquitecturas compartidas (una arquitectura para varios fabricantes) con frecuente compatibilidad entre las sucesivas generaciones.

Por último en los GPPs de alto desempeño las características dinámicas tales como ejecuciones superescalares, caches y predicción de saltos son fuertemente utilizadas, mientras que en los DSPs de alto desempeño se utilizan en menor medida dado que estas características complican el desarrollo del software para aplicaciones de procesamiento digital de señales en tiempo real[18].

4.1 DSPs vs ASICs vs FPGAs

Los Circuitos Integrados de Aplicación Especifica o ASICs (por su sigla en ingles Application Specific Integrated Circuits) son dispositivos optimizados en consumo y velocidad para realizar una aplicación concreta. Tiene la desventaja de que no son reconfigurables, sus características no pueden ser cambiadas o actualizadas durante el desarrollo de una aplicación dado que los fabricantes configuran el dispositivo según los requerimientos del usuario. Por otra parte los DSPs pueden ser actualizados sin cambiar la constitución física del chip; con solo cambios en el software se pueden reducir costos en el desarrollo de un producto y se pueden realizar mejoras de los códigos fuentes durante el desarrollo del mismo.

[18] Ibid., p. 50.

En resumen, los Circuitos Integrados de Aplicación Especifica pocas veces se utilizan en aplicaciones de procesamiento de señales en tiempo real. Se emplean comúnmente como interfaces de bus o aceleradores funcionales en sistemas basado en DSPs.

Los Arreglos de compuertas programables en campo o FPGA (por su sigla en inglés Field Programmable Gate Arrays, son matrices de compuertas lógicas eléctricamente programables que se caracterizan por tener alta densidad de compuertas, un número grande de entradas y salidas definidas por el usuario en un esquema de interconexión flexible y en un entorno de desarrollo similar al de una matriz de compuertas. Además tiene la capacidad de ser reconfigurable dentro de un sistema, lo que puede ser una gran ventaja en aplicaciones que necesitan múltiples versiones en un desarrollo, lo que permite rapidez para comercializar un determinado producto. Asimismo ofrecen un gran potencial para desempeñar una operación específica debido a sus circuitos lógicos dedicados. Sin embargo las FPGAs son significativamente más costosas y típicamente disipan más potencia que un DSP con la misma funcionalidad. Por eso, aunque las FPGAs son la tecnología escogida en aplicaciones como infraestructuras inalámbricas, los DSPs son usados por lo general junto con FPGAs para brindar gran flexibilidad, mejor desempeño y precio y menor consumo de energía.

5. PROGRAMACIÓN DE DSPs

5.1 Lenguaje ensamblador

Un ensamblador traduce el código fuente en lenguaje de máquina (secuencia de unos y ceros), que son las instrucciones que un DSP puede ejecutar. En la mayoría de los casos es una traducción uno a uno, cada línea del lenguaje ensamblador es traducida en una sola instrucción al DSP.

5.1.1 Ventajas de programar en Lenguaje ensamblador

Existen varias razones por las cuales un programador de DSPs debe utilizar el lenguaje ensamblador:

- El lenguaje ensamblador le brinda un control completo al programador, dado que le permite optimizar la velocidad de ejecución y el uso de memoria.
- Los requerimientos en desempeño típicamente exigen que los programadores aprovechen al máximo los recursos del DSP, y para pequeñas partes del código, la mejor manera de lograr esto es programar en lenguaje ensamblador.

- Si se programa en ensamblador con eficiencia y el DSP permite la ejecución completa del algoritmo en la memoria interna, se evita el uso de memorias externas con lo cual se reducen costos en el desarrollo y adicionalmente se evita el retraso que se produce cuando se accesa una memoria externa.

5.1.2 Desventajas de programar en Lenguaje ensamblador

Basado en los hechos anteriores, parecería que la programación en lenguaje ensamblador es la elección correcta. Si embargo existen desventajas muy significativas al programar en este lenguaje:

- La programación en lenguaje ensamblador requiere que el programador codifique los algoritmos de procesamiento digital de señales a un nivel muy detallado, lo que resulta en un código relativamente largo.
- Desarrollar un algoritmo en lenguaje ensamblador típicamente toma mucho tiempo, dado que se requiere entender en detalle las funciones de los registros, las instrucciones especializadas, los accesos a memoria y las interrupciones. Además los errores de programación son más difíciles de ver y la depuración de errores es más difícil.
- El código fuente escrito en ensamblador no puede ser transportado a nuevas arquitecturas fácilmente. Si se elige un procesador diferente es código por lo general debe ser reescrito.

En otras palabras es más difícil mantener y agregar nuevas funcionalidades a un programa escrito en lenguaje ensamblador dado su dependencia a la arquitectura del DSP[19].

De cualquier modo programar en lenguaje ensamblador es muy tedioso. Si un determinado proceso puede ser automatizado, un programador de DSPs podría concentrarse en tareas de más alto nivel. Una manera de conseguirlo es programar en lenguaje C, que es el lenguaje de alto nivel preferido para la programación de DSPs, y que gradualmente está reemplazando la programación en lenguaje ensamblador.

De hecho en los DSPs más recientes, el lenguaje C es el único método de programación viable, dado que escribir un código en ensamblador para estos dispositivos es muy complicado.

5.2 Lenguaje C

Cuando se programa en lenguaje C se manipulan variables abstractas sin ninguna referencia a un hardware en particular. Un compilador es un programa usado para transformar directamente el código fuente escrito en C, en lenguaje máquina. Esto requiere que el compilador asigne al hardware locaciones de memoria a cada una de las variables abstractas que son referenciadas.

[19] RIDDER, Robert. Programming digital signal processors with high-level languages. IN: DSP engineering. 2000

5.2.1 Ventajas de programar en Lenguaje C

Programar en lenguaje C ofrece varias ventajas sobre la programación en ensamblador:

- Los ambientes de programación del lenguaje C se encargan de muchas tareas de gestión interna del DSP. Esto mejora la confiabilidad de un programa de aplicación, especialmente en áreas donde la prueba y validación es difícil.
- El código fuente del lenguaje C es más fácil de leer y entender. Esto ayuda al programador a identificar errores y facilita el trabajo de agregar y actualizar algunas características a un programa existente.
- La programación en lenguaje C aísla al programador del hardware.
- Esto hace la programación mucho más fácil y permite al código fuente ser transportado entre diferentes procesadores. La clave de programar en C radica en que el programador no necesita entender la arquitectura del procesador; el conocimiento de este se lo deja al compilador.
- La gran ventaja de utilizar lenguaje C es su popularidad, dado que casi todos los procesadores tienen un compilador C lo que permite transportar y reutilizar el código fuente con solo compilarlo nuevamente.

5.2.2 Desventajas de programar en Lenguaje C

Existen varias razones por las cuales un programador puede evitar el uso de C para programar un DSP:

- Las antiguas arquitecturas de DSPs las cuales fueron diseñadas para lenguaje ensamblador no son muy apropiadas para ser programadas en C. Por ejemplo muchos lenguajes de alto nivel como C, requieren accesos aleatorios a elementos en la pila del sistema, y algunas arquitecturas no proveen instrucciones para este propósito.
- El código escrito en C usualmente requiere una memoria más larga que el lenguaje ensamblador, lo que resulta en un hardware más costoso.
- El lenguaje C no fue inicialmente creado para aplicaciones de procesamiento de señales. Adicionalmente, en su forma original no presenta soporte para formatos fraccionales de punto fijo.
- Es más difícil de optimizar un programa elaborado en C, puesto que su eficiencia depende en gran medida de la calidad del compilador.

5.3 Migración de C a C++

El lenguaje de programación C++ fue desarrollado para soportar la programación orientada a objetos. Este se ha convertido el lenguaje elegido para programar gran número de aplicaciones en el campo de la informática.

Sin embargo, el uso de C++ en la programación de DSPs está menos difundido. Las razones principales son:

- Ineficiencia en el código generado por los compiladores C++.
- Disponibilidad limitada de compiladores C++ para los DSPs populares.

5.3.1 Ventajas de C++

Las ventajas de C++ sobre C son:

- Habilidad de proveer un nivel de abstracción más alto, con lo cual se reduce el tiempo de desarrollo.
- Estructura modular obligatoria, lo que produce un mejor diseño del código y expansibilidad.
- Mejor encapsulamiento de módulos, lo que conlleva a una mejor reutilización del código.
- Compatibilidad con C, lo que permite fácil migración.

5.3.2 Desventajas de C++

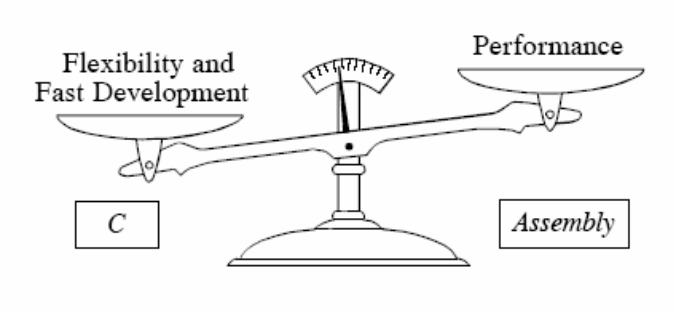
- La programación C++ puede volverse compleja.
- C++ tiene un flujo de programación menos intuitivo, debido al llamado automático de constructores y destructores.

- Los diseños más generales conducen a grandes requerimientos de memoria[20].

De acuerdo a todo lo anterior surge la siguiente pregunta: ¿Cuál es el mejor lenguaje para programar un DSP? La respuesta depende de lo que es más importante en una aplicación particular.

Si se necesita flexibilidad y rápido desarrollo, la respuesta es C. Por otro lado, se recomienda el uso de ensamblador si se necesita el mejor desempeño posible. Este compromiso se ilustra en la siguiente figura.

Figura 5.1. Ensamblador vs. C



A continuación se describen algunas consideraciones que se deben tener en cuenta al programar un DSP:

[20] Ibid,.p 69.

1. ¿Qué tan complicado es el programa? Si es largo y confuso, se aconseja el uso de C. Si es pequeño y simple, ensamblador puede ser una buena elección.
2. ¿Se requiere la máxima velocidad del DSP? Si es así, ensamblador aprovechará al máximo el desempeño del dispositivo. Para aplicaciones menos exigentes, se puede usar C.
3. ¿Cuantos programadores trabajan en la aplicación? Si el proyecto es lo suficientemente largo para más de un programador, hay que apoyarse en C y usar ensamblador en líneas de código para segmentos críticos.
4. ¿Qué es más importante, el precio del producto o el costo de desarrollo? Si es el precio del producto, se escoge ensamblador; si es el costo de desarrollo se escoge C.
5. ¿Qué sugiere el fabricante que se debe usar? hay que tener presente este concepto dado que el fabricante aconseja que un determinado procesador es más fácil de programar en un lenguaje que en otro, y la omisión de este consejo puede conducir a largos meses de entrenamiento para programar un procesador en particular[21].

5.4 Programación gráfica

Históricamente, los algoritmos escritos en lenguaje C o ensamblador con sus respectivos compiladores y ensambladores son usados para producir el código máquina que interpreta el DSP. El desarrollo de aplicaciones usando estas técnicas de programación implica cierta experiencia por parte del programador y por lo general mayor trabajo y tiempo.

[21] SMITH, Op. cit., p. 20.

Otra desventaja es que la documentación y la descripción del sistema por lo general muestran tareas independientes.

Por eso, surge la necesidad de desarrollar una metodología que permita el desarrollo de aplicaciones de forma clara y didáctica que permita obtener el código máquina empleando menos tiempo.

La programación grafica o por diagrama de bloques permite que los algoritmos de procesamiento digital de señales sean creados por métodos gráficos, simplemente conectando los respectivos bloques funcionales, que son considerados como cajas negras, pero cada uno de los cuales son responsables de implementar tareas especificas. La unión de estos bloques funcionales se utiliza para producir directamente el código máquina que entiende el DSP, tal como lo ilustra la siguiente figura[22].

Figura 5.2. Programación grafica

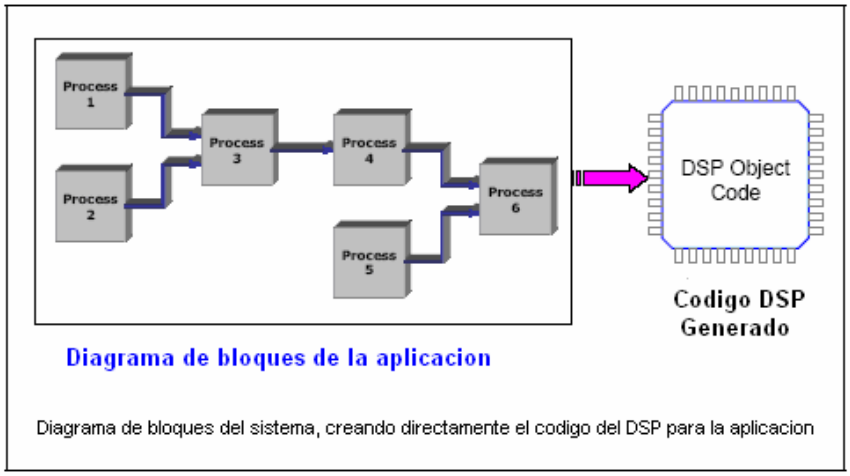


Diagrama de bloques del sistema, creando directamente el codigo del DSP para la aplicacion

[22] www.hyperception.com/pdf/wp/us/graphical_programming_for_dsps.pdf

La creación de aplicaciones de procesamiento digital de señales basadas en la programación gráfica permite el desarrollo de diseños válidos de manera rápida y fácil, y se convierte en una ventaja para aquellos diseñadores que tienen poca experiencia en lenguajes de programación textuales como C o ensamblador.

A demás esta técnica permite el mantenimiento y la reutilización del código porque al basarse en la unión de diagramas de bloques facilita la comprensión a futuros diseñadores de mejorar o ampliar una determinada aplicación.

Hyperception una compañía de la National Instruments ha estudiado las bondades que ofrece la programación gráfica y ha diseñado una herramienta de desarrollo que explota esta técnica llamada RIDE/VAB®(Real time Integrated Development Environment/ Visual Application Builder), que produce directamente el código máquina que maneja el DSP desde una construcción gráfica y posee todas las herramientas de software para optimizar el diseño de una aplicación de procesamiento digital de señales. Entre las aplicaciones en donde se utiliza la programación gráfica se encuentran: Procesamiento de imágenes, procesamiento de habla, procesamiento de audio, control de motores, robótica y aplicaciones inalámbricas.

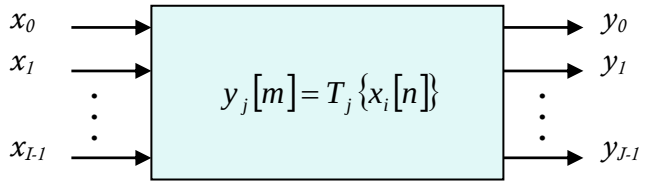
6. METODOLOGIA DE DESARROLLO DE SISTEMAS EN TIEMPO REAL

6.1 Definición de Tiempo Real.

Las aplicaciones de procesamiento digital de señales tienen como objetivo extraer información útil de una serie de señales de entrada, representadas como secuencias de números. Las muestras de una señal de entrada son procesadas por bloques denominados tramas para lo cual se requiere cierto tiempo computacional. El procesamiento en tiempo real completa las operaciones necesarias para calcular la salida asociada a una trama, en un tiempo que no excede la duración de la trama.

Como ejemplo considérese un sistema de procesamiento digital de señales que realiza una transformación sobre un conjunto de señales de entrada xi[n] $0 \leq i \leq I-1$, obtenidas a partir del muestreo de señales continuas, con una frecuencia de muestreo fx=1/Tx, y como salida se proporciona un segundo conjunto de señales yj[m] $0 \leq j \leq J-1$. Este segundo conjunto de señales puede tener un período de espaciamiento de las muestras Ty diferente al de las señales de entrada, pero tanto Tx como Ty permanecen constantes a lo largo de los intervalos de análisis. La transformación realizada por el sistema puede observarse en la gráfica a continuación:

Figura 6.1. Esquema de un sistema de procesamiento de señales



En la situación más general, cada muestra de salida yj[m] puede depender de todas las muestras de las señales de entrada, es decir de xi[n], $-\infty < n < \infty$. En la práctica, la dependencia de yj[m] sobre xi[n] está limitada a un rango mucho menor de n. Si se considera que la salida y0[m] depende de los valores de la entrada x0[n], $0 \leq n \leq L-1$. Estas muestras forman una trama de datos de longitud L, que se extiende en un intervalo de L*Tx segundos.

Para llevar a cabo los cálculos de la transformación T0 y obtener y0[1], se requiere un cierto tiempo de cómputo Tc, a partir de que la muestra x0[L-1] haya sido recibida. Si el tiempo de cómputo es menor que el tiempo que demoran en producirse las L muestras requeridas para el cálculo (Tc < L*Tx), es decir, si el procesamiento finaliza antes de que la última muestra que contribuye para el cálculo de y0[2] esté lista, las operaciones para el siguiente marco pueden iniciarse en el momento en que x0[2L-1] sea recibida. Y de esta manera, para una longitud de trama L y tiempo de cómputo constantes Tc, el sistema podrá operar continuamente sin errores de sincronismo. La señal de salida tendrá un período: Ty = L*Tx y cada muestra se producirá con un retardo de Tc segundos después de que la trama de datos de la señal de entrada se haya recibido.

Un sistema con Tc > L*Tx no puede operar continuamente, porque las muestras de la señal de entrada acumularían cada vez en mayor proporción antes de ser procesadas, hasta que llega el punto en que se pierden datos. Si $Tc \leq L*Tx$ se dice que el sistema opera en tiempo real[23].

[23] ACKENHUSEN, John. Real time signal processing: design and implementation of signal processing systems. New Jersey: Prentice Hall, 1999.

6.2 Características de un sistema de tiempo real

Un sistema en tiempo real (STR) es un sistema informático que interactúa repetidamente con su entorno físico, responde a los estímulos que recibe del mismo dentro de un plazo de tiempo determinado.Estos sistemas son capaces de realizar un procesamiento de datos en línea, es decir, recibe las señales de entrada, las procesa y entrega resultados inmediatamente. A continuación se presentan algunas características de este tipo de sistemas:

Requisitos temporales: El tiempo de respuesta es fundamental en un sistema de tiempo real. Sus requerimientos de tiempo de respuesta dependen básicamente del ancho de banda de las señales sobre las que se opera y varían de unos sistemas a otros. Existen tres tipos de requisitos temporales que dependen la aplicación:

1. Tiempo Real estricto (Hard real time): en el cual todas las acciones debe ocurrir dentro del plazo especificado, de no ser así los resultados pueden ser catastróficos, por ejemplo el control de vuelo de un avión.
2. Tiempo Real flexible (Soft real time): en el cual se pueden perder plazos de vez en cuando y la acción sigue siendo valida, pero el valor de la respuesta decrece con el tiempo, por ejemplo adquisición de datos.
3. Tiempo Real firme (Firm real time): se pueden perder plazos ocasionalmente. Una respuesta tardía no tiene valor, por ejemplo perdida de una trama en sistemas de audio o video.

Cabe resaltar que en un mismo sistema puede haber componentes con distintos tipos de requisitos temporales.

Determinismo temporal: Un STR por lo general, interactúa con variables físicas, las cuales en la mayoría de los casos, tienen un comportamiento no determinístico, con eventos que ocurren de forma asincrónica, concurrente y en un orden impredecible. Por lo tanto, un sistema en tiempo real debe estar en capacidad de atender y sincronizar estos eventos y realizar las operaciones sin exceder el tiempo máximo de procesamiento permitido (plazo), es decir, ser capaz de realizar acciones en intervalos de tiempo determinados.

Concurrencia (simultaneidad de acciones, paralelismo): La concurrencia permite que dispositivos físicos controlados funcionen al mismo tiempo, pero esta operación adiciona complejidad al diseño del software.

Seguridad y fiabilidad: un STR debe ser altamente confiables, ya que muchos de estos sistemas desempeñan un papel importante en donde están operando y una falla en su funcionamiento puede traer consecuencias graves, por ejemplo, pérdidas de vidas humanas, pérdidas económicas, daños medioambientales[24].

6.3 Factores que influyen en la complejidad de los algoritmos

Un algoritmo es un método para transformar un conjunto de datos en otro. En el diseño de un STR deben considerarse ciertos aspectos que incrementan la complejidad de los algoritmos.

[24] CONTRERAS, Sonia y CASTELLANOS, Germán. Detección Activa de voz orientada a la clasificación de fonemas aislados. Bucaramanga, 2003. p 45-53. Tesis de Maestría. Universidad Industrial de Santander. Facultad de Ingeniería Físico- Mecánicas.

- Rendimiento: El rendimiento en un sistema está determinado por el número de operaciones que deben realizarse, la cantidad de datos sobre los cuales se realizan estas operaciones y el tiempo disponible para ello.
- Rango numérico y precisión: los datos en un sistema digital se representan por medio de códigos binarios de longitud fija. La cantidad de bits empleados para representar un dato determina su precisión. Con las operaciones se incrementa el rango numérico de los datos, ya que por ejemplo, al sumar dos cantidades de N bits se obtiene un número de N+1 bits y al multiplicarlas, un número de N*N bits. Por tanto, si se dispone de un número fijo de bits, los resultados que exceden este número deben ser reducidos en tamaño, bien sea por medio de escalamiento, redondeo o truncamiento; lo cual afecta su precisión. Otro aspecto que debe tenerse en cuenta es que a medida que se incrementa el número de bits, las operaciones son más demoradas.
- Operaciones de más de un ciclo de reloj: las operaciones lógicas, suma y multiplicación, cuentan con un hardware dedicado en los procesadores digitales de señales, lo cual permite que puedan realizarse en un solo ciclo de reloj. Sin embargo, cálculos matemáticos,como los logaritmos y las funciones trigonométricas, requieren de múltiples aproximaciones para obtener el resultado.
- Dependencia de datos: cuando los cálculos dependen de los resultados de otras operaciones, hay dificultad para lograr paralelismo de tareas y de datos. Cuando se presentan retardos en la actualización de un valor que es requerido por una tarea deben insertarse estados de espera.
- Tiempo de vida de los datos y precedencia: las operaciones que usan un dato una vez y luego lo descartan son más apropiadas para implementarlas con algoritmos de procesamiento seriales, porque se ahorra espacio de memoria y se reduce la latencia de resultados. La precedencia se refiere a que las primeras etapas de un algoritmo deben completarse antes de iniciar las siguientes etapas.

Como ejemplo puede considerarse un sistema de reconocimiento de voz, en el cual la señal debe pasar por las etapas de preproceso, extracción de características y reconocimiento, que deben ejecutarse en orden hasta terminar. La precedencia reduce la habilidad para alcanzar paralelismo de tareas.

6.4 Desarrollo de algoritmos de proceso en tiempo real

El desarrollo de algoritmos de procesamiento digital de señales comienza en un entorno en tiempo diferido como MATLAB®, debido a que un software especializado brinda las herramientas necesarias para llevar a cabo el proceso de planteamiento y depuración de los algoritmos. Una vez se ha verificado que el algoritmo cumple los objetivos para los cuales fue diseñado y su desempeño es el adecuado, se procede a su adecuación y posterior implementación para ejecución en tiempo real. Como primera medida debe realizarse una descripción completa del algoritmo. A continuación, éste debe ser adaptado a la arquitectura del DSP seleccionado.

En esta fase es importante estudiar la manera de optimizar el programa con base en los recursos disponibles, para obtener la mayor eficiencia posible. Los pasos generales del proceso de desarrollo se resumen a continuación:

1. Simplificar el algoritmo: Aplicar transformadas rápidas. Aprovechar regularidad. Reordenar estructuras.

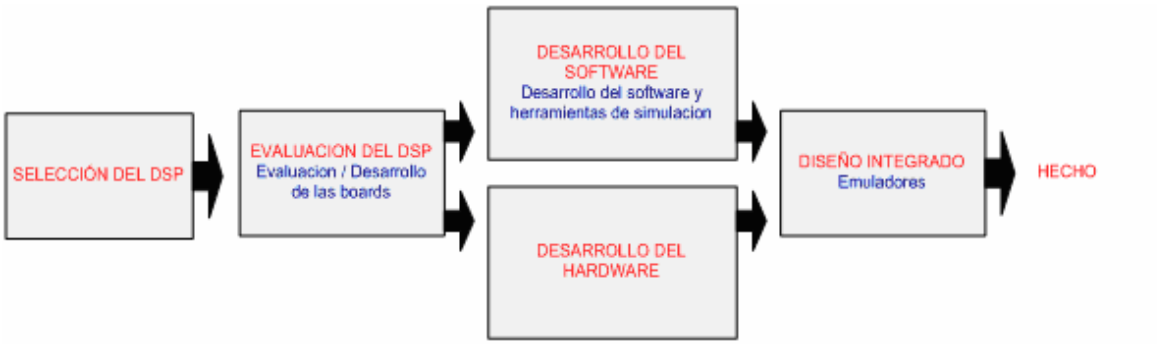
2. Desarrollar el diagrama de flujo de datos: Particionar el algoritmo en módulos. Seleccionar módulos en hardware o software. Realizar simulaciones de desempeño. Determinar la temporización de los módulos. Determinar la comunicación entre módulos. Definir la línea de tiempo de ejecución.
3. Mejorar la descripción del diagrama de flujo de datos: Desarrollar lazos. Reducir y optimizar. Actualizar temporización y comunicación entre módulos.
4. Caracterizar la arquitectura: Recursos y velocidad del procesador (o los procesadores). Enlaces de comunicación.
5. Adaptar el algoritmo al procesador (o sistema multiprocesador): Desarrollar el programa. Desarrollar la línea de tiempo de ejecución. Simular. Medir desempeño, temporización, E/S, comunicación. Identificar cuellos de botella.
6. Chequeo y validación del sistema: Verificar que no se excede la capacidad del procesador, memoria o E/S[25].

6.5 Sistemas embebidos de tiempo real

Cuando se pretende desarrollar una aplicación embebida, la metodología de desarrollo es la siguiente:

[25] Ibid,. p. 80

Figura 6.2. Flujo de diseño de una aplicación embebida



Como primera medida se debe elegir un DSP que se adapte mejor a la aplicación y para eso se hace referencia a los criterios de selección mencionados como son: Formato de datos, Ancho de palabra, velocidad, organización de la memoria, soporte técnico y facilidad de desarrollo, consumo y costo.

Como segunda instancia se evalúan las características del DSP seleccionado y las herramientas de software y hardware que provee el fabricante y que serán utilizadas en el desarrollo de la aplicación.

Para el desarrollo del software se tiene en cuenta la metodología de desarrollo de algoritmos en tiempo real expuesta. Pueden utilizarse cualquiera de las técnicas de programación descritas en párrafos anteriores como lo son ensamblador, lenguaje C o incluso la programación gráfica. En la etapa de desarrollo del hardware puede incluirse la tarjeta del DSP y otros componentes que sirven de interfaz con las señales del mundo real.

Por último en la etapa del diseño integrado, convergen el desarrollo del software y hardware para crear el proyecto final. Un emulador es una herramienta que se usa para probar el diseño final.

7. EJEMPLO DE APLICACIÓN SOBRE EL DSP56303EVM DE MOTOROLA

7.1 Características del DSP

El DSP56303 es una miembro de la familia DSP5600 de procesadores digitales de señales CMOS de Motorola. Sus características de alto desempeño son:

- Núcleo de la Familia DSP5600

✓ Desarrolla 66/80/100 millones de instrucciones por segundo (MIPS) usando un reloj interno de 66/80/100 MHz.
✓ El código fuente es compatible con el núcleo de la familia DSP5600.
✓ El conjunto de instrucciones es altamente paralelo.
✓ Posee un multiplicador-acumulador paralelo de 24x24 bits completamente segmentado o pipelined.
✓ Posee un desplazador combinatorio (barrel shifter) paralelo de 56 bits.
✓ Soporta operaciones aritméticas de 24 o 16 bits.
✓ Los modos de direccionamiento que maneja están optimizados para aplicaciones de procesamiento digital de señales.
✓ Posee una cache de instrucciones on-chip.
✓ Presenta un controlador DMA concurrente de seis canales on- chip.
✓ Presenta un PLL (Phase Lock Loop) on-chip.
✓ Presenta un modulo de emulación (OnCE^{TM}) on-chip.

✓ Puerto JTAG (Joint Test Action Group)

- Memorias on-chip

- ✓ Los tamaños de las memorias: la RAM de programa, la cache de instrucciones, la RAM X de datos, la RAM Y de datos son programables:

Tabla 7.1. Configuración de la memoria

Instruction Cache	Switch Mode	Program RAM Size	Instruction Cache Size	X Data RAM Size	Y Data Ram Size
disabled	disabled	4096 × 24-bit	0	2048 × 24-bit	2048 × 24-bit
enabled	disabled	3072 × 24-bit	1024 × 24-bit	2048 × 24-bit	2048 × 24-bit
disabled	enabled	2048 × 24-bit	0	3072 × 24-bit	3072 × 24-bit
enabled	enabled	1024 × 24-bit	1024 × 24-bit	3072 × 24-bit	3072 × 24-bit

- Expansión de memoria off-chip

- ✓ Expansión de la memoria de datos a dos espacios de memoria de 256Kx24 bits de palabras
- ✓ Expansión de la memoria de programa a un espacio de memoria de 256Kx24 bits de palabras.
- ✓ Puerto de expansión de memoria externa.
- ✓ Chip select que no requiere circuitería adicional para conectarse a las SRAMs o SSRAMs
- ✓ Controlador DMA on-chip que no requiere circuitería adicional para conectarse al a la DRAM.

- Periféricos on-chip

- ✓ Interfaz paralela de 8 bit HI8, interfaz en bus compatible ISA, que provee una efectiva solución en precio para aplicaciones sin requerir el bus PCI.

- ✓ Dos Enhanced Synchronous Serial Interfaces (ESSI).
- ✓ Una interfaz de comunicación serial SCI con generador de velocidad en baudios.
- ✓ Módulo Timer triple
- ✓ Hasta 34 pines de entada /salida de propósito general (GPIO), programados dependiendo de cuales periféricos son habilitados.

- Disipación de energía

- ✓ Diseño CMOS de muy baja potencia.
- ✓ Modos de baja potencia wait y stop.

Para una descripción detallada del DSP56303 ir al manual[26].

7.1.1 Características generales del sistema de desarrollo

El DSP56303 Evaluation Module (DSP56303EVM) está diseñado como una plataforma de bajo costo para desarrollar productos de hardware y software en tiempo real para soportar la nueva generación de aplicaciones en comunicaciones inalámbricas, telecomunicaciones y productos multimedia usando en procesamiento de voz, datos y fax, videoconferencias, aplicaciones de audio, control y en procesamiento digital de señales en general.

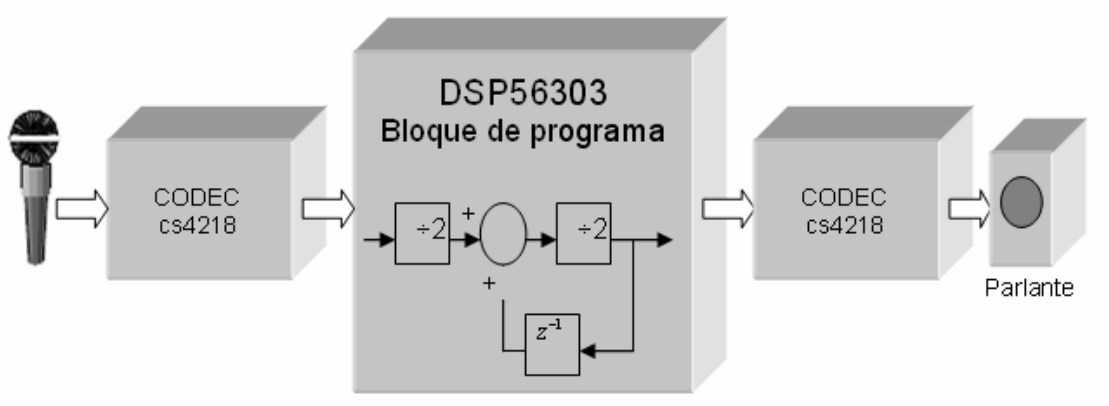
[26] Cannon, William. DSP56303EVM User's Manual and Reference guide. Motorola.

El usuario puede cargar el software al chip o a la RAM de la tarjeta para ejecutar y depurar el programa. También puede conectar hardware, como memorias externas, conversores ADC o DAC, para el desarrollo de una aplicación. La precisión de 24 bit del DSP56303 combinada con los 32K de SDRAM externa y el CS4218 16bit stereo audio codec hace que la tarjeta DSP56303EVM se ideal para implementar y demostrar muchos algoritmos de procesamiento de audio y comunicaciones, así como para aprender la arquitectura y conjunto de instrucciones del DSP56303. En el Anexo B se muestra el diagrama en bloques del DSP303EVM.

7.2 Estructura general de la aplicación

A continuación se presenta la implementación de un programa que genera un ECO sobre una señal de entrada usando un número de muestras retrasadas en el tiempo. El diagrama de bloques del sistema es el siguiente:

Figura 7.1. Diagrama en bloques de la aplicación



Para la implementación del eco en el DSP, en primera instancia una señal de voz se suministra al codec que realiza la conversión análogo/digital, y luego la señal digitalizada pasa hacia el DSP56303, donde el bloque de programa consiste en dividir por dos la señal de entrada para mantener la estabilidad y luego se le suma a ésta una muestra retrasada en el tiempo. La suma de la señal es de nuevo dividida por dos y se envía al codec donde se realiza la conversión digital/análogo y se escucha el efecto en los parlantes.

El desarrollo de esta aplicación se basó en al nota de aplicación denominada Programming the CS4218 CODEC for Use With DSP56300 Devices by Thomas Lay, en donde se explica detalladamente la utilización y configuración del CODEC para esta aplicación en particular.

En el Anexo C se presenta la parte principal del programa fuente de la aplicación, en lenguaje ensamblador.

8. CONCLUSIONES

Los DSP poseen arquitecturas especialmente diseñadas para acelerar los intensos cálculos matemáticos utilizados en la mayoría de sistemas de procesamiento digital de señales. La organización del sistema de memorias, ya sea la arquitectura Von Neuman, Harvard o Super Harvard que muestran la interconexión de la memoria de datos y la memoria de programa con la CPU, así como también el tipo de instrucciones que utiliza la CPU, CISC, RISC o VLIW, brindarán mejoras en el desempeño del procesador en términos de la velocidad de ejecución de instrucciones.

Además, la utilización de técnicas arquitectónicas como el Pipelining o Segmentación y el SIMD permiten mejorar el desempeño global de algunos algoritmos ya que estas permiten explotar el paralelismo de instrucciones, es decir, la ejecución de varias instrucciones simultáneamente, en el caso del Pipelining, y la ejecución de una instrucción con un conjunto de datos diferentes, en el caso de SIMD.

Los Dsp se utilizan en muchas aplicaciones, desde radar y sonar hasta la electrónica de consumo. Características básicas como el formato aritmético, velocidad, organización de memoria o la arquitectura interna hacen que sean o no adecuados para una aplicación en particular, así como otras que no hay que olvidar, como pueden ser el costo, consumo o la disponibilidad de una amplia gama de herramientas de desarrollo.

La tendencia es que en el mercado de los DSPs, los fabricantes van incorporando procesadores con características que cada vez más están adaptadas a las particularidades de las diferentes aplicaciones.

En este sentido, aunque a nivel de prestaciones varios DSP puedan reunir los requisitos necesarios exigidos por una aplicación, hay que ponderar la importancia requerida de los criterios de selección mencionados anteriormente y tomarlos como marco de referencia al momento de adquirir el procesador final.

La principal diferencia existente entre los DSPs y los procesadores de propósito general es que un DSP está optimizado para realizar tareas de procesamiento digital de señales que requieren de manipulaciones matemáticas a altas velocidades y en tiempo real, mientras que los microprocesadores y microcontroladores están diseñados para realizar tareas de propósito general o tareas orientadas al control. Además la principal diferencia de un DSP con un ASIC, es que este último no se puede reconfigurar en campo mientras que un DSP solo con cambios en software se puede actualizar durante el desarrollo de una aplicación. Y por último y FPGA y un DSP están optimizados para realizar funciones parecidas y la diferencia radica en que la forma de programar una FPGA está orientada al hardware, es decir, a la interconexión de compuestas lógicas y bloques de memoria para realizar una tarea especifica, además su costo y consumos son mayores que los de un DSP de la misma funcionalidad.

El cumplimiento del requisito de operar en tiempo real de un sistema involucra una serie de factores tales como requisitos temporales, concurrencia, seguridad y fiabilidad y otros como rendimiento, precisión, operaciones de más de un ciclo de reloj, dependencia y precedencia de datos que aumentan la complejidad del sistema de procesamiento de señales. Estos factores dependen de las características del hardware del sistema, su arquitectura, la velocidad de operación y tiempo de conversión A/D y D/A, capacidad de memoria, entre otros, que introducen retardos y disminución de ganancias en el momento de finalizar la aplicación. Por esto además de los requerimientos de software, el Hardware asociado a las tareas de procesamiento de señales son de vital importancia en el momento de realizar una aplicación embebida.

BIBLIOGRAFÍA

- SMITH, Steven W. The scientist and engineer's guide to digital signal processing. 2 ed. California.

- CONTRERAS, Sonia y CASTELLANOS, Germán. Detección Activa de voz orientada a la clasificación de fonemas aislados. Bucaramanga, 2003. p 45-53. Tesis de Maestría. Universidad Industrial de Santander. Facultad de Ingeniería Físico- Mecánicas.

- ACKENHUSEN, John. Real time signal processing: design and implementation of signal processing systems. New Jersey: Prentice Hall, 1999.

- STALLINGS, William. Organización y arquitectura de computadores. 5 ed. Madrid: Prentice- Hall, 2000.

- LAPSLEY, Phil; BIER, Jeff; SHOHAM, Amit and LEE, Edward. Dsp processor fundamentals: architectures and features". California: Berkeley Design Technology, Inc. 1994-1996

- SALAZAR, Jordi. Procesadores digitales de señal (dsp), Arquitecturas y Criterios de selección. Universidad Politécnica de Cataluña. Faculta de Ingeniería electrónica. Centro de sistemas y sensores electrónicos.

- SKOLNICK, David and LEVINE, Noan. Digital signal processing: an introductory course in dsp system design. 1997.

- EYRE, Jennifer and BIER, Jeff. The evolution of dsp processors. Berkeley Design Technology, Inc.

- RIDDER, Robert. Programming digital signal processors with high-level languages. IN: DSP engineering. 2000.

- Choosing a DSP Processor. Berkeley Design Technology, Inc. 1996-2000.

- Cannon, William. DSP56303EVM User's Manual and Reference guide. Motorola.

- www.semiconductors.philips.com/ acrobat/other/vliw-wp.pdf

- www-gti.det.uvigo.es/~pedro/pub/sodtr/ps/cap09.pdf

- www.hyperception.com/pdf/ wp/us/graphical_programming_for_dsps.pdf

ANEXOS

Anexo A. Guía De Selección De Dsps.

CHIPS										
Vendor	Family	Floating, Fixed, or Both	Data Width	Instruction Width	Core Clock Speed [1]	**BDTImark2000™** *BDTIsimMark2000™* [2]	Total On-Chip Memory, Bytes	Core Voltage	Unit Price [3]	Notes
Agere Systems	DSP1641x	Fixed point	16 bits	16/32 bits	285 MHz	**1360** [4]	388 K–644 K	1.2, 1.575	$52–61	Dual-MAC architecture
Analog Devices	ADSP-218x	Fixed point	16 bits	24 bits	80 MHz	**240**	20 K–256 K	1.8	$4–24	Many family members w/ assorted peripherals
	ADSP-219x	Fixed point	16 bits	24 bits	160 MHz	**410**	20 K–160 K	2.5	$10–24	Enhanced version of the ADSP-218x
	ADSP-2116x/ 2126x (SHARC)	Floating point	32/40 bits	48 bits	200 MHz	**950**	128 K–768 K	1.2, 1.8	$10–106	Features SIMD, strong multiprocessor support
	ADSP-2136x (SHARC)	Floating point	32/40 bits	48 bits	333 MHz	n/a	896 K	1.2	$17–36	SHARC with a lengthened pipeline for higher clock speeds
	ADSP-BF5xx (Blackfin)	Fixed point	16 bits	16/32 bits	750 MHz	**4190** [4,5]	84 K–328 K	0.8–1.45, 1.0–1.6	$5–40	Dual-MAC DSP with variable speed and voltage
	ADSP-TS20x (TigerSHARC)	Both	8/16/32/40 bits	32 bits	600 MHz	**6400** [5]	512 K–3 M	1.0, 1.2	$47–197	4-way VLIW with SIMD capabilities; uses eDRAM
Freescale [6]	DSP563xx	Fixed point	24 bits	24 bits	275 MHz	**820**	24 K–649 K	1.25, 1.6, 1.8, 3.3	$4–47	Many audio-oriented parts; binary-compatible with ’560xx
	DSP56F8xx	Fixed point	16 bits	16 bits	40 MHz [7]	**110**	28 K–152 K	2.5, 3.3	$3–12	Contains many microcontroller-like features
	DSP5685x/ MC56F83xx	Fixed point	16 bits	16 bits	120 MHz	*340*	22 K–300 K	1.8, 2.5	$4–17	Enhanced version of the ’568xx
	MSC71xx (SC1400)	Fixed point	16 bits	16 bits	200 MHz	**2240**	88 K–408 K	1.2	$12–24	Based on SC1400 licensable core (see below)
	MSC81xx (SC140)	Fixed point	16 bits	16 bits	400 MHz	**4490** [4]	514 K–1440 K	1.0, 1.2, 1.6	$76–205	Based on SC1400-compatible core; most chips use 4 cores
Intel	PXA255/26x (XScale)	Fixed point	16/32 bits	16/32 bits	400 MHz	**930**	66 K–32 M	1.0–1.3	$19–68 [3]	ARM-compatible; features SIMD support, “stacked” flash
	PXA27x (XScale/ Wireless MMX)	Fixed point	16/32 bits	16/32 bits	624 MHz	**2140**	322 K–64 M	0.85–1.55	$32–? [3]	Adds 64-bit-wide SIMD operations and "stacked" SDRAM
LSI Logic	LSI40x (ZSP400)	Fixed point	16/32 bits	16 bits	200 MHz	**940**	96 K–252 K	1.2, 1.8	$8–15	Based on ZSP400 licensable core (see below)
NEC	µPD77050 (SPXK5)	Fixed point	16 bits	16/32 bits	250 MHz	*1770*	400 K	0.9–1.5	$15	Dual-MAC DSP with variable speed and voltage

Renesas [8]	SH76xx (SH2-DSP)	Fixed point	16 bits	16/32 bits	100 MHz	**280**	20 K–24 K	1.9, 3.3	$10–15	Hybrid DSP/microprocessor based on SH2-DSP
	SH772x (SH3-DSP)	Fixed point	16 bits	16/32 bits	200 MHz	**490**	32 K	1.7, 1.8, 1.9, 2.0	$11–17	Hybrid DSP/microprocessor based on SH3-DSP
	SH775x (SH-4)	Both	16/32 bits	16 bits	240 MHz	**750**	32 K	1.5	$16–25	Based on SH-4 licensable core (see below)
Texas Instruments	TMS320C24x/ TMS320F24x	Fixed point	16 bits	16/32 bits	40 MHz	n/a	13 K–69 K	3.3	$2–8	Hybrid microcontroller/DSP
	TMS320C28x/ TMS320F28x	Fixed point	32 bits	16/32 bits	150 MHz	n/a	40 K–294 K	1.8, 1.9	$5–15	Hybrid microcontroller/DSP; assembly-compatible w/ 'C24x
	TMS320C54x	Fixed point	16 bits	16 bits	160 MHz	**500** [4]	24 K–1280 K	1.5, 1.6, 1.8, 2.5	$4–107	Many specialized instructions
	TMS320C55x	Fixed point	16 bits	8–48 bits	300 MHz	**1460**	80 K–376 K	1.2–1.6, 1.26, 1.6	$5–19	Dual-issue, dual-MAC DSP; assembly-compatible w/ 'C54x
	TMS320C62x	Fixed point	16 bits	32 bits	300 MHz	**1920**	72 K–896 K	1.5, 1.8	$9–102	8-way VLIW, dual-MAC DSP
	TMS320C64x	Fixed point	8/16 bits	32 bits	1 GHz	**9130**	160 K–1056 K	1.1, 1.2, 1.4	$18–219	Adds quad-MAC capabilities and specialized operations to 'C62x
	TMS320C67x	Floating point	32 bits	32 bits	300 MHz	**1470**	72 K–264 K	1.2, 1.26, 1.4, 1.8, 1.9	$14–105	Floating point version of 'C62x

CORES										
Licensor	Family	Floating, Fixed, or Both	Data Width	Instruction Width	Core Clock Speed [1]	**BDTImark2000™** *BDTIsimMark2000™* [2]	Total Core Memory Space, Bytes	Core Voltage	Process	Notes
ARM	ARM7	Fixed point	32 bits	16/32 bits	133 MHz	*140*	4 G	1.2	0.13µm	Widely licensed 32-bit microprocessor core
	ARM9	Fixed point	32 bits	16/32 bits	250 MHz	*310*	4 G	1.2	0.13µm	Adds separate data bus for data access, deeper pipeline to ARM7
	ARM9E	Fixed point	16/32 bits	16/32 bits	250 MHz	*520*	4 G	1.2	0.13µm	ARM9 enhanced with single-cycle MAC unit
	ARM10E	Fixed point	16/32 bits	16/32 bits	325 MHz	n/a	4 G	1.2	0.13µm	Adds load/store unit, branch prediction, 64-bit buses
	ARM11	Fixed point	16/32 bits	16/32 bits	550 MHz	n/a	4 G	1.2	0.13µm	Adds SIMD and deeper pipeline

CEVA [9]	CEVA-TeakLite	Fixed point	16 bits	16 bits	190 MHz	n/a	256 K	1.2	0.13µm	Single-MAC, single-issue DSP core
	CEVA-Teak	Fixed point	16 bits	16 bits	180 MHz	n/a	8 M	1.2	0.13µm	Dual-MAC DSP core
	CEVA-Palm	Fixed point	16/20/24 bits	16/32 bits	180 MHz	n/a	32 M	1.2	0.13µm	Selectable data width, dual-MAC, dual-issue DSP core
	CEVA-X1620	Fixed point	8/16 bits	16/32 bits	450 MHz	*3620*	4 G	1.2	0.13µm	8-way VLIW, dual-MAC DSP core
LSI Logic	ZSP200	Fixed point	16/32 bits	16 bits	260 MHz	n/a	256 K	1.2	0.13µm	2-way superscalar variant of the ZSP400
	ZSP400	Fixed point	16/32 bits	16 bits	260 MHz	**1220**	256 K	1.2	0.13µm	4-way superscalar DSP core
	ZSP500	Fixed point	16/32 bits	16/32 bits	340 MHz	*2690*	64 M	1.2	0.13µm	Second-generation ZSP; 4-way superscalar
	ZSP600	Fixed point	16/32 bits	16/32 bits	310 MHz	n/a	64 M	1.2	0.13µm	Second-generation ZSP; 6-way superscalar
Philips	CoolFlux	Fixed point	24 bits	32 bits	175 MHz	n/a	640 K	1.2	0.13µm	Dual-MAC core targets low-power audio applications
StarCore	SC1200	Fixed point	16 bits	16 bits	340 MHz	*2690*	4 G	1.2	0.13µm	Dual-MAC, 4-issue variant of the SC1400
	SC1400	Fixed point	16 bits	16 bits	305 MHz	**3420**	4 G	1.2	0.13µm	Synthesizable version of quad-MAC, 6-issue SC140
SuperH [11]	SH-4	Both	16/32 bits	16 bits	266 MHz	**830**	4 G	1.2	0.13µm	Superscalar microprocessor with 3D geometry instructions
	SH-5	Fixed point	16/32 bits	16/32 bits	400 MHz	*1560*	4 G	1.2	0.13µm	Microprocessor with SIMD, optional floating-point
Tensilica	Xtensa V/ Vectra	Fixed point	8/16/24 bits [12]	16/24 bits	275 MHz [13]	n/a	4 G	1.0	0.13µm	Customizable core with optional DSP features
	Xtensa LX/ Vectra LX	Fixed point	18 bits [12]	16/24/32/64 bits	370 MHz [13]	*6150* [14]	4 G	1.2	0.13µm	VLIW-based customizable core; compatible with Xtensa V

NOTES:

[1] Chips: clock speed for fastest family member. Cores: worst-case clock speed.

[2] The BDTImark2000 and BDTIsimMark2000 provide a summary measure of DSP speed; for both scores, higher is faster. Both scores are calculated with the same formula, but BDTIsimMark2000 scores may use projected clock speeds. See http://www.BDTI.com/bdtimark/BDTImark2000.htm for additional information and scores.
Note: BDTImark2000 scores are shown in **bold** and BDTIsimMark2000 scores in *italic*.

[3] Unit prices based on vendors' quotations for 10,000 quantities. Except for Intel parts, prices are as of the second quarter of 2004. Intel prices are as of the first quarter of 2004; Intel has not provided full pricing data for the PXA27x.

[4] Score for one core. Some family members contain multiple cores. Details available from BDTI.

[5] Score does not apply to some family members, which use slightly different architectures. Details available from BDTI.

[6] Freescale is the former semiconductor division of Motorola.

[7] Instruction rate. The DSP56F8xx requires two clock cycles per instruction cycle.

[8] Renesas was formed by the merger of Hitachi Semiconductor with Mitsubishi Electric.

[9] CEVA was previously known as ParthusCeva, which was formed by the merger of the core licensing division of DSP Group with Parthus Technologies.

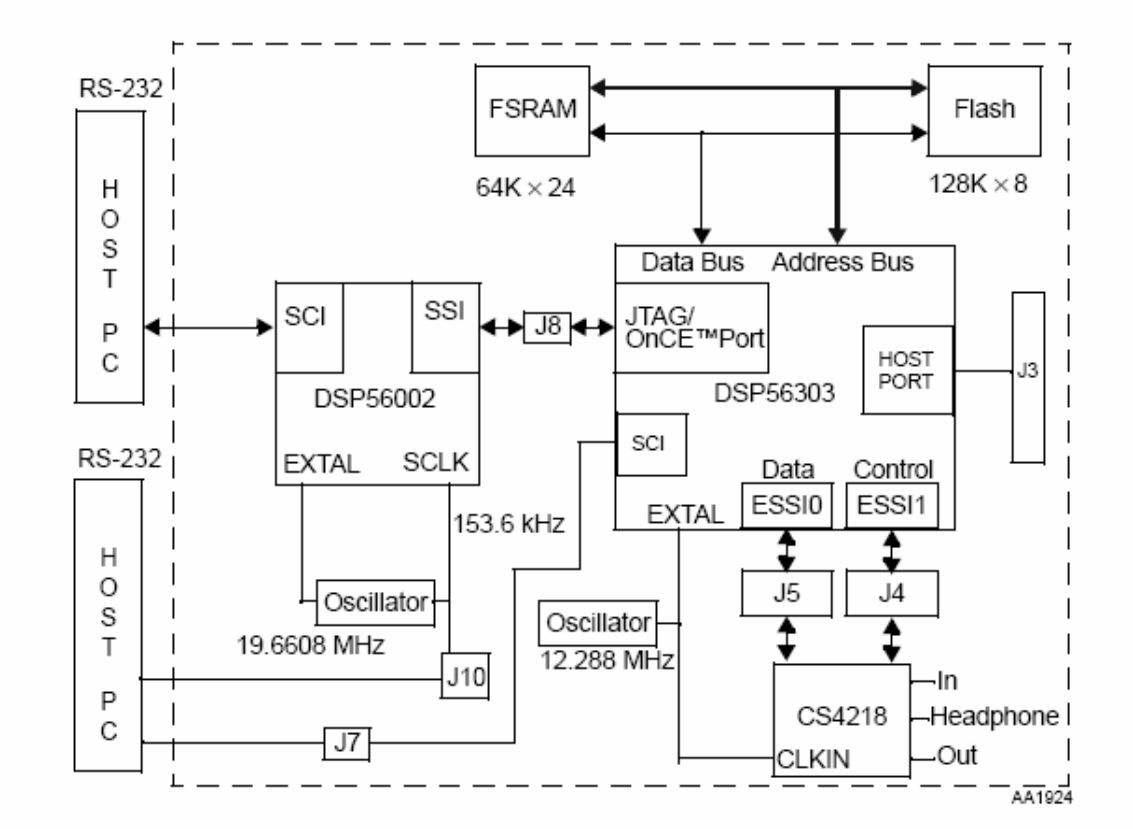
[11] SuperH is an independent processor core licensing company formed by Hitachi and STMicroelectronics.

[12] Native multiplier width(s) available in the Vectra (or Vectra LX) DSP engine. Users may add custom instructions that support other data widths.

[13] Clock speeds vary depending on configuration.

[14] Score assumes use of 12 custom instructions that expand the area of the core by 16%. Licensees may require greater or lesser degrees of customization.

Anexo B. Estructura General De La Dsp56303evm



Anexo C. Código Del Programa De La Aplicacion

```
;*************************************************************************
    nolist
    include 'ioequ.asm'
    include 'intequ.asm'
    include 'ada_equ.asm'
    include 'vectors.asm'
        list
 ;*************************************************************************

;---Buffer for talking to the CS4218

    org   x:$0
RX_BUFF_BASE        equ    *
RX_data_1_2    ds      1                    ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4    ds      1                    ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE        equ      *
TX_data_1_2    ds      1                    ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4    ds      1                    ; data time slot 3/4 for TX ISR (right audio)

RX_PTR       ds    1                       ; Pointer for rx buffer
TX_PTR       ds    1                       ; Pointer for tx buffer

CTRL_WD_12    equ   MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34    equ   MIN_LEFT_GAIN+MIN_RIGHT_GAIN

    org    p:$100
START
main
    movep  #$040006,x:M_PCTL                ; PLL 7 X 12.288 = 86.016MHz
    ori    #3,mr                            ; mask interrupts
    movec  #0,sp                            ; clear hardware stack pointer
    move   #0,omr                           ; operating mode 0
    move   #$40,r7                          ; initialize stack pointer
    move   #-1,m7                           ; linear addressing
    jsr    ada_init                         ; initialize codec

    move   #$0400,r4                        ; start echo buffer at $400
    move   #$03FF,m4                        ; make echo buffer 1024 deep

    clr    a                                ; clear a
    rep    #$03FF                           ; clear the echo buffer
    move   a,l:(r4)+

echo_loop

    jset   #3,x:M_SSISR0,*                  ; wait for rx frame sync
    jclr   #3,x:M_SSISR0,*                  ; wait for rx frame sync
        clr a
```

```
        clr b
        move    x:RX_BUFF_BASE,a            ; receive left
        move    x:RX_BUFF_BASE+1,b          ; receive right
        asr     a       x:(r4),x0           ; divide them by 2 and get oldest
        asr     b       y:(r4),y0           ; samples from buffer
        add     x0,a                        ; add the new samples and the old
        add     y0,b
        asr     a                           ; reduce magnitude of new data (to ensure stability)
        asr     b
        move    a,x:(r4)                    ; save the altered samples
        move    b,y:(r4)+                   ; and bump the pointer
        move    a,x:TX_BUFF_BASE            ; transmit left
        move    b,x:TX_BUFF_BASE+1          ; transmit right

        jmp     echo_loop

        include 'ada_init.asm'              ; used to include codec initialization routines
echo
    end
```