T
621.382
V1L
c.2





CIB-ESPOL

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"Diseño e implementación en software y en hardware de un decodificador Viterbi para el estándar IEEE802.11a WLAN"

TESIS DE GRADO

Previa a la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentada por:

Ricardo Villafuerte Calderón

Luis Oñate Malo

Nury Ramírez Cely



CIB-ESPOL

GUAYAQUIL – ECUADOR

AÑO

2007

AGRADECIMIENTO

Agradecemos:

Primeramente a Dios por todas sus bendiciones dadas para la terminación de este trabajo.

Al PhD. Moisés Tacle por su apoyo personal e institucional, por su confianza y por habernos abierto una puerta en nuestra carrera profesional.

Al Ing. Pedro Vargas, por su valiosa gestión dentro de la ESPOL y por su continuo apoyo en la culminación de nuestra carrera.

A los doctores José Luis Leyva y Jesús Vásquez del Centro de Investigación y Estudios Avanzados -CINVESTAV- Unidad Guadalajara, por la oportunidad de estudios brindada.

A los ingenieros del Centro de Diseño Electrónico -CDE- del Instituto Tecnológico y de Estudios Superiores de Monterrey -ITESM- Campus Guadalajara, por su confianza y asesoría en la implementación de este proyecto.

DEDICATORIA

A mi familia y amigos,
Por su constante apoyo
Y por siempre estar conmigo
En todos los años de mi carrera
RICARDO VILLAFUERTE CALDERÓN

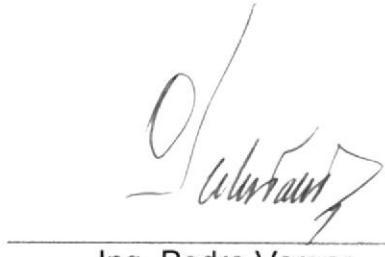
A Dios, La Virgen Santísima y Don Bosco,
A mi Madre y a mi familia
Que son la razón de mi vida,
A mis amigos por su apoyo incondicional,
A todos mi eterna y sincera gratitud
Por su enorme participación en la
Consecución de esta importante
Meta en mi vida.
LUIS ALBERTO OÑATE MALO

A Dios,
A mis padres,
A mis tíos y
A mis amigos,
Que con cariño y comprensión
Me ayudaron, impulsaron y
Siempre me apoyaron
Durante mi carrera.
NURY RAMÏREZ CELY

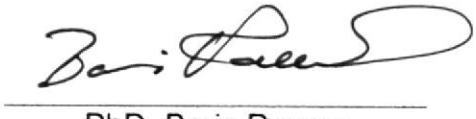
TRIBUNAL DE GRADUACIÓN



Ing. Holger Cevallos
SUBDECANO DE LA FIEC



Ing. Pedro Vargas
DIRECTOR DE TESIS



PhD. Boris Ramos
MIEMBRO PRINCIPAL



MSc. Rebeca Estrada
MIEMBRO PRINCIPAL

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL" (Reglamento de Graduación de la ESPOL).

Ricardo Villafuerte Calderón

Luis Oñate Malo

Nury Ramírez Cely

RESUMEN



CIB-ESPOL

El presente trabajo de tesis presenta el diseño y la implementación en software y en hardware de un decodificador de Viterbi para el estándar de comunicaciones inalámbricas IEEE 802.11a.

El prototipo del decodificador de Viterbi fue implementado en VHDL, lo que constituye el diseño en hardware, además fue implementado sobre el conjunto de instrucciones de un procesador MIPS de la serie R2000 también diseñado e implementado para este proyecto, constituyendo el diseño en software. Ambas implementaciones tienen las siguientes características: Decodificador convolucional de decisión suave para la entrada y de decisión dura para la salida, longitud restrictiva de 7 y tasa de codificación de 1/2.

La tesis esta compuesta por 4 capítulos. El capítulo 1 corresponde al fundamento teórico del estándar IEEE 802.11a y su relación con los códigos convolucionales, también expone la teoría del Algoritmo del Viterbi empleado en un decodificador inalámbrico. El capítulo 2 introduce los conceptos básicos detrás de la arquitectura y diseño de un procesador MIPS segmentado. El capítulo 3 muestra los detalles de las implementaciones, primero del decodificador en hardware, luego la construcción del procesador MIPS, y en base a este último, la implementación del decodificador en

software. El capítulo 4 despliega los reportes y análisis de los resultados de las dos implementaciones. Finalmente, culminamos la tesis con nuestras conclusiones y recomendaciones.

Las implementaciones fueron sintetizadas usando las herramientas de Xilinx y Altera, y fueron probados sobre cuatro diferentes familias de FPGAs. La verificación funcional permitió comprobar que el diseño satisface los requerimientos planteados.

ÍNDICE GENERAL

RESUMEN VI
ÍNDICE GENERAL VIII
ABREVIATURAS XIII
INDICE DE FIGURAS XV
INDICE DE TABLAS XVIII
INTRODUCCIÓN 1

1. Fundamentos Teóricos del decodificador Viterbi 4

1.1. Estándares WLAN 4

1.2. Generalidades del estándar IEEE 802.11a 5

1.2.1. OFDM 6

1.2.2. Transmisor-Receptor IEEE 802.11a 9

1.3. Códigos Convolucionales 14

1.3.1. Generación 17

1.3.2. Respuesta Impulso y Función de Transferencia 19

1.3.3. Representación 20

1.3.4. Distancia Mínima 23

1.4. Decodificación de Máxima Verosimilitud: Algoritmo de Viterbi 24

1.4.1. Algoritmo de Viterbi Decodificación Dura y Suave 27

1.4.2. Probabilidades de Error 30

1.4.3. Arquitectura de un Decodificador Viterbi 33

1.4.3.1. Unidad de Cálculo de Métricas de 33

Rama –BMU

1.4.3.2. Unidad de Cálculo de Métricas de Nodo –ACS 35

1.4.3.3. Unidad de Manejo de bits sobrevivientes –SMU 37

2. Arquitectura del Procesador MIPS 41

2.1. Arquitectura del conjunto de instrucciones de MIPS 41

2.1.1. Tipos de instrucciones 42

2.1.2. Formatos de Instrucción 43

2.2. Diseño de un procesador MIPS de un solo ciclo 44

2.2.1. Unidades Funcionales 46

2.2.2. Diseño de una ALU 47

2.2.3. Camino de datos 48

2.2.4. Unidad de Control 52

2.2.5. Excepciones e Interrupciones 56

2.3. Segmentación 57

2.3.1. Camino de Datos segmentado 59

2.3.2. Control de un MIPS segmentado 62

2.3.3. Riesgos por Dependencia de Datos 65

2.3.4. Control de Riesgos por Dependencia de 69

Datos: Burbujas

2.3.5. Reducción de Riesgos por Dependencia de Datos: Anticipación 70

2.3.6. Riesgos de Control 74

2.3.7. Rendimiento de Sistemas Segmentados 75

3. Implementación del Decodificador Viterbi en Software y en Hardware 78

3.1. Implementación del decodificador Viterbi en VHDL 79

3.1.1. BMU 79

3.1.2. ACS 84

3.1.3. SMU 86

3.1.4. Cama de pruebas 94

3.2. Implementación de un procesador MIPS segmentado 98

3.2.1. Descripción de componentes por etapas de Segmentación 98

3.2.2. Unidad de control principal 105

3.2.3. Control de la ALU 105

3.2.4. Unidad de anticipación para datos 107

3.2.5. Recorrido de las instrucciones en el camino de datos 109

3.2.6. Comparador 113

3.2.7. Multiplicador 115

3.2.8. Unidad de anticipación para saltos 116

3.2.9. UART 118

3.2.10. Unidad de Programación externa 123

3.2.11. Manejo de interrupciones 127

3.3. Desarrollo en C de un canal de comunicación inalámbrica con codificación de canal 132

3.3.1. Descripción del programa 132

3.3.2. Obtención de las curvas de BER en Matlab para diferentes parámetros 138

3.4. Implementación del decodificador Viterbi en el procesador MIPS 145

3.4.1. Traducción del algoritmo en C a lenguaje máquina 145

3.4.2. Descripción de la Implementación 147

3.4.3. Cama de pruebas 153

4. Análisis y Resultados 154

4.1. Resultados de la implementación en SW 154

4.1.1. Reportes de síntesis del MIPS sobre los FPGA 155

4.1.2. Reporte de estimación de consumo de potencia 157

del MIPS

4.1.3. Análisis de Tiempo de las operaciones del decodificador 160

4.1.4. Resultados de cama de prueba: Gráficas de Probabilidad de Error 163

4.2. Resultados de la implementación en HW 169

4.2.1. Reportes de síntesis del decodificador Viterbi sobre los FPGAs 170

4.2.2. Reporte de estimación de consumo de potencia del decodificador Viterbi 178

4.2.3. Simulaciones 180

4.2.4. Resultados de cama de prueba: Gráficas de Probabilidad de Error 184

4.3. Propuestas de Co-diseño HW-SW del decodificador de Viterbi 189

CONCLUSIONES Y RECOMENDACIONES

ABREVIATURAS

IEEE	Institute of Electrical and Electronics Engineers. Instituto de Ingenieros Eléctricos y Electrónicos.
WLAN	Wireless Local Area Networks. Redes Inalámbricas de Área Local
FPGA	Field Programmable Gate Array. Arreglo de Compuertas de Campos Programables
MIPS	Microprocessor without Interlocked Pipeline Stages, Microprocesador con etapas de segmentación sin bloqueo
OFDM	Orthogonal Frequency Division Multiplexing. Modulación por División Ortogonal de Frecuencia.
VHDL	Very High Speed Integrated Circuit Hardware Description Language. Lenguaje de Descripción de Hardware para circuitos integrados de muy alta velocidad.
BMU	Branch Metric Unit. Unidad de cálculo de métricas de rama.
ACS	Add-Compare-Select. Suma, Compara y Selecciona.
SMU	Survivor Management Unit. Unidad de Manejo de Bits Sobrevivientes.
TB	Trace-Back. Rastreo hacia atrás.
ALU	Arithmetic Logic Unit. Unidad Aritmético Lógica.
UART	Universal Asinchronous Receiver/Transmiter. Transmisor/Receptor Asíncrono Universal
Mbps	Mega bits per second. Mega bits por Segundo.
BER	Bit Error Rate. Tasa de Error de bits.
SW	Software
HW	Hardware
3GPP	The 3rd Generation Partnership Project. Proyecto Conjunto de Tercera Generación
GSM	Global System for Mobile communications. Sistema Global para comunicaciones móviles
HiperLAN/2	High Performance Radio LAN version 2. Alto Desempeño en Redes de Área Local versión 2.
MMAC	Mobile Multimedia Access Communications. Comunicaciones de Acceso Móvil
BPSK	Binary Phase-Shift Keying. Desplazamiento en Fase Binaria
QPSK	Quadrature Phase-Shift Keying. Modulación por cambio de fase en cuadratura
M-QAM	Multiple Quadrature-Amplitude Modulation. Modulación Múltiple de Amplitud en Cuadratura
IFFT	Inverse Fast Fourier Transformation. Transformada Rápida Inversa de Fourier

DC	Direct current. Corriente Directa
IS95	Interim Standard 95. Estándar Interino 95. (Primer estándar para redes CDMA)
CDMA	Code Division Multiple Access. Multiplexación por División de código
WCDMA	Wideband Code Division Multiple Access. Multiplexación por División de código de Banda Ancha.
CPI	Cycles per Instruction. Ciclos por instrucción
LIFO	Last input first output. Ultimo que ingresa-primero en salir
SNR	Signal Noise Rate. Relación Señal-Ruido
dB	Decibelios
LUTS	Look Up Tables
I/O	Input/Outputs
BRAM	Block Random Access Memory
Pe	Probabilidad de Error
SOC	System on Chip. Sistema sobre un chip.



CIB-ESPOL

ÍNDICE DE FIGURAS

Figure 1.1	Configuración de un Sistema OFDM	8
Figure 1.2	Diagrama de Bloques de un transmisor-receptor IEEE802.11a	10
Figura 1.3	Codificador Convolucional	16
Figura 1.4	Codificador Convolucional (3,1,3) de 4 estados	18
Figure 1.5	Codificador Convolucional (2,1,7) de 64 estados para el estándar IEEE 802.11a.	19
Figura 1.6	Diagrama de Estados	22
Figura 1.7	Diagrama de Trellis	23
Figura 1.8	Modelo de canal con decodificación dura y suave	28
Figura 1.9	Decisiones del decodificador dura y suave	32
Figura 1.10	Diagrama de bloques de unidades principales de un decodificador Viterbi	33
Figura 1.11	Transición de estados en una sección del diagrama de Trellis	35
Figura 1.12	Diagrama de bloques de Arquitectura de unidad ACS	36
Figura 1.13	Arquitectura de unidad de rastreo hacia atrás	39
Figura 2.1	Diseño de una ALU de un bit con solo tres operaciones: ADD, AND y OR.	48
Figura 2.2	Camino de datos sencillo con unidad de control.	54
Figura 2.3	Instrucciones en ejecución en el camino de datos de un solo ciclo suponiendo ejecución segmentada.	62
Figura 2.4	Camino de datos segmentado.	63
Figura 2.5	Dependencias segmentadas en una secuencia de cinco instrucciones utilizando caminos de datos simples para mostrar las dependencias.	68
Figura 2.6	Una versión de la figura 2.5, mostrando la forma en que realmente se inserta las detenciones en la segmentación	71
Figura 2.7	Efecto de la anticipación para corregir las dependencias de datos	73
Figura 2.8	Desempeño de los diferentes caminos de datos en cuanto a la frecuencia de reloj y CPI.	77
Figura 3.1	Diagrama de Bloques de interconexión de unidades BMU	84
Figura 3.2	Diagrama de Bloques de interconexión de unidades ACS	86
Figura 3.3	Diagrama de bloques de interconexión de unidades TB	89
Figura 3 4	Funcionamiento de Bloques TB en el diseño del decodificador	91
Figura 3.5	Diagrama de bloques de conexión de las LIFOS	93
Figura 3.6	Cama de Prueba	94
Figura 3.7	Imagen de Lena	95

Figura 3.8	Unidad de Anticipación dentro del procesador segmentado	108
Figura 3.9	Esquema implementado para programar el MIPS	127
Figura 3.10	Selección de la dirección de la rutina de interrupción	128
Figura 3.11	Esquema de elección del PC cuando ocurre una interrupción	129
Figura 3.12	Esquema de selección de dato de entrada al banco de registros cuando ocurre interrupción	130
Figura 3.13	Decodificador Viterbi, r = ½	139
Figura 3.14	Decodificador Viterbi, r = 1/3	140
Figura 3.15	Decodificador Viterbi, r = ¼	141
Figura 3.16	Curvas de diferente Cuantización	142
Figura 3.17	Curvas de diferente Cuantización. Sin multiplicador	143
Figura 3.18	Decodificador de Viterbi con tramas variables	144
Figura 3.19	Rutina de TB sobre el procesador MIPS	152
Figura 4.1	Sumario de consumo de Potencia	158
Figura 4.2	Potencias subtotales del MIPS	159
Figura 4.3	Consumo de Corriente del MIPS	159
Figura 4.4	Datos Termales	160
Figura 4.5	Tiempo de Métrica de rama y nodo	161
Figura 4.6	Tiempo de TB 64 bits	161
Figura 4.7	Tiempo total de ejecución del decodificador	163
Figura 4.8	Curvas de Probabilidad de Error de bits para diferentes niveles de SNR de la implementación en SW	164
Figura 4.9	Imagen de Lena con SNR=10 dB en la implementación de decodificador en SW	165
Figura 4.10	Imagen de Lena con SNR=5 dB en la implementación de decodificador en SW	166
Figura 4.11	Imagen de Lena con SNR=2 dB en la implementación de decodificador en SW	166
Figura 4.12	Imagen de Lena con SNR=0 dB en la implementación de decodificador en SW	167
Figura 4.13	Reporte de consume de potencia del decodificador Viterbi usando Power Analyzer de Altera	179
Figura 4.14	Reporte de consumo de potencia del decodificador Viterbi usando XPower de Xilinx.	179
Figura 4.15	Simulación en que se muestra el retardo inicial del decodificador antes de obtener el primer bit de salida.	181
Figura 4.16	Resultado de la simulación del decodificador de Viterbi en el que se muestra las métricas de nodo.	183
Figura 4.17	Curvas de Probabilidad de Error de bits para diferentes niveles de SNR de la implementación en HW	186
Figura 4.18	Imagen de Lena con SNR= 10 dB en la implementación de decodificador en HW	186

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL ESPOL
CIB-ESPOL

Figura 4.19 Imagen de Lena con SNR= 5 dB en la implementación de decodificador en HW 187
Figura 4.20 Imagen de Lena con SNR= 2 dB en la implementación de decodificador en HW 187
Figura 4.21 Imagen de Lena con SNR= 0 dB en la implementación de decodificador en HW 188
Figura 4.22 Diagrama de Flujo del proceso de Co-diseño 192
Figura 4.23 Esquema de codiseño para optimización en área. 195
Figura 4.24 Esquema de codiseño para optimización en tiempo. 196

ÍNDICE DE TABLAS

Tabla I.	Estándares WLAN OFDM	4
Tabla II.	Principales Parámetros del Estándar IEEE802.11a	6
Tabla III.	Combinaciones de desempeño en el estándar 802.11 a	13
Tabla IV.	Campos de las instrucciones	45
Tabla V.	Señales básicas proporcionadas por la Unidad de control	52
Tabla VI.	Retardo para cada etapa para diferentes tipos de instrucciones	58
Tabla VII.	Operaciones de la ALU en la arquitectura MIPS implementada	102
Tabla VIII.	Otras señales de la Unidad de Control	106
Tabla IX.	Señal ALUCtrl de acuerdo a las combinaciones de entradas de ALUOp y function	107
Tabla X.	Valores de Divisor para las tasas de Baudios soportadas por la UART implementada	120
Tabla XI.	Selección de la paridad y baudios de acuerdo a los bits de la instrucción de configuración de la UART	122
Tabla XII.	Instrucciones MIPS del código TB del decodificador Viterbi	146
Tabla XIII	Resultado de síntesis para el procesador MIPS	156
Tabla XIV.	Resultados de Síntesis de Unidad de Cálculo de Métrica de Ramas -BMU	171
Tabla XV.	Resultados de Síntesis de Unidad de Cálculo de Métrica de Nodos -ACS	172
Tabla XVI.	Resultados de Síntesis de la Unidad de Rastreo hacia atrás -TB	174
Tabla XVII.	Resultados de Síntesis del Decodificador Viterbi en VHDL	176

INTRODUCCIÓN

En comunicaciones digitales los datos transmitidos se presentan en forma binaria, estos datos son modulados hasta obtener ondas analógicas que son transmitidos a través de un canal hacia un receptor. En el canal el ruido y la interferencia corrompen la señal transmitida, la cual vuelve a convertirse en datos binarios o bits en el receptor. Algunos errores de bits pueden ocurrir si la interferencia es muy fuerte, por eso se usa codificación de canal para prevenir estos errores. La codificación de canal significa que bits adicionales se añaden a los datos transmitidos para que luego estos bits puedan ser usados cuando se reconstruye la secuencia de datos transmitida en el receptor. Hay muchos métodos para codificación de canal empleados en su mayoría para los últimos estándares de comunicaciones inalámbricas como: 3GPP, GSM y WLAN, por ejemplo codificación convolucional y entre ellas la decodificación Viterbi, que representa una de las técnicas de codificación de canal más atractivas y tiene aplicación en muchos sistemas de comunicación satelitales y espaciales. El algoritmo de Viterbi fue introducido en el año de 1968 por Andrew J. Viterbi.

Uno de los principales objetivos de esta tesis fue satisfacer los requerimientos del estándar IEEE 802.11a WLAN, diseñado para proveer altas velocidades en redes de comunicaciones inalámbricas en un corto

rango. El estándar tiene una amplia variedad de altas tasas de datos disponibles hasta 54 Mbps; donde es obligatorio para todos los productos tener tasas de 6, 12, 24, dependiendo de los parámetros seleccionados para el sistema. Para alcanzar estas tasas de datos se aplican diferentes combinaciones de técnicas de modulación y tasa de codificación. Particularmente, para este trabajo se usa una tasa de codificación de ½ donde la tasa de datos que se debe alcanzar es de 24Mbps.

En este proyecto se construyó un decodificador Viterbi con entrada de decisión suave y salida de decisión dura, con una Longitud Restrictiva de 7, lo que equivale a 64 estados. La implementación tuvo dos objetivos: ser realizada en software y en hardware. Para el primer objetivo, primeramente se diseñó e implementó un procesador MIPS de 32 bits sobre un FPGA. Luego, la codificación en lenguaje C del algoritmo de Viterbi, permitió un análisis de los diferentes parámetros del estándar y a partir de este código se obtuvo el conjunto de instrucciones de MIPS R2000 que serían ejecutadas y verificadas sobre el procesador. Para la implementación en hardware, se codificó el algoritmo de Viterbi en el lenguaje de descripción de hardware VHDL. Ambas implementaciones sintetizadas en Xilinx y Altera fueron probadas sobre cuatro familias diferentes de FPGAs para analizar el consumo de potencia, recursos y tiempo sobre esta tecnología. Finalmente se analizó la factibilidad de un Co-diseño Hardware-Software para obtener un

sistema combinado del decodificador Viterbi que logre cumplir los requerimientos del estándar.

Este proyecto de Tesis fue patrocinado en su totalidad por el Centro de Diseño Electrónico -CDE- del Instituto Tecnológico y de Estudios Superiores de Monterrey -ITESM- Campus Guadalajara.

CAPÍTULO 1

FUNDAMENTOS TEÓRICOS DEL DECODIFICADOR VITERBI

1.1 Estándares WLAN

Actualmente, existen tres estándares WLAN que utilizan OFDM (la técnica OFDM será revisada brevemente en secciones siguientes) para sus especificaciones de capa física, ellos se listan en la siguiente tabla.

Tabla I. Estándares WLAN OFDM

Estándar	Región de Operación
IEEE802.11a	Norte América
HiperLAN/2	Europa
MMAC	Japón

CIB-ESPOL

HiperLAN/2 y MMAC ofrecen tasas de datos variables desde 6 Mbps hasta 54Mbps en la banda de los 5GHz. La principal diferencia entres estos estándares es la capa MAC usada por cada uno. No obstante, este no es el objetivo de este proyecto ya que estamos más

relacionados con la capa física de estos estándares, específicamente el estándar IEEE802.11a.

1.2 Generalidades del Estándar IEEE802.11a

El estándar 802.11a es el estándar para operaciones WLAN hasta tasas de datos de 54 Mbps en los 5GHz (5.12 – 5.825), banda de infraestructura de información no licenciada, el cual está diseñado para proveer corto rango, y altas velocidades para comunicaciones de redes inalámbricas. La capa MAC es la misma en 802.11 y 802.11b, pero este no usa la técnica de Espectro Ensanchado en la capa física. En vez de esto, usa OFDM, el cual es la base para tener altas tasas de datos. Este estándar usa 300MHz de ancho de banda y el espectro se divide en tres secciones, cada sección está sujeta a una restricción de máxima potencia permitida [1].

El estándar 802.11a tiene una amplia variedad disponible de altas tasas de datos: 6, 9, 12, 18, 24, 36, 48 y 54 Mbps, donde es obligatorio para todos los productos tener disponibles 6, 12 y 24 Mbps tasas de datos. Una lista de los principales parámetros para el sistema se presenta en la tabla II. La selección de una tasa de transmisión se determina por un esquema de adaptación del enlace, un proceso de

selección de la mejor tasa de codificación y esquema de modulación basado en condiciones de canal [1].

Tabla II. Principales Parámetros del Estándar IEEE 802.11a

Estándar IEEE 802.11a	
Tasa de Datos	6, 9, 12, 18, 24, 36, 48, 54 Mbps
Modulación	BPSK, QPSK, 16-QAM, 64-QAM
Tasas de Codificación	1/2, 9/16, 2/3, 3/4
Número de Sub-portadoras	52
Número de Tonos pilotos	4
Duración del símbolo OFDM	4 us
Intervalo de Guarda	800 us
Espaciado entre Sub-portadoras	312.5 kHz
Ancho de banda de la Señal	16.66 MHz
Espaciado del Canal	20 MHz

1.2.1 OFDM

Multiplexación por División Ortogonal de Frecuencia –OFDM- es una técnica de modulación ampliamente usada en muchos esquemas de comunicación inalámbrica, para alcanzar altas

tasas de datos mientras combate el desvanecimiento por multi-canales en comunicaciones inalámbricas.

El concepto de OFDM se basa en expandir los datos a ser transmitidos sobre un gran número de portadoras ortogonales. Las portadoras ortogonales son transmitidas simultáneamente, esto es, el canal completo es ocupado por medio de la suma agregada de las sub-bandas angostas y ortogonales.

Una señal OFDM se construye asignando un flujo de bits paralelos a aquellas sub-bandas portadoras, transmitiendo muchos símbolos en paralelo, la duración del símbolo se incrementa proporcionalmente, lo cual reduce los efectos de la interferencia intersímbolos –ISI- causados por el ambiente desvanecedor de Rayleigh [2].

Configuración de un Sistema OFDM

El diagrama de bloques de un sistema OFDM se muestra en la figura 1.1. La entrada es un flujo de datos seriales binarios. Esto se codifica usando cualquier modulación apropiada (BPSK, QPSK, M-QAM).

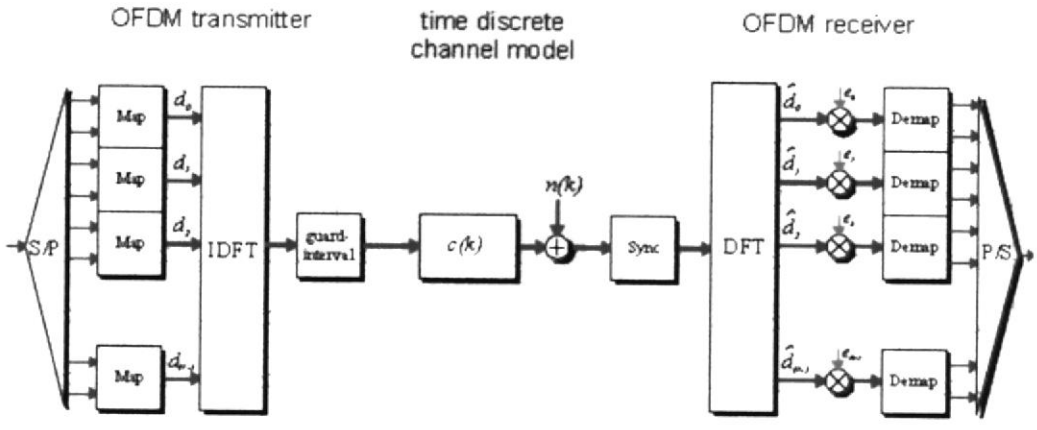


Figura 1.1 Configuración de un Sistema OFDM

Por ejemplo, si se usa una modulación M-QAM, el flujo de datos se convierte en un flujo de datos multinivel. La entrada binaria será transformada en una señal multinivel reduciendo la tasa de símbolo a:

$$D = \frac{R}{\log_2 M} \quad \text{sym/sec}$$

donde R es la tasa de bits del flujo de datos en bits/sec.

Si los datos seriales se convierten en paralelos, la tasa de datos se reduce por N, donde N es el número de canales paralelos. De aquí que, estos canales paralelos son esencialmente canales de bajas tasas de datos y debido a que son canales angostos, ellos

experimentan desvanecimiento plano. Esta es la principal ventaja de la técnica OFDM.

Estos flujos de datos paralelos se someten a una IFFT y luego se suman. Esto constituye la modulación OFDM. Después de la modulación OFDM, se inserta un intervalo de guarda para suprimir la ISI causada por la distorsión multi-camino. Este intervalo de guarda es también llamado prefijo cíclico. Un prefijo cíclico es la copia de la última parte del símbolo OFDM, el cual es añadido al símbolo transmitido. Esto hace periódico al símbolo transmitido, lo que juega un rol decisivo para identificar correctamente las tramas, y así para evitar la interferencia inter-símbolo e inter-portadora [2].

1.2.2 Transmisor-Receptor IEEE 802.11a

El diagrama de bloques de la capa física del estándar 802.11a se muestra en la figura 1.2.

La entrada de datos binaria es inicialmente enviada a un revolvedor de secuencia pseudo-aleatoria de longitud 127. El

propósito del revolvedor es revolver una secuencia larga de 1s y 0s. Esto ayuda a la recuperación de tiempo del receptor.

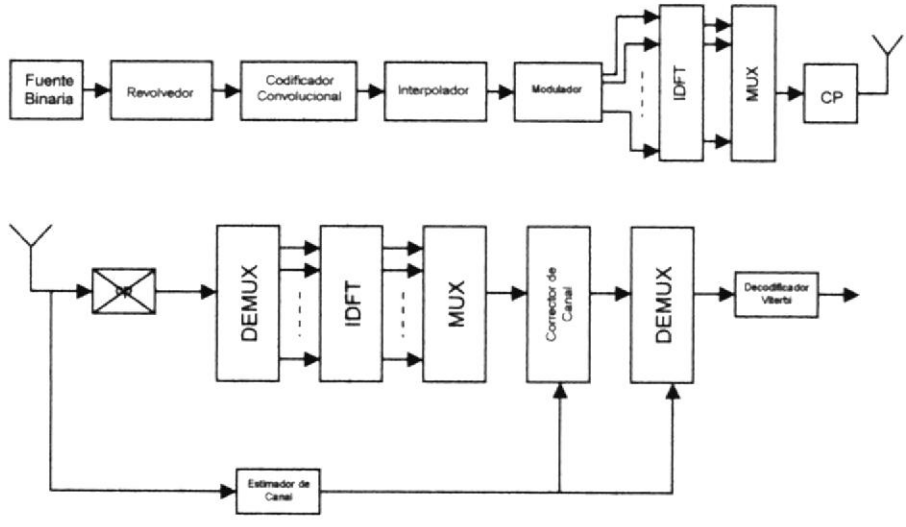


Figura 1.2 Diagrama de Bloques de un transmisor-receptor IEEE802.11a

La señal es entonces enviada al codificador convolucional. Los parámetros para este codificador según el estándar es (2,1,7) mostrada en la figura 1.5. Las otras tasas descritas en la tabla II se alcanzan perforando la salida del codificador. La señal codificada es enviada a un interpolador. La idea del interpolador es dispersar un bloque de datos en frecuencia, así el bloque entero no experimenta desvanecimiento profundo en el canal. Esto previene ráfagas de errores en el receptor.

CIB-ESPOL

Los códigos luego del interpolador se agrupan para formar símbolos. Los símbolos luego son modulados usando una de los esquemas indicados en la tabla II. Por tanto, BPSK usa un bit a la vez, QPSK, dos bits a la vez, 16-QAM usa 4 bits a la vez y 64-QAM usa 6 bits a la vez por símbolo. Los símbolos modulados son entonces proyectados a las sub-portadoras de 64 puntos-IFFT, con lo cual se crea un símbolo OFDM. En el estándar IEEE802.11a el ancho de banda es típicamente restringido a 20MHz. Es importante que no haya picos espectrales fuera de este ancho de banda. Debido a esta razón, las 64 sub-portadoras no pueden ser usadas, en vez de esto, se usan solo 48 sub-portadoras para datos, 4 para tonos pilotos y las 12 restantes no son usadas. Fuera de estas 12, 11 están en los arcos y 1 está en el centro. Esta portadora central no es usada porque es DC y queda fuera de la operación IFFT. Por tanto, solo se usan 48 sub-portadoras de 64. Las 4 sub-portadoras piloto son usadas para determinan el desplazamiento en fase que sufre por la señal portadora durante una transmisión a través del canal. Esta información será usada en el receptor para coincidir exactamente con las frecuencias portadoras.

La salida de la IFFT se convierte en una secuencia serial y un intervalo de guarda o prefijo cíclico es añadido. Entonces el símbolo completo OFDM es la suma del prefijo cíclico más la duración del símbolo útil. Después de que el prefijo cíclico es añadido, todo el símbolo OFDM es transmitido a lo largo del canal. Esto constituye un paquete. Debería notarse que mientras la duración del prefijo cíclico es más larga que la respuesta impulso del canal, la interferencia intersímbolo es eliminada. Esta es la ventaja más grande de OFDM en que los multi-caminos son mínimos. Esta es la razón porque IEEE802.11a alcanza un alto desempeño de 54 Mbps, el cual es mucho más alto que lo que se alcanza con otros sistemas OFDM.

El proceso en el receptor es justamente al revés. Sin embargo, la primera cosa que el receptor necesita hacer es alcanzar una sincronización en tiempo. Realmente, OFDM es una técnica de modulación en el dominio de la frecuencia. De aquí que es esencial tener un estimado seguro del desplazamiento en frecuencia causado por la inestabilidad del oscilador en el receptor. Además, se necesita estimar el canal tan precisamente como sea posible. Estimando el canal, el receptor puede de-modular, de-interpolar y alimentar la señal al algoritmo de Viterbi

para la corrección de errores. La modulación y tasas de codificación usada permiten al estándar 802.11a soportar las tasas de datos enlistadas en la tabla II. Más específicamente, la tabla III lista las diferentes tasas de datos alcanzadas por diferentes combinaciones de modulación y tasas de códigos convolucionales [3].

Tabla III. Combinaciones de desempeño en el estándar 802.11 a

Combinaciones de desempeño en IEEE802.11a					
Tasa de Datos (Mbps)	**Modulación**	**Tasa de Codificación (R)**	**Bits codificados por sub-portadora**	**Bits codificados por símbolo OFDM**	**Bits de datos por símbolo OFDM**
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

1.3 Códigos Convolucionales

Codificación de canal es la operación más crítica en el procesamiento de la señal, cuando se quiere proteger la señal contra errores de transmisión. El objetivo de la codificación de canal es mejorar el desempeño de la tasa de error de bit –BER- de la potencia limitada y/o canales de banda limitada añadiendo redundancia estructurada a los datos transmitidos. En otras palabras, codificación de canal involucra añadir bits de protección a los datos usando algoritmos matemáticos más o menos complejos. Estos bits de protección son transmitidos junto con el dato. Esto hace incrementar la velocidad de datos requerida y el ancho de banda de la transmisión, sin embargo, este incremento en ancho de banda es el precio que se tiene que pagar para tener la protección contra errores de transmisión. Estos bits extra permiten al receptor detectar cualquier error de transmisión que haya ocurrido, y corregirlos [4][5].

Existen diferentes tipos de estrategias de codificación de canal: codificación de bloques, codificación convolucional y codificación concatenada. Entre estos, la combinación serial de codificación de bloques y códigos convolucionales junto con el adecuado interpolador en tiempo/frecuencia, se llama concatenación serial, y ha llegado a ser

la elección preferida en actuales estándares de comunicación inalámbrica. Recientemente, una nueva generación de codificación concatenada paralela, llamada turbo códigos, parece ser muy promisoria para algunas aplicaciones OFDM.

En Telecomunicaciones, los códigos convolucionales son talvez los códigos de canal más populares, como un solo bloque o como una parte de un esquema concatenado. Códigos convolucionales deben su popularidad a su buen desempeño y flexibilidad para alcanzar diferentes tasas de codificación. Estos códigos convolucionales son uno de los códigos de canal más ampliamente usados en los esquemas actuales; la mayoría de sistemas celulares lo usan. (GSM,IS-95). Los estándares IEEE 802.11a e HiperLAN/2 WLAN también y el estándar IEEE802.11b incluye un modo opcional que los usa [6][7].

En los códigos convolucionales cada *m*-bits de información a ser codificado se transforma en *n*-bit símbolos, donde *m/n* es la Tasa de Codificación y la transformación es una función de los últimos *k* símbolos, donde *K* es la Longitud Restrictiva del código.



CIB-ESPOL

Un código convolucional se genera pasando la secuencia de información a ser transmitida a través de un registro de desplazamiento lineal de estados finitos. En general, el registro de desplazamiento consiste de K (k-bit) estados y *n* funciones lineales algebraicas generadoras (Polinomios Generadores), como se muestra en la figura 1.3.

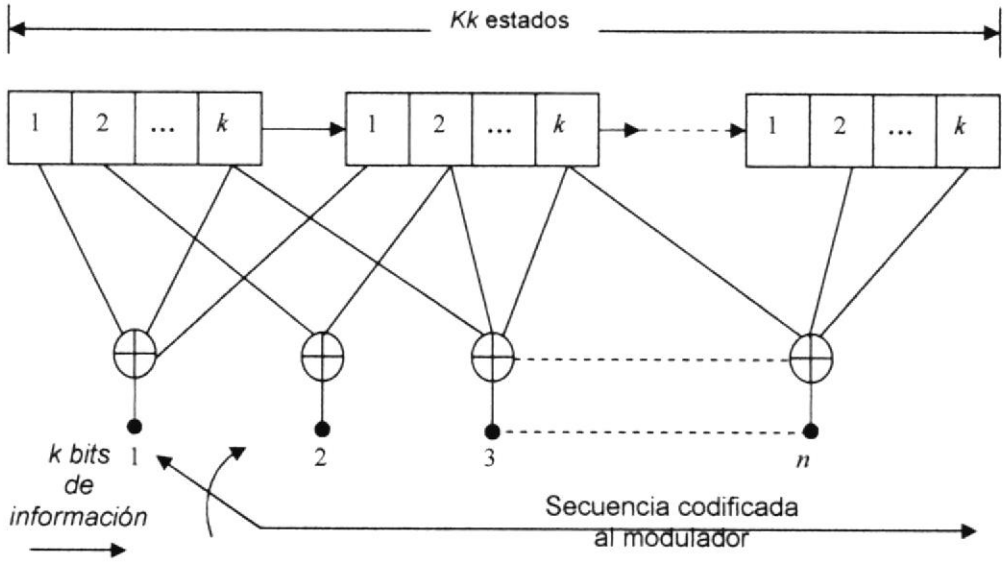


Figura 1.3 Codificador Convolucional

El dato de entrada al codificador, la cual se asume que es binario, es desplazado a lo largo del registro de desplazamiento de *k* bits a la vez. El número de bits de salida por cada *k*-bit de secuencia de entrada es *n* bits. Consecuentemente, la tasa de codificación está definido como $R_c = k/n$ $(n>k)$.

1.3.1 Generación

En códigos convolucionales, en vez de operar los símbolos como un bloque, los códigos operan continuamente sobre flujos de símbolos, esto es, el codificador tiene memoria. Esto significa que la n-ésima generada por el código convolucionales es una función no solo de la entrada K sino también de la previa entrada K-1. Esto se logra alimentando la secuencia de entrada k continuamente a través de un registro de desplazamiento k de longitud K-1.

Un método para describir un código convolutivo está dado por su matriz generadora que consiste de un conjunto de vectores de longitud fija llamado palabras de código. La longitud de estas palabras es el número de elementos en el vector y es denotado por *n*. Los elementos de una palabra de código se seleccionan de un alfabeto de *q* símbolos o elementos. Como una alternativa para especificar esta matriz, se usa una representación equivalente funcionalmente en la cual especificamos un conjunto de *n* vectores, un vector para cada sumador de módulo-2 (pueden ser implementados como compuertas EXOR). Cada vector tiene *Kk* dimensiones y

contiene las conexiones del codificador al sumador de módulo-2. Un 1 en la i-ésima posición del vector indica que el correspondiente estado en el registro de desplazamiento está conectado al sumador de módulo-2, y un 0 en una posición dada indica que no existe una conexión entre ese estado y el sumador.

Una convención breve típicamente empleada en la descripción de códigos convolucionales es (n, k, K). La figura 1.4 muestra un ejemplo de un código convolucional (3,1,3) y la figura 1.5 muestra un codificador de mínima distancia (2,1,7) usado para el estándar IEEE 802.11a, cuyos polinomios generadores son 133 y 171 en notación octal.

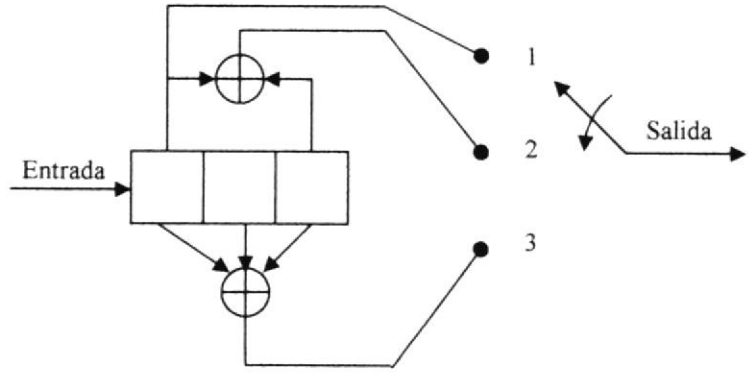


Figura 1.4 Codificador Convolucional (3,1,3) de 4 estados

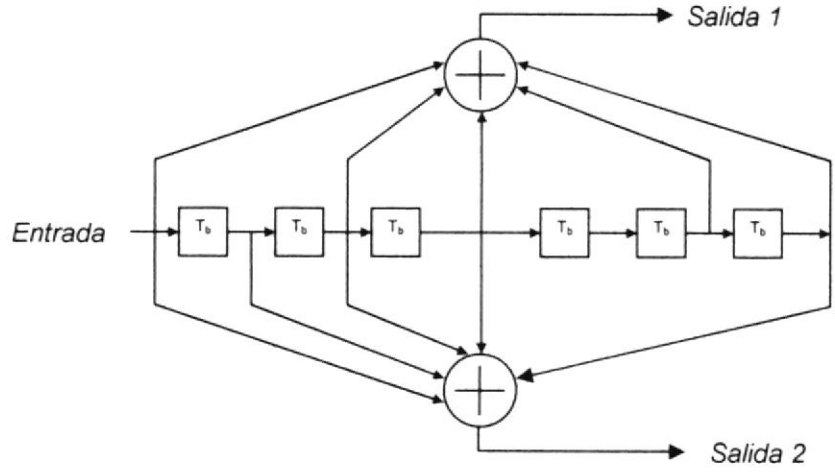


Figura 1.5 Codificador Convolucional (2,1,7) de 64 estados para el estándar IEEE 802.11a.

1.3.2 Respuesta Impulso y Función de Transferencia

Un codificador convolutivo es llamado así porque realiza una convolución de la trama de entrada con la respuesta impulso del codificador.

$$Y_i^j = \sum_{k=0}^{\infty} h_k^j x_{i-k}$$

Donde x es una secuencia de entrada, y^j es una secuencia de la salida j y h^j es su respuesta impulso.

CIB-ESPOL

Un codificador convolucional es un sistema lineal discreto invariante en el tiempo. Cada salida de un codificador pude ser descrito por su propia función de transferencia, la cual esta muy relacionada al polinomio generador. La respuesta impulso está conectada con la función de transferencia a través de la transformada Z.

Las funciones de transferencia para el primer codificador de la figura 1.5 (no recursivo) son:

$$H_1(z) = 1,$$
$$H_2(z) = 1 + z^{-2},$$
$$H_3(z) = 1 + z^{-1} + z^{-2},$$

Las funciones de transferencia para el codificador recursivo son:

$$H_1(z) = \frac{1 + z^{-1} + z^{-3}}{1 + z^{-2} + z^{-3}},$$

$$H_2(z) = 1$$

1.3.3 Representación

Hay tres métodos alternativos que son usados a menudo para describir un código convolucional, son: diagrama de árbol, diagramas de estado, diagrama de Trellis. [2]

Dado que el diagrama de árbol es una representación que casi no se usa actualmente no se describe en este trabajo, pero si el diagrama de estados que es la base para el diagrama de Trellis.

Diagrama de estados

El diagrama de estados es un simple gráfico de los posibles estados del codificador y las posibles transiciones de un estado a otro. Por ejemplo tenemos el diagrama de estados del ejemplo de la figura 1.5 en la figura 1.6, donde $\alpha \rightarrow \beta$ denota la transición del estado α a β cuando el bit de entrada es un 1. Los tres bits cercanos a cada línea de diagrama de estados representa la secuencia de salida. La línea punteada indica que la entrada es un 1 y la línea sólida que es un 0.

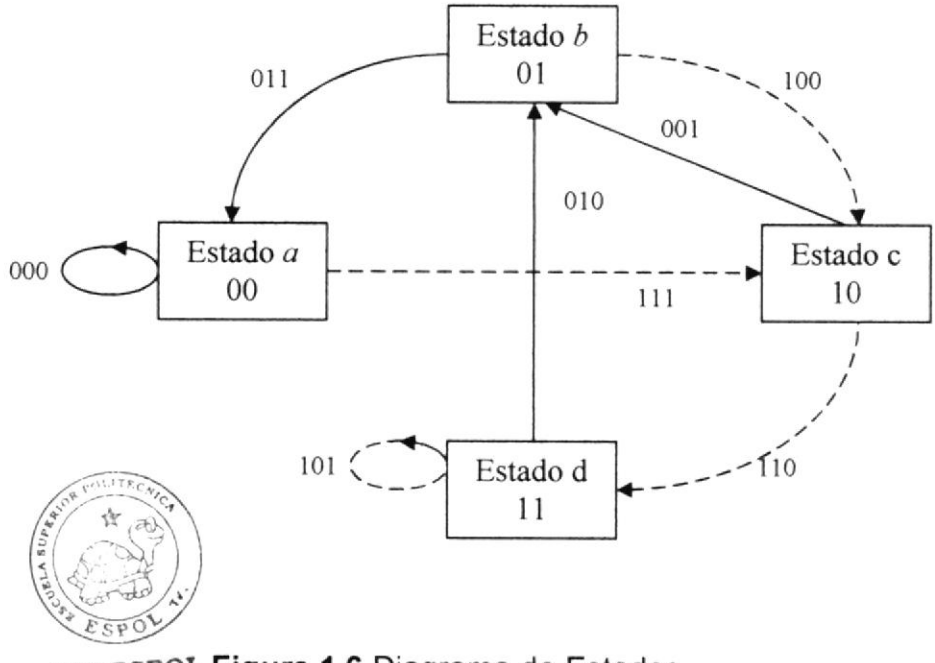
Diagrama de Trellis

Los códigos convolucionales pueden ser descritos por medio de un diagrama de Trellis el cual muestra la evolución en tiempo de los estados del codificador durante el proceso de

codificación. Este diagrama de Trellis sirve para decodificar la secuencia de información transmitida.

0 1 0 1 0 1 0 1

a → a, a → c, b → a, b → c, c → b, c → d, d →b, d →d



CIB-ESPOL **Figura 1.6** Diagrama de Estados

La figura 1.7 representa el diagrama de Trellis de la figura 1.5 y 1.6. Se observa que el diagrama usa la convención que una línea sólida denota la salida generada por el bit de entrada 0 y las líneas punteadas la salidas generadas por un bit de entrada 1. En este ejemplo observamos que después del transiente inicial, el Trellis contiene cuatro nodos en cada estado,

correspondientes a los cuatros estados del registro de desplazamiento *a, b, c, d.*

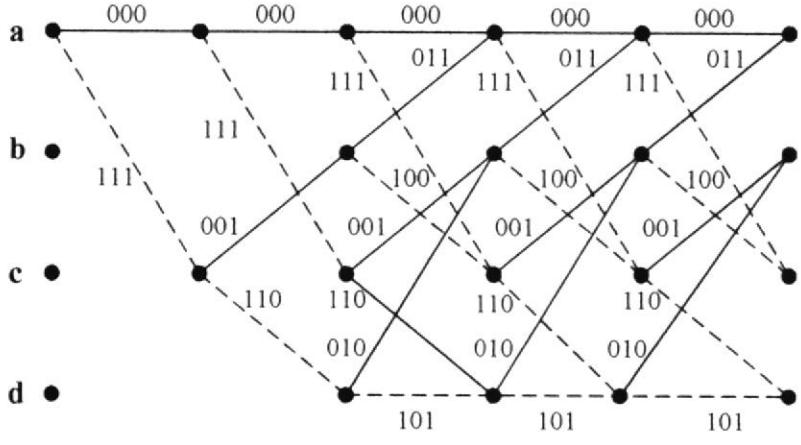


Figura 1.7 Diagrama de Trellis

Después del segundo estado, cada nodo en el Trellis tiene dos caminos entrantes y dos salientes. De los dos caminos entrantes, uno corresponde a una bit entrada 0 y el otro camino si la entrada es 1.

1.3.4 Distancia Mínima

Supongamos C_i y C_j son dos palabras de código cualquiera en un código de bloque (n , k). Una medida de la diferencia entre las dos palabra de código es el número de elementos correspondientes o posiciones en la cual difieren. Esta medida

es llamada la *distancia de Hamming* entre las dos palabras de código y se denota d_{ij}. Claramente d_{ij} para $i \neq j$ satisface la condición $0 \leq d_{ij} \leq n$. El valor más corto del conjunto $\{dij\}$ para $M = 2^k$ $(k<n)$ palabras de código se llama *distancia mínima* del código y se denota como d_{min}.

Dado que la distancia de Hamming es una medida de la separación entre dos palabras códigos, está íntimamente relacionado con el coeficiente de correlación cruzada entre las correspondientes par de ondas generadas y las palabras de código, es decir, cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores.

1.4 Decodificación de Máxima Verosimilitud: Algoritmo de Viterbi

Para una cierta secuencia de salida de un código convolucional $U^{(m)}$ existe una secuencia recibida, denominada Z, que resulta de la aparición de errores por efecto del ruido sobre la secuencia original U.

El decodificador óptimo es tal que compara las probabilidades condicionales $P(Z/U^{(m)})$ de que la secuencia recibida Z corresponda a

una secuencia conocida $U^{(m)}$ y toma la decisión de elegir la secuencia que tiene la mayor probabilidad de elegir la secuencia que tiene la mayor probabilidad de coincidir con Z:

$$P(Z/U^{(m')}) = max_{U(m)} P(Z/U^{(m)}) \quad (x)$$

Este es el criterio de máxima verosimilitud, el que está de acuerdo con el hecho intuitivo de asignar la secuencia conocida en el decodificador que más se parece a la recibida.

La aplicación de este criterio a los códigos convolucionales se enfrenta con el hecho de que un gran número de secuencias posibles debe ser considerado. Para una secuencia de L bits se tienen 2L posibilidades. El decodificador de máxima verosimilitud elige una secuencia $U^{(m')}$ del conjunto posible, que tenga la mayor probabilidad de parecerse al vector recibido.

Si el canal no tiene memoria, el ruido es aditivo blanco y gaussiano [2], por lo tanto, cada símbolo está afectado de una forma independiente de los otros. Para un código convolucional de velocidad 1/n la función de probabilidad que mide la verosimilitud respecto del vector recibido es:

$$P(Z/U^{(m)}) = \prod_{i=1}^{\infty} P(Z_i/U_i^{(m)}) = \prod_{i=1}^{\infty} \cdot \prod_{j=1}^{n} P(z_{ij}/u_{ij}^{(m)}) \quad (x)$$

Donde Z_i es la *i-ésima* rama de la secuencia recibida Z, $U_i^{(m)}$ es la *i-ésima* rama de la secuencia de una palabra de código $U^{(m)}$, z_{ij} es el *j-ésimo* símbolo de código de Z_i y u_{ij} es el *j-ésimo* símbolo de código de $U_i^{(m)}$, donde cada rama está constituida por n símbolos de código.

El proceso de decodificación consiste en la elección de una secuencia que maximice la función verosimilitud. El algoritmo de decodificación de Viterbi realiza la detección por máxima verosimilitud. Para eso utiliza las propiedades del Trellis de un código convolucional. El algoritmo intenta reducir la complejidad del cálculo evitando tener en cuenta la totalidad de las secuencias posibles. El procedimiento consiste en calcular la distancia entre la señal recibida en el instante t_i y los caminos o ramas entrantes del Trellis en ese instante en el estado analizado. En la medida que este criterio se va aplicando se evalúa la secuencia que tiene la menor distancia respecto de la recibida, de forma que la secuencia de máxima verosimilitud finalmente aparece. El camino o secuencia de máxima verosimilitud, que es al mismo tiempo aquel que presenta la menor distancia

respecto del recibido, se denomina camino o secuencia sobreviviente, la cual es almacenada. De esta forma, el algoritmo de Viterbi trata de encontrar el camino con mínimo error. Decir entonces que la secuencia es la de máxima verosimilitud, es también decir que se trata de aquella secuencia que tiene la menor distancia respecto de la recibida.

1.4.1 Algoritmo de Viterbi Decodificación Dura y Suave


CIB-ESPOL

La decodificación de decisión dura y suave se refiere al tipo de cuantización usado sobre los bits recibidos. La decodificación de Decisión Dura usa 1 bit de cuantización para los valores del canal de recepción. La decodificación de Decisión Suave en cambio usa múltiples bits de cuantización para los valores del canal de recepción. Para una decodificación de decisión suave ideal (cuantización infinita), los valores del canal de recepción son directamente usados por el decodificador, tal como lo muestra la figura 1.8.

El algoritmo de Viterbi de decisión dura utiliza el modelo de canal simétrico binario es decir dos valores posibles en el receptor. Dado que el algoritmo de Viterbi usa la función de

máxima verosimilitud, existen solo dos caminos en el diagrama de Trellis que pueden ser escogidos como el de menor costo o distancia de Hamming entre la palabra recibida y la palabra esperada.

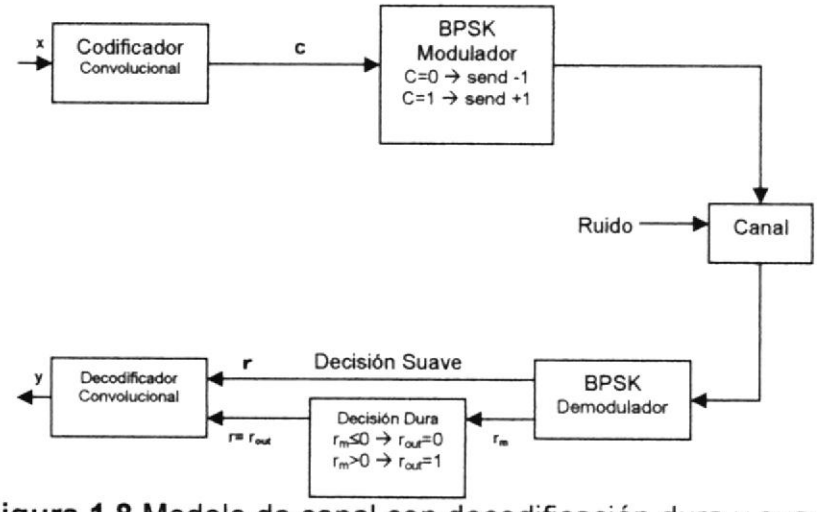


Figura 1.8 Modelo de canal con decodificación dura y suave

Así mismo utiliza el Trellis para calcular las métricas de nodo e ir asignando y escogiendo las métricas parciales de menor o mayor valor dependiendo si los valores binarios son escogidos convencionalmente. Los caminos sobrevivientes son almacenados y el resto es descartado en el diagrama de Trellis. El algoritmo de Viterbi de decisión dura puede ser implementado como sigue:

$S_{k,t}$ es el estado en el diagrama de Trellis que corresponde al estado S_k en el instante t . A cada estado en el Trellis se le asigna un valor denotado $V(S_{k,t})$

1. (a) Se inicia en t = 0.

   (b) Se inicializa $V(S_{0,0}) = 0$ y el resto $V(S_{k,t}) = +\infty$.

2. (a) $t = t+1$.

   (b) Calculamos las métricas para todos los caminos que van al estado Sk en el tiempo t. Primero encontramos la t-ésima métrica de rama calculando la distancia de Hammning.

   Segundo, calculamos la t-ésima métrica de nodo parcial sumando el valor previo del nodo y la nueva métrica de rama V(Sk,t-1)M(r t | y t).

3. (a) Asignamos $V(S_{k,t})$ a la mejor métrica de camino que llega al estado S_k en el tiempo t. convencionalmente esta mejor métrica se refiere al menor valor.

   (b) Si existe una igualdad cualquier métrica puede ser escogida.

4. Almacenamos la mejor métrica de camino y su bit sobreviviente asociado con el estado del camino.

5. Si t es menor que la profundidad de decodificación más número de estados menos 1.

Regresamos al paso 2.

El resultado del algoritmo de Viterbi es un único camino de Trellis que corresponde a la palabra de máxima verosimilitud.

En la decodificación de decisión Suave el receptor no asigna cero o uno a cada bit recibido sino más bien cuantización múltiple o infinita. Idealmente, la secuencia recibida es cuantizada infinitamente y es usada directamente en el decodificador de Viterbi de decisión suave. El algoritmo de Viterbi de decisión Suave es similar al de decisión Dura excepto que en el literal 2(b) se usa distancia Cuadrada Euclidiana en las métricas de rama en lugar de distancia de Hamming.

Generalmente con decodificación de decisión suave se obtiene una ganancia de aproximadamente 2 dB de codificación sobre la decodificación de decisión dura en canales Gaussianos [2].

1.4.2 Probabilidad de error

Dada una única observación y en el receptor, existe una secuencia x enviada, llamada x_{max}, dentro del conjunto de

secuencias enviadas posibles de x que hace máxima la función de verosimilitud. Es decir, la secuencia x_{max} es la que tiene más probabilidad de haber generado la observación y.

En el diagrama de Trellis del codificador, se observa que en cada nodo confluye más de una rama. El algoritmo de Viterbi permitirá escoger en cada tiempo t_i, cual es la rama que acumula el camino más verosímil. Asumiendo que el canal de transmisión no tiene memoria y que el ruido afecta a cada uno de los símbolos recibidos de forma independiente, se puede aplicar la función logaritmo a la función verosimilitud,

$$f(y_j \mid x_j) = \prod_{k=1}^{n} f(y_{jk} \mid x_{jk})$$

así, se obtiene la métrica de rama:

$$\mu_j = \mu(x_j, y_j) = \sum_{k=1}^{n} \log f(y_{jk} \mid x_{jk}) = \log \prod_{k=1}^{n} f(y_{jk} \mid x_{jk})$$

El canal gaussiano dispone de un alfabeto discreto de entrada y un alfabeto continuo de salida, considerado de $-\infty$ a $+\infty$. Además, según el modelo simplificado de canal que utilizamos, se añade ruido gaussiano aleatorio, llamado AWGN.

Así, considerando una señal binaria a la entrada del canal, la salida puede tomar infinitos valores. A efectos prácticos se

puede considerar la señal recibida en términos de la variable aleatoria y. En la Figura 1.8 se representa la probabilidad de la señal recibida condicionado al hecho de haber enviado un '0' *f(y |'0')* y la probabilidad al hecho de haber enviado un '1' *f(y |'1')* cuando se introduce ruido AWGN al canal.

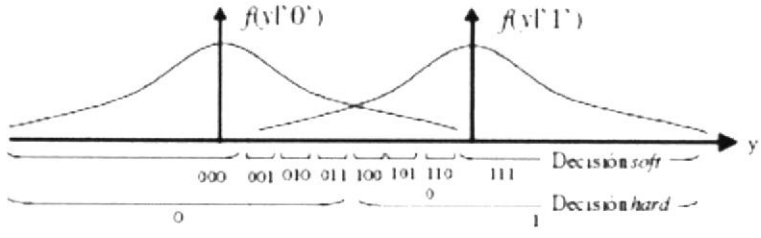


Figura 1.9 Decisiones del decodificador dura y suave.

El demodulador aplicado a un decodificador suave cuantifica una graduación entre los dos niveles de señal del emisor '0' y '1'. Así, la decisión de si un símbolo es '1' o '0' se suaviza (decisión suave). En la figura 1.9, el decodificador Viterbi interpretará el símbolo '000' como un '0' con máxima probabilidad y el símbolo '111' como un '1' con máxima probabilidad. En medio se encuentran el resto de símbolos. La métrica se maximiza con el producto escalar del código recibido respecto al asociado a cada rama [2].

1.4.3. Arquitectura del decodificador

Un decodificador Viterbi está compuesto principalmente por tres bloques funcionales: unidad de Cálculo de Métrica de Ramas -BMU-, Unidad de Cálculo de Métrica de Nodos que realiza las funciones de Suma, Compara y Selecciona -ACS-, y unidad que administra los bits sobrevivientes -SMU-, como se muestra en la figura 1.10 [8][9][10].

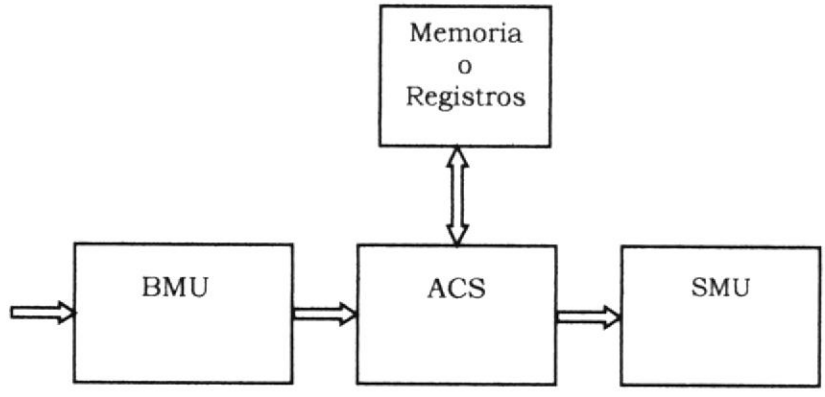


Figura 1.10 Diagrama de bloques de unidades principales de un decodificador Viterbi

1.4.3.1 BMU

La unidad de cálculo de métrica –BMU- por sus siglas en inglés, se usa para calcular las métricas de rama. Una métrica de rama es el valor de diferencia entre los símbolos codificados recibidos y las correspondientes

etiquetas de rama del encoder del Trellis. Estas etiquetas de rama son los símbolos que serían esperados y están dados por la salida del encoder como resultado de las transiciones de estado.

En decodificación de decisión dura, el método de cálculo se denomina Distancia de Hamming. La distancia de Hamming d(X, Y) entre dos palabras X y Y. Es igual a número de elementos diferentes. El máximo valor de la distancia de Hamming corresponde al número de bits en el símbolo, el mínimo valor es cero cuando el símbolo recibido y el símbolo del encoder son iguales.

En decodificación de decisión suave existe otro algoritmo llamado Distancia Euclidiana, donde el símbolo de entrada se representa X y el símbolo del encoder por Y, y la Distancia Euclidiana es calculada por la siguiente fórmula.

$$BM = \sum_{i=0}^{n-1} (rec_i - esp_i)^2$$

La arquitectura básica del bloque BMU, varía entonces dependiendo del tipo de decisión que se desee.



CIB-ESPOL

1.4.3.2 ACS

Esta unidad calcula los nuevos valores de métricas de nodo y los valores de decisión.

Debido a que cada estado puede ser alcanzado desde dos estados diferentes de etapas anteriores, como lo muestra la figura 1.11 hay dos posibles métricas de nodo llegando a un estado actual, entonces esta unidad suma la métrica de rama con la métrica de nodo, selecciona la menor y toma una decisión; de allí su nombre de suma, compara y selecciona o por sus siglas en inglés. –ACS. [9].

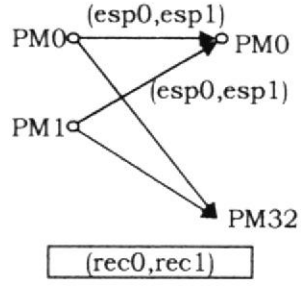


Figura 1.11 Transición de estados en una sección del diagrama de Trellis

Esta unidad guarda el resultado de la suma seleccionada como métrica de nodo para el estado actual. El valor de

decisión puede ser representado por un bit, si la métrica de nodo seleccionada viene del estado de debajo de los dos posibles estados del diagrama de Trellis, el bit de decisión es uno. Si la métrica seleccionada viene del estado de arriba, el bit de decisión es cero.

El diagrama de bloques de la arquitectura de la unidad ACS se presenta en la figura 1.12 [9].

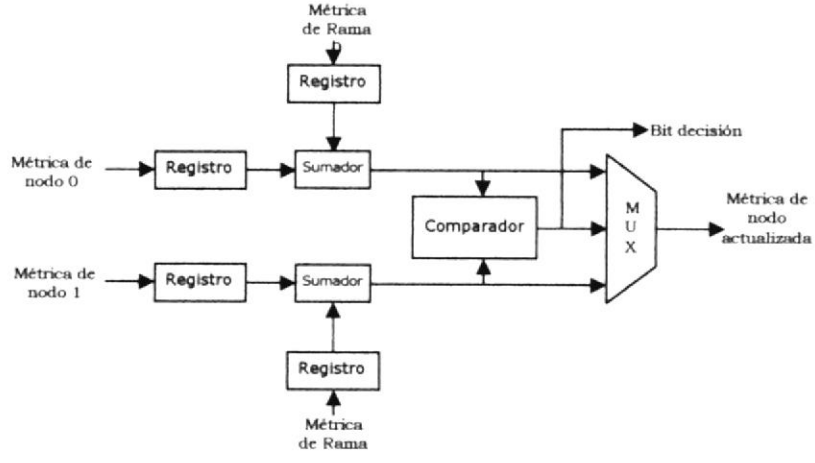


Figura 1.12 Diagrama de bloques de Arquitectura de unidad ACS

Donde los registros ayudan a evitar lazos combinatoriales en la realimentación, a la vez que almacenan el valor de las métricas de nodo utilizadas para realizar las actualizaciones de las mismas por cada flanco de reloj.

Se tiene dos bloques sumadores, los cuales operan con las métricas de nodo obtenidas en un estado anterior y las métricas de rama provenientes del bloque BMU. El comparador, selecciona cual de las sumas mencionadas es la de menor valor.

Esta comparación da como resultado el nuevo bit decisión que indica si el camino de menor métrica se obtiene por la rama superior o inferior de las dos que llegan a un nuevo nodo como se observa en la figura. Este bit decisión a la vez actúa como selector del multiplexor, con el que se elige la nueva métrica de nodo. [11]

1.4.3.3. SMU

Esta unidad maneja la secuencia de bits sobrevivientes generados por el ACS para encontrar los datos decodificados. Generalmente para realizar esta operación existen dos técnicas llamadas: Intercambio de Registros y Rastreo hacia atrás –TB. El esquema de Rastreo hacia atrás presenta un menor costo en

hardware que lo que demanda el método de Intercambio de registros.

El rastreo hacia atrás se realiza "moviéndose" en el diagrama de Trellis desde el último estado hasta el primero. [12]

El método de rastreo hacia atrás almacena información de los caminos en forma de un arreglo de punteros recursivos. Es ventajoso pensar en la memoria del rastreo hacia atrás como si estuviera organizada en una estructura bidimensional, con filas y columnas. El número de columnas es igual al número de estados $N=2^n$. Cada fila almacena el resultado de N comparaciones correspondientes a una etapa en el diagrama de Trellis. Debido a que cada flujo de símbolos es, en general, semi-infinito, las localidades de almacenamiento son reutilizadas periódicamente.

La arquitectura de la unidad de rastreo hacia atrás está formado básicamente por una memoria y una etapa de selección, la memoria almacena los vectores decisión provenientes del bloque ACS, mientras que la etapa de

CIB-ESPOL

selección elije cual de esos 64 bits que forma un vector decisión, será considerado como un bit de salida.

La figura 1.13 presenta el esquema del bloque TB.

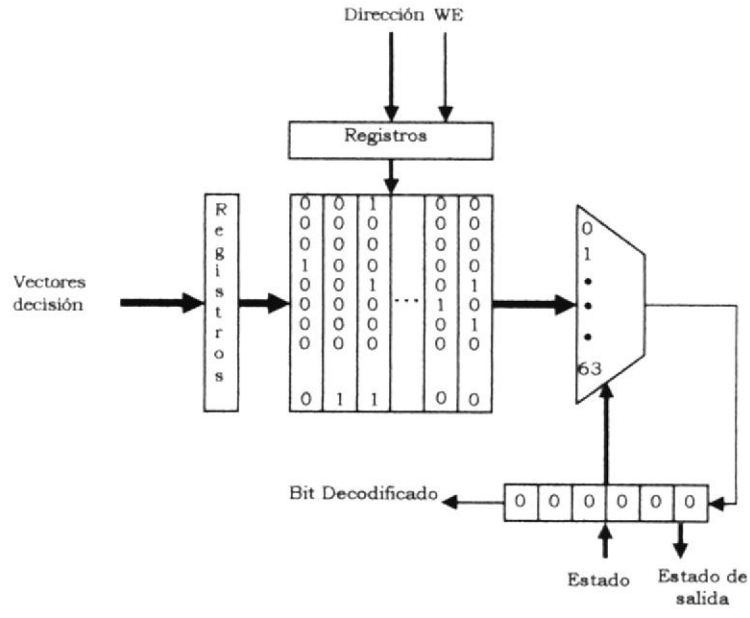


Figura 1.13 Arquitectura de unidad de rastreo hacia atrás.

La etapa de selección está formada por un registro de desplazamiento y un multiplexor, en el registro de desplazamiento se carga inicialmente el estado que presente una menor métrica de nodo, dicho estado funciona como selector para el multiplexor, el cual va a elegir el bit correspondiente a ese estado. Ese nuevo bit elegido espera en la entrada serial del registro de

desplazamiento y con el siguiente flanco de subida de reloj, ingresa como bit menos significativo del registro, es decir se tiene un nuevo estado. Luego se lee el siguiente vector decisión y se repite la operación hasta haber leído por completo la memoria. [13]

CAPÍTULO 2

ARQUITECTURA DEL PROCESADOR MIPS

Para dar órdenes al hardware del computador hay que hablar en su lenguaje. Las palabras del lenguaje de máquina se denominan instrucciones, y su vocabulario se denomina Conjunto de Instrucciones.

2.1 Arquitectura del Conjunto de instrucciones de MIPS

El diseño del hardware del computador debe ser capaz de soportar operaciones aritméticas, lógicas y de control; las cuales en MIPS constan de tres operandos, dos fuentes y uno destino, por ejemplo la instrucción ADD a, b, c indica que se realiza la suma entre b y c y el resultado se almacena en a.

Estos operandos se obtienen de localidades llamadas registros. En una arquitectura MIPS se tiene un total de 32 registros cada uno de los cuales tienen una longitud de 32 bits. Para representarlos se utiliza la notación $0, $1,..., $31 [14].

2.1.1 Tipos de Instrucciones

Instrucciones aritméticas

Instrucciones para las operaciones aritméticas que sólo se realizan en los registros.

Instrucciones de transferencia de datos

Son las instrucciones que transfieren datos entre la memoria y los registros. Para acceder a una palabra en memoria, la instrucción debe suministrar su dirección. La memoria es realmente un gran arreglo unidimensional, donde la dirección actúa como índice de ese arreglo.

La instrucción de transferencia de datos que desplaza un dato desde memoria hasta un registro se denomina cargar (load). La instrucción complementaria a la de cargar se denomina almacenar (store); ésta transfiere datos de un registro a memoria.

Instrucciones de control

Lo que distingue a un computador de un simple calculador es su posibilidad de tomar decisiones. Basándose en los datos de entrada y en los valores creados durante el cálculo, se ejecutan diferentes instrucciones. Estas son las instrucciones de saltos condicionales.

Instrucciones que soportan procedimientos en hardware

Un procedimiento o subrutina es una forma en la que los programadores estructuran los programas. Estas instrucciones proporcionan un método para ir a un procedimiento y después volver desde ese procedimiento a la instrucción inmediatamente siguiente al punto de llamada. Las instrucciones que soportan los procedimientos son las instrucciones de salto.

2.1.2 Formatos de Instrucción

Las instrucciones de MIPS tienen un tamaño constante de 32 bits, los cuales se encuentran divididos en *campos* que se ordenan de diferente manera dependiendo del formato de la

instrucción. A continuación se presentan los formatos de las instrucciones y la tabla IV describe cada uno de estos campos [15].

Formato Tipo R

Opcode	rs	rt	rd	Shift amount	Function
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Formato Tipo I

Opcode	rs	rt	inmediato
6 bits	5 bits	5 bits	16 bits

Formato Tipo J

Opcode	dirección
6 bits	26 bits



CIB-ESPOL

2.2 Diseño de un procesador MIPS de un solo ciclo

La implementación del procesador incluye la arquitectura del conjunto de instrucciones de MIPS y diversas estrategias de implementación que determinarán factores importantes para medir su desempeño como el ciclo de reloj y el número de ciclos de reloj por instrucción [14].

Tabla IV. Campos de las instrucciones

Campo	Descripción
Opcode	Indica la operación que realiza la instrucción
Rs	primer registro operando
Rt	Para instrucciones Tipo R es el segundo registro operando. Para una instrucción con operandos inmediatos y para una carga es el registro destino de la operación. Para un almacenamiento es el dato a ser guardado en memoria.
Rd	Para instrucciones tipo R es el registro del operando destino; obtiene el resultado de la operación
Shift Amount	cantidad de desplazamientos a realizarse
Function	selecciona la variante de la operación del campo op
Inmediato	Para un salto condicional representa el desplazamiento del salto. Para una instrucción inmediata representa el segundo operando. Para una instrucción de acceso a memoria representa sumado al valor de Rs la dirección a la cual se va a acceder.
Dirección	Para una instrucción Tipo J, es la dirección a la cual hará el salto.

Para implementar las instrucciones aritmético-lógicas, de referencia a memoria, y de salto de MIPS, se necesitan dos pasos básicos independientemente del tipo de instrucción:

1. Enviar el contador de programa (PC) a una memoria que contenga el código para buscar la instrucción.
2. Leer uno o dos registros utilizando los campos de instrucción para seleccionar cuáles registros leer.

2.2.1 Unidades funcionales

Las unidades funcionales en la implementación de MIPS constan de dos tipos diferentes de elementos lógicos: elementos secuenciales y elementos combinatoriales.

Los elementos que operan sobre datos son todos combinatoriales. La ALU, por ejemplo, es un elemento combinatorial.

Un elemento secuencial tiene algún tipo de memoria interna además tiene por lo menos dos entradas y una salida. Las entradas requeridas son el valor del dato, que se va a escribir, y

el reloj que determina cuando se almacena el dato de entrada. La salida proporciona el valor que se escribió en el reloj anterior [16].

2.2.2 Diseño de una ALU

La Unidad aritmético Lógica –ALU- es el músculo del computador, el dispositivo que realiza las operaciones aritméticas y lógicas.

Como la palabra de MIPS es de 32 bits, se necesita una ALU de 32 bits. Se asume que a partir de una ALU de 1 bit se puede crear la ALU deseada.

Una ALU se construye a partir de los bloques de construcción and, or, inversor y multiplexor. Si se quiere construir una ALU de tres operaciones: and, or y add. Para la función de suma se construye un bloque con las entradas a, b y CarryIn y las salidas Sum y CarryOut. Combinando este bloque con los componentes de las operaciones lógicas AND y OR tenemos el circuito de la figura 2.1.

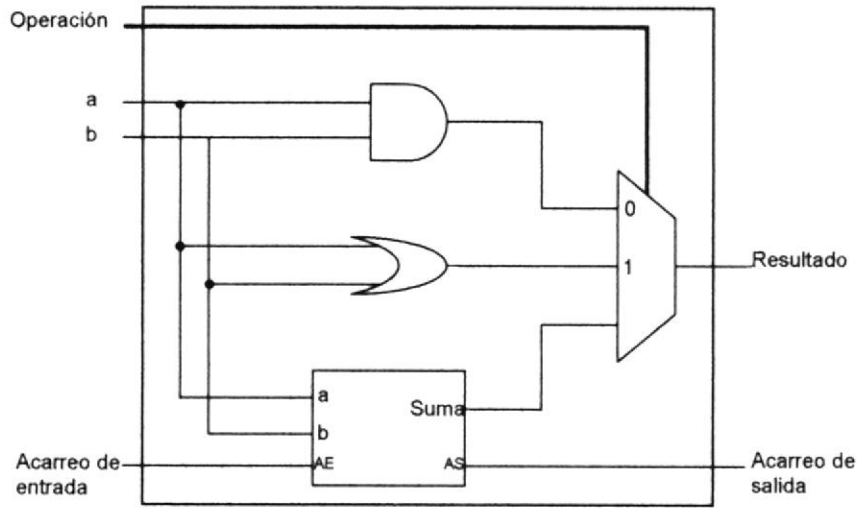


Figura 2.1 Diseño de una ALU de un bit con solo tres operaciones: add, and y or

Para añadir más operaciones se expande el multiplexor controlado por la línea Operación.

2.2.3 Camino de datos

Se puede empezar un diseño de camino de datos examinando los componentes principales necesarios para ejecutar cada tipo de instrucción MIPS y a partir de estos elementos construir las secciones del camino de datos.

El primer elemento que se necesita es un dispositivo para almacenar las instrucciones de un programa. Una unidad de

memoria, que es un elemento de estado y se utiliza para que contenga y suministre instrucciones dada una dirección.

La dirección de la instrucción también debe estar en un elemento de estado, que se llamará contador de programa (PC). Finalmente, se necesitará un sumador para incrementar el PC a la dirección de la siguiente instrucción.

Todas las instrucciones tipo R leen registros, realizan una operación en la ALU sobre los contenidos del registro y escriben el resultado en un registro.

Los 32 registros del procesador se almacenan en una estructura denominada *archivo o banco de registros*. Un banco de registros es una colección de registros en la que cualquiera de ellos puede ser leído o escrito si se especifica el número de registro en el archivo.

Debido a que las instrucciones de formato R tienen tres operandos de registro (Rs, Rt, Rd) para cada instrucción se requiere leer dos palabras de datos del archivo de registros y escribir una palabra de datos en el archivo de registros.

Las instrucciones de carga y almacenamiento calculan una dirección de memoria sumando el registro base (Rs) al campo de desplazamiento de 16 bits con signo, contenido en la instrucción. Si la instrucción es un almacenamiento, el valor que se va a almacenar en memoria también debe ser leído en el archivo de registros. Si la instrucción es una carga, el valor leído en memoria debe ser escrito en el archivo de registros en el registro que se especifica.

Las instrucciones de salto condicional, por ejemplo la instrucción BEQ tiene tres operandos, dos registros que se comparan para la igualdad, y un desplazamiento de 16 bits utilizado para calcular la dirección destino del salto con respecto a la dirección de la instrucción de salto.

Se puede combinar los caminos de datos para cada tipo de instrucción y añadir el control para completar la implementación.

El camino de datos más sencillo podría ejecutar todas las instrucciones en un ciclo de reloj. Esto significa que ningún recurso del camino de datos puede ser utilizado más de una vez por instrucción y que cualquier elemento que se necesite más de

ESCUELA SUPERIOR POLITÉCNICA ESPOL
CIB-ESPOL

una vez debe estar duplicado. Por tanto, se necesita una memoria para las instrucciones separada de los datos.

Aunque algunas unidades funcionales necesitan ser duplicadas cuando se combinen los caminos individuales para cada tipo de instrucción, muchos elementos pueden ser compartidos por diferentes flujos de instrucciones.

Para compartir un elemento del camino de datos entre dos tipos de instrucciones diferentes, puede que se necesiten múltiples conexiones a la entrada de un elemento y una señal de control para seleccionar las entradas. Normalmente esto se hace con un multiplexor o selector de datos.

El camino de datos combinado incluye una memoria de instrucciones y una memoria separada para datos, se requiere también un sumador y una ALU, ya que el sumador se utiliza para incrementar el PC, mientras que la ALU se utiliza para ejecutar la instrucción en el mismo ciclo de reloj.

2.2.4 Unidad de Control

La unidad de control debe ser capaz de tomar las entradas y generar una señal de escritura para cada elemento de estado, el control del selector para cada multiplexor, y el control de la ALU. Estas líneas de control son:

Tabla V. Señales básicas proporcionadas por la Unidad de control

Señal	Función
MemRead	Habilita la lectura de memoria
MemWrite	Habilita la escritura en memoria
ALUScr	Selecciona el segundo operando de la ALU. Este puede provenir del registro Rt o del valor inmediato de la instrucción.
ALUOp	Indica al Control de la ALU el tipo de operación a realizar.
RegDst	Selecciona el registro destino. Rd si es una operación de registros o Rt si no lo es.
RegWrite	Habilita escritura en el banco de registros
PCScr	Selecciona el siguiente valor del Contador de Programa
MemtoReg	Selecciona el dato a escribirse en el banco de registros, este puede provenir del resultado de la ALU o de memoria.
Jump	Indica si la instrucción es de salto incondicional.

La unidad de control principal debe saber inicializar todas las líneas de control y para esto se basa en el campo de código de operación de la instrucción.

Para controlar directamente la ALU es necesaria una unidad de control adicional para indicar que tipo de operación va a realizar. Esto lo hace tomando en cuenta el campo de función de la instrucción además de la señal ALUOp que genera la unidad de control principal.

La figura 2.2 muestra un camino de datos sencillo con la unidad de control y el Control de la ALU [17].

Se debe tener presente que esta implementación es combinatorial, es decir no se trata de una serie de cuatro pasos distintos. El camino de datos opera realmente en un solo ciclo de reloj.

En este diseño de un único ciclo, el ciclo de reloj debe tener la misma duración para cada instrucción y está determinado por el camino más largo posible del conjunto de instrucciones a ejecutarse. Este camino es ciertamente una instrucción de carga,

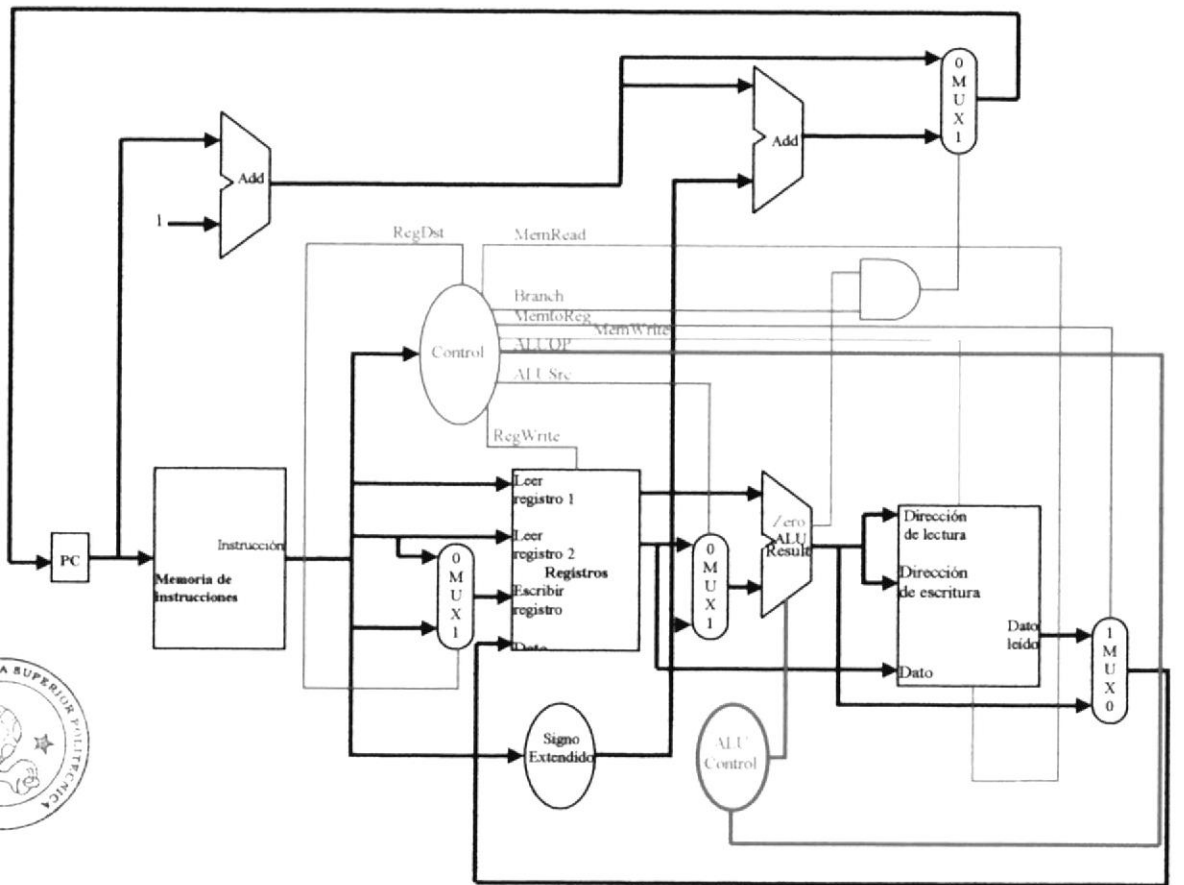


Figura 2.2. Camino de datos sencillo con unidad de control.

CIB-ESPOL

que utiliza cinco unidades funcionales en serie: la memoria de instrucciones, el archivo de registros, la ALU, la memoria de datos y el archivo de registros. Por esto, el rendimiento global de una implementación de un solo reloj probablemente no sea muy bueno, ya que algunos tipos de instrucciones pueden realizarse en un ciclo de reloj más corto.

Además con esta implementación monociclo, cada unidad funcional puede utilizarse solo una vez por reloj; por tanto, algunas unidades deben estar duplicadas, elevando el coste de la implementación.

Para evitar estas dificultades, se utilizan técnicas de implementación que tengan un ciclo de reloj más corto o que requieran múltiples ciclos de reloj para cada instrucción. Otra técnica de implementación denominada segmentación (pipelining) utiliza un camino de datos similar al de un solo ciclo de reloj y solapa la ejecución de múltiples instrucciones para lograr un aumento del rendimiento.

2.2.5 Excepciones e Interrupciones

El control es el aspecto más complicado del diseño del procesador: es la parte más difícil de diseñar. Por su parte, lo más arduo del control es implementar las excepciones e interrupciones, que son eventos diferentes a los saltos que cambian el flujo normal de ejecución de las instrucciones.

Una excepción es un evento inesperado del procesador, el desbordamiento aritmético, invocaciones al sistema operativo desde el programa del usuario o mal funcionamiento de hardware son un ejemplo de excepción.

Una interrupción es un evento que también provoca un cambio inesperado en el flujo de control pero proviene del exterior del procesador. Las interrupciones las utilizan los dispositivos de entrada y salida para comunicarse con el procesador [14].

2.3 Segmentación

Segmentación es una técnica de implementación en la cual múltiples instrucciones se solapan durante su ejecución. Actualmente, es la clave para hacer procesadores rápidos.

La segmentación no reduce el tiempo que se necesita para completar una instrucción individual; incrementa el número de instrucciones que se ejecutan simultáneamente. Por tanto, la segmentación mejora la productividad de las instrucciones en lugar del tiempo de ejecución de las instrucciones individuales [18].

El tiempo necesario para que una instrucción avance un paso a lo largo de la segmentación es idealmente un ciclo de reloj. La duración del ciclo de reloj está determinada por el tiempo necesario para la etapa más lenta de la segmentación, porque todas las etapas deben proceder a la misma velocidad [14].

Bajo condiciones ideales, la mejora de velocidad debida a la segmentación es igual al número de etapas; una segmentación de cinco etapas es cinco veces más rápida, sin embargo, habitualmente las etapas están equilibradas de manera imperfecta.

Si consideramos la duración de la operación para las unidades funcionales más importantes (su asume que los multiplexores, unidad de control, accesos al PC y unidad de extensión de signo no tienen retardo) de la implementación del camino de datos y control principal ya revisados; el tiempo necesario para cada una de las siete instrucciones se muestra en la tabla siguiente:

Tabla VI. Retardo para cada etapa para diferentes tipos de instrucciones

Tipo de instrucción	**Memoria de instrucciones**	**Lectura de registros**	**Operación ALU**	**Memoria de datos**	**Escritura de Registros**	**Tiempo Total**
LW	0 ns	5 ns	10 ns	10 ns	5 ns	40 ns
SW	10 ns	5 ns	10 ns	10 ns		35 ns
Formato R	10 ns	5 ns	10 ns	-	5 ns	30 ns
BEQ	10 ns	5 ns	10 ns	-	-	25 ns

Debido a que el diseño de un solo ciclo debe permitir la instrucción en el peor caso –la instrucción más lenta de la tabla anterior- el tiempo necesario para cada instrucción es de 40 ns.

Si se tiene presente los pasos del flujo de instrucciones revisados anteriormente, se ve que las instrucciones toman un máximo de cinco pasos, esto sugiere que una segmentación de cinco etapas sería un buen punto de partida.

Todas las etapas emplean un solo ciclo de reloj, así el ciclo de reloj debe ser suficientemente largo para acomodar la operación más lenta. Igual que en el diseño con un solo ciclo de reloj, debe tomarse el ciclo del peor caso, aun cuando algunas instrucciones puedan ejecutarse en 25ns, el ciclo de reloj en la ejecución segmentada debe ser el correspondiente al peor caso, de 10ns, aún cuando algunas etapas necesiten sólo 5 ns.

2.3.1 Camino de Datos segmentado

Partiendo del camino de datos monociclo, se puede hacer la división de una instrucción en cinco etapas, lo que significa una segmentación de cinco etapas, que a su vez significa que cinco instrucciones se estarán ejecutando durante un solo ciclo de reloj.

Así, se debe separar el camino de datos en cinco partes, el nombre que se le da a cada parte corresponde a una etapa de la ejecución de la instrucción:

1. IF: Búsqueda de la instrucción.
2. ID: Decodificación de la instrucción y búsqueda de registros.
3. EX: Ejecución y cálculo de la dirección efectiva.



CIB-ESPOL

4. MEM: Acceso a memoria.

5. WB: Post-escritura.

Las instrucciones y los datos se desplazan generalmente de izquierda a derecha a través de las cinco etapas cuando completan una ejecución. Sin embargo, hay dos excepciones a este flujo de instrucciones de izquierda a derecha:

La etapa de post-escritura, que vuelve a colocar el resultado en el archivo de registros en la mitad del camino de datos.
La selección del siguiente valor del PC, seleccionando entre el PC incrementado y la dirección de salto de la etapa MEM.

El flujo de datos de derecha a izquierda no afecta a la instrucción actual; solamente las instrucciones posteriores de la segmentación están influenciadas por estos dos desplazamientos de datos. Esto será discutido en secciones posteriores.

Una forma de mostrar lo que ocurre en la ejecución segmentada es pretender que cada instrucción tenga su propio camino de datos, y después colocar estos caminos de datos en una secuencia temporal para mostrar sus relaciones.

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL
ESPOL
CIB-ESPOL

La figura 2.3 muestra la ejecución de instrucciones de carga (lw) visualizando sus caminos de datos individuales en una línea de tiempo común. La figura parece sugerir que tres instrucciones necesitan tres caminos de datos, podemos usar la técnica de añadir registros para que mantengan los datos ya que partes del camino pueden ser compartidas durante la ejecución de la instrucción.

Por ejemplo, en el diseño monociclo la memoria de instrucciones es utilizada cada vez por solo una instrucción, mientras dura la ejecución de esa instrucción. Con la segmentación, como se muestra en la figura 2.3, la memoria de instrucciones se utiliza solamente durante una de las cinco etapas de la instrucción, permitiendo que sea compartida por otras instrucciones durante las cuatro etapas restantes. Para conservar el valor de una instrucción individual en sus cuatro etapas restantes, el valor leído en memoria de instrucciones se debe almacenar en un registro.

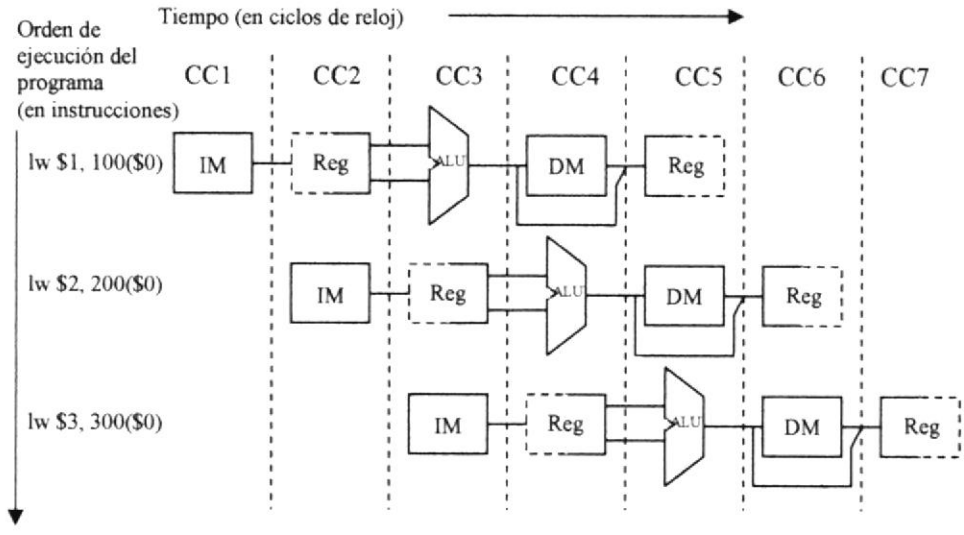


Figura 2.3. Instrucciones en ejecución en el camino de datos de un solo ciclo suponiendo ejecución segmentada.

Argumentos similares se aplican a cada etapa de la segmentación, así se debe colocar registros entre cada etapa de la segmentación.

La figura 2.4 muestra el camino de datos segmentado.

2.3.2 Control de un MIPS Segmentado

Las líneas de control utilizadas en el camino de datos segmentado son las mismas líneas usadas para un control de un solo ciclo vistos en la implementación de la unidad de control, con la diferencia de que ahora esas líneas de control viajarán a través

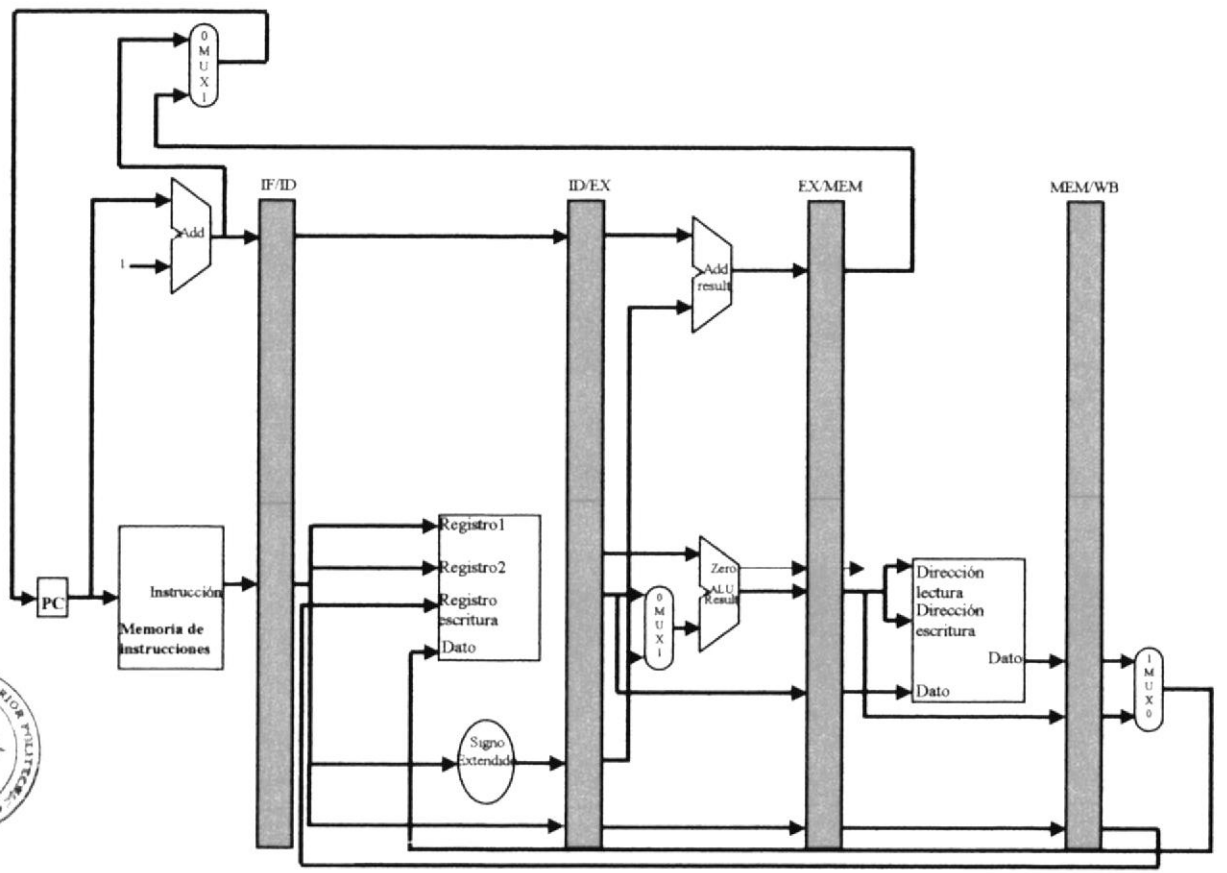


Figura 2.4 Camino de datos segmentado.

CIB-ESPOL

de los registros de segmentación dependiendo del componente activo al cual cada una de ellas se encuentre asociada [18].

Podemos dividir las líneas de segmentación en cinco grupos de acuerdo con las etapas de segmentación:

1. *Búsqueda de la instrucción*: Las señales de control para leer la memoria de instrucciones y para escribir el PC están siempre asertadas, así que no hay control especial en esta etapa de la segmentación.

2. *Decodificación de la instrucción/búsqueda de registros*: Como en la etapa anterior ocurre lo mismo en cada ciclo de reloj, así que no hay líneas de control opcionales que inicializar.

3. *Ejecución*: Las señales que se van a inicializar son *RegDst, ALUop* y *ALUSrc*. Las señales seleccionan el registro Result, la operación de la ALU, y/o un registro o un inmediato con signo extendido para la ALU.

4. *Etapa de memoria*: Las líneas de control que se inicializan en esta etapa son *Branch*, *MemRead* y *MemWrite*. Estas señales las

inicializan las instrucciones branch on equal (beq), loadword (lw), y storeword (sw), respectivamente.

5. *Postescritura*: Las dos líneas de control son *MemtoReg*, que decide entre enviar el resultado de la ALU o el valor de memoria a los registros, y *Regwrite*, que escribe el valor escogido.

Implementar el control significa inicializar las nueve señales de control con estos valores en cada etapa para cada instrucción. La forma más sencilla de hacer esto es extender los registros de segmentación para que incluyan la información de control. Como las líneas de control comienzan en la etapa de ejecución, podemos crear la información de control durante la decodificación de la instrucción [18].

2.3.3 Riesgos por dependencias de datos

Se tiene la siguiente secuencia de instrucciones:

```
SUB     $2,$1,$3
AND     $12,$2,$5
OR      $13,$6,$2
ADD     $14,$2,$2
SW      $15,100($2)
```

Las 4 instrucciones luego de la instrucción SUB, tienen como operando al registro $2, o en otras palabras, esas 4 instrucciones dependen del resultado obtenido con esa primera operación.

El problema surge debido a que las operaciones AND, OR, ADD y SW, necesitan el valor contenido en el registro $2. Se muestra a continuación un diagrama de cómo se ejecutarían estas instrucciones conforme avanzan los ciclos de reloj.

Para mantener el orden adecuado en el tiempo, este camino de datos segmentado descompone el banco de registros en dos mitades lógicas: lectura de registros durante ID y escritura de registros durante WB. Esta división tiene sentido, porque las mitades están unidas lógicamente solamente cuando se lee y escribe en el mismo registro. Por ahora es útil pensar en la mitad de la lectura y la mitad de la escritura como recursos separados.

La figura 2.5 muestra lo que sucede con el registro $2 conforme se ejecuta el conjunto de instrucciones anteriormente mostradas.

Se asume que antes de la primera instrucción el valor del registro $2 es 10 y el resultado de la instrucción SUB es -20.

ESCUELA SUPERIOR POLITÉCNICA ESPOL
CIB-ESPOL

Para la operación AND, se requiere que el valor que esté presente en el registro $2 sea el valor obtenido luego de la operación SUB, es decir, -20. Este resultado es requerido en el ciclo CC3, pero como se observa en la figura, dicho valor estará presente en el registro $2 en el ciclo CC6. De igual manera las instrucciones OR y ADD, requieren de este valor en ciclos de reloj anteriores a CC6, entonces, como resultado estas operaciones se ejecutarán con valores errados en los operandos, con la operación SW no hay problema de dependencia porque este lee luego de que la operación SUB ya terminó el ciclo de escritura en registros.

Realizando un análisis más detallado nos damos cuenta que estas dependencias ocurren cuando la dirección de uno o ambos registros operando (rs o rt) en la etapa de decodificación, es igual al registro destino de una operación ejecutada anteriormente la cual no ha completado su ciclo, es decir, se encuentra en la etapa de ejecución, en la etapa de memoria o en la etapa de escritura en registros, es decir, tenemos dependencia de datos cuando se cumple al menos una de las siguientes condiciones:

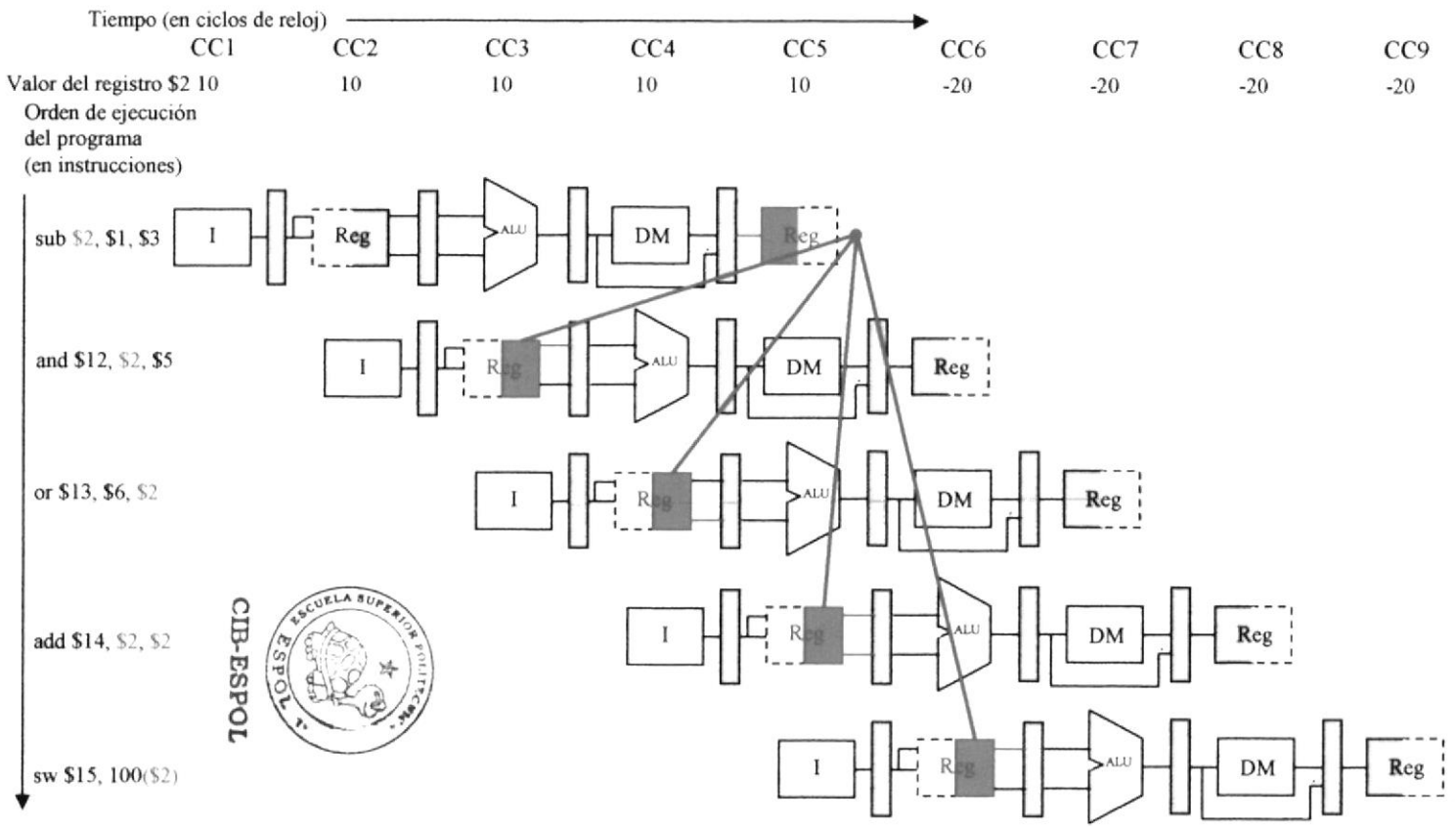


Figura 2.5 Dependencias segmentadas en una secuencia de cinco instrucciones utilizando caminos de datos simples para mostrar las dependencias.

1a. ID/EX.WriteRegister = IF/ID.ReadRegister1

1b. ID/EX.WriteRegister = IF/ID.ReadRegister2

2a. EX/MEM.WriteRegister = IF/ID.ReadRegister1

2b. EX/MEM.WriteRegister = IF/ID.ReadRegister2

3a. MEM/WB.WriteRegister = IF/ID.ReadRegister1

3b. MEM/WB.WriteRegister = IF/ID.ReadRegister2

Siempre y cuando la instrucción escriba en el banco de registros, es decir, la señal RegWrite esté a un nivel alto.

2.3.4 Control de riesgos por dependencias de datos: Burbujas

Una forma de controlar la dependencia de datos es mediante la acción de detener las instrucciones de la segmentación hasta que el dato requerido por las instrucciones subsiguientes se guarde en el registro destino determinado.

Estas detenciones reciben comúnmente el nombre de burbujas y deben darse cuando el resultado de una operación es requerido en operaciones subsiguientes cuando aún no ha sido guardado en el registro destino.

Las detenciones consisten en desactivar las señales de control en el registro de segmentación ID/EX. El efecto de realizar esto es que no se escribe en ningún registro ni en memoria si los valores de control son todos cero [14].

Con la implementación de las burbujas el diagrama de operaciones vs. Ciclos de reloj visto se modifica como lo muestra la figura 2.6.

2.3.5 Reducción de Riesgos por Dependencia de Datos: Anticipación

Detener la segmentación garantiza una ejecución correcta, pero el coste de la exactitud es un rendimiento más bajo. Si se examina más cuidadosamente el ejemplo anterior, se observa que el valor necesario para la instrucción AND, como entrada a la ALU en el ciclo de reloj 4, existe realmente en el campo ALUResult del registro de segmentación EX/MEM de la instrucción SUB.

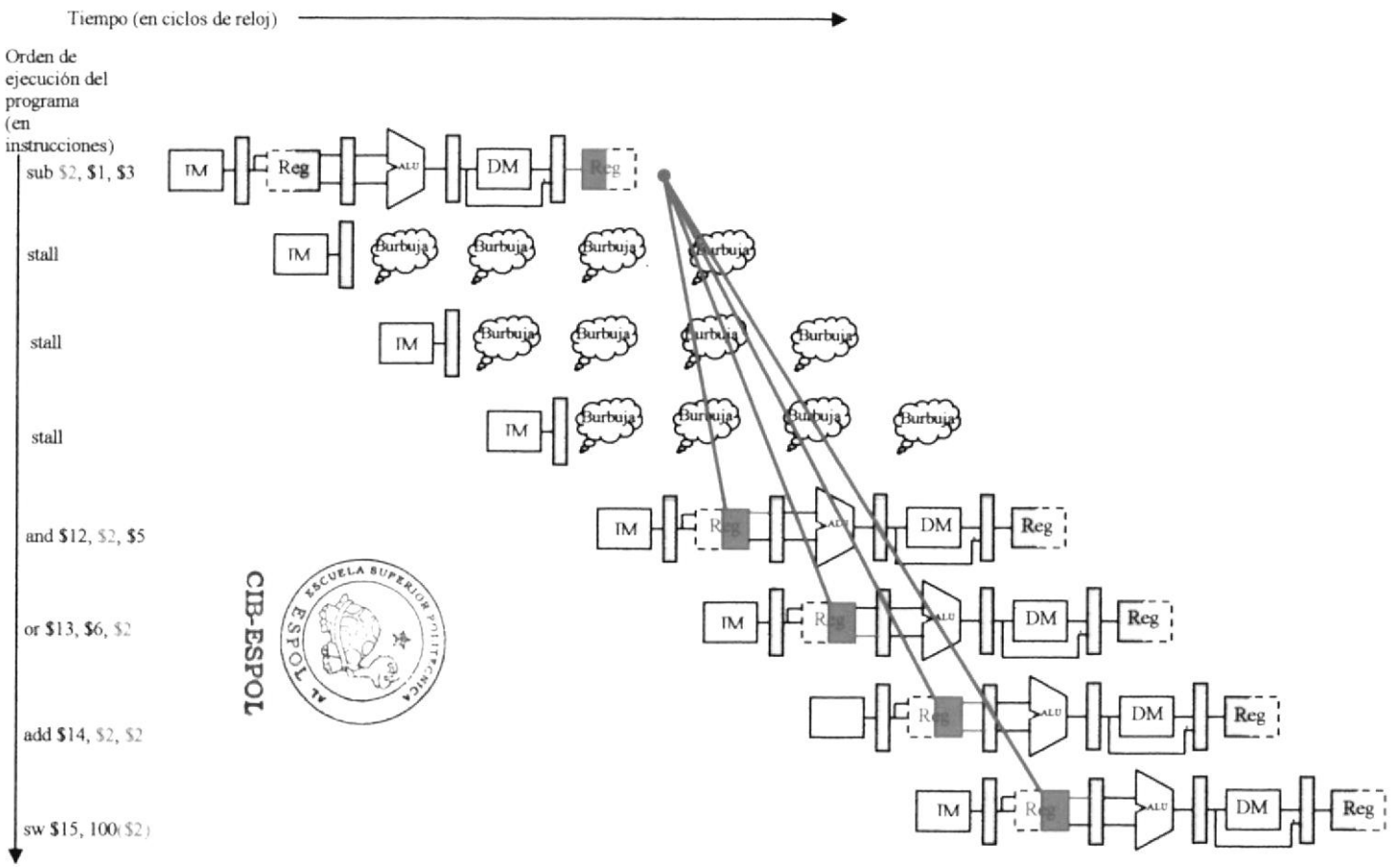


Figura 2.6 Una versión de la figura 2.5, mostrando la forma en que realmente se inserta las detenciones en la segmentación.

Análogamente, la entrada de la ALU para la siguiente instrucción OR se puede encontrar en el registro de segmentación MEM/WB de la instrucción *sub*. Si cambiamos el archivo de registros para que suministre el valor escrito si la lectura y escritura se hacen en el mismo registro, el camino de datos también puede suministrar el operando para la instrucción ADD.

La figura 2.7 muestra las dependencias entre los registros de segmentación y las entradas a la ALU para la misma secuencia de código vista anteriormente. El cambio es que la dependencia comienza en un registro de segmentación en lugar de esperar que la etapa de WB escriba los registros reales. El dato requerido está en los registros de segmentación para que eventualmente sea utilizado por instrucciones posteriores, sugiriendo un *atajo* que puede reducir pérdidas de rendimiento por la detención. Si podemos tomar las entradas a la ALU desde cualquier registro de segmentación y no necesariamente desde ID/EX, la segmentación puede proceder sin detenciones [18].

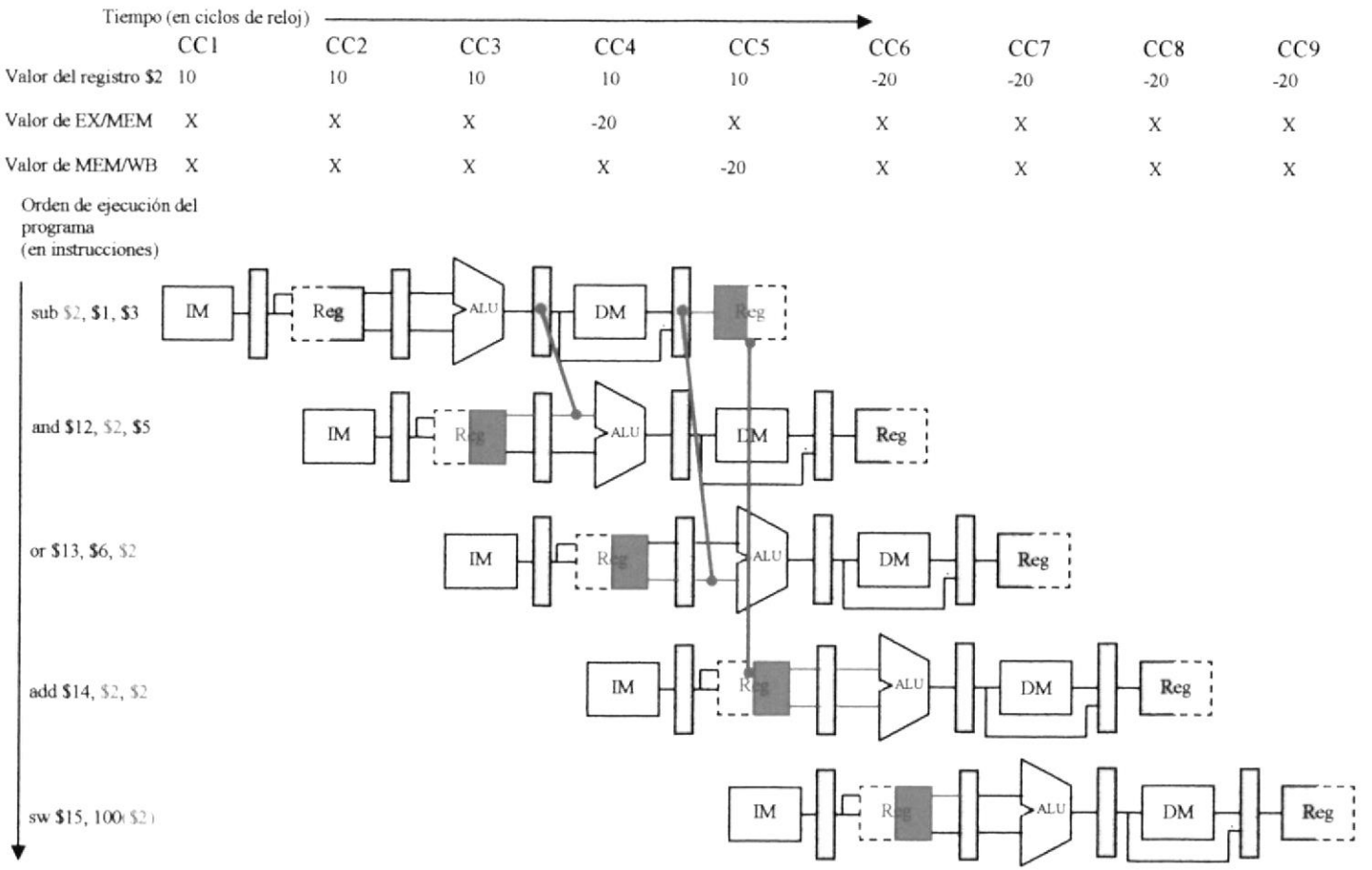


Figura 2.7 Efecto de la anticipación para corregir las dependencias de datos.

Esta técnica, que utiliza resultados temporales en lugar de esperar que los registros sean escritos, se denomina *anticipación (forwarding).* Añadiendo multiplexores a la entrada de la ALU y suministrando un control similar a la unidad de detección de riesgos podemos correr la segmentación rápidamente en presencia de estos riesgos por dependencias de datos.

Este control de anticipación estará en la etapa EX, porque los multiplexores de anticipación de la ALU se encuentran en esa etapa. Así debemos pasar los números de registro desde la etapa ID vía el registro de segmentación ID/EX, para determinar si existen valores de anticipación.

2.3.6 Riesgos de control

Otro de los riesgos que existen en un esquema segmentado es el riesgo de control o riesgo de saltos. Este ocurre cuando se tiene una instrucción del tipo salto condicional, en el cual se debe esperar a que la instrucción realice la comparación en la etapa EX, para saber si el salto debe tomarse o no.

Una solución sencilla es la de detener todo el pipeline hasta que se conozca el resultado de la comparación, esto es una solución simple y nos cuesta ciclos de reloj perdidos en el caso de que el salto no debiera tomarse.

Otra solución es la de suponer siempre que el salto no se va a realizar y permitir que las instrucciones subsiguientes a la instrucción de salto se ejecuten, y en caso de que el salto deba tomarse se efectúa un *Flush*, es decir, las instrucciones que ingresaron detrás de la instrucción de salto deben eliminarse, ya que no deben ejecutarse si el salto se realiza, esto se logra desasertando las señales de control en los registros de segmentación donde se encuentran las instrucciones que no debieron ejecutarse.

2.3.7 Rendimiento de los sistemas segmentados.

La razón de diseñar procesadores segmentados es conseguir mayor rendimiento, y como hemos visto, la segmentación reduce el tiempo medio de ejecución por instrucción. Los riesgos limitan las ganancias que se obtienen de la

segmentación, pero las técnicas hardware y software han sido proyectadas para evitar estos límites.

El escritor de compiladores debe comprender la segmentación de la máquina objeto para que consiga el mejor rendimiento; en otro caso, detenciones inesperadas pueden echar a perder las ventajas rendimiento segmentado [14].

La segmentación mejora el tiempo de ejecución por instrucción. Dependiendo de si se comienza con un camino de datos monociclo o multiciclo, esta reducción puede considerarse que proviene de disminuir la duración del ciclo de reloj o de disminuir el número de CPI. Comparando con el sencillo camino de datos monociclo, la segmentación reduce la duración del ciclo de reloj de este camino de datos. La figura 2.8 muestra el efecto sobre el CPI y la frecuencia de reloj para los caminos de datos, monociclo, multiciclo y segmentado, ofreciendo este último un CPI bajo y una frecuencia de reloj rápida [14].

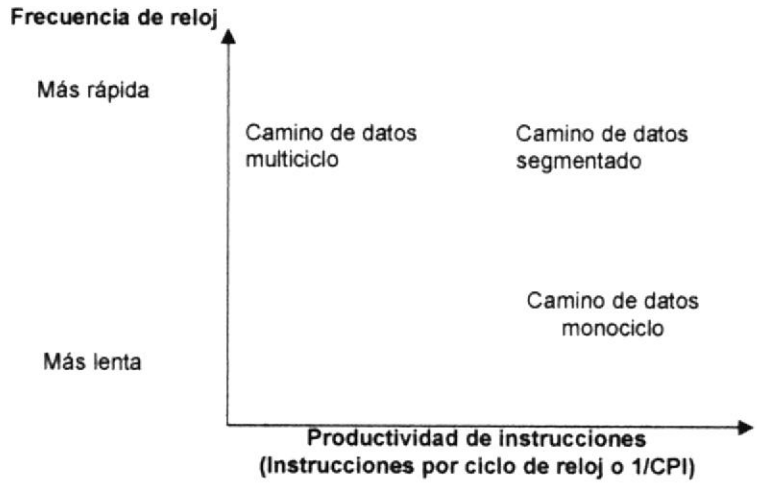


Figura 2.8 Desempeño de los diferentes caminos de datos en cuanto a la frecuencia de reloj y CPI.

CIB-ESPOL

CAPÍTULO 3

IMPLEMENTACIÓN DEL DECODIFICADOR VITERBI EN HARDWARE Y EN SOFTWARE



CIB-ESPOL

Este trabajo de tesis propone dos implementaciones del decodificador Viterbi, en hardware y en software.

La implementación en hardware se realizó sobre un FPGA por medio del lenguaje de descripción de hardware VHDL.

La implementación en software usó como plataforma el procesador MIPS anteriormente descrito, el cual también fue implementado sobre un FPGA como se detalla en las siguientes secciones.

Además se describe la implementación en lenguaje C de un canal inalámbrico con codificación de canal parametrizable, que sirve como modelo de referencia para comprobar los estudios teóricos y evaluar el rendimiento de las implementaciones. La función que corresponde al decodificador de Viterbi dentro de este modelo, sirve como código fuente para generar las instrucciones ejecutadas en el procesador MIPS.

Para entender mejor los detalles de las implementaciones presentadas en este capitulo, se recomienda revisar los códigos que se encuentran en el CD de soporte anexo a este documento.

3.1 Implementación del decodificador Viterbi en VHDL

A continuación se describirán los detalles de la implementación del código en VHDL de las unidades principales del decodificador Viterbi, basados en la arquitectura básica que propone el capítulo 1.

3.1.1 BMU

Para la implementación del decodificador de Viterbi de entrada de decisión suave, las entradas pasan primero por un bloque de ecualización antes de ingresar al decodificador.

En este caso se tienen dos entradas de tres bits por cada símbolo cuantificado, ya que la tasa de codificación es de ½. Esto quiere decir que por cada bit de dato de la secuencia de entrada, se transmitirán dos. Según la figura 1.8 que muestra la curva del Algoritmo de Viterbi para diferentes niveles de

cuantización se sabe que este nivel de 3 bits es adecuado para alcanzar prácticamente el mismo desempeño que con más bits para representar el símbolo.

Como se indicó en el capítulo 1, dada una decodificación de decisión suave, el bloque BMU se calculará las métricas de rama por medio de la Distancia Euclidiana, la cual obedece a la siguiente ecuación:

$$BM = \sum_{i=0}^{n-1} (rec_i - esp_i)^2$$

Como se observa según la ecuación, esta operación requiere de muchas multiplicaciones, lo cual incurre en un excesivo de recursos de hardware para este bloque. Por lo que para esta implementación se hizo un tratamiento a la expresión anterior de manera que se pueda eliminar dichas multiplicaciones, como se muestra a continuación:

$$BM = \sum_{i=0}^{n-1} (rec_i - esp_i)^2$$

$$BM = (rec_0 - esp_0)^2 + (rec_1 - esp_1)^2$$

$$BM = rec_0^2 - 2rec_0 esp_0 + esp_0^2 + rec_1^2 - 2rec_1 esp_1 + esp_1^2$$

Ya que se sabe que estos valores serán utilizados para compararse en la unidad ACS y así actualizar el valor de una nueva métrica de nodo, analizando esa comparación se tiene:

$$PM = \min (PMi + rec_0^{\,2} - 2rec_0 esp_0 + esp_0^{\,2} + rec_1^{\,2} - 2rec_1 esp_1 + esp_1^{\,2},$$
$$PMj + rec_0^{\,2} - 2rec_0 esp_0 + esp_0^{\,2} + rec_1^{\,2} - 2rec_1 esp_1 + esp_1^{\,2})$$

$$PM = \min (PMi - 2rec_0 esp_0 + esp_0^{\,2} - 2rec_1 esp_1 + esp_1^{\,2},$$
$$PMj - 2rec_0 esp_0 + esp_0^{\,2} - 2rec_1 esp_1 + esp_1^{\,2})$$

Entonces al comparar las dos posibles métricas de nodo de los estados previos, se observa que el dato recibido es el mismo para ambas por lo cual no influye en la comparación y no es necesario incluir ese valor en el cálculo efectuado por el BMU. De donde, tenemos que el valor final a ser calculado viene dado por:

$$BM = -2rec_0 esp_0 + esp_0^{\,2} - 2rec_1 esp_1 + esp_1^{\,2}$$

Se sabe que los valores esperados solo pueden tomar los valores de 3 o -4 debido a que son la representación en tres bits (donde el bit más significativo es el bit de signo) de un 0 o un 1 obtenidos a la salida del codificador, entonces tendremos cuatro casos posibles para el cálculo de las nuevas métricas de rama:

Si esp0= 3 y esp1= 3

$$BM = 18 - 6\,rec_0 - 6rec_1$$

Si esp0= 3 y esp1= -4

$$BM = 25 - 6\,rec_0 + 8\,rec_1$$

Si esp0= -4 y esp1= 3

$$BM = 25 + 8\,rec_0 - 6rec_1$$

Si esp0= -4 y esp1= -4

$$BM = 32 + 8\,rec_0 + 8rec_1$$

De esta manera, en el código en VHDL de la unidad BMU se pregunta si los valores esperados son los descritos anteriormente para ver cual de las 4 operaciones es la que se debe realizar y así se evita las multiplicaciones que implicaba elevar al cuadrado los valores esperados.

Por otro lado, las multiplicaciones por 8 y por 6, se realizan usando desplazamientos a la izquierda que son mucho menos costos de efectuar que las multiplicaciones. Por ejemplo, en el código de la multiplicación por 6, se desplaza una vez a la izquierda, lo que equivale a multiplicar por 2 y se desplaza dos veces a la izquierda, lo que equivale a multiplicar por 4, estas

dos palabras luego se suman y se obtiene la palabra multiplicada por 6.

La métrica de rama a la salida de esta unidad está representada por 8 bits. El tamaño de la métrica de rama tiene fuerte relación con el tamaño de la métrica de nodo, como se explicará en la siguiente sección.

Según las especificaciones del estándar el decodificador Viterbi debe tener 64 estados, por lo que se tienen 128 ramas en la transición de un estado a otro en el diagrama de Trellis. Entonces, para esta implementación, se tienen 128 bloques BMU interconectados entre sí. Las etiquetas de rama o símbolos esperados se guardan como constantes en cada uno de estos bloques.

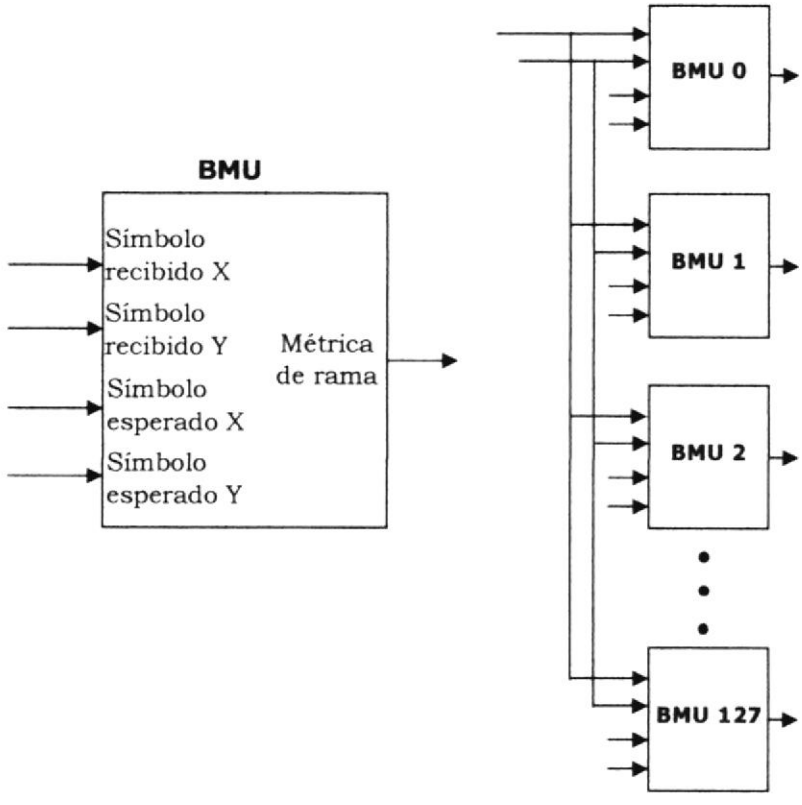


Figura 3.1 Diagrama de Bloques de interconexión de unidades BMU

3.1.2 ACS

Como ya se ha expuesto, esta unidad ACS realiza la operación de suma de la métrica de rama con la del nodo, selecciona la menor y de acuerdo a esta selección designa el bit de decisión. Por lo tanto, un bloque ACS recibe de entrada la métrica de rama de 8 bits y la métrica del nodo de 14 bits. Las salidas de este bloque son el bit decisión y la métrica del nodo que va a actualizarse por medio de registros.

Dado que se tienen 64 estados que necesitan actualizar sus métricas de nodo, en la implementación se interconectan 64 bloques ACS, así se obtiene de forma paralela 64 bits decisión, que forman el llamado vector decisión el cual ingresará a la memoria del siguiente bloque del decodificador, el Rastreo hacia atrás (o TB por sus siglas en inglés de Trace Back).

La selección del ancho de la métrica de nodo de 14 bits tiene que ver con la cantidad de bits del flujo de entrada, debido a que el cálculo de una métrica de nodo es acumulativo, es decir, se suma una cantidad a la métrica de nodo anterior, cada vez que un nuevo dato ingresa al decodificador. El diseño asume un flujo de 256 bits lo que causa que en el peor de los casos se llegue a un valor de métrica de nodo de -8192 el cual para ser representado necesita de 14 bits. Esto se justificará en las simulaciones.



CIB-ESPOL

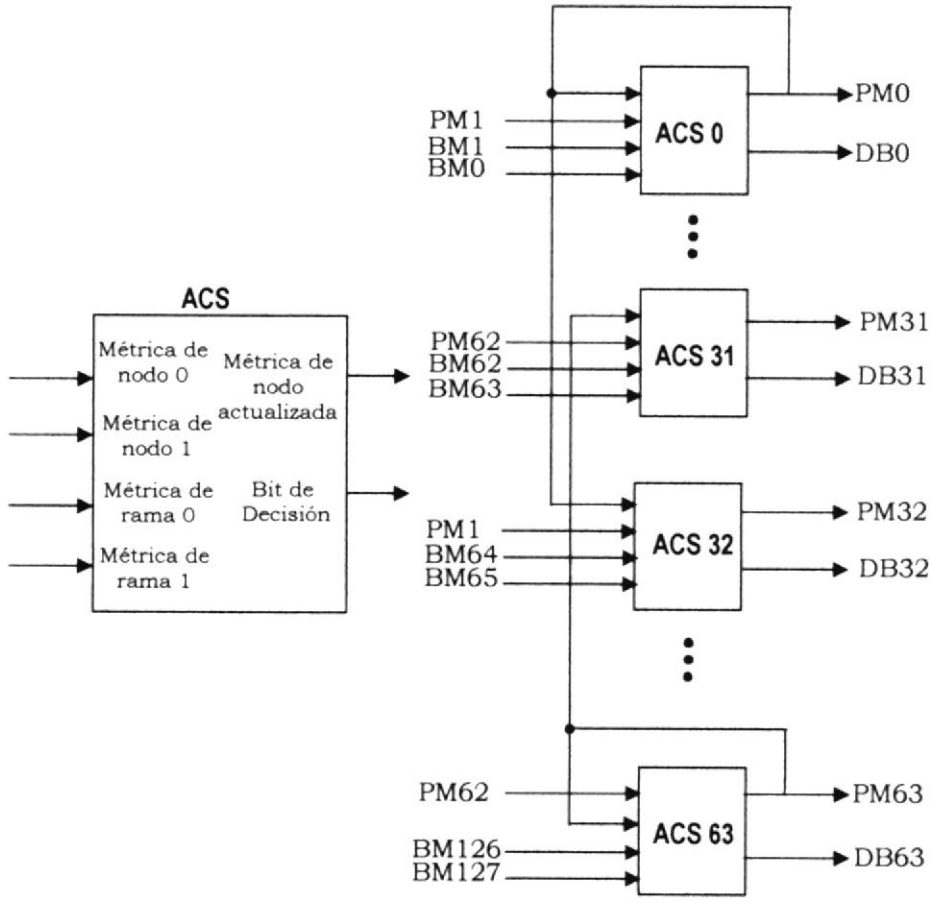


Figura 3.2 Diagrama de Bloques de interconexión de unidades ACS

3.1.3 TB

Una dificultad que se presenta en esta implementación del Trace Back, es el hecho de que mientras más grande sea la trama de entrada, más grande deberá ser el tamaño de la memoria, ya que por cada dato que ingresa, se obtiene un

nuevo vector decisión que debe ser almacenado. Es decir, si la trama de entrada es de un tamaño de 1024 datos, la memoria deberá tener entonces un tamaño de 1024 localidades de memoria.

El método de rastreo hacia atrás almacena información de los caminos en forma de un arreglo de punteros recursivos. Es ventajoso pensar en la memoria del rastreo hacia atrás como si estuviera organizada en una estructura bidimensional, con filas y columnas. El número de columnas es igual al número de estados $N=2^n$. Cada fila almacena el resultado de N comparaciones correspondientes a una etapa en el diagrama de Trellis. Debido a que cada flujo de símbolos es, en general, semi-infinito, las localidades de almacenamiento son reutilizadas periódicamente.

Para solucionar este inconveniente, en la implementación se hace uso de una propiedad llamada *Longitud de Convergencia del Diagrama de Trellis [2]* la cual básicamente nos dice que no es necesario esperar a que se haya ingresado toda la trama de datos para empezar a obtener bits decodificados a la salida del la unidad de rastreo hacia atrás.

Utilizando entonces esta propiedad dividimos la unidad, en cuatro etapas como se muestra en la figura 3.3. La implementación de esta etapa del decodificador se realizará entonces utilizando 4 bloques TB cada uno de ellos con una memoria RAM síncrona de un solo puerto de 64 x 64 bits (debido a que se tienen 64 estados, y por lo tanto 64 bits por cada vector decisión) generadas con la herramienta IP Core Generador de Xilinx.

En la figura 3.3 se muestra la interconexión de unidades TB para el decodificador Viterbi de 64 estados.

Antes de entrar en el detalle del funcionamiento del bloque de rastreo hacia atrás, vale describir los siguientes términos:

- Rastreo hacia atrás de convergencia.- El funcionamiento del bloque TB es el descrito en el capítulo 1, aprovechando la propiedad de convergencia del Trellis, el registro de corrimiento de la etapa de selección se inicializa con un estado cualquiera.



CIB-ESPOL

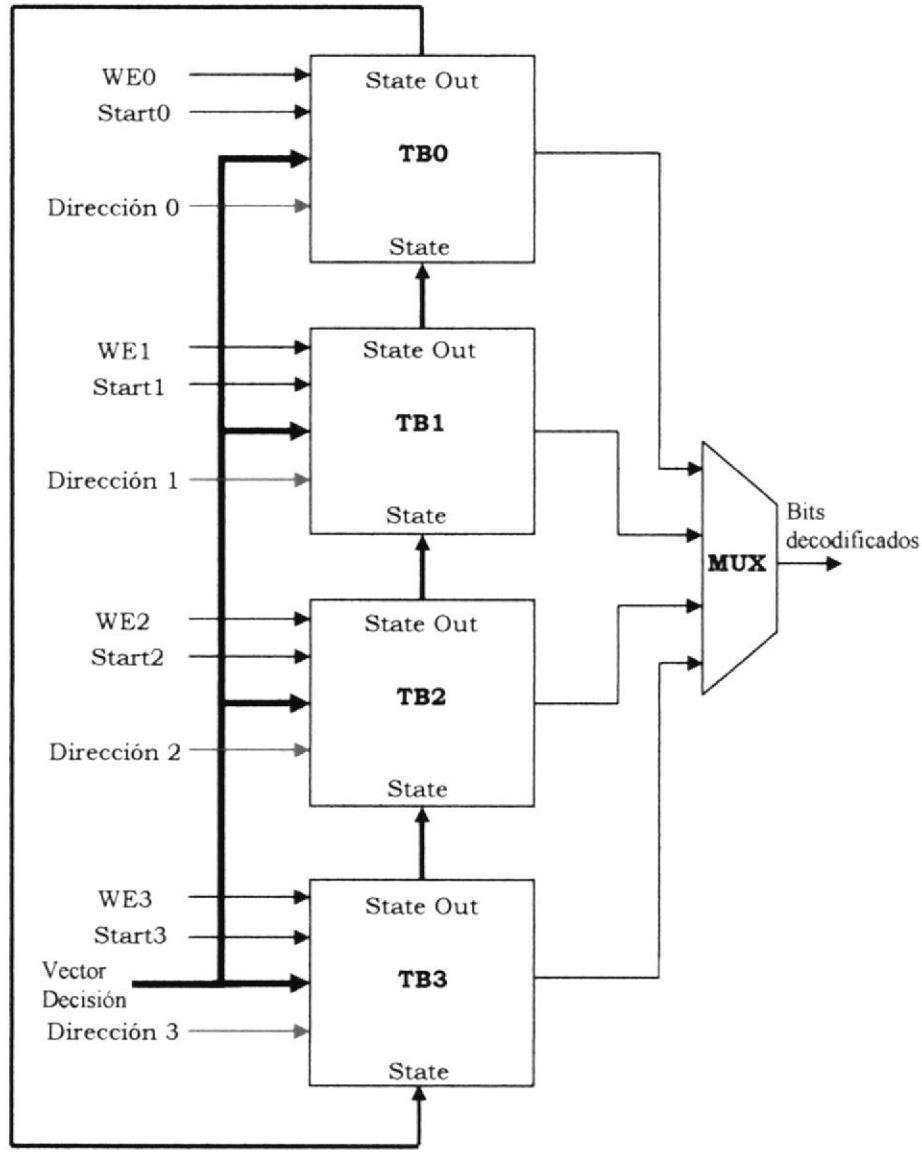


Figura 3.3 Diagrama de bloques de interconexión de unidades TB

Al finalizar de leer toda la memoria RAM el estado que se encuentra almacenado en el registro de desplazamiento es el estado de convergencia que

servirá como punto de partida para que el bloque TB siguiente inicie su rastreo hacia atrás.

- Rastreo hacia atrás de decodificación.- El registro de corrimiento se inicializa con el estado de convergencia proporcionado por un bloque TB anterior, a partir de este y siguiendo el funcionamiento normal de un bloque de rastreo hacia atrás como se describió en el capítulo 1, cada ciclo de reloj el bit más significativo del registro de corrimiento será considerado un bit decodificado.

El funcionamiento en conjunto de los 4 bloques TB mostrados en la figura 3.3 es el que se observa en la figura 3.4. Se nota que para el instante t=t7 el conjunto de operaciones a realizarse es el mismo que en el instante t=t3, de igual manera para el instante t=t8 serán las mismas instrucciones que para t=t4, y así sucesivamente.

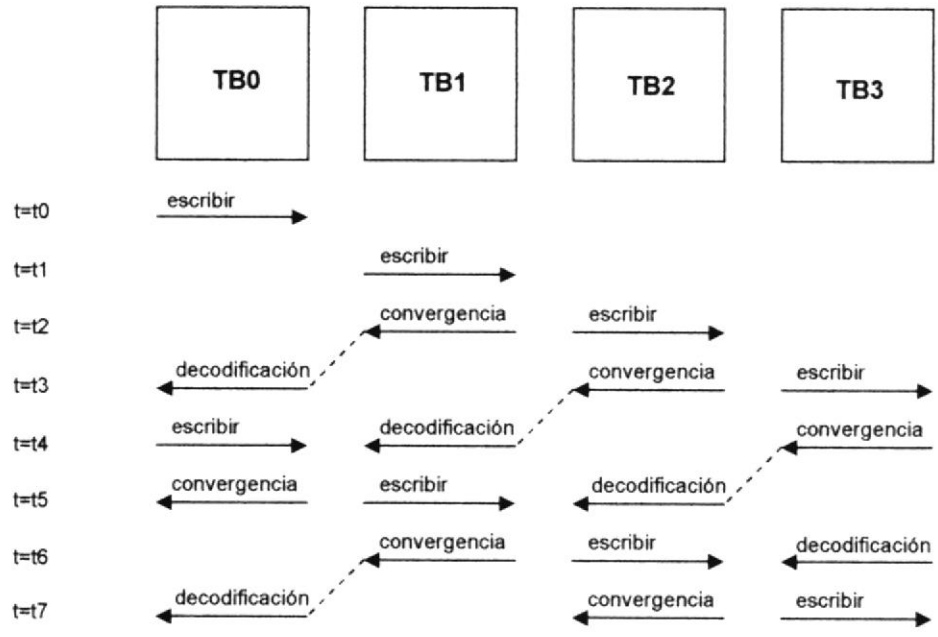


Figura 3.4 Funcionamiento de Bloques TB en el diseño del decodificador

Una forma de implementar el control para este conjunto de operaciones a realizarse en cada uno de los bloques TB es mediante el uso de una máquina de estados que controle las señales habilitadoras de escritura (WE) de cada una de las memorias, así como las direcciones desde las que se va a leer o en las que se van a escribir vectores decisión en cada una de ellas (recordar que para escribir se lo hace desde la dirección 0 hasta la 63 mientras que para las operaciones de rastreo hacia atrás tanto para convergencia como para decodificación se lo hace en sentido contrario) y finalmente una señal para controlar

el multiplexor que elige de cual de los bloques TB se obtendrán los bits decodificados.

Dado que la operación de lectura en el rastreo hacia atrás se realiza desde la dirección más significativa a la menos significativa, los bits decodificados saldrán en grupos de 64 (que es el tamaño de la memoria en cada etapa de la implementación de TB) pero en orden inverso, es decir, si ingresamos 256 bits de datos la salida será en este orden:
63-0, 127-64, 191-128, 255-192.

Para solucionar este inconveniente, se implementó dos bancos de registros de 64 localidades, como se muestra en la figura 3.5, los cuales se empiezan a llenar una vez que se obtienen los primeros bits decodificados, los cuales son guardados desde la dirección 0 a la 63, una vez que este banco de registros está lleno, se empieza a leer pero en sentido contrario, es decir, desde la dirección 63 a la dirección 0, con ello se consigue que se ordenen los bits decodificados. Estos actúan como una LIFO.

A la vez que se lee la primera LIFO, se escribe en la segunda, luego cuando esta última se llena, se empieza a leer de la misma forma como se hizo con el primera LIFO, ordenando los bits del 64 al 127 y sigue así el procedimiento hasta que sean leídos todos los bits de la trama de bits decodificados.

El multiplexor que se observa selecciona siempre la LIFO del cual se están leyendo los bits decodificados.

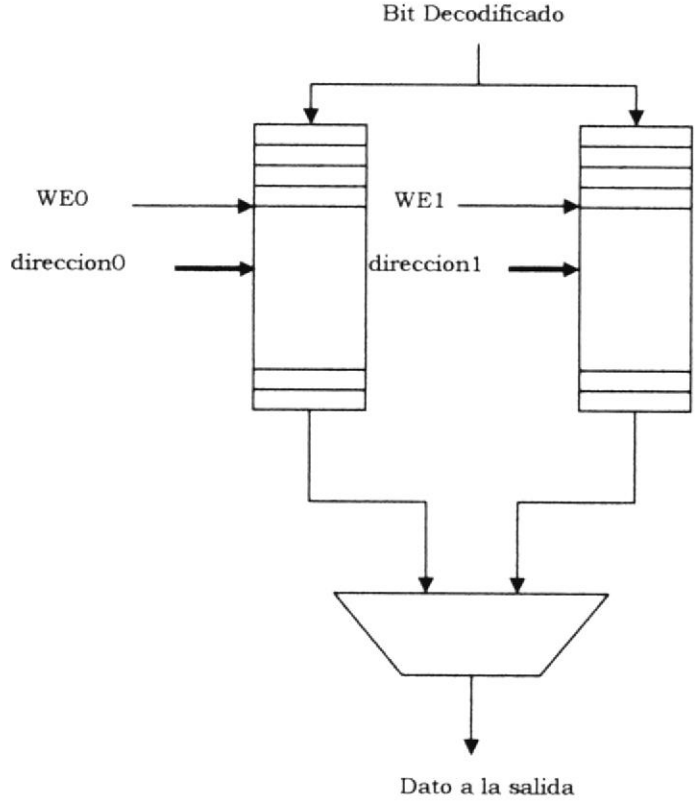


Figura 3.5 Diagrama de bloques de conexión de las LIFOS

3.1.4 Cama de Prueba

Una vez comprobado el correcto funcionamiento del decodificador Viterbi, se diseñó una cama de prueba que pueda utilizarse para la implementación en software y en hardware que permita obtener una mejor evaluación del desempeño del algoritmo sobre una secuencia de datos más extensa y continua.

La estructura de la cama de prueba consta de tres partes básicamente: Una computadora, un cable serial y la tarjeta de desarrollo del FPGA, tal como lo muestra la figura 3.6

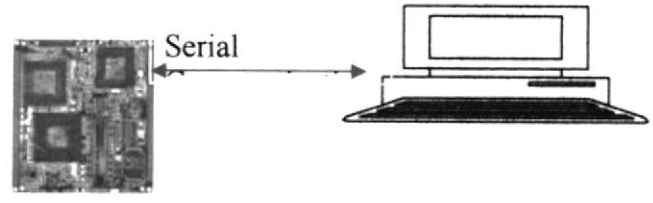


Figura 3.6 Cama de Prueba

La cama de prueba trabaja conjuntamente con el modelo de un canal de comunicación inalámbrico con codificación de canal, descrito en la sección 3.3. El programa que modela este canal

CIB-ESPOL

recibe el mensaje con la información original. En este caso, se usó una imagen como mensaje.

La imagen a ser procesada se muestra en la figura 3.7. Esta imagen es un archivo de extensión RAS que a través de una rutina hecha en C se convierte en un archivo binario, el cual es dividido en secuencias de 256 bits a la entrada del transmisor.

Cada una de estas secuencias se codifica según las especificaciones del estándar y luego por medio de una función en C, se le asigna diferentes niveles de ruido blanco gaussiano, modelando así un canal ruidoso.



Figura 3.7 Imagen de Lena

Después en el receptor cada secuencia se cuantifica también por medio de una función y se generan los símbolos codificados. Cuando se complete este proceso para todas las secuencias de 256 bits, se forma el archivo completo de símbolos codificados. Este archivo es utilizado en la cama de prueba como entrada al decodificador.

La cama de prueba incluye un programa hecho en lenguaje C que configura el puerto serial y permite enviar el archivo al decodificador.

El decodificador recibe los datos, los procesa y devuelve a la computadora los bits decodificados, estos son comparados con el mensaje de entrada, es decir con el archivo binario de la imagen, con esto se obtienen los resultados de probabilidad de error que permitan obtener curvas de BER.

Dado que el tamaño de la imagen de Lena es de 2MB y se codifica a una tasa de 1/2, el tamaño del archivo a la entrada del decodificador es el doble, es decir 4MB en total. El diseño del decodificador no puede recibir directamente un archivo de este tamaño, por tanto, por medio de una rutina en lenguaje C,

la información se divide y se envía en paquetes de 512 símbolos, espera hasta recibir los 256 bits decodificados y está listo para enviar el siguiente paquete.

Este proceso es ejecutado con 16 archivos con diferente nivel de SNR, que van desde -5dB (que se considera un canal muy ruidoso) hasta 10 dB (un canal libre de ruido).

Luego con una rutina en C que compara bit por bit el archivo decodificado con la secuencia original de la imagen de Lena, se obtiene una probabilidad de error para los diferentes niveles de SNR.

Cuando se tienen los 16 valores de probabilidad de error, con ayuda de Matlab se genera una curva de BER que permitirá evaluar el desempeño de la decodificación con las simulaciones previamente estudiadas.

3.2 Implementación de un procesador MIPS segmentado

3.2.1 Descripción de Componentes por etapas de segmentación

La primera aproximación de la implementación del MIPS sobre un FPGA se realizó en VHDL conforme a la arquitectura segmentada revisada en el capítulo 2.

A continuación se hará una descripción de los detalles de la implementación del código para el camino de datos segmentado y la unidad de control, esto es cumpliendo con las 5 etapas de segmentación sugeridas para el procesador:

1. IF: Búsqueda de la instrucción.
2. ID: Decodificación de la instrucción y búsqueda de registros.
3. EX: Ejecución y cálculo de la dirección efectiva.
4. MEM: Acceso a memoria.
5. WB: Post-escritura.

Búsqueda de la instrucción

En esta parte de la segmentación se tiene la unidad de memoria para almacenar las instrucciones de un programa. En el diseño,

esta memoria es síncrona de 1024 localidades de 32 bits, por lo que la entrada para la dirección tiene un ancho de 10 bits y la salida para la instrucción es de 32 bits. Fue generada por medio de la herramienta IPcore Generador que ofrece ISE de Xilinx.

Además se tiene el contador de programa –PC-, registro que guarda la dirección de la siguiente instrucción a ejecutarse y finalmente, un sumador para incrementar el PC a la dirección de la siguiente instrucción.

Al final de esta etapa se tiene el registro de segmentación IF/ID que separa esta etapa de la siguiente.

Decodificación de la instrucción y búsqueda de registros

En esta etapa se encuentra el banco o archivo de registros, que tiene un total de 4 entradas y 2 salidas. Las tres entradas de los números de registros (una para escritura y dos para lectura) son de 5 bits para especificar uno de los 32 registros ($2^5 = 32$), mientras que la entrada de dato y las dos salidas de datos son cada una de 32 bits.

El archivo de registros siempre tiene a la salida el contenido de cualquier número de registro que se encuentre en las entradas. Sin embargo, las escrituras están controladas por la señal de control de escritura *RegWrite*, que debe estar lógicamente en alto para que se produzca una escritura cuando se presenta el flanco de reloj.

El registro en el cual se va a escribir el resultado de alguna operación, puede estar dado por el campo Rd o Rt dependiendo del tipo de instrucción, para esta elección se implementa un multiplexor el cual es controlado por la señal *RegDst* la cual es un '1' lógico cuando la instrucción es tipo R, seleccionando Rd como registro destino y '0' cuando es una operación inmediata.

Las instrucciones tipo I tienen el campo *Inmediato* de 16 bits, el cual actúa como segundo operando de la operación a realizarse, o en el caso de un salto condicional para el desplazamiento del PC. En ambos casos este operando debe ingresar a la ALU o a un sumador, respectivamente. Tanto la ALU y el sumador reciben dos operandos de 32 bits, por lo cual se tiene una unidad de *Extensión con Signo* la cual concatena 16 bits delante del operando de 16 bits del campo Inmediato de la instrucción, bajo

el criterio de que si el bit más significativo de este campo es un '1' lógico, entonces para conservar el signo, se añaden 16 unos caso contrario se añaden 16 ceros.

En esta etapa se encuentra en el registro de segmentación ID/EX.

Ejecución y cálculo de la dirección efectiva

En esta etapa de la segmentación se encuentra la ALU o Unidad Aritmético Lógica, la cual tiene cuatro entradas, dos de 32 bits que representan los operandos; una entrada para el campo Shift Amount que se usa en caso de que se efectúe una operación de desplazamiento. Este campo indica la cantidad de desplazamientos a realizarse y finalmente una entrada llamada ALUCtrl de 4 bits la cual le indica a la ALU que operación debe realizar, de acuerdo a la tabla VII.

La ALU tiene una salida de 32 bits que representa el resultado de la operación realizada.

CIB-ESPOL

Tabla VII. Operaciones de la ALU en la arquitectura MIPS implementada

ALU Ctrl	Operación
0000	Add
0001	Sub
0010	And
0011	Shift Right Logical
0100	Shift Left Logical
0101	Set On less
0110	Or
0111	Xor
1000	Shift Right Arithmetical
1001	Load Upper Inmediate
1010	Set On Less Unsigned

Dado que el segundo operando de la ALU puede ser el valor del registro Rt (para instrucciones tipo R) o el dato inmediato contenido en la instrucción (para instrucciones formato tipo I), existe un multiplexor que elige cual de los operandos va a entrar a la ALU. La señal selectora de este multiplexor es la señal *ALUSrc* proveniente de la unidad de control, la cual es '1' cuando se trata de una instrucción formato tipo I, y cero cuando es una instrucción tipo R.

En esta etapa de ejecución se encuentra también el Control de la ALU. Para el cálculo de la dirección efectiva en el caso de un salto, en esta etapa además de la ALU se tiene un sumador de 32 bits, el cual suma el valor del PC más el valor del campo

Inmediato de la instrucción, el que fue extendido con signo a 32 bits en la etapa de decodificación.

Acceso a memoria

En esta etapa del camino de datos se encuentra la memoria de datos, implementada por medio de dos memorias SRAM del Spartan3 Starter Kit, cada una de ellas es una memoria asíncrona de 256k localidades cada una de ellas de 16 bits.

Estas memorias no fueron implementadas en código VHDL para optimizar el diseño evitando un exagerado consumo de recursos. Ambas memorias se conectan en paralelo, es decir, las señales habilitadoras LB, UB, CE, las señales para lectura y escritura y los bits de dirección de ambas memorias se conectan entre sí. Las entradas y salidas de esta memoria y su funcionamiento se pueden consultar en el manual de usuario del FPGA.

Con esto, los 16 bits de datos de una memoria y los 16 de la otra actuarán como los bits menos significativos y los más significativos, respectivamente, de una palabra de datos de MIPS, la cual es de un tamaño de 32 bits.

La lectura y escritura está controlada por las señales WE y OE, las cuales son activas en bajo y son generadas por la unidad de control. WE se genera cuando se quiere realizar una escritura en memoria (store) mientras que la señal OE se genera cuando se desea leer un dato desde la memoria (load).

Dado que la memoria de datos tiene un bus de datos bidireccional, es necesario el uso de un búfer bidireccional de tres estados de 32 bits, el cual debe estar en estado de alta impedancia cuando se recibe un dato desde la memoria hacia el MIPS.



CIB-ESPOL

Post-escritura.

En esta etapa se elige si el dato que se va a almacenar en el banco de registros es el resultado de una operación de la ALU o el dato proveniente de un acceso a memoria. La señal que controla la selección del multiplexor encargado de esta elección es la señal MemtoReg generada por la unidad de control cuando el dato que se va a guardar en los registros proviene de la memoria.

3.2.2 Unidad de control principal

Como se explicó en el capítulo 2, la unidad de control tiene la función de controlar accesos a memoria, escritura o lectura en registros y el tipo de operación a realizarse en la ALU, para ello, recibe como entrada los 6 bits del código de operación y con ellos decide el valor que debe tener cada una de las señales encargadas de controlar cada una de esas acciones: '0', '1' o irrelevante ('X').

En la implementación realizada a más de las señales descritas en el capítulo 2, requirió de otras cuantas señales para su completo funcionamiento. Estas se detallan en la tabla VIII.

3.2.3 Control de la ALU

La unidad de Control de la ALU recibe como entradas el campo función de 6 bits de las instrucciones y la señal ALUOp de 4 bits, proveniente de la unidad de control principal. Como salida genera la señal ALUCtrl cuyo número de bits debe ser tal, que alcance para cubrir todas las operaciones que tiene que ejecutar la ALU.

En la implementación, esta señal es de 4 bits de acuerdo a la tabla VIII.

Tabla VIII. Otras Señales de la unidad de control

Señal	Función
RST_IF_ID	Pone todos los campos de las instrucciones cero (Flush).
JAL	Indica que se realizó la operación Jump and Link (JAL).
LD	Habilita transmisión por la UART.
CDM	Selecciona si el dato que ingresa a memoria es el contenido del registro Rt o el dato recibido por la UART.
En_reg5	Habilita el registro 5 bits que contiene información de la paridad y de la razón de baudios de la UART.
Cntrl_interr	Selecciona la dirección de inicio de la subrutina de interrupción como valor de entrada al PC.
Mul_Sel	Escoge al resultado de la multiplicación como dato de salida de la etapa de ejecución.
Sel_Dir_Int	Elige la dirección a la que se debe retornar luego de una interrupción.
UB_LB_CE_L	Habilita los dos bytes de cada una de las localidades de las dos SRAM usadas como memorias de datos.

La salida *ALUCtrl*, indica a la ALU qué tipo de operación ejecutar, por ejemplo la señal *ALUCtrl* tendrá el mismo valor para una instrucción ADD que para una instrucción LW o SW, dado que en estos tres casos la ALU debe efectuar una suma, como se muestra en la tabla IX.

Tabla IX. Señal *ALUCtrl* de acuerdo a las combinaciones de entradas de *ALUOp* y *function*

ALUOp	function	ALUCtrl
0000	xxxxxx	0000
0010	xxxxxx	0110
0011	xxxxxx	0010
0101	xxxxxx	0000
0110	xxxxxx	0111
0111	xxxxxx	1001
1000	xxxxxx	0101
1001	xxxxxx	1010
0100	100000	0000
0100	100001	0000
0100	100010	0001
0100	100100	0010
0100	100101	0110
0100	100110	0111
0100	000000	0100
0100	000011	1000
0100	000010	0011
0100	101010	0101
0100	101011	1010

3.2.4 Unidad de anticipación para datos

Para reducir el riesgo por dependencia de datos se implementa una unidad de anticipación. Esta unidad controla dos multiplexores, a la salida del banco de registros. Con la ayuda de la figura 3.8 se describe el criterio para generar las señales de selección de esos multiplexores.

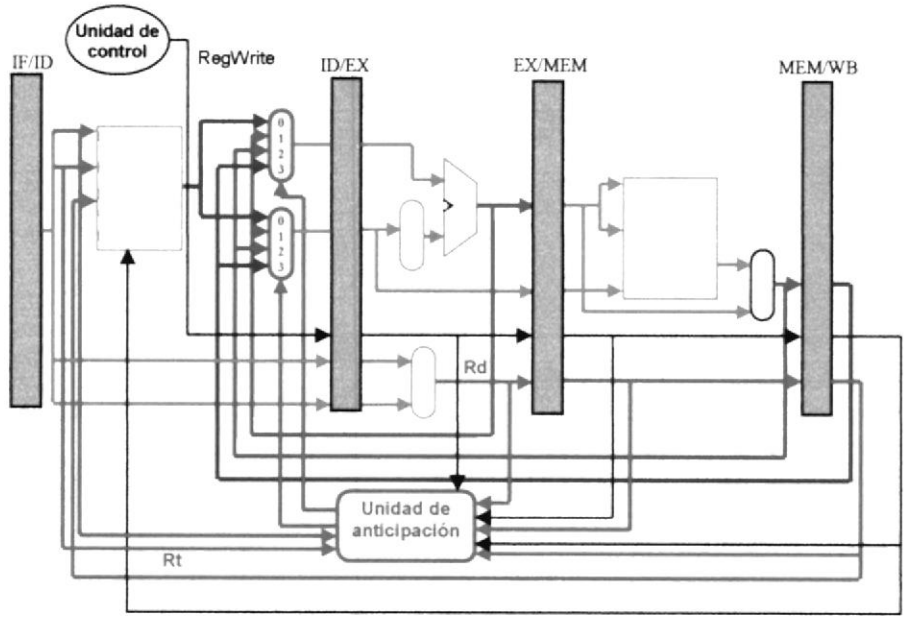


Figura 3.8 Unidad de Anticipación dentro del procesador segmentado

La Unidad de anticipación verifica si alguno de los registros operando (Rs o Rt) requeridos por una instrucción coincide con el registro destino de otra que aún no ha escrito su resultado en el banco de registros, esto se lleva a cabo comparando el valor de Rd que se encuentra en las etapas de ejecución, acceso a memoria y post-escritura, con los valores de Rs y Rt de la etapa de decodificación. Además la Unidad pregunta por el valor de la señal RegWrite en las etapas ejecución, acceso a memoria y post-escritura para cerciorarse que las instrucciones que se encuentran en estas etapas deben en realidad escribir su resultado en los registros.

ESCUELA SUPERIOR POLITÉCNICA ESPOL
CIB-ESPOL

Una complicación es que el valor de Rd que se va a comparar sea el mismo para dos o más etapas de la segmentación. Por ejemplo, cuando sumamos un vector de números en un solo registro, una secuencia de instrucciones leerá y escribirá en el mismo registro. En este caso, la prioridad se elige el valor de Rd que se encuentre en la etapa más cercana a la de decodificación.

3.2.5 Recorrido de las instrucciones en el camino de datos

Luego de revisar las etapas de segmentación y los componentes básicos del diseño, se explica a continuación el recorrido del camino de datos para las diferentes instrucciones que gobiernan el procesador MIPS.

El camino de datos para una instrucción tipo R, empieza por obtener los operandos del banco de registros, luego éstos se operan en la ALU y finalmente el resultado obtenido se almacena en el banco de registros.

Para el caso de una instrucción de carga se obtiene el valor del registro base, luego este valor es sumado en la ALU con el dato inmediato proveniente de la instrucción, el cual es extendido con

signo de 16 a 32 bits. Este resultado es la dirección de la cual se obtiene el dato de memoria que finalmente se almacena en el banco de registros.

En el caso de una instrucción de almacenamiento la obtención de la dirección se efectúa de igual manera. El dato a ser guardado en esa dirección proviene del contenido del registro Rt.

Para las instrucciones de salto condicional, el camino de datos debe efectuar básicamente dos operaciones. Primero se obtienen los dos operandos del banco de registros y ambos pasan a la ALU para efectuar una comparación, lo cual no es más que una resta que enciende una señal que indica si el resultado de esa resta es cero. Además se efectúa la suma del valor del registro PC más el desplazamiento (offset) que proviene de la instrucción, previa extensión a 32 bits con signo. Se necesita además un multiplexor para indicar si se toma o no el salto.

La instrucción de salto incondicional opera sustituyendo una parte del PC por los 26 bits inferiores de la instrucción. Esto se realiza sencillamente concatenando ceros al desplazamiento del salto incondicional.

Para incluir el camino de datos para una instrucción de salto incondicional se debe concatenar los primeros 26 bits de la instrucción de salto con los 6 bits más significativos del PC. Para esto se usa un multiplexor adicional para seleccionar la fuente del nuevo valor del PC, que es el PC incrementado en uno, el PC destino del salto condicional, o el PC destino del salto incondicional.

A continuación revisemos un ejemplo de funcionamiento de la Segmentación para mostrar como ésta funciona en cada una de sus etapas, se considera la instrucción de carga LW, sus cinco etapas son:

Búsqueda de la instrucción: La instrucción se va a leer en memoria utilizando la dirección del PC y después se va a colocar en el registro de segmentación IF/ID. La dirección del PC se incrementa en uno y después se vuelve a cargar en el PC para que sea leída en el siguiente ciclo de reloj. Esta dirección incrementada también se guarda en el registro de segmentación IF/ID.

Decodificación de la instrucción y lectura de registros: El registro IF/ID suministra el campo inmediato de 16 bits que tiene el signo extendido a 32 bits y los números de registro para leer los dos registros. Estos tres valores se almacenan en el registro ID/EX junto con la dirección del PC incrementado.

Ejecución y cálculo de la dirección efectiva: La instrucción de carga lee el contenido del registro y el inmediato de signo extendido del registro de segmentación ID/EX y los suma en la ALU.

Memoria: Se lee la memoria de datos utilizando la dirección del registro de segmentación EX/MEM y se coloca en el registro MEM/WB

Post-escritura: La etapa final de una instrucción de carga, el dato se lee en el registro de segmentación MEM/WB y escrito en los registros en medio del camino de datos.

Este recorrido de las instrucciones de carga muestra que cualquier información necesaria en la etapa siguiente de la segmentación debe ser pasada a esa etapa vía un registro de

segmentación; en otro caso la información se pierde cuando la instrucción siguiente entra en esa etapa de la segmentación.

La primera implementación del MIPS se realizó conforme la arquitectura segmentada que se acaba de describir, sin embargo a lo largo del diseño se encontraron ciertos detalles en la segmentación los cuales requirieron cambios e inclusiones en la arquitectura inicial. A continuación se detalla los nombres de los módulos agregados y se describe los casos que resuelven.

3.2.6 Comparador

En la sección correspondiente a riesgos de control descrita en el capítulo 2 de la arquitectura del MIPS, se vieron dos soluciones, primero, detener todo el pipeline hasta que se haya efectuado la comparación en la ALU, y segundo asumir que el salto no se efectuará y continuar con la ejecución de las instrucciones normalmente, y en caso de que el salto se efectúe, realizar un *flush* para que las instrucciones subsiguientes a la instrucción del salto no se ejecuten.

Esto supone un retraso de tres ciclos de reloj (debido a que la comparación que se realiza en la ALU está en la tercera etapa, la de Ejecución) por cada vez que se tome el salto. Esto parece no representar mayor problema, pero en el caso de un lazo que deba efectuarse por varios cientos de veces, esto representa un aumento considerable en el tiempo de ejecución del conjunto de instrucciones que se encuentran dentro del lazo. Debido a esto se hicieron modificaciones al diseño de tal manera que ese retraso disminuya de tres a un solo ciclo de reloj.

Para ello primero se hizo la modificación de que la comparación no se efectúe en la etapa de ejecución sino más bien en la misma etapa de decodificación, para lo cual se agregó la unidad llamada *comparador*. Este fue el cambio que causó que los multiplexores usados junto con la unidad de anticipación se coloquen ahora en la etapa de decodificación de la instrucción y no en la de ejecución como estaban inicialmente ubicados.

La unidad comparador recibe como entradas el opcode o código de operación de la instrucción y los dos operandos a ser comparados, y genera la salida *salto*. El opcode sirve para saber si el salto condicional es del tipo BEQ, BNEQ, BEZ, etc. Luego

analiza la relación existente entre ambos operandos y si coincide con el tipo de salto dado por el opcode la salida *salto* se coloca a un valor de un 1 lógico.

Por ejemplo, para el caso de un BEQ A, B (salte si A es igual a B) se tiene que el opcode es 000100 entonces se pregunta si el opcode es igual a 000100 y si A es igual a B, si se cumplen estas dos condiciones entonces la unidad genera la salida *salto*, y de igual manera para los demás saltos condicionales que soporta esta arquitectura MIPS.

3.2.7 Multiplicador

Se agregó una unidad de multiplicación para el MIPS, ya que efectuar una multiplicación por medio de algoritmos genera un retardo considerable debido al número de instrucciones que deben ejecutarse para obtener un producto. La unidad implementada es un multiplicador de dos operandos de 16 bits, y una salida de 32 bits sin etapas de pipeline (con lo cual es capaz de realizar multiplicaciones en menos de un ciclo de reloj) generado con el IP Core Generator del ISE 8.2.



CIB-ESPOL

3.2.8 Unidad de anticipación para saltos incondicionales

Otro caso de dependencias de datos que no controla el esquema segmentado del MIPS inicialmente propuesto es el caso de la siguiente secuencia de instrucciones:

```
JAL 0x0FFFFFF
ADD $3,$5,$3
JR  $31
```

La instrucción JAL efectúa el salto incondicional a la dirección indicada en el campo de dirección de la misma instrucción y almacena la dirección de la instrucción siguiente en la dirección 31 del archivo de registros. La instrucción JR salta hacia la dirección dada por el contenido del registro operando. Entonces, en este conjunto de instrucciones se ejecuta la instrucción JAL, se almacena en el registro 31 la dirección de la instrucción ADD, luego se hace un *flush* a la siguiente instrucción y finalmente según lo indica la instrucción JR, debería regresar a ejecutar la instrucción ADD.

Para ello se necesitaría que la instrucción JAL ya haya completado las etapas de la segmentación, sin embargo según la secuencia de instrucciones dada, esta instrucción se encuentra



en la etapa de memoria mientras la instrucción JR se encuentra en la etapa de decodificación donde se lee el registro 31. Es decir, se tiene el problema de que la dirección de la siguiente instrucción aún no está disponible en el registro 31 cuando es requerida por la instrucción JR.

Para resolver esto, se creó la unidad de anticipación para saltos incondicionales, la cual opera de manera similar a la unidad de anticipación para dependencia de datos, revisada en el capítulo del MIPS. Tiene como salida una señal llamada *Mux_JF* que controla un multiplexor que va a seleccionar los diferentes valores que puede tomar la dirección de la instrucción siguiente para la entrada del PC. Así para el ejemplo mostrado, esta unidad pregunta si el registro requerido por la instrucción JR es el registro 31, y si hay una operación JAL que se encuentre en cualquiera de las cuatro primeras etapas de la segmentación. De ser así, la salida de esta unidad envía directamente el contenido del registro 31 o lo que es lo mismo la dirección a la cual debe saltar, desde cualquier etapa de la segmentación al registro PC.

3.2.9 UART

El MIPS incorpora a su arquitectura una UART con una tasa de baudios configurable desde 1200 a 115200 baudios, basada en el estándar RS232.

Las operaciones de recepción y transmisión son realizadas mediante la implementación de dos máquinas de estado que controlan ambos procesos por separado, permitiendo de esta manera que la comunicación sea full dúplex.

Para obtener las diferentes tasas de baudios se hace el siguiente razonamiento:

El tiempo de bit para una tasa de baudios dada, está dado por:

$$T_{bit} = \frac{1}{Baudios}$$

Ya que esta razón de Baudios va a ser implementada con un contador, este tiempo de bit debe ser igual al periodo de reloj del FPGA multiplicado por un valor constante el cual será el valor límite de ese contador, entonces:

$$T_{bit} = \frac{1}{Baudios} = K\frac{1}{f_{CLK}}$$

De donde:

$$K = \frac{f_{CLK}}{Baudios}$$

En la implementación para obtener este valor de K se ha usado dos contadores en cascada, debido a que puede llegar a ser demasiado grande, uno de los cuales es de 4 bits, es decir cuenta de 0 a 15 y el otro con un valor límite llamado *Divisor* que va a depender de la razón de Baudios utilizada.

$$Divisor = \frac{1}{16}\left(\frac{f_{CLK}}{Baudios}\right)$$

Por ejemplo si a frecuencia del FPGA en el cual se va a implementar el MIPS es de 50MHz se tiene que:

$$Divisor = \frac{3125000}{Baudios}$$



CIB-ESPOL

En la tabla X se muestra los valores a los cuales debe llegar el segundo contador para las diferentes tasas de baudios.

A continuación se describe el funcionamiento de las máquinas de estado encargadas de la transmisión y recepción de la UART.

Tabla X. Valores de Divisor para las tasas de Baudios soportadas por la UART implementada

Baudios	Divisor
1200	2603
2400	1301
4800	651
9600	325
19200	162
38400	81
57600	51
115200	27

Describiendo las máquinas de estado utilizadas se tiene que para la transmisión, se empieza con un estado *Idle*, en el cual se espera por la señal de inicio *LD*, la cual indica que existe un dato listo para ser transmitido, y se pasa al siguiente estado que es el estado *Load*. En este estado el dato que se encuentra en la entrada de la UART, se almacena en el búfer de transmisión, junto con el bit de inicio y el bit de paridad que también es calculado. Luego se espera por la señal *TopTx*, la cual indica que ya ha transcurrido el tiempo de bit para la razón de baudios seleccionada. Una vez que llega la señal *TopTx*, se pasa al estado *Shift*. En este estado es donde se efectúa la transmisión propiamente dicha. Cada vez que ha transcurrido un tiempo de bit, se efectúa un desplazamiento hacia la derecha del dato que se encuentra en el búfer de transmisión por la salida Tx, hasta que el búfer quede vacío, es decir, hasta que se haya hecho tantos desplazamientos como número de bits tenga el búfer. Una

vez que han transmitido todos los bits, se pasa al estado Stop el cual es solo un estado de transición hacia el estado *Idle*.

Mientras se esté transmitiendo permanece encendida la señal *TxBusy* indicando que el transmisor se encuentra ocupado.

Para la recepción, se inicia al igual que sucede con la transmisión, con un estado *Idle*, en el cual se espera por el bit de inicio, una vez que este llega se pasa al estado *Start*, en el cual se verifica si en realidad se trata del bit de inicio y no únicamente de una espuria, de ser así se pasa al estado error y se vuelve a esperar por un nuevo bit de inicio, de lo contrario el siguiente será el estado *Edge*, que es un estado de transición que dura medio tiempo de bit para hacer el muestreo de cada bit recibido en el estado *Shift*. Esto se hace ya que cada muestreo debe hacerse en la mitad de cada bit recibido y no al inicio ni al final del mismo, para estar seguros de que se trata de un bit y no de un glitch. Al final del estado *Shift*, cada nuevo bit muestreado se ingresa al búfer de recepción el cual es un registro de corrimiento que va realizando un desplazamiento a la derecha conforme van ingresan los bits. Finalmente se pasa al estado *Stop* donde se



CIB-ESPOL

activa la señal *RXRdy* la cual indica que existe un nuevo dato presente en el registro o búfer, es decir a la salida de la UART.

Para la programación de la UART, se tiene una instrucción con Opcode "100110". Los tres bits menos significativos de esta instrucción seleccionan la velocidad en baudios, y el bit tres y el bit cuatro seleccionan la paridad. La tabla XI muestra las combinaciones de estos 5 bits para seleccionar la paridad y los baudios.

Tabla XI. Selección de la paridad y baudios de acuerdo a los bits de la instrucción de configuración de la UART

BITS de la instrucción de configuración UART	**Velocidad en Baudios**
B2 B1 B0	
000	115.200
001	57.600
010	38.400
011	19.200
100	9.600
101	4.800
110	2.400
111	1.200
B4 B3	**Paridad**
00	Par
01	Impar
1X	Sin paridad

Estos bits pasan a un registro temporal que tiene como habilitador la señal *en_reg5* generada por la unidad de control

cuando detecta esta instrucción de configuración, luego de un pulso de reloj; estos pasan a las entradas de la UART.

3.2.10 Unidad de programación externa del MIPS

La memoria del procesador MIPS puede estar inicializada con un conjunto de instrucciones que realicen un programa específico o puede programarse serialmente por medio de una unidad externa al MIPS. Esto se realiza con la ayuda de la UART y una máquina de estados.



CIB-ESPOL

Cuando se está en modo de programación esta máquina de estados envía la señal *RST_MIPS* al procesador que hace que este permanezca reseteado, es decir, que no realice ninguna operación mientras se está escribiendo un nuevo programa en la memoria de instrucciones. La máquina de estados obedece al siguiente formato de ingreso de las instrucciones, el orden en el que deben ingresar los caracteres es el siguiente:

I, que indica el inicio de una secuencia de instrucciones. D, indica el inicio del campo de la instrucción, luego se espera por el ingreso de 8 caracteres de datos que representan la

instrucción en hexadecimal. A, inicial el campo de dirección donde se grabará la instrucción, luego 3 caracteres que representan la dirección en hexadecimal. S, para indicar que ya se puede grabar la instrucción en la dirección indicada y F, que indica el fin del conjunto de instrucciones a grabarse en memoria.

Una secuencia de instrucciones se vería de la siguiente manera:

ID0231FFFB**A**032**SD**04000002**A**052**S**...**F**

Donde se tienen dos instrucciones: la primera instrucción es la palabra "0231FFFB" y se va a almacenar en la dirección "032", la siguiente instrucción es "04000002" y su dirección "052".

Una vez que se ingresa el carácter "F" el MIPS pasa del modo de programación al modo de funcionamiento normal.

El funcionamiento de la máquina de estados es básicamente el de un detector de secuencia, el cual debe ir validando cada uno de los caracteres que forman parte del formato de entrada de las instrucciones visto anteriormente. Cada transición de

estados no se realiza por cada ciclo de reloj sino cada vez que se recibe un nuevo caracter en la UART.

El estado inicial es el estado *Ini* en el cual se espera por el ingreso del caracter 'I', que indica el inicio de la secuencia de instrucciones. Una vez que ha llegado el caracter 'I' se pasa al estado *Data* en el cual se espera por el carácter 'D'. En los siguientes 8 estados *DH0, DH1,...,DH7*, se recibe los 8 caracteres que representan los 8 dígitos hexadecimales de la instrucción, en estos estados se verifica si estos datos ingresados corresponden a los códigos ASCII del '0' al '9' y de la 'A' hasta la 'F', estas representaciones ASCII se transforman en dígitos hexadecimales de 4 bits, por ejemplo el código ASCII de la 'A' es el 41 en hexadecimal y es transformado a la cadena de bits "1010". Estos dígitos ingresan en un registro de 32 bits, los cuales representan la instrucción. En el estado *Addr* se espera por el caracter 'A' y en los siguientes tres estados *AH0, AH1* y *AH2* se almacena la dirección en un registro de 10 bits, no sin antes realizar una verificación similar a la que se realizó con los caracteres que representan la instrucción. Luego se pasa al estado *Save* en el cual se envía la dirección y la instrucción almacenadas a las entradas del MIPS, *Dir_Instr* e

Inst_IN respectivamente, además de una señal de control de escritura en memoria *WE (write enable)*.

Estas señales son recibidas por la memoria de instrucciones y a la vez se pregunta si se desea ingresar una nueva instrucción (mediante el caracter 'D') o si ya es el fin de la secuencia de instrucciones a ser almacenadas (mediante el caracter 'F'). Si se va a almacenar una nueva instrucción se regresa al estado *Data*, mientras que si es el fin de la secuencia de instrucciones se avanza al estado *Fin*, el cual es solo un estado de transición hacia el estado *Ini*.

Si en cualquiera de los estados se recibe un caracter que no es el esperado, o si en los estados DH0 a DH7 o AH0 a AH2 se reciben caracteres que no representen dígitos hexadecimales, se pasa al estado *Error*, sin almacenar ninguna instrucción.

A continuación en la figura 3.9 se muestra el esquema implementado para programar el MIPS, donde se observa la UART y la máquina de estados anteriormente descritos.

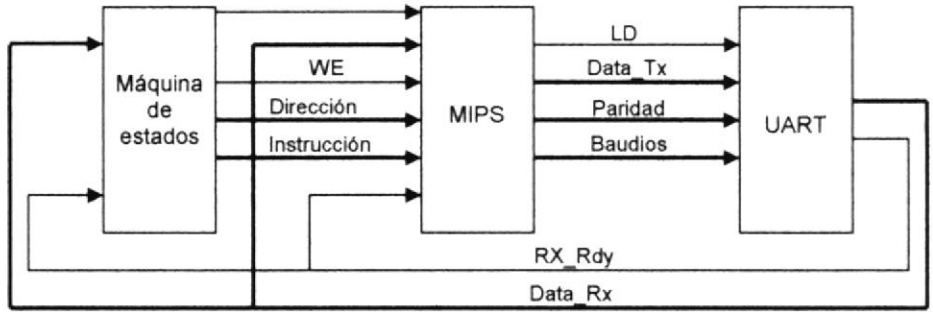


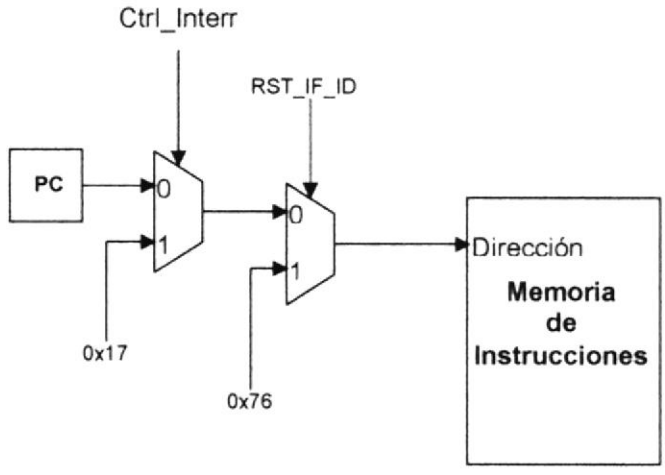
Figura 3.9 Esquema implementado para programar el MIPS

3.2.11 Manejo de interrupciones

El MIPS tiene una interrupción que se origina cada vez que estando en modo de funcionamiento normal (cuando no se está programando) se recibe un dato por medio de la UART. Entonces, la interrupción se presenta cuando la UART genera la señal *RXRdy*.

El manejo que se le da a esta interrupción es el siguiente:

La señal *RXRdy* ingresa a la Unidad de Control del MIPS, la cual genera la señal *Cntrl_Interr* que actúa como selectora de un multiplexor como se muestra en la figura 3.10



CIB-ESPOL

Figura 3.10 Selección de la dirección de la rutina de interrupción.

Como se puede observar en la figura 3.8 cuando la señal *CNTrl_Interr* es un cero lógico se selecciona el valor dado por el PC, caso contrario se selecciona una dirección fija en la cual se encuentra la primera instrucción de la sub-rutina de interrupción.

La primera instrucción de la sub-rutina de interrupción es una instrucción JAL, que guarda la dirección de la siguiente instrucción a ejecutarse. Para este caso se necesita que se almacene la dirección de la instrucción que se iba a ejecutar en el momento que llegó la interrupción y no la dirección de la instrucción subsiguiente al JAL.

Se observa en la figura 3.11 los componentes que intervienen en el manejo de una interrupción. Cuando esta ocurre suceden las siguientes dos acciones:

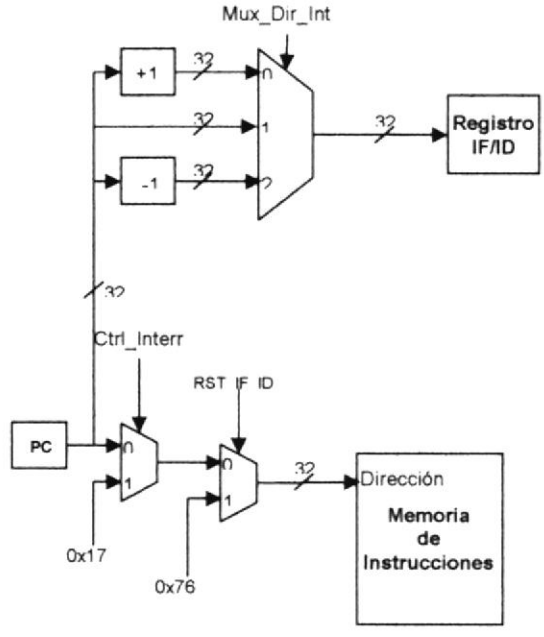


Figura 3.11 Esquema de elección del PC cuando ocurre una interrupción

1. La señal *CNT_Interr*, selecciona la dirección 0x17 debido a que ahí se encuentra la primera instrucción de la sub-rutina de interrupción.

2. El multiplexor MUX_DIR_INT selecciona el valor de PC para que sea almacenado en el registro IF/ID.

Con estas dos acciones descritas, se consigue que se ejecute la instrucción JAL, pero la dirección que se encuentra en el registro IF/ID es la de la instrucción que se iba a ejecutar en el momento de la interrupción.

Después de esto hace pasar el valor de PC por todo el esquema segmentado y al final se llega a un multiplexor en el cual por medio de la señal de control *MEM/WB_JAL* se selecciona este valor y se coloca el mismo como dato de entrada al banco de registros, como se muestra en la figura 3.12.

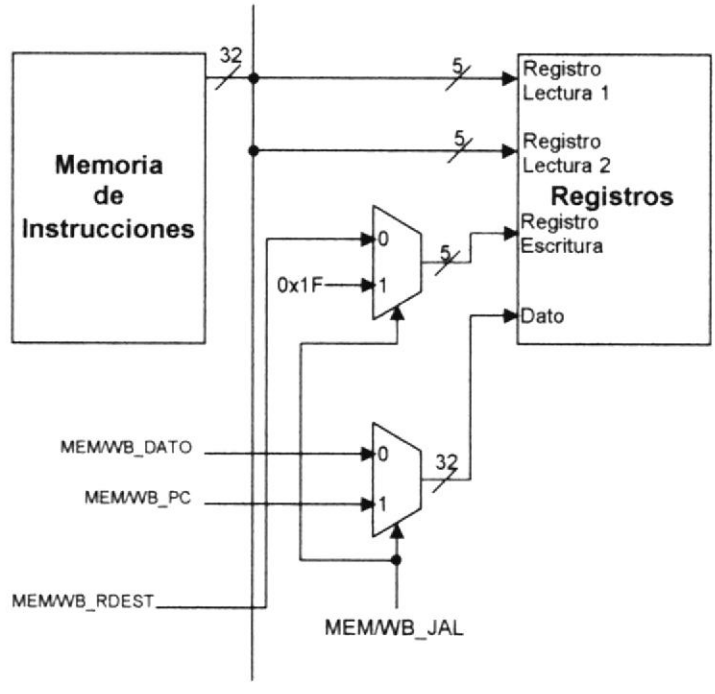


Figura 3.12 Esquema de selección de dato de entrada al banco de registros cuando ocurre interrupción

Entonces, al ejecutar la instrucción JR $31, al final de una subrutina de interrupción, se regresa el PC a la dirección de la instrucción que iba a ejecutarse en el momento en que ocurrió la interrupción, la cual fue almacenada previamente en el banco de registros.

Dentro de esta subrutina de interrupción intervienen dos operaciones de MIPS que son para uso con coprocesadores, que no se han descrito anteriormente, estas son las instrucciones SWC2 y LWC2. La primera de estas instrucciones se encarga de almacenar el dato que se encuentra en el búfer de recepción en la memoria de datos, mientras que la segunda de ellas será usada para enviar un dato de la memoria al búfer de transmisión.

Con la instrucción de recepción, el dato presente en el búfer de recepción, primero se concatena con otros 24 bits debido a que solo tiene 8 bits y la entrada de datos de la memoria es de 32 bits, luego se coloca en un multiplexor cuya entrada selectora es la señal *cdm* la cual avanza junto con esta instrucción a través de todo el esquema segmentado seleccionando este dato recibido como entrada a la memoria de datos.

La instrucción de transmisión obtiene un dato de memoria de 32 bits y toma solo los últimos 8 de estos para enviarlos por la UART, generando además la señal LD la cual avanza por todo el esquema segmentado y sale junto con el dato hacia la UART, teniendo entonces todas las entradas necesarias para iniciar la transmisión.

3.3 Desarrollo en C de un canal de comunicación inalámbrica con codificación de canal

3.3.1 Descripción del programa

El programa desarrollado en C obedece a una simulación de un canal de comunicaciones parametrizable que se divide en cinco partes básicas: Fuente de los datos, codificación, transmisión y recepción BPSK en un canal Gaussiano, decodificación Viterbi, Cálculo de BER. Las líneas de códigos del programa principal y las librerías se encuentran en el CD de soporte adjunto a esta tesis.



CIB-ESPOL

Cabe indicar que a través de la ejecución del programa se van guardando diferentes archivos para su posterior depuración o comparación. El programa usa una librería adicional de funciones que son invocadas en la función principal y demás rutinas para simplicidad y ahorro de líneas de código del mismo.

Al ejecutar el programa el usuario debe ingresar los diferentes requerimientos que son necesarios para la correcta ejecución de la simulación. Dentro esos parámetros que se piden están:

1. Si se desea guardar archivos de depuración.
2. Número de veces que desea ejecutar el algoritmo.
3. La longitud restrictiva (K) donde se computa la cantidad de estados que tendrá el decodificador.
4. El tamaño del flujo de datos de entrada que deseamos transmitir.
5. La cantidad de memorias de TB.
6. La fuente de los datos de entrada que puede ser una secuencia aleatoria, una entrada de un archivo en forma binaria o una secuencia de ceros.
7. La tasa de codificación (r).

8. Los polinomios generadores que dependen de k y de n para computar la cantidad y largo de los vectores generadores y deben ser ingresados en forma decimal.
9. El número de bits de cuantización del ecualizador.
10. Los valores de la señalización binaria antipodal normalmente -1 y 1.

Una vez ingresado estos parámetros se procede a codificar los datos de entrada con los polinomios generadores creando un nuevo arreglo de datos *r* veces más grande que la secuencia de entrada. Estos a su vez serán convertidos en señales antipodales (1,-1) para la transmisión en modulación BPSK.

Una vez transformados a los nuevos símbolos se procede a introducirles ruido blanco gaussiano por cada símbolo transmitido en un barrido de -5dB a 10dB. En la recepción se recibe toda la secuencia enviada por el canal por cada transmisión hecha con un diferente nivel SNR. Es decir se va obtener al final del programa, 16 secuencias recibidas donde por cada una se realiza la cuantificación, la decodificación de Viterbi y el análisis de BER.

En la cuantificación se hace una ponderación de los valores recibidos en la transmisión dependiendo del rango trazado por el parámetro de bits de cuantificación, en nuestro caso este valor es 3 lo que traza un rango de 8 posibles valores de 000 a 111 en forma binaria que formará la nueva secuencia de entrada de la función del decodificador. El valor de "011" representa el valor con mayor probabilidad de haber sido '0' y "100" la mayor probabilidad de haber sido '1'. El valor con la menor probabilidad de haber sido '0' es "000" y el de menor probabilidad de haber sido '1' es "111".

En el proceso de decodificación se toma los datos ya cuantificados y los parámetros necesarios para el desarrollo del algoritmo tales como tasa de codificación, polinomios generadores, longitud restrictiva, tamaño de los datos, número de memorias de TB y los bits de cuantificación.

Primeramente se calcula las métricas de rama esperadas y sus respectivos valores cuadrados usando los polinomios generadores que serán utilizados posteriormente para la distancia euclidiana. Después se inicializa los primeros valores

de las métricas de nodo con cero en el primer valor y un valor muy grande las demás.

La función de ACS es el siguiente cálculo realizado tomando como parámetros la secuencia cuantificada de entrada, las métricas de ramas y de nodos, *r, K* y el número de estados. Entrega los vectores de decisión y el valor de las métricas de nodos acumulada hasta ese momento de la operación y el estado donde se dio la menor métrica. En esta operación se calcula distancia euclidiana entre el valor esperado y el valor recibido. El término cuadrado del valor recibido no se toma en cuenta en la operación ya que este es un factor común entre las dos ramas que llegan al mismo estado por ende en la comparación se anulan logrando así evitar un cálculo más en el computo. Los vectores decisión toman un valor de cero cuando la rama superior es menor que la rama inferior que llega a un mismo estado y uno si es lo contrario. Esta operación se repite para todos los estados del Trellis. La última operación es determinar el estado de menor métrica en el instante evaluado.

Una vez calculado un vector decisión pasa a ser almacenado en la memoria de vectores sobrevivientes respectiva,

dependiendo cuantas de estas memorias fueron declaradas al comienzo de la ejecución. Si se utiliza más de una memoria cuando se llena la primera se activa una bandera que indica que la memoria 1 ya fue llenada, dando lugar para que empiece a llenar la siguiente memoria. Cuando por lo menos dos memorias está llenas, se procede a hacer el primer TB de convergencia y de decodificación y así obtener los primeros bits decodificados, es decir, a la memoria dos se le hace TB de convergencia para obtener el estado inicial más probable para el siguiente TB de decodificación en la memoria 1, dando paso a los primeros bits decodificados cuyo tamaño depende de cómo fue divida la memoria según el tamaño de los datos de entrada y el parámetro de números de memorias.

Las siguientes operaciones de TB se dan en la misma forma, como la memoria 1 ya fue utilizada, los siguiente vectores de decisión entrantes serán almacenados en esta misma memoria. Una vez llena, en esta memoria se efectúa el TB de convergencia debido a que la memoria 2 ya está lista para el TB de decodificación. Y así sucesivamente se irá repitiendo este procedimiento dependiendo de cuantas memorias fueron requeridas al comienzo.

Al final como el último tramo de datos no tiene una memoria posterior de donde hacer TB de convergencia, se utiliza el estado menor métrica calculado en la función de ACS, para tomarlo como estado de partida inicial del TB de decodificación y de esta forma obtenemos los últimos bits decodificados.

Cuando tenemos toda la secuencia decodificada se compara bit por bit con la secuencia original de entrada para tener la probabilidad de error de esa secuencia a un determinado nivel de SNR y se va guardando en un archivo que nos servirá posteriormente para hacer curvas de BER para diferentes parámetros de codificación.

3.3.2 Obtención de las curvas de BER en Matlab para diferentes parámetros

Para comprender mejor el comportamiento de algoritmo de Viterbi y de las capacidades de corrección según sus características, se obtuvieron varias curvas de probabilidad de error para analizar y comprobar las diferentes ventajas y desventajas tanto en potencia y en complejidad que se obtienen cuando se utiliza codificación de canal.

La figura 3.13 nos muestra para una secuencia de entrada de 1 millón de datos, un decodificador de Viterbi con punto flotante, tasa de codificación 1/2 para diferentes longitudes restrictivas desde K= 3 hasta K=8. La línea punteada indica una transmisión sin codificación de canal.

Entre las observaciones se puede destacar la ganancia de probabilidad de error de una señal transmitida sin codificar con respecto a las codificadas para un rango de 0dB a 10dB. También se puede ver cómo a medida que se aumenta K, la curva se compacta más logrando mejor probabilidad de error.

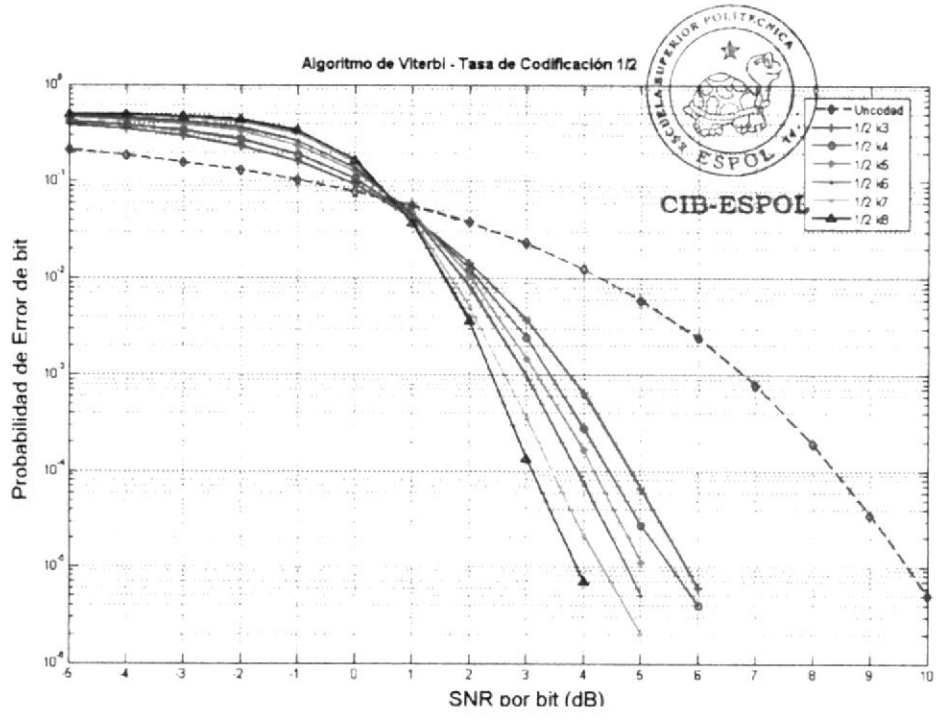


Figura 3.13 Decodificador Viterbi, r = 1/2

En la figura 3.14 se tiene una tasa de codificación de 1/3 pero con las mismas características de datos y longitud restrictiva. Se puede apreciar una ganancia de probabilidad de error de bit con respecto a la anterior de tasa 1/2 para cada k a un SNR dado.

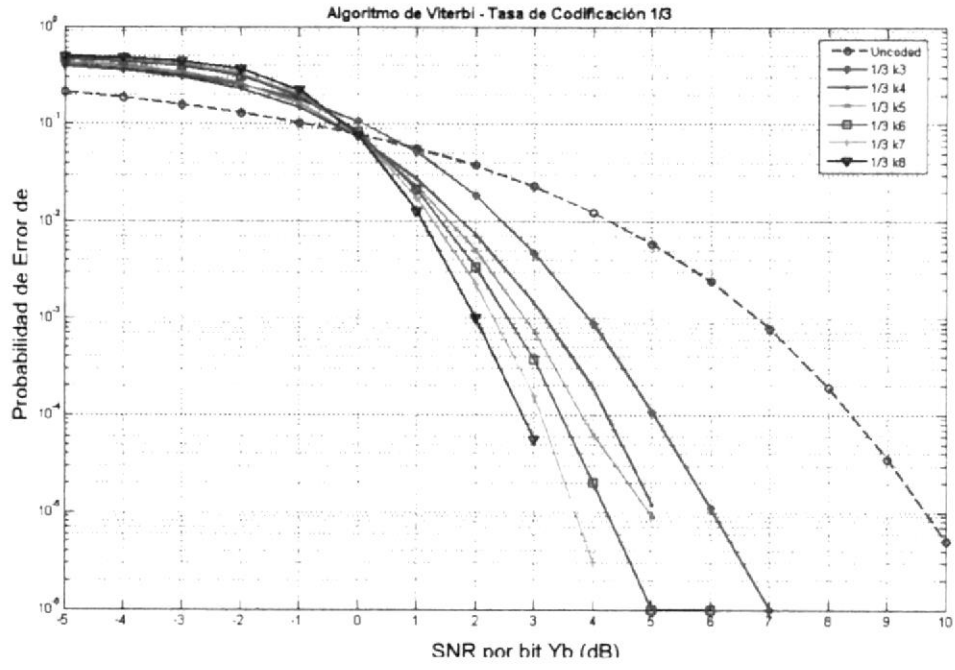


Figura 3.14 Decodificador Viterbi, r = 1/3

Así mismo para una tasa de 1/4 se obtienen un margen de ganancia con respecto a las anteriores tal como lo muestra la figura 3.15. Si se continúa evaluando para más tasas de codificación, la diferencia de ganancia con respecto a los otros se va haciendo muy pequeña, razón por la cual no es necesario

seguir mostrando más gráficos porque no se justifica el nivel de dificultad a la ganancia obtenida.

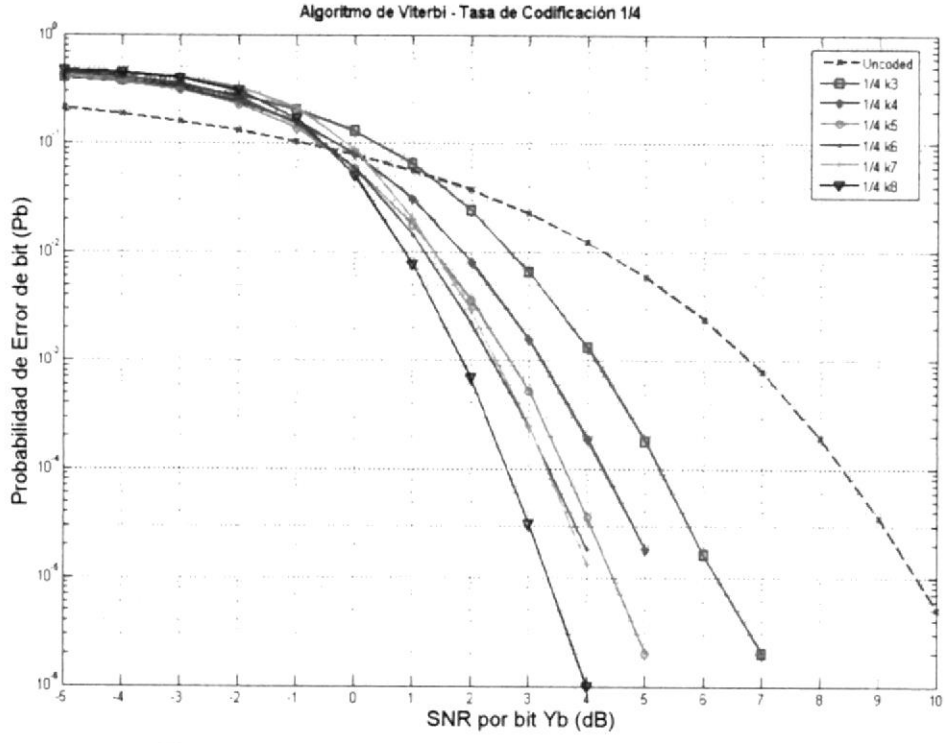


Figura 3.15 Decodificador Viterbi, r = 1/4

La relación de ganancia de ruido con la cantidad de bits de cuantificación fue otro análisis importante que se hizo usando este programa. Si se compara diversas decodificaciones para 1,2,3,4,5,6,7 bits de cuantificación (Q=2,4,8,16,32,64,128) se observa que en la figura 3.16 se logra la peor relación de BER cuando solo usamos 1 bit de cuantificación.

Como se ve en la figura 3.16 a medida que aumenta el nivel de cuantificación la curva se cierra más, obteniendo mejor

ganancia hasta una diferencia deseable de 0,5dB con respecto a la curva de referencia. A partir de Q=32 casi todas las curvas convergen al mismo trazado obteniendo cada vez menos margen de ganancia favorable, con respecto a la complejidad de diseño y de espacio que involucra hacer una implementación con mayor índice de cuantificación.

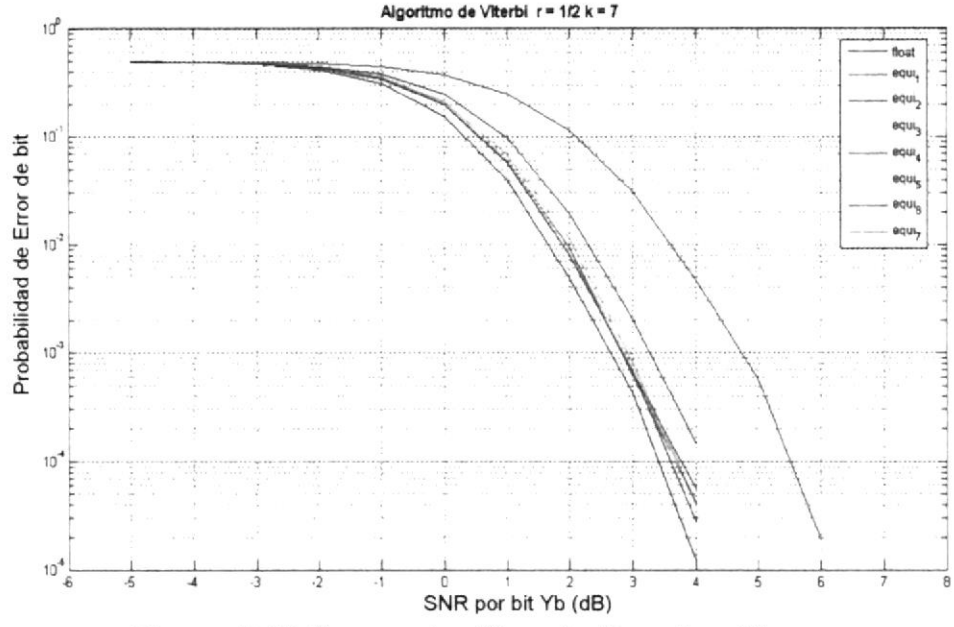


Figura 3.16 Curvas de diferente Cuantización

Entre las consideraciones de la implementación de hardware y software fue comprobar si era estrictamente necesario mantener la función de multiplicación para el cálculo de distancia euclidiana, o dejar expresada la fórmula de distancia euclidiana sólo en términos de sumatorias con la intención de ahorrar espacio y tiempo en la implementación, sin reducir

ESCUELA SUPERIOR POLITECNICA ESPOL
CIB-ESPOL

notablemente el desempeño o ganancia de probabilidad de error.

Implementando las sumatorias, se observa en la figura 3.17 que solo las cuantificaciones de 1 y 2 bits son las más afectadas, el resto de curvas permanecen ligeramente distantes de las curvas de la figura anterior 3.16.

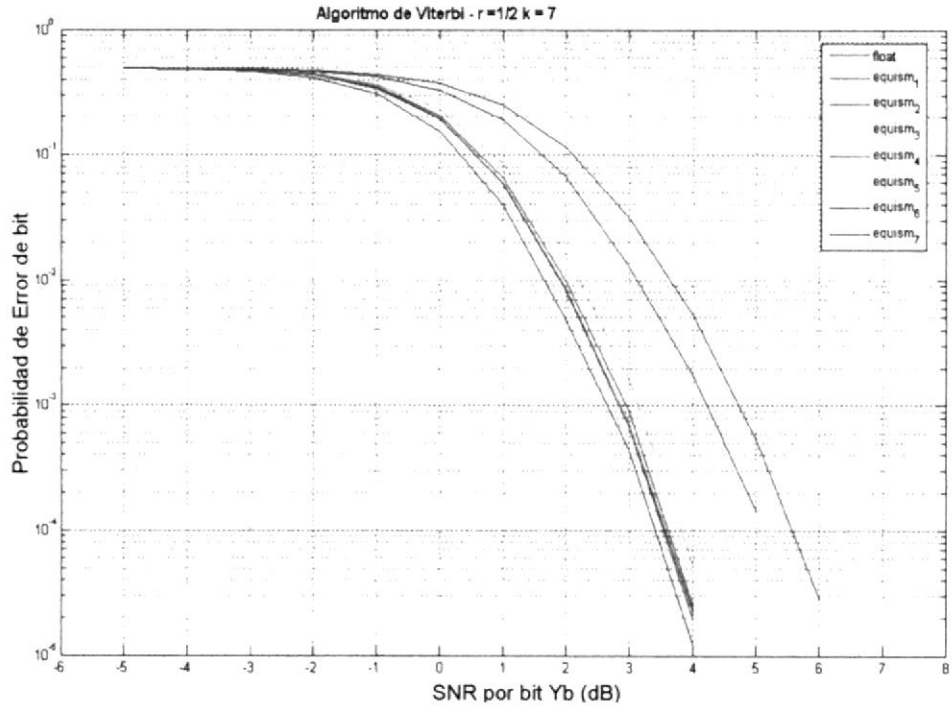


Figura 3.17 Curvas de diferente Cuantización. Sin multiplicador

Un factor importante que ayuda a obtener una mejor probabilidad de error es el tamaño de la longitud del Trellis. Mientras mayor es esta longitud más parecida será a la curva

de referencia ideal (tamaño total de los datos de entrada) y por ende realizará un mejor desempeño.

La figura 3.18 muestra el comportamiento de las curvas por el hecho de dividir todo el flujo de datos en paquetes de 256, 512, 1024, 4096, 16384 bits, logrando un mejor rendimiento a medida que crece el tamaño. Por ejemplo el de mayor tamaño fue el paquete de 16384, el de 256 bits, aunque es el que menos ganancia de error posee con respecto a los demás, sigue teniendo una ganancia aceptable.

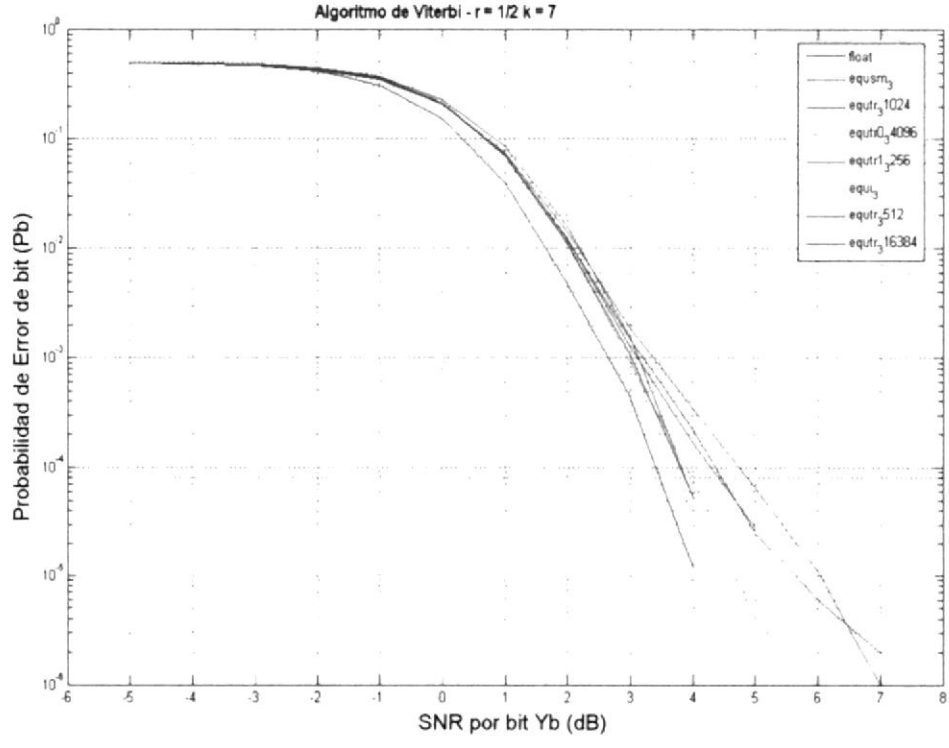


Figura 3.18 Decodificador de Viterbi con tramas variables

3.4 Implementación del decodificador Viterbi en el procesador MIPS

3.4.1 Traducción del algoritmo en C a lenguaje máquina

Existe una amplia gama de programas para el desarrollo de sistemas de procesadores con arquitectura MIPS que incluye herramientas y librerías previsto a acelerar el desarrollo de aplicaciones de alto rendimiento y calidad sobre estos núcleos. Uno de los programas líderes en este campo es el SDE que está construido bajo herramientas de GNU que incluye librerías de C y C++ así como también interfaces convenientes.

La versión de Windows que se usa para este proyecto esta construido bajo una ventana de ambiente de programación Linux llamada Cygwin, distribuida bajo los términos GNU. Cygwin es una herramienta basada en línea de comando que puede ser ejecutada desde una consola Windows, y una vez instalada, se puede correr sobre esta plataforma los diferentes componentes de SDE, utilizando los comandos propios de este sistema.

El SDE-lite fue usado en el proyecto para traducir las líneas de código en C de la función del decodificador Viterbi a instrucciones de MIPS y lenguaje de máquina.

La tabla XII muestra la traducción de la rutina en C de la operación de TB a código ensamblador y de máquina obtenida de la herramienta de compilación gráfica del SDE-*Lite*.

Tabla XII. Instrucciones MIPS del código TB del decodificador Viterbi

330	lw	t5,1153(sp) (carga la direcc base de seq cod)	8fad0481,
331	move	t7,a0 (base del decision windows respectivo)	00807821,
332	li	t0,63 (for i=63)	2408003f,
333	li	t6,1 (para if ==1)	240e0001,
334	sll	t9,a1,0x6 para obtener 40,80,C0...FC0 indice de decwind	0005c980,
335	addu	t4,t9,t0 +i donde calculo donde empieza el TB	03286021,
336	addu	t2,t4,t7 la suma con la base de decicsionwindows	018f5021,
337	addu	v1,a3,t0 starting_address + i a3=i-convergenlenght	00e81821,
338	lw	a0,0(t2)	8d440000,
339	sll	t2,a1,0x1 (initial_state << 1)	00055040,
33a	srl	t8,a1,0x5 (initial_state >> 5)	0005c142,
33b	addiu	t1,t2,1 ((initial_state << 1) + 1)	25490001,
33c	addu	v1,v1,t5 (calc. para base de seq decodificada)	006d1821,
33d	addiu	t4,t0,-1 disminuye for	250cffff,
33e	andi	t3,t8,0x1 (initial_state >> 5) & 1	330b0001,
33f	andi	a1,t1,0x3f ((initial_state << 1) + 1) & 63	3125003f,
340	bne	a0,t6,0x344 <TB+20> if(dec_window[initial_state][i] = 1)	148e0003,
341	move	v0,v1	00601021,
342	b	0x346<TB+22>	10000003,
343	sw	t3,-63(v1) guarda la secuencia decodificada	ac6bffc1,
344	andi	a1,t2,0x3f (initial_state << 1) & 63	3145003f,
345	sw	t3,-63(v0) guarda la secuencia decodificada	ac4bffc1,
346	sw	a1,0(a2) guardo el small state en memoria para el sig TB	acc50000,
347	bgez	t4,0x334 <TB+4> for k>=0	0581ffec,
348	move	t0,t4 retorno de índice	01804021,
349	jr	ra	03e00008,
34a	nop		00000000,

Gracias a esta herramienta de compilación es posible escribir cualquier programa en C y obtener las instrucciones en ensamblador para MIPS como también su respectivo código de máquina.

3.4.2 Descripción de la Implementación

Para efecto de optimización del tiempo de operación del proceso total y espacio en memoria se escogió cuidadosamente las líneas de código que representan el algoritmo de decodificación de Viterbi que representen la menor cantidad de instrucciones de máquina posibles.

El programa principal del archivo de C consta de tres partes esenciales la declaración de variables y constantes, la adquisición y la entrega de datos a través de la UART y la operación principal del algoritmo de Viterbi.

Se hizo el seguimiento de las instrucciones obtenidas para saber como fueron traducidas las operaciones y asignaciones en C. Es por eso que la mayoría de instrucciones del algoritmo posee una explicación breve a que operación corresponde en la

ejecución del programa, tal como se lo puede observar en la traducción completa del código que se lo puede encontrar en el CD de soporte.

Todas las instrucciones que generó este código fueron almacenadas en la memoria de instrucciones del MIPS (1 kB).

La primeras 16 localidades de memoria están reservadas para la declaración de variables y constantes globales que serán utilizadas a lo largo del programa. Los siguientes 576 localidades están reservadas para las constantes de métricas iniciales de nodo y los valores esperados de las métricas de rama con su respectivo valor cuadrado. Las siguientes 16 localidades están reservadas para una rutina de lazo infinito que se ejecuta mientras no este realizando la rutina de decodificación y está esperando la entrada de datos.

Cuando existe una de entrada de datos al procesador, se genera una interrupción, que hace que este salte a otro espacio reservado de memoria, donde se encuentra una rutina que se encarga de almacenar la información de entrada en la memoria de datos y de llevar la cuenta de datos ingresados hasta

completar los 512 símbolos. Una vez completados los datos necesarios, esta rutina salta a la función principal de decodificación.

Al inicio de la rutina principal se añadió una pequeña operación extra de conversión, debido a que como los datos ingresados por el puerto serial corresponden a caracteres ASCII, este los convierta a sus respectivos valores equivalentes de cuantificación válidos para el cálculo de las métricas de ramas y de nodos. Esto son valores enteros de -4 a 3, como se explicó en detalle en la implementación del hardware del decodificador.

Existen dos rangos de 4KB de memoria para guardar 64 vectores de decisión de 64 localidades. Cada bit de decisión es guardado cada 64 localidades en la memoria, es decir, para el primer bit de decisión del vector 1 se ubica en la localidad 0, el segundo bit del primer vector en la localidad 64, el tercero en la localidad 128 y así sucesivamente, el primer bit del segundo vector lo hará empezando en la localidad 1 y sus bits de decisión siguientes cada 64 localidades o sea en 1, 65, 129..., por ende el último vector guardará su primer bit de decisión en la localidad 63 y el resto de bits cada 64.



CIB-ESPOL

El motivo de esta asignación es la solución que da el compilador para simular un arreglo bidimensional de enteros y poder recorrer y obtener más fácilmente la información de la memoria.

Una vez que se llenan las dos primeras memorias con los 128 vectores de decisión se puede empezar a realizar el TB de convergencia en la segunda memoria y el de decodificación en la primera. Para esto se invoca a la subrutina de rastreo hacia atrás (TB) que se encuentra en una localidad de memoria previamente asignada.

Los siguientes 64 vectores de decisión pasan a ser almacenados en la memoria 1 que ya fue utilizada para el primer TB de decodificación. Una vez llena, se procede ahora a hacer TB de convergencia en la primera memoria para luego hacer un TB de decodificación en la segunda memoria. Como ahora los datos de la segunda memoria ya fueron utilizados, los últimos 64 bits entrantes se llenan en esta memoria.

Ahora la segunda memoria se le realiza un TB de convergencia para que la primera memoria pueda hacer un TB de

decodificación de los penúltimos 64 bits decodificados. Para la decodificación de los últimos 64 bits no se invoca un TB de convergencia sino que se ejecuta una rutina de búsqueda del nodo con la menor métrica; para que este sea el estado de partida para el último TB de decodificación. El proceso puede observarse en la figura 3.19.

Cuando están listos los 256 bits decodificados entra a ejecutarse la rutina de transmisión de datos por la UART a la PC. Una vez que son transmitidos todos los bits, la rutina salta a la localidad 0 de la memoria de instrucciones para ejecutar de nuevo otro proceso.

Unos de los retos de la implementación fue programar de la manera más eficiente posible para que al traducir el código a instrucciones en ensamblador, no aparezcan instrucciones de una versión superior que provoque cambiar la estructura básica del MIPS o tratar de buscar un funcionamiento equivalente con las instrucciones ya conocidas.

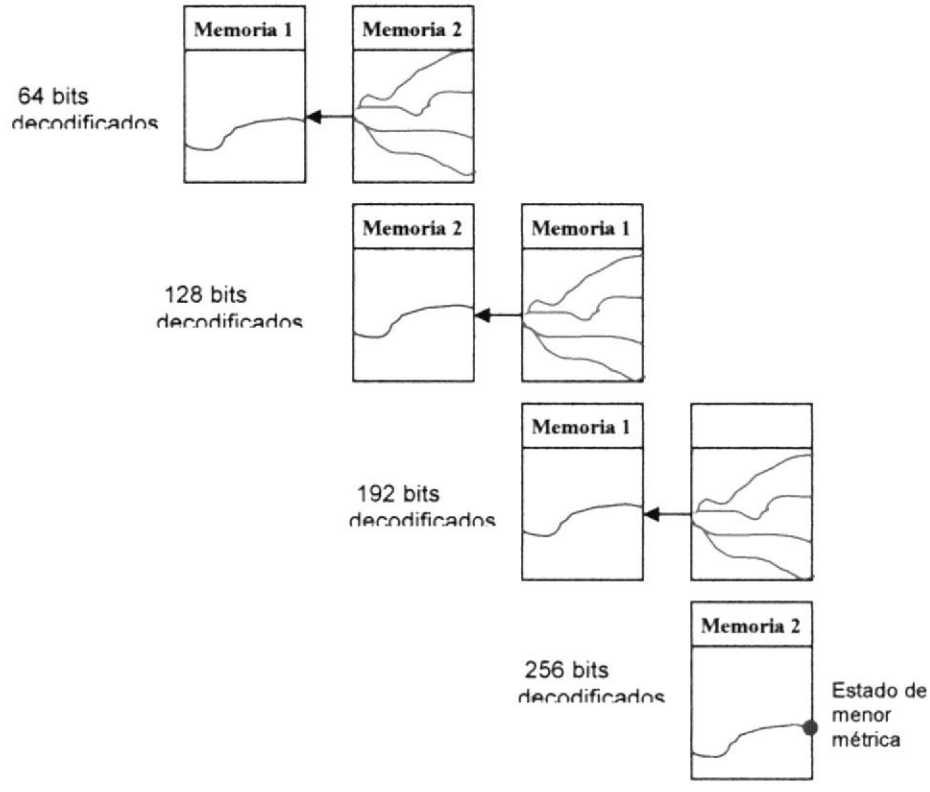


Figura 3.19 Rutina de TB sobre el procesador MIPS

En la traducción al código de máquina se realizaron algunos cambios en los valores inmediatos de la instrucción debido al manejo de una distribución diferente de localidades de memoria tanto para la de datos como para la de instrucciones. Es decir en la compilación de una estructura estándar de MIPS se asume que se usa una memoria de 8 bits por localidad tanto para los datos como para las instrucciones produciendo instrucciones con cálculos de direccionamiento inmediato múltiplo de 4 lo cual no serviría para nuestra implementación que maneja localidades de 32 bits.

También se hizo cambios a las direcciones de memoria de todo el programa, las que se adecuaron convenientemente a nuestras necesidades de espacio para las dos memorias SRAM que usa el MIPS.

3.4.3 Cama de Prueba

La cama de prueba para la implementación del decodificador Viterbi en software tiene el mismo esquema y funcionamiento que la cama de prueba implementada para el decodificador en hardware.

CAPÍTULO 4

ANÁLISIS Y RESULTADOS

4.1 Resultados de la implementación en SW

A continuación se muestra un análisis comparativo de los resultados de síntesis del MIPS sobre tres familias diferentes de FPGAs en tres herramientas de síntesis tales como: el ISE 7.1 y 8.2i de Xilinx, QuartusII de Altera, y Mentor Graphics. Analizando consumo de área del FPGA y máxima frecuencia de operación del diseño.

Se presentan tablas que resumen los resultados de área que se mide en la cantidad de LUTs, I/Os y BRAM utilizados. Un LUT es un bloque básico de un FPGA donde se implementan funciones combinatoriales. Los I/Os proveen una interfaz bidireccional entre un pin de entrada/salida y la lógica interna del FPGA. Dentro de un I/O existen básicamente tres caminos que pueden ser recorridos por una señal determinada: el camino de entrada, de salida y el de tres estados. Los BRAMs o bloques RAM están organizados como bloques de 18kbits síncronos y configurables. Si se desea crear memorias de mayor capacidad se puede cambiar el ancho de los

datos o el tamaño de la memoria, mediante la conexión en cascada o en paralelo de estos bloques [19].

A partir de estos resultados, se elige aquel FPGA que presente mejor resultado en cuanto a frecuencia para realizar una estimación de potencia, simulaciones y finalmente la implementación de la cama de prueba del decodificador Viterbi en el MIPS.

4.1.1 Reporte de síntesis del MIPS

El propósito fundamental de analizar la síntesis del MIPS para varias familias de FPGAs es investigar cuanto sería su máxima velocidad de operación, es decir, la cantidad de instrucciones que puede ejecutar por segundo.

Las familias de FPGAs de Xilinx son el Spartan 3 y Spartan 3E y de Altera el Cyclone II.

La tabla XIII muestra el resultado de síntesis para las diferentes herramientas de síntesis en las tres familias de FPGAs utilizadas. Aunque el factor espacio dado por la cantidad de LUTs utilizados, es importante, el factor tiempo es



CIB-ESPOL

Tabla XIII. Resultado de síntesis para el procesador MIPS

Herramientas de Síntesis	***Quartus***			***ISE***			***Precision Synthesis***		
FPGA	**Recursos**		**Máxima Frecuencia**	**Recursos**		**Máxima Frecuencia**	**Recursos**		**Máxima Frecuencia**
	LUTs	**I/O**	**MHz**	**LUTs**	**I/O**	**MHz**	**LUTs**	**I/O**	**MHz**
Cyclone II **EP2C35F672C6**	68%	44%	64,41	-	-	-			
Spartan3 **xs3S200FT256**	-	-	-	30%	80%	270.64	28.52%	79.77%	247.709
Spartan3E **xs3S500EFG320**	-	-	-	14%	12%	256.54	11.83%	78.41%	133.708

el que más relevancia tiene para saber hasta que velocidad puede correr el procesador MIPS. La tabla muestra que el Spartan3 de Xilinx entrega la mayor frecuencia de operación mientras que el CycloneII de Altera brinda la menor frecuencia de reloj de los tres FPGAs.

Otro parámetro importante de destacar en la tabla XIII es la cantidad de recursos de área que utiliza la implementación del MIPS en las diferentes tarjetas, sin contar el uso de la memoria externa de datos. El FPGA Spartan3E es la tarjeta que consume menor porcentaje de recursos de hardware, en cambio, la tarjeta Cyclone II consume más de la mitad de sus recursos de área para la misma implementación.

4.1.2 Reporte de estimación de consumo de potencia del MIPS

A partir de los resultados de la sección anterior se eligió el FPGA Spartan3 de Xilinx para realizar la implementación del decodificador Viterbi en el MIPS, y obtener una estimación del consumo de potencia del procesador.

La herramienta de Xilinx provee un programa de estimación de consumo de energía que permite obtener un análisis detallado del consumo de potencia de todo el dispositivo. Este tipo de análisis es muy importante cuando un dispositivo va ser utilizado en aplicaciones de bajo consumo. La figura 4.1 muestra la potencia total consumida por el procesador MIPS según los parámetros intrínsicos de voltaje dinámico y estático como también la corriente consumida por la aplicación.

	Voltage (V)	Current (mA)	Power (mW)
Vccint	1.2		
Dynamic		9.08	10.90
Quiescent		20.00	24.00
Quiescent		15.00	37.50
Vccaux	2.5		
Dynamic		0.00	0.00
Vcco25	2.5		
Dynamic		0.00	0.00
Quiescent		0.00	8.00
Total Power			72.40
Startup Current (m		0.00	

Figura 4.1 Sumario de consumo de Potencia del MIPS

La figura 4.2 indica el consumo individual de los recursos utilizados en la implementación, como es la potencia consumida por el reloj, por los búfers de entrada y salida, las puertas lógicas, y las señales.

Clocks Power (mW)	3.73
IOs Power (mW)	6.91
Inputs Power (mW)	0.22
Logic Power (mW)	0.02
Outputs Power (mW)	0.00
Signals Power (mW)	0.01

Figura 4.2 Potencias subtotales del MIPS

Otro importante parámetro de medición es el desglose del consumo de corriente de la tarjeta para saber que unidades es la que mas demanda corriente y poder hacer modificaciones a la aplicación si uno de los requerimientos así lo exigiera. La figura 4.3 detalla el consumo de corriente de cada uno de los recursos de harware utilizados en la aplicación.

Clocks Current (mA)	3.11
clk_BUFGP/IBUFG Current (mA)	3.11
IOs Current (mA)	5.76
Vcco25 Current (mA)	0.00
Inputs Current (mA)	0.18
Logic Current (mA)	0.02
Outputs	
Vcco25 Current (mA)	0.00
Signals Current (mA)	0.01

Figura 4.3 Consumo de Corriente del MIPS

La temperatura de operación es un parámetro muy importante en los casos donde las condiciones de trabajo son muy extremas, es por eso que esta herramienta calcula un

estimado de temperatura en la juntura y el empaquetamiento dado una temperatura ambiente de trabajo. Figura 4.4

Ambient Temperature (°C)	25
Junction Temperature (°C)	27.43
Case Temperature (°C)	27.03
Part Type	Commercial
Airflow (LFM)	0
Package	ft256
Total Power (mW)	72.40

Figura 4.4 Datos Termales

4.1.3 Análisis de Tiempo de las operaciones del decodificador

Con la implementación del decodificador Viterbi sobre el MIPS, es decir sobre el Spartan3 de Xilinx y la ayuda de un analizador lógico, se hizo el análisis de tiempo de cada una de las operaciones del decodificador.

Las figuras que se muestran a continuación miden el tiempo en alto (encerrado en el círculo rojo) de la señal A3(7), que corresponde al estado de un pin de salida del MIPS. Esta señal se activa con una instrucción extra de prueba ejecutada al inicio de cada rutina a medir y se desactiva al final de la misma.



CIB-ESPOL

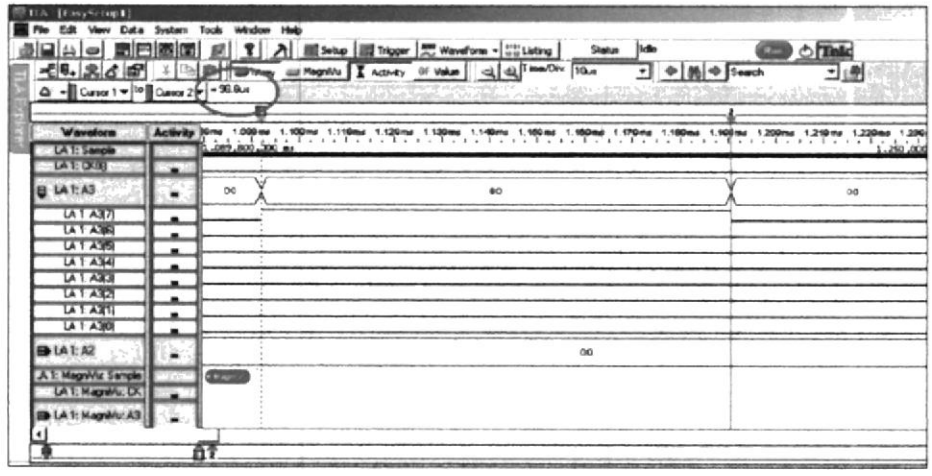


Figura 4.5 Tiempo de Métrica de rama y nodo

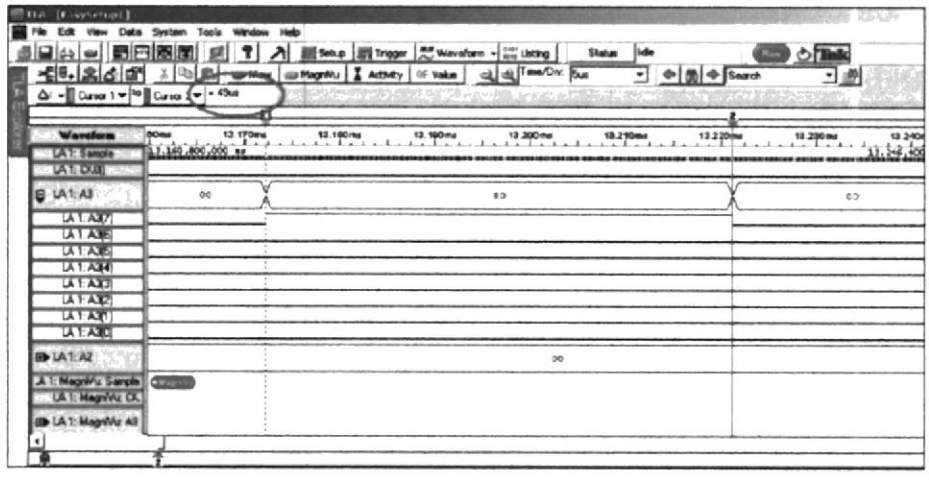


Figura 4.6 Tiempo de TB 64 bits

La figura 4.5 muestra la medición de tiempo real del cálculo de las métricas de nodo y rama (BMU y ACS) para una frecuencia de operación del MIPS de 50 MHZ, ya que el Spartan3 trabaja a esta frecuencia.

La ejecución completa de un conjunto de 60 instrucciones para la operación ACS de una sola métrica del nodo oscila en un tiempo aproximado de 98.66 microsegundos. Este tiempo incluye la ejecución de subrutinas de salto y lazos entre las 60 instrucciones. Mientras que con el analizador lógico se midió un tiempo de 98.6 microsegundos, tal como lo indica la figura 4.5. Este es el proceso más tardado de todo el decodificador debido a que solo contamos con una ALU dentro del MIPS.

El proceso de TB como se muestra en la figura 4.6 presenta la menor cantidad de tiempo de todo el proceso de Viterbi, 48.98 microsegundos, para una ejecución de decodificación de 64 bits.

La medición de tiempo de todo el proceso de decodificación de una trama de 256 bits se presenta en la figura 4.7 que indica que se lleva un total de 26.4634 milisegundos de ejecución en terminar el proceso que corresponden a 1.323.170 instrucciones realizadas.

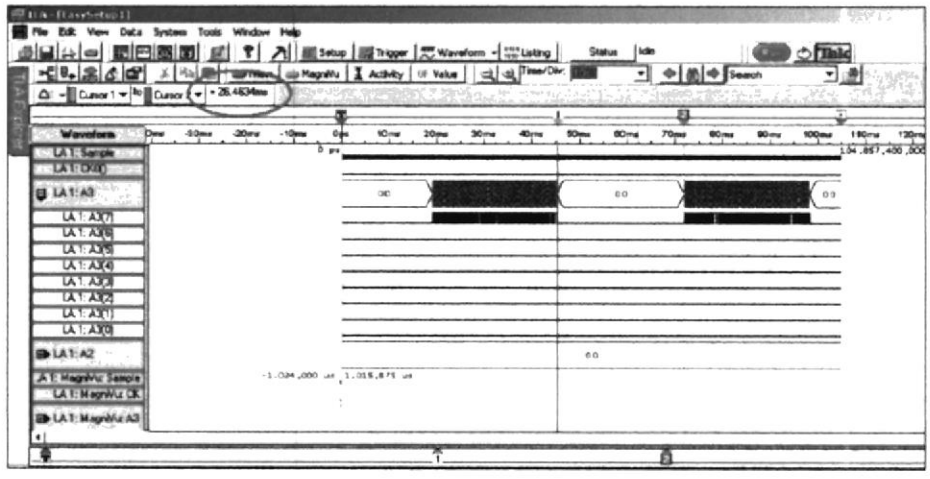


Figura 4.7 Tiempo total de ejecución del decodificador

4.1.4 Resultados de la cama de pruebas: Gráficas de Probabilidad de Error

En esta sección se mostrarán los resultados de la implementación del decodificador Viterbi en software. Como se explicó en la sección 3.4.3 acerca de la implementación de la cama de prueba, un programa en C permitió recibir los bits decodificados y comparar estos datos con la secuencia original y así obtener una curva real de probabilidad de error en base al desempeño del decodificador.

La figura 4.8 contiene la curva de BER de la implementación hecha en software y las curvas presentadas en la sección 3.3 que representan a la decodificación hecha en punto fijo y

punto flotante (ideal). De esta curva se puede resaltar la diferencia de ganancia de probabilidad de error a partir de los 3 dB, entre la curva del decodificador en software con respecto a la curva de referencia de punto fijo. Esto se debe básicamente a la forma de decodificar los últimos 64 bits de información que utiliza una subrutina de búsqueda del estado con menor métrica que se mejora con la solución implementada en hardware con los bits de cola como se explicará en la siguiente sección.

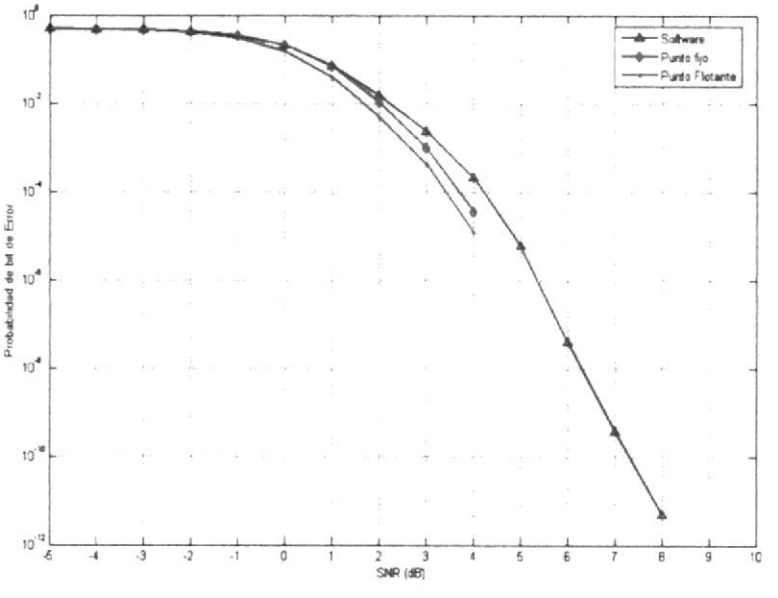


Figura 4.8 Curvas de Probabilidad de Error de bits para diferentes niveles de SNR de la implementación en SW

Con los archivos que contienen los bits decodificados también se puede reconstruir las imágenes y observar si existen irregularidades en las figuras para los diferentes niveles de

SNR a las que fueron sometidas. Por ejemplo las figuras 4.9, 4.10, 4.11 y 4.12 muestran una comparación entre la simulación de una transmisión sin codificación de canal y los resultados de la cama de prueba de una imagen con niveles de SNR de 10, 5 ,2 y 0 dB.

Se puede apreciar que para las imágenes que fueron decodificadas con SNR de 10 y 5 dB y 2 dB, presentan una gran diferencia entre los resultados obtenidos de la simulación con los resultados obtenidos en la cama de prueba.



Sin codificación **Implementado**

Figura 4.9 Imagen de Lena con SNR=10 dB en la implementación del decodificador en SW

ESCUELA SUPERIOR POLITÉCNICA ESPOL
CIB-ESPOL



Sin Codificación **Implementado**

Figura 4.10 Imagen de Lena con SNR= 5 dB en la implementación del decodificador en SW



Sin Codificación **Implementado**

Figura 4.11 Imagen de Lena con SNR= 2 dB en la implementación del decodificador en SW

La imagen con un SNR de 2 dB presenta puntos más visibles de error y empieza a notarse una ligera distorsión de entre los resultados simulados sin codificación y los procesados por el MIPS.



Sin Codificación **Implementado**

Figura 4.12 Imagen de Lena con SNR=0 dB en la implementación del decodificador en SW

En la imágenes con SNR de 0 dB (canal ruidoso) se puede observar que están más claramente distorsionadas. Esto comprueba los resultados previos obtenidos en la sección 3.3.2 sobre la capacidad correctiva a partir de un nivel SNR de 1dB, mientras que las imágenes a 0 dB, a pesar que ambas están distorsionadas, la imagen con codificación de canal no mejora la calidad de la imagen; la curva de BER en la figura 3.13 de la señal sin codificación presenta mejor características de probabilidad de error para este rango de nivel de SNR de -5 dB a 0 dB.

Aunque pareciera no tener sentido el hecho que la recepción sin codificación presente menor probabilidad de error en el

rango de -5 dB a 0 dB con respecto a la señal codificada, en fines reales de aplicación se considera que en términos generales un canal ruidoso está a 3 dB de nivel SNR, por lo tanto no representa una desventaja analizar los datos a esos niveles de SNR sino más bien a la capacidad correctiva que adquieren los códigos convolutivos a partir de los 1 dB de SNR.

Por eso para diferentes aplicaciones esta referencia cambia, por ejemplo, en aplicaciones de voz donde no se envía mucha información por segundo, un canal se considera ruidoso si no se permite tener una Pe de 10E-5. Para transmisión de video, en cambio, donde se envía mucho más información que en voz se necesita garantizar una Pe de 10E-6.

Las figuras 4.9 hasta 4.12 son congruentes con la figura 4.8 donde se observa que para niveles de SNR mayores a 5 dB, la imagen no presenta errores visiblemente significativos, para los casos de menor nivel de SNR la decodificación con estos parámetros ya no es tan efectiva, por lo que si se desea conseguir un mejor desempeño se deben utilizar parámetros

más exigentes o técnicas más sofisticadas y complejas como los Turbo Códigos [3].

Dependiendo la exigencia de la aplicación se puede obtener un mejor desempeño del decodificador aumentando la longitud de la trama de entrada, los bits de cuantificación o la tasa de codificación.

4.2 Resultados de síntesis de la implementación en HW

Se presenta a continuación un análisis comparativo de los resultados de síntesis del código en VHDL del decodificador Viterbi para el caso de Longitud Restrictiva K=7 y Tasa de codificación 1/2, con entrada de decisión suave de 3 bits.

Las herramientas de síntesis usadas fueron: ISE 7.1 y 8.2i de Xilinx, QuartusII de Altera, y Mentor Graphics; y fue probado sobre cuatro familias diferentes de FPGAs, a diferencia de la implementación en SW probada en tres FPGAs.

4.2.1 Reporte de síntesis del decodificador Viterbi sobre los FPGAs

Primeramente se muestran y analizan por separado los resultados de síntesis de cada una de las operaciones del decodificador, y luego los resultados de la implementación completa. Las tablas que se presentan resumen los resultados de la síntesis para recursos de área: LUTs, I/Os y BRAMs y desempeño en frecuencia.

Las familias de FPGAs: CycloneII, Spartan3 y Spartan3E tienen una frecuencia de reloj de 50MHz, mientras que el VirtexIIPro es de 100MHz.

Para la unidad de cálculo de métrica de ramas –BMU, en la tabla XIV, se observa que este bloque es de relativamente bajo uso de recursos de hardware en las cuatro familias de FPGAs utilizadas. El elevado número de buffers I/O se debe al gran número de entradas y salidas que tiene este bloque, esto permite sugerir que ninguna de las familias de FPGAs podría soportar la implementación única de esta operación, ya que esta estadística es sólo para entradas y salidas externas

al diseño. En la implementación del decodificador completo, es decir, los tres bloques juntos, estas entradas y salidas son interconexiones internas que no afectarán en realidad el número de I/Os.

Tabla XIV. Resultados de Síntesis de Unidad de Cálculo de Métrica de Ramas -BMU

Herramienta de Síntesis	*Quartus*		*ISE*		*Mentor Graphics*	
FPGA	Recursos		Recursos		Recursos	
	LUTs	**I/O**	**LUTs**	**I/O**	**LUTs**	**I/O**
***Cyclone II* EP2C35F672C6**	0.2%	219.79%	-	-	0.48%	217.47%
***Spartan3* xs3S200FT256**	-	-	2%	597%	2.63%	596.53%
***Spartan3E* xs3S500EFG320**	-	-	<0.1%	445%	0.47%	531.96%
***Virtex2Pro* xc2vp307ff896**	-	-	<0.1%	185%	-	-

Se puede observar en la misma tabla que la implementación del BMU que menos recursos usa se obtuvo en la síntesis proporcionada por la herramienta Xilinx para el VirtexIIPro.

El bloque de Cálculo de Métricas de Nodo -ACS, es el bloque con la mayor cantidad de recursos utilizados, esto se puede observar en la tabla XV por medio de la cantidad de LUTs utilizados. La tabla también muestra una gran utilización de búfers I/O sin embargo, como se explicó anteriormente la

CIB-ESPOL

cantidad de I/Os de cada uno de los bloques interiores del decodificador, no influye en realidad en la cantidad de I/Os de la implementación completa.

La última operación del decodificador es la unidad que administra los bits sobrevivientes y hace el rastreo de ellos hacia atrás –TB-, esta implementación arrojó los resultados, que se detallan en la tabla XVI, donde se observa que la mayor frecuencia de operación para esta parte del decodificador está dada por el FPGA VirtexIIPro, la cual es de 217.436 MHz, además de utilizar la menor cantidad de recursos de hardware.

Vale destacar que en el reporte de esta operación se indica la cantidad de bloques BRAM, debido al uso de las cuatro memorias para almacenar los vectores decisión que provienen del ACS, y por las dos LIFOS que se usan para ordenar los bits decodificados.

Tabla XV. Resultados de Síntesis de Unidad de Cálculo de Métrica de Nodos -ACS

Herramienta de Síntesis	*Quartus*			*ISE*			*Mentor Graphics*		
FPGA	Recursos		Máxima Frecuencia	Recursos		Máxima Frecuencia	Recursos		Máxima Frecuencia
	LUTs	**I/O**	**MHz**	**LUTs**	**I/O**	**MHz**	**LUTs**	**I/O**	**MHz**
***Cyclone II* EP2C35F672C6**	23.66%	612.98%	-	-	-	-	19.05%	606.95%	487.092
***Spartan3* xs3S200FT256**	-	-	-	121%	1666 %	283.66	145.86%	1665.90%	279.096
***Spartan3E* xs3S500EFG320**	-	-	-	50%	1242%	233.526	60.15%	1637.50%	129.870
***Virtex2Pro* xc2vp307ff896**	-	-	-	14%	518%	505.676	-	-	-

Tabla XVI. Resultados de Síntesis de la Unidad de Rastreo hacia atrás -TB

Herramienta de Síntesis	*Quartus*				*ISE*				*Mentor Graphics*			
FPGA	Recursos			Máxima Frecuencia	Recursos			Máxima Frecuencia	Recursos			Máxima Frecuencia
	LUTs	**I/O**	**BRAM**	**MHz**	**LUTs**	**I/O**	**BRAM**	**MHz**	**LUTs**	**I/O**	**BRAM**	**MHz**
***Cyclone II* EP2C35F672C6**	< 1 %	15 %	-	121.15	-	-	-	-	0.92%	14.95%	-	156.715
***Spartan3* xs3S200FT256**	-	-	-	-	6%	41%	50%	149.338	6.51%	41.04%	0%	149.858
***Spartan3E* xs3S500EFG320**	-	-	-	-	2%	31%	30%	106.281	2.68%	40.34%	-	103.563
***Virtex2Pro* xc2vp307ff896**	-	-	-	-	<1%	12%	2%	217.436	-	-	-	-

Finalmente, la tabla XVII muestra los resultados de síntesis de la implementación completa en VHDL del decodificador Viterbi. Se observa que la máxima frecuencia para el decodificador está dada en la familia VirtexIIPro y es de 163.825 MHz o Mbps porque por cada ciclo de reloj se obtiene un bit decodificado. Además la implementación sobre este FPGA presenta el menor porcentaje de uso de los recursos de hardware.

De estos resultados se puede resaltar que esta implementación satisface el requerimiento de velocidad del decodificador según el estándar IEEE 802.11a, la cual es de 48 Mbps, como lo muestra la tabla II del capítulo 1.

Esta tasa de datos alcanzada tiene estrecha relación con los parámetros usados en el codificador, estos son: polinomios generadores de 133 y 171 en base octal, longitud restrictiva de 7 y tasa de codificación de 1/2. Para determinadas aplicaciones que obedecen a otros estándares, se utilizan otros polinomios generadores que ofrezcan diferentes poderes correctivos.



CIB-ESPOL

Tabla XVII. Resultados de Síntesis del Decodificador Viterbi en VHDL

Herramienta de Síntesis	***Quartus***				***ISE***				***Mentor Graphics***		
FPGA	Recursos			Máxima Frecuencia	Recursos			Máxima Frecuencia	Recursos		Máxima Frecuencia
	LUTs	**I/O**	**BRAM**	**MHz**	**LUTs**	**I/O**	**BRAM**	**MHz**	**LUTs**	**I/O**	**MHz**
Cyclone II **EP2C35F672C6**	**17 %**	**3 %**	**10 %**	**87.84**	-	-	-	-	23.37%	2.74%	98.551
Spartan3 **xs3S200FT256**	-	-	-	-	161%	7%	50%	104.896	152.27%	6.94%	61.935
Spartan3E **xs3S500EFG320**	-	-	-	-	66%	5%	30%	83.785	62.79%	6.82%	49.324
Virtex2Pro **xc2vp307ff896**	-	-	-	-	21%	2%	2%	163.825	-	-	-

Por ejemplo, para el estándar GSM se usa un código de canal de 16 estados, para el estándar W-CDMA 256 estados y para WiFi se usa el código convolucional implementado en este trabajo [8].

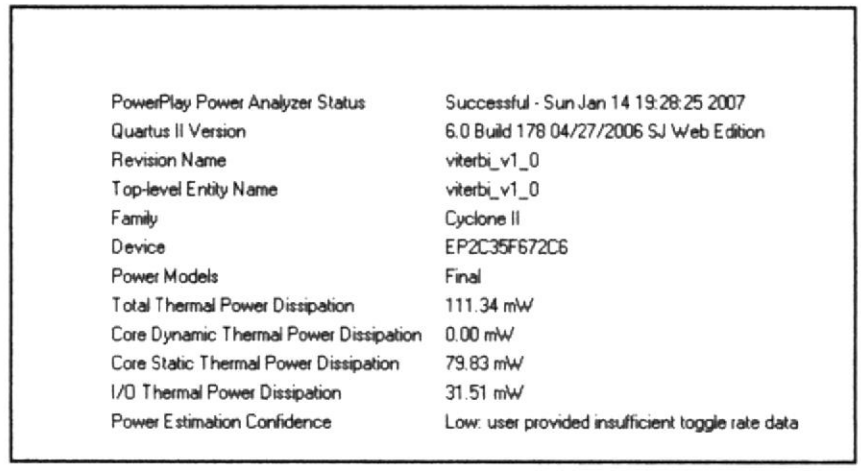
Los criterios usados y descritos en el capítulo 3 de la implementación del decodificador Viterbi, como en el caso de la unidad BMU donde se evitan las multiplicaciones para el cálculo de la distancia Euclidiana, o en el ACS donde se usan registros en vez de memorias para guardar las métricas del nodo actualizadas, y en la unidad TB, donde se usan 4 memorias aplicando la Propiedad de Longitud de Convergencia de Trellis, ayudan a cumplir el requerimiento de desempeño del estándar y aseguran un preciso funcionamiento del decodificador en la detección y corrección de errores, como se demostrará en secciones siguientes.

El decodificador Viterbi fue finalmente implementado y verificado en el VirtexIIPro, específicamente el XC2VP30-7ff896 a una velocidad de grado 7. Por lo tanto, las simulaciones y estimación de consumo de potencia que se presenta a continuación se basan en este FPGA.

4.2.2 Reporte de estimación de consumo de potencia del decodificador Viterbi

Se presenta a continuación el análisis de consumo de potencia para la implementación del decodificador Viterbi en hardware sobre un FPGA VirtexIIPro, mediante el uso de la herramienta XPower de Xilinx.

En la figura 4.14 se presenta el resultado obtenido, el cual es de 569 mW lo que representa un valor elevado si lo comparamos con el consumo de potencia de la implementación realizada sobre el MIPS. Por ello se decidió realizar un análisis de consumo de potencia para otro FPGA sobre el cual se pudiera diseñar la implementación, y se eligió al CycloneII. Para estimar el consumo de potencia de este FPGA se usó la herramienta PowerPlay Power Analyzer de Altera. La figura 4.13 muestra el resultado del consumo aproximado de potencia del CycloneII el cual fue de 111.34 mW.



PowerPlay Power Analyzer Status	Successful - Sun Jan 14 19:28:25 2007
Quartus II Version	6.0 Build 178 04/27/2006 SJ Web Edition
Revision Name	viterbi_v1_0
Top-level Entity Name	viterbi_v1_0
Family	Cyclone II
Device	EP2C35F672C6
Power Models	Final
Total Thermal Power Dissipation	111.34 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	79.83 mW
I/O Thermal Power Dissipation	31.51 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Figura 4.13 Reporte de consumo de potencia del decodificador Viterbi usando Power Analyzer de Altera.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		569
Vccint 1.50V:	99	149
Vccaux 2.50V:	167	418
Vcco25 2.50V:	1	3
Clocks:	49	73
Inputs:	1	1
Logic:	0	0
Outputs:		
Vcco25	0	0
Signals:	0	0
Quiescent Vccint 1.50V:	50	75
Quiescent Vccaux 2.50V:	167	418
Quiescent Vcco25 2.50V:	1	3

Figura 4.14 Reporte de consumo de potencia del decodificador Viterbi usando XPower de Xilinx.

En cuanto al análisis de porque existe tal diferencia en el consumo de potencia entre ambos FPGAs vale acotar que estas mediciones no fueron realizadas con el mismo valor

de frecuencia de operación en ambos casos, ya que la implementación para el FPGA CycloneII de Altera, esta fue de 50MHz, mientras que para el FPGA VirtexIIPro de Xilinx, era de 100 MHz, y mientras más alta es la frecuencia más alto es el consumo de potencia del FPGA.

Otro punto a tener en cuenta para que haya existido esta diferencia de potencia para ambas implementaciones es la diferencia en el proceso tecnológico de fabricación con el que estén hechas las FPGA porque mientras menor sea el proceso normalmente se tiene un menor consumo. El Cyclone II fue elaborado con un proceso de 90nm mientras que el Virtex2Pro con uno de 130 nm.

4.2.3 Simulaciones

El diseño del decodificador Viterbi en Hardware fue simulado usando la herramienta ModelSim XE 6.1e.

La figura 4.15 muestra parte de la simulación del decodificador de Viterbi. Se observan la señal de reloj (*Clk*), con un periodo de 10ns, que corresponde al periodo de reloj

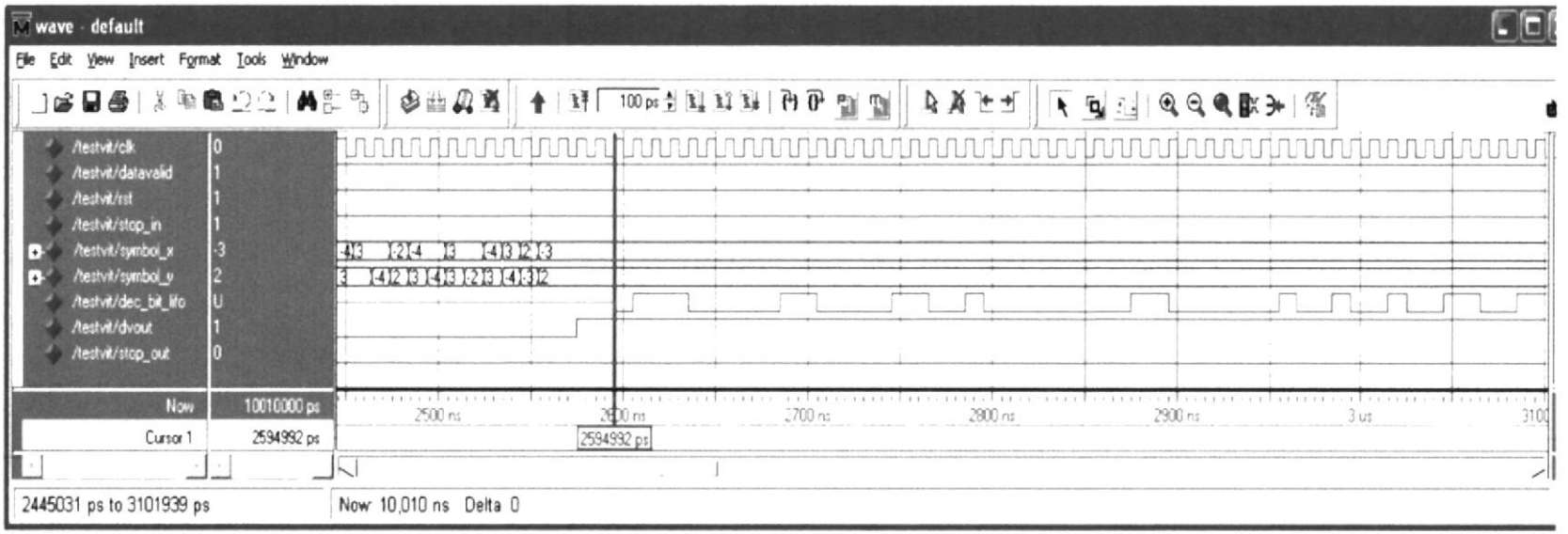


Figura 4.15 Resultado de la simulación del decodificador de Viterbi en el que se muestra el retardo inicial antes de obtener el primer bit de salida.

CIB-ESPOL

al cual opera el VirtexIIPro. Para que el decodificador funcione la señal de reset del decodificador (*rst*) la cual es activa en nivel bajo y las señales de habilitación *Datavalid* y *Stop_In* deben encontrarse en un nivel de un '1' lógico. Se observa además las señales *Symbol X* y *Symbol Y* de 3 bits cada una, las cuales representan las dos entradas de datos cuantificados del decodificador.

El primer bit de la secuencia de salida decodificada, representado en la figura 4.15 por la señal *dec_bit_lifo*, se lo obtiene luego de un retardo de 260 ciclos de reloj, lo cual representa un tiempo de 2.6 microsegundos, antes de lo cual esta señal se encuentra en estado 'U' (undefined), ya que no existe ninguna salida generada. Se observa además que luego de este retardo se obtiene un bit de salida por cada ciclo de reloj, lo que significa que el decodificador de Viterbi entrega una tasa de bits de 100 Mbps.

Una observación relevante que puede notarse por medio de la simulación se observa en la figura 4.16, donde se justifica el uso de 14 bits para representar las métricas del nodo a la salida del ACS.

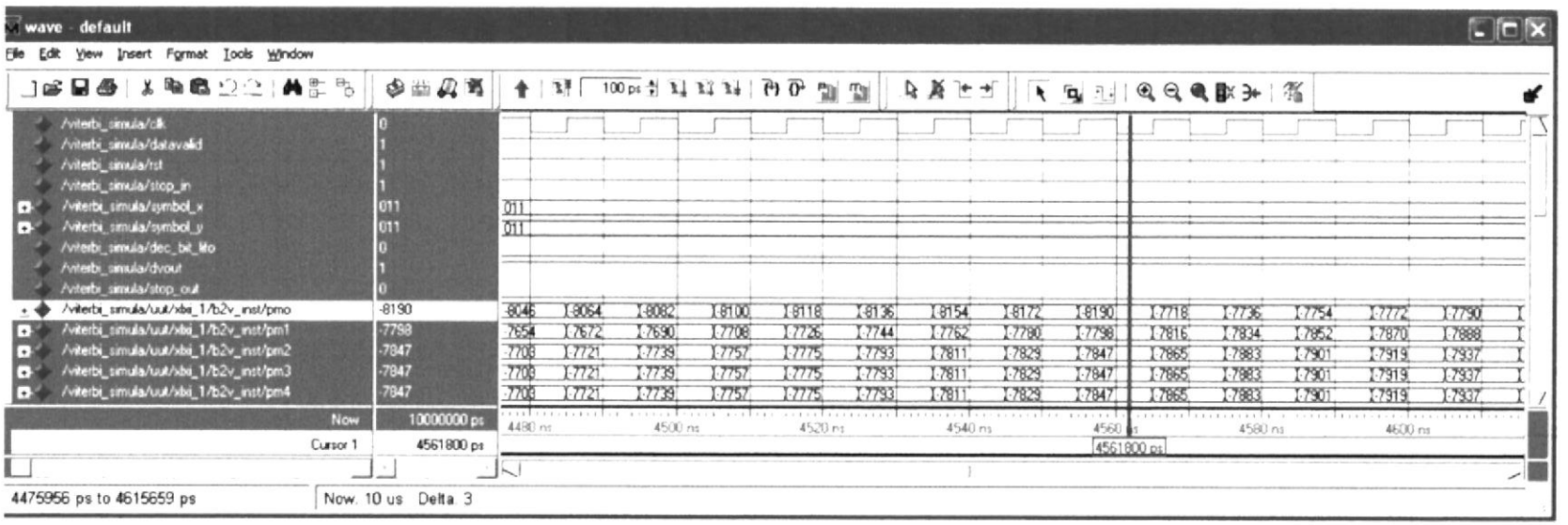


Figura 4.16 Resultado de la simulación del decodificador de Viterbi en el que se muestra las métricas de nodo.

Con una secuencia de 256 pares de símbolos a la entrada del decodificador, el máximo valor que puede tomar una métrica del nodo cero, *pm0*, como se muestra, es de -8190, cuando las entradas *Symbol_X* y *Symbol_Y* siempre tienen un valor binario de "011". Entonces, con esta cantidad de bits se pueden representar métricas del nodo desde -8192 hasta 8191 (2^{13} = 8192).

4.2.4 Resultados de cama de pruebas: Gráficas de Probabilidades de Error

La cama de pruebas para la implementación en hardware se construye también como se describe en la sección 3.4.3.

Una observación importante durante la realización de estas pruebas de la implementación en hardware, fue la necesidad de incluir ceros al final de cada secuencia del mensaje, estos bits de cola sirven para forzar al codificador regresar al estado cero y de esta forma la transmisión de una secuencia sea independiente de la transmisión de otras. El número de ceros que se añade depende del número de elementos de retardo del codificador, en este caso 6. Por lo tanto, cada secuencia de 256 bits del archivo

binario de la imagen de Lena a la entrada del transmisor, contiene 250 bits de información y 6 bits de cola.

Es importante destacar que en las pruebas del decodificador en software no se añaden estos bits de cola al final de cada secuencia.

La figura 4.17 demuestra el gran desempeño y aproximación de las curvas de BER de la implementación en hardware con respecto a las curvas de referencia tanto en punto fijo como la flotante o ideal.

Las curvas de BER del diseño y de punto fijo mantienen una margen de ganancia menor de 0,5 dB con respecto a la curva ideal a partir de 0 dB de SNR, lo que en aspectos prácticos de implementación no representa un mayor compromiso de desempeño.

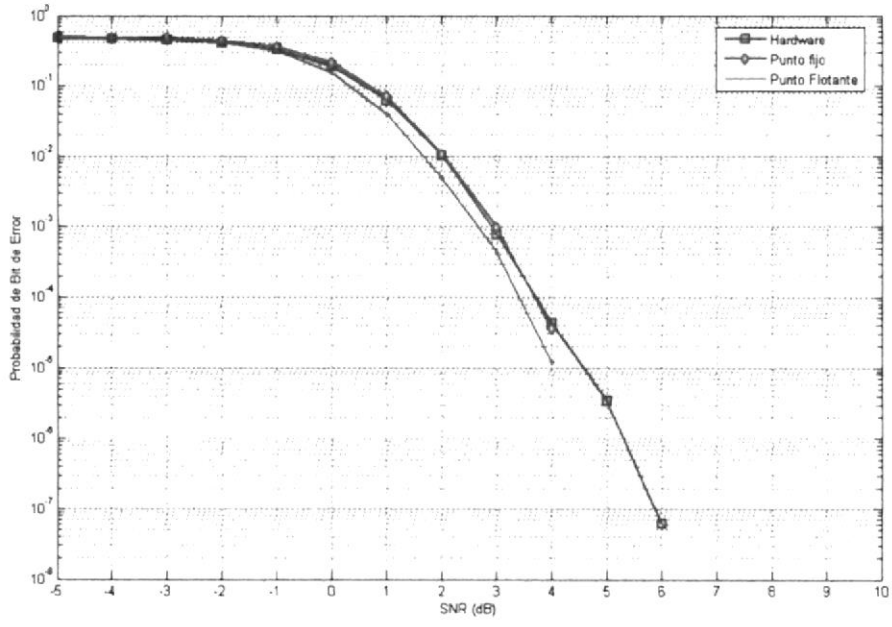


Figura 4.17 Curvas de Probabilidad de Error de bits para diferentes niveles de SNR de la implementación en HW.

Para observar la corrección de errores del decodificador, se presentan a continuación las imágenes de Lena obtenidas con los archivos decodificados a diferentes niveles de SNR.



Sin Codificación **Implementado**

Figura 4.18 Imagen de Lena con SNR= 10 dB en la implementación del decodificador en HW

CIB-ESPOL



Sin Codificación Implementado

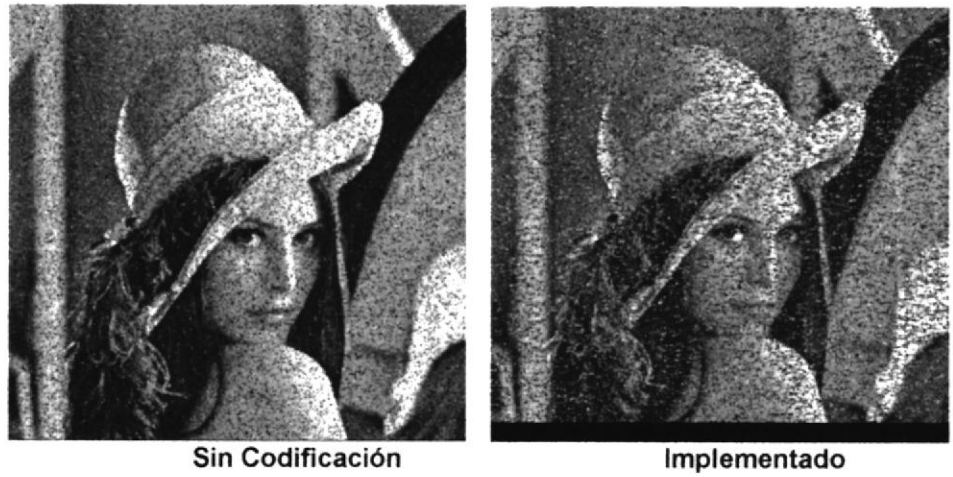
Figura 4.19 Imagen de Lena con SNR= 5 dB en la implementación del decodificador en HW



Sin Codificación Implementado

Figura 4.20 Imagen de Lena con SNR= 2 dB en la implementación del decodificador en HW

En las figuras 4.18 a 4.21 se puede observar las diferentes imágenes reconstruidas con diferentes valores de SNR.



Sin Codificación Implementado

Figura 4.21 Imagen de Lena con SNR= 0 dB en la implementación del decodificador en HW

Se puede notar que para un nivel de SNR de 0 dB la imagen sin decodificación es más clara que la imagen que se obtiene a la salida del decodificador, lo cual es congruente con las curvas de la figura 3.13 del capítulo 3, en la cual se observa que para niveles de SNR menores a 1 dB, un canal sin codificación muestra un mejor desempeño que uno que si la tenga.

Luego se observa que para los niveles de 2, 5 y 10 dB la imagen que se obtiene a la salida del decodificador muestra menor número de "puntos" que representan el ruido, en comparación a la imagen que se obtiene en un canal sin codificación.

Al igual que en la implementación de SW del decodificador, se establece la relación entre la curva de BER de la figura 4.17 con las figuras de Lena, donde se observa que para un canal considerado muy ruidoso (niveles de SNR de 0 dB) el decodificador de Viterbi demuestra poca capacidad de corrección de errores, no así para un canal a un SNR de 5 y 10 dB, donde se aprecia que la imagen casi no presenta errores.

Se observa también que entre las imágenes que se transmiten y las decodificadas, existe un pequeño recorte de la imagen en la parte inferior debido a los 48 kbits de información ocupados por los bits de cola. En pocas palabras se puede decir que la desventaja de utilizar este recurso es que se afecta la tasa de datos de la recepción, es decir se transmiten menos bits de información por paquete; mientras mejora la probabilidad de error.



CIB-ESPOL

4.3 Propuesta de Co-diseño Hardware-Software del decodificador Viterbi

Actualmente ha crecido la tendencia hacia la aplicación de metodologías que integren concurrentemente técnicas de diseño para desarrollar sistemas digitales combinados, estos sistemas complejos

son los actuales sistemas embebidos, que necesitan ser implementados con combinación de software y hardware.

Una plataforma de implementación común para sistemas embebidos es un SOC. Un SOC consiste de uno o más procesadores de software programable combinado con hardware de aplicación específica en un solo circuito integrado.

El concepto de Co-diseño de Hardware/Software se refiere a un proceso de diseño de sistemas que combine perspectivas de hardware y de software desde las primeras etapas para explotar la flexibilidad del diseño y la eficiente ubicación de funciones. Es un modelo de diseño sistemático para sistemas digitales complejos, los que tienen que ser descritos con estrictas restricciones en tiempo, área, tamaño, confiabilidad y costo, en el que se pretende construir un óptimo marco de diseño. El desafío del co-diseño es la optimización de todas estas características lo que implica establecer ciertos compromisos entre hardware y software que cumplan con los objetivos de diseño del sistema.

La práctica de diseñar concurrentemente los componentes de hardware y software incluye algunas etapas: especificación, partición,

síntesis y verificación. En la figura 4.22 se muestra el flujo del proceso de Co-diseño.

Siguiendo con el flujo de Co-diseño, tenemos que como primer paso se encuentra la Especificación. La especificación para un sistema embebido consiste en tres partes: opciones de tecnología deseada, descripción de comportamiento de la aplicación y exigencias de diseño. [22]

En el caso de las dos implementaciones del decodificador Viterbi, la tecnología deseada describe cuales procesadores programables pueden ser usados para ejecutar software, cuales equipos de hardware pueden ser sintetizados. En particular, nuestra investigación generó como resultado dos propuestas de implementaciones de Co-diseño Hardware/Software para el decodificador de Viterbi, que usan la implementación del decodificador sobre el procesador MIPS, como software programable y la implementación en VHDL sobre un FPGA como hardware específico.



CIB-ESPOL

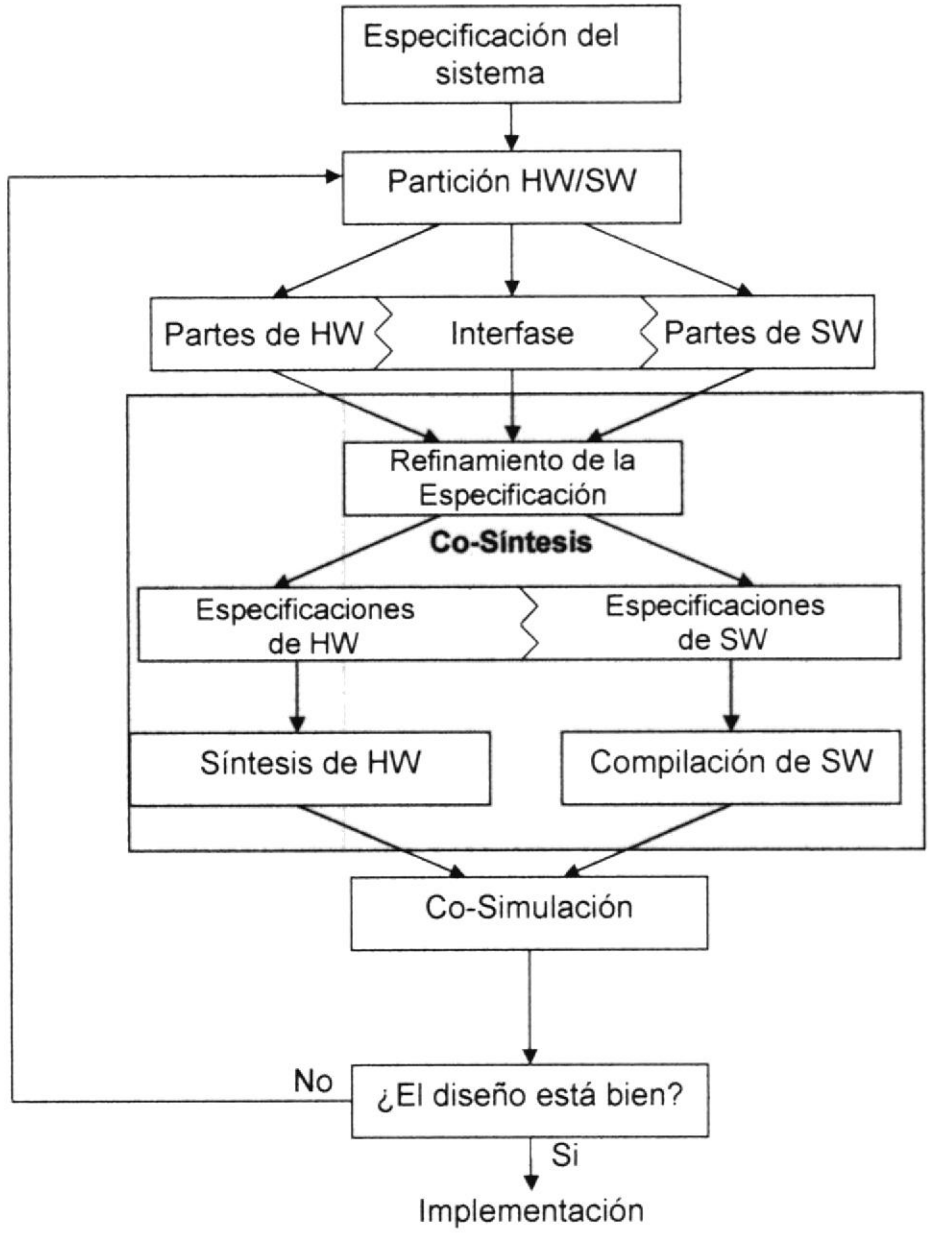


Figura 4.22 Diagrama de Flujo del proceso de Co-diseño

La descripción de comportamiento puede ser realizada por medio de un modelo o de un lenguaje de especificación [20]. En este caso, se usa el lenguaje de programación C, para la descripción de

comportamiento en software y VHDL, para la descripción de comportamiento en hardware.

Las exigencias de diseño pueden incluir tamaño del FPGA, tamaño del programa o memoria de datos y exigencias de desempeño.

En esta etapa, los desarrolladores no toman decisiones sobre cómo construir los componentes; simplemente conectan jerárquicamente bloques básicos hasta llegar a un modelo preliminar aceptable que esté de acuerdo con las especificaciones iniciales. [22]

Siguiendo en el flujo de Co-diseño, se tiene la etapa de Partición. Realizar una partición entre hardware y software se refiera al proceso de decidir para cada subsistema de un sistema principal, si es que la funcionalidad requerida brinda más ventajas implementándola en hardware o en software. A breves rasgos la partición consiste en separar cierto número de procedimientos y variables en grupos, con el objetivo de cumplir con las exigencias requeridas para un determinado programa.

Particionar un sistema en hardware y software afecta el costo y desempeño total de dicho sistema.

Una implementación en Hardware, brinda un alto desempeño debido a su velocidad y a la ejecución paralela de las operaciones, pero se incurre en el costo adicional de la fabricación de los ASICs o tamaño y costo del FPGA.

Una implementación en Software puede correr en procesadores de alto desempeño que satisfagan los requerimientos, pero sería necesario un procesador de tal poder de procesamiento, que no sería práctico en un sistema embebido de comunicaciones móviles.

Para realizar una partición se puede partir de dos formas, empezar con todas las funcionalidades en software y mover a hardware aquellas porciones que son críticas en tiempo de ejecución; o empezar con todas las funcionalidades en hardware y mover porciones a una implementación en software. [23]

La primera propuesta de Co-diseño se basa en la optimización de área del diseño combinado y la segunda en la optimización de tiempo, como se muestra en las figuras 4.23 y 4.24.

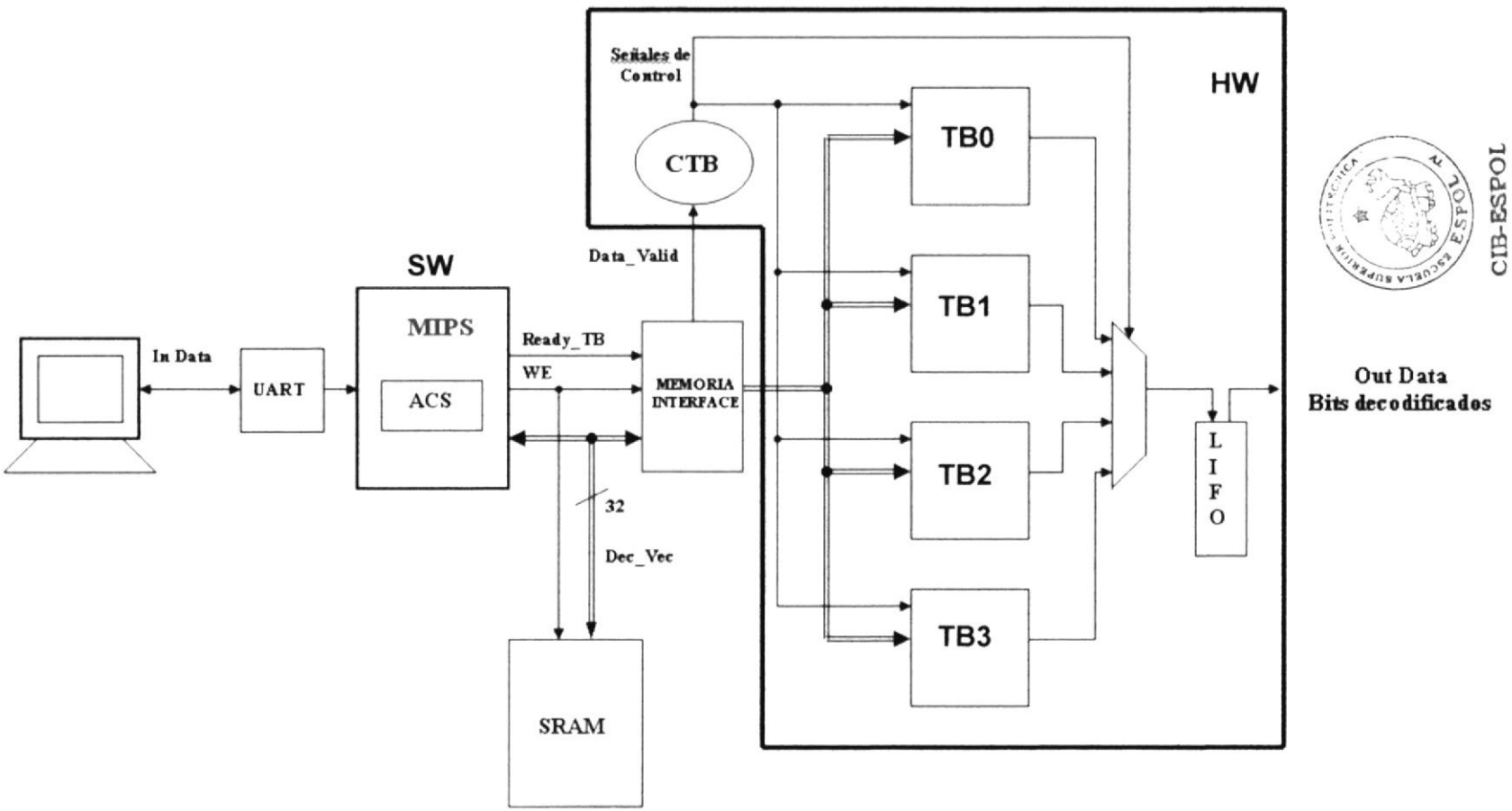


Figura 4.23 Esquema de codiseño para optimización en área.

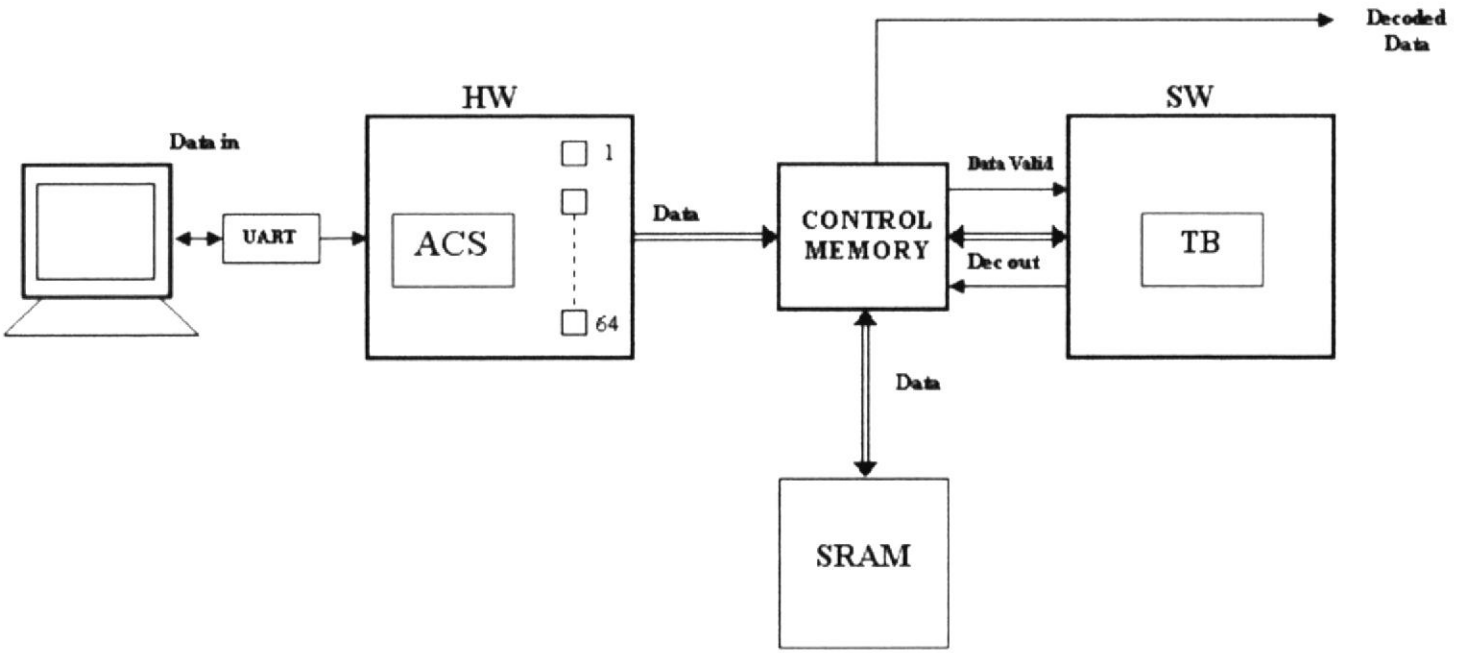


Figura 4.24 Esquema de codiseño para optimización en tiempo.

Estas propuestas pueden ser consideradas en un trabajo futuro, en esta parte solo se muestra un esquema básico de las interconexiones entre los diferentes bloques que estarían involucrados en el codiseño.

El caso en el cual se desea optimizar el área del diseño, se empieza por implementar el decodificador en hardware, es decir, se construye el decodificador en su totalidad en código VHDL. Luego se verifica si se cumplen los requerimientos de hardware, por ejemplo si se desea diseñar sobre un FPGA determinado, se verifica si el número de compuertas disponibles en ese FPGA es el suficiente para soportar el diseño en su totalidad.

En base a este criterio y de acuerdo a los resultados de la tabla XV, se aprecia que para los FPGAs Spartan 3 y Spartan3E, no existe el suficiente número de compuertas para soportar el diseño, es decir, que si este hubiera sido el requerimiento inicial de hardware, no se hubiera cumplido con el mismo.

Entonces se analiza, cual es él o los bloques que presentan excesivo uso de recursos. En el caso de la implementación del decodificador en hardware se analizaron los tres bloques, BMU, ACS y SMU en base a las tablas XII, XIII y XIV respectivamente. Se observa entonces que el

bloque que más recursos consume el de la unidad de cálculo de las métricas de nodo -ACS-.

Una vez identificado el bloque con mayor consumo de recursos en hardware, se procede a implementarlo mediante código en la plataforma de software que en este caso sería el MIPS.

El esquema aproximado de un primer Co-diseño Hardware-Software el cual optimice área se muestra en la figura 4.23. Se aprecia que el ACS se encuentra codificado en la implementación en software, es decir en el MIPS y ya no es parte del hardware del decodificador, y la operación de TB sería realizada por la implementación en hardware.

Según el programa que sigue la implementación en software del decodificador Viterbi, el MIPS entrega un bit decisión a la vez y no los 64 en paralelo como lo hace la implementación en hardware, debido a que el procesador trabaja de manera secuencial. Entonces los bits decisión se envían en serie, y no en vectores de 64 bits como requiere recibirlos el bloque TB en hardware.

Para solucionar este inconveniente se implementa la unidad de interfaz de memoria, la cual debería encargarse de recibir los bits

decisión y agruparlos en un vector de 64 bits y luego enviar este vector al bloque TB, este bloque no efectúa operación alguna mientras no reciba la señal habilitadora de dato válido llamada en la implementación *Datavalid*, la cual es generada también por la interfaz de memoria cada vez que haya agrupado los 64 bits y están listos para ser leídos por el SMU.

Por otra parte, si se quiere optimizar el tiempo utilizado por el decodificador, se procede primero a implementar el diseño como un programa en software, en este caso en el MIPS. Luego se verifica si se cumplen los requerimientos de frecuencia, de no ser así se analiza cual es el bloque que más tiempo toma en hacer una operación y se lo implementa en hardware.

Según las figuras 4.5 y 4.6 se puede observar que la operación del decodificador que genera un mayor retardo es el cálculo de métricas de nodo y de rama. Por esto se propone como Co-diseño Hardware-Software para optimización en tiempo, que el BMU y ACS sean implementados en hardware, mientras que el bloque TB quedaría en la implementación en software, tal como lo muestra la figura 4.24.

De esta manera se obtiene un vector decisión por cada ciclo de reloj, es decir, cada 10 nanosegundos (en caso de que sea implementado en un FPGA de 100Mhz), lo cual representa un tiempo mucho menor a 98.66 microsegundos que es el tiempo que tarda el MIPS en generar esos vectores decisión.

Luego esos vectores decisión, pasan a una unidad de control de memoria, la cual debería transmitirlos de manera serial hacia el bloque TB el cual se encuentra implementado en el MIPS. Una vez que el MIPS termine de realizar la operación de rastreo hacia atrás estos bits decodificados pueden ser enviados hacia el exterior o hacia la unidad de control de memoria para que esta los envíe a la aplicación que los requiera.



CIB-ESPOL

CONCLUSIONES Y RECOMENDACIONES

Luego de haber finalizado el presente trabajo podemos concluir que:

1. Pudimos comprobar la importancia que tienen las técnicas codificación de canal en los actuales estándares de comunicación inalámbricas, su aplicación ayuda a reducir considerablemente errores en la recepción, si se compara con sistemas que no aplican codificación. como lo indica la figura 3.11 donde se observa el comportamiento de un canal sin codificación y con codificación para diferentes valores de longitudes restrictivas.

2. Los estudios realizados con las diferentes simulaciones de canal nos permitieron analizar y elegir las mejores configuraciones y parámetros para obtener un decodificador lo más eficiente posible, evitando una implementación muy compleja y a la vez sin sacrificar recursos. Entre el más importante se destaca la elección de los bits de cuantificación para la decisión suave que permitió apenas una diferencia de ganancia de 0.5 dB con respecto a la ideal, como se muestra en la figura 3.14.

3. La implementación del decodificador de Viterbi en hardware tiene una tasa de decodificación de bits de 100 Mbps y puede operar a una velocidad máxima de 163.82 MHz, como lo muestra la tabla XVII en el capítulo 4, con esto se cumple con el requerimiento de velocidad del estándar IEEE 802.11a que corresponde a una tasa de bits de 48 Mbps, para una tasa de codificación de ½ y longitud restrictiva K=7.

4. Para aplicaciones de alta velocidad en el procesamiento de señales usando el mismo estándar, puede lograrse mayores tasas de datos hasta 54 Mbps, si se combina otro tipo de modulación y tasa de codificación, tal como lo muestra la tabla III.

5. La implementación del decodificador Viterbi en software constituye para nosotros un desafío cumplido, debido a que nuestro diseño del procesador MIPS fue capaz de soportar eficazmente una gran carga de operaciones y manejo de datos. El desarrollar un programa sobre una plataforma estándar beneficia que sea portable para otras generaciones de MIPS y pueda ser reutilizado para otras aplicaciones de control y corrección de errores siendo esto una buena aportación al campo de codificación de canal.

6. Como trabajo futuro podemos recomendar implementar el decodificador Viterbi como un sistema combinado Hardware-Software y de esta manera introducirse más a fondo en el campo de sistemas embebidos.

7. Otro trabajo interesante sería implementar el esquema completo de un transmisor-receptor en hardware, teniendo el transmisor y receptor en diferentes implementaciones.

8. Puede considerarse también el desarrollo de un decodificador Viterbi configurable, para diferentes estándares de comunicaciones inalámbricas, así como lo demuestran trabajos anteriores [8][9][10][11][12][13] que sirvieron de referencia para este proyecto.

9. El diseño e implementación del procesador MIPS permite su escalabilidad, es decir agregar funcionalidades adicionales como por ejemplo memoria caché, que soporte paralelismo o incluirle co-procesadores de punto flotante.

El desarrollo de este proyecto ha sido por demás una experiencia enriquecedora, que nos permitió poner en práctica los conceptos adquiridos en la etapa universitaria en la ESPOL, adquirir nuevos conocimientos y

relacionar la teoría con la práctica. Además nos permitió interactuar y aprender con profesionales destacados de la industria del diseño electrónico en Guadalajara, México.

BIBLIOGRAFÍA

1. IEEE standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements Part II: wireless LAN medium access control (MAC) and physical layer (PHY) specifications. Standard Number: EEE Std802.11g-2003 (Amendment to IEEE Std 802.11, 1999 Edn. (Reaff 2003)

2. Proakis John. *"Digital Communications"*. McGraw Hill. Cuarta Edición

3. Costello T., Shu L., Error Control Coding: Fundamentals and Applications, Prentice-Hall, Octubre 2004.

4. Johannesson Rolf, Kamil Sh. Zigangirov *"Fundamentals of Convolutional Coding"*.

5. MacKay David J.C. *"Information theory, inference and learning algorithms"*. 2003

6. Introducción a Códigos Convolutivos
   http://en.wikipedia.org/wiki/Convolutional_code

7. Codificación de canal y conceptos generales.
   http://www.isa.cie.uva.es/proyectos/codec/teoria1.html

8. Savolainen Marko, Nurmi Jari "Reusable Viterbi Decoder Implementation" Institute of Digital and Computer Systems, Tampere University of Technology, Finlandia.

9. Rasheed Rizwan, Menouni Hayar Aawatif. "Reconfigurable Viterbi Decoder for Mobile Platform" Mobile Communications Department, Institute Eurecom. France

10. Feygin G. , Chow P., Gulak G., Chappel J., Goodes G., Hall O., Sayes A., Singh S.t, Smith M., Wilton S., A VLSI Implementation of a Cascade Viterbi Decoder with Traceback.

11. Tang J., Parhi K., Viterbi Decoder for High-Speed Ultra-Wideband Communication Systems, Dept. of Electrical and Computer Engineering. University of Minnesota, 2005.

12. Benaissa M., Zhu Y., A Novel High-Speed Configurable Viterbi Decoder for Broadband Access, Hindawi Publishing Corporation, Septiembre 2003.

13. Ashry M., Salih A., Abd-El-Galil, An Implementation Scheme for a Viterbi Decoder,

14. Patterson David A, Hennessy John L. *"Computer Organization and Design: The Hardware/Software Interface".* Tercera Edición.

15. Arquitectura MIPS
http://en.wikipedia.org/wiki/MIPS_architecture

16. Hennessy John L. "*Computer Architecture: A Quantitative Approach".* Tercera Edición.

17. Referencia de Instrucciones MIPS
http://www-id.imag.fr/Laboratoire/Membres/Marangozova-Martin_Vania/teaching/NACHOS/NACHOS_j/NACHOS_DOC/MIPSir.html

18. Organización de Computadoras: Segmentación
http://www.cise.ufl.edu/~mssz/CompOrg/CDA-pipe.html

19. Xilinx, DS099 Spartan-3 Complete datasheet, Abril 26, 2006.

20. G. Bosman "A Survey on Hardware/Software Codesign Representation Models". Vrije Universiteit

21. De Micheli and Rajesh K., "Hardware/software co-design". Proceedings of the IEEE, 85(3):349–365, Marzo 1997.

22. Schulz Stephan, Rosemblit Jerzy, Mrva Michael, Buchenrieder Klaus, Model Based Codesign. IEEE Computer Society Press. Los Alamitos, CA, USA. Agosto 1998.

23. Vahid Frank, Partitioning Sequential Programs for CAD Using a Three-Step Approach, ACM Transactions on Design Automation of Electronic Systems, Vol. 7, No. 3, Julio 2002.

24. Edwards S., Lavagno L., Lee E., Sangiovanni- Vincetelli A., "Design of Embedded Systems: Formal Models, Validation and Synthesis". Proceedings of the IEEE, Vol.85, No.3, pp 366-390, Marzo1997.

ESPOL
CIB-ESPOL