INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

Trabajo Terminal
2014 – B024

"Procesador de Punto Flotante para Análisis de Señales"

Que para obtener del Título de:

"Ingeniero en Sistemas Computacionales"

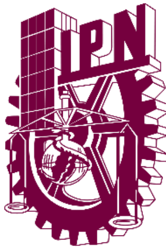
Presenta

Morales Rodríguez Úrsula Samantha.

Directores

Castillo Cabrera Gelacio
Molina Lozano Herón

México D.F. a 17 de Diciembre de 2015.



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA

TT 2014 – B024 *17 de Diciembre de 2015.*

Documento Técnico

“Procesador de Punto Flotante para Análisis de Señales”

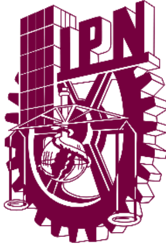
Presenta
Morales Rodríguez Úrsula Samantha[1]

Directores
Dr. Castillo Cabrera Gelacio. **Dr. Molina Lozano Herón.**

RESUMEN

El presente trabajo consiste en diseñar y programar utilizando FPGA’s un procesador que realice operaciones de punto flotante para la realización de tareas específicas. Además, se utilizarán componentes externos como podrían ser sensores que generen señales convertidas a formato digital, es decir, la conversión analógico-digital no es propia del procesador. La señal sensada podría contener información específica desde códigos de identificación, hasta señales biológicas. La tarea a realizar por parte del procesador, será una de las que se mencionan a continuación: promedio de un conjunto de datos, identificación de códigos o identificación de umbrales.

[1] umoralesr1100@alumno.ipn.mx



ESCUELA SUPERIOR DE CÓMPUTO

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE FORMACIÓN INTEGRAL E INSTITUCIONAL

COMISIÓN ACADÉMICA DE TRABAJO TERMINAL

México, D.F. a 12 de Enero de 2016.

DR. FLAVIO ARTURO SÁNCHEZ GARFIAS
PRESIDENTE DE LA COMISIÓN ACADÉMICA
DE TRABAJO TERMINAL
P R E S E N T E

Por medio del presente, se informa que la alumna que integra el ***TRABAJO TERMINAL 2014 – B024***, titulado **"Procesador de Punto Flotante para Análisis de Señales"** concluyó satisfactoriamente su trabajo.

Los discos (DVDs) fueron revisados ampliamente por sus servidores y corregidos, cubriendo el alcance y el objetivo planteados en el protocolo original y de acuerdo a los requisitos establecidos por la Comisión que Usted preside.

ATENTAMENTE.

Dr. Castillo Cabrera Gelacio	**Dr. Molina Lozano Herón**

ADVERTENCIA.

“Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan.”

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.
Información adicional sobre este reporte técnico podrá obtenerse en:
La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

AGRADECIMIENTOS.

Mi principal agradecimiento en todos y cada uno de los aspectos de mi vida es al Universo, por haber conspirado de una manera perfecta y con las condiciones idóneas para que el día de hoy me encuentre presente y disfrutando de cada logro obtenido en mi recorrido, tal y como el espacio y el tiempo me lo han permitido durante mi estadía.

Agradezco de una manera muy especial a mis padres por haberme dado la vida y formar parte de la mayor conspiración en ella. Gracias a María del Carmen Rodríguez Sanabria por ser la mejor mamá del mundo, amorosa, cariñosa, paciente y protectora, todos mis logros conseguidos son gracias a ella, pues sin ella, no habría podido llegar hasta donde ahora me encuentro, gracias infinitas a mi guía, a mi hermosa, linda e inteligente madre. Gracias a José Rafael Morales Lozano por ser mi guia en la vida, por su cariño, por sus sabios consejos y por todo el apoyo brindado para que pudiera terminar otra etapa más en mi vida, gracias a mi mentor, es decir, gracias a mi padre.

Específicamente, agradezco de todo corazón a mis directores del presente trabajo terminal, pues gracias a su asesoría y dirección fue posible cumplir con los objetivos planteados desde un principio, y con ello una culminación exitosa en el proyecto. Agradezco de manera muy especial al Dr. Castillo Cabrera Gelacio por todo el apoyo brindado incondicionalmente, por su paciencia y la confianza brindada hacia mi persona, por su dirección, guía y preocupación para que mi aprendizaje fuera el adecuado y así poder desarrollar un buen trabajo, tal y como se logró. Del mismo modo agradezco al Dr. Molina Lozano Herón por su paciencia y comprensión, así como por todo el conocimiento brindado, proporcionándome una base sólida para el desarrollo del presente trabajo terminal.

Estoy agradecida con todos los profesores de la Escuela Superior de Cómputo con quien tome clase, porque es gracias a ellos y a su enseñanza que yo aprendí lo necesario para tener un título profesional el día de hoy. Así mismo, agradezco en general a todos los profesores que formaron parte de mi aprendizaje durante todo mi camino escolar recorrido. Agradezco a todos mis compañeros de mis distintas escuelas, porque de ellos también tuve un aprendizaje que me forma hoy en día. Gracias a mis hermanos por su apoyo y por ayudarme en mis trabajos cuando lo necesitaba, pues gracias a ellos también pude superarme en este aspecto de mi vida. En general, agradezco a todas aquellas personas que me apoyaron durante todo mi camino para poder alcanzar esta meta.

Dedico este trabajo terminal a mis padres, pues solo ellos son merecedores de lo mejor de mí, ya que este trabajo terminal es resultado de sus esfuerzos. También lo dedico a mi abuelo Jesús Rodríguez Garduño, pues fue a él a quien le prometí terminar mi carrera en contra de todas las adversidades encontradas. Lo dedico en general a toda mi familia. Dedico el presente trabajo terminal de manera muy especial y con todo el amor que el infinito en acto me permite, a los frutos de mi razón de ser, pues ellos son el motor por quienes he luchado y siempre lucharé para alcanzar cada meta en mi vida y así poder brindarles lo mejor de mí y del mundo entero para trascender.

Índice General

1. CAPITULO 1 16
1.1. INTRODUCCIÓN. 16
1.2. DEFINICIÓN DE LA PROBLEMÁTICA. 17
1.3. OBJETIVO. 18
1.3.1. Objetivo General. 18
1.3.2. Objetivos Específicos. 18
1.4. ESTADO DEL ARTE. 18
2. CAPITULO 2. 21
2.1 MARCO TEORICO. 21
2.1.1 Arquitectura de Computadoras. 21
2.1.2 Procesador 21
2.1.3 Arquitecturas Harvard y Von Neumann 22
2.1.4 Procesadores CISC y RISC 24
2.1.5 Procesador MIPS 25
2.1.6 Arquitectura MIPS 25
2.1.7 Microarquitecturas MIPS 27
2.1.8 Paralelismo 27
2.1.9 Transistores MOS 28
2.1.10 Fabricación CMOS 30
3 CAPITULO 3 31
3.1 Dispositivos Programables 31
3.1.1 CPLD 31
3.1.2 FPGA 32
3.2 Tecnologías de Desarrollo. 34
3.2.1 Lattice Diamond 34
3.2.2 ispLEVER Classic Project Navigator 35
3.2.3 ISE Design Suite 14.6 35
3.2.4 Dsch Lite 3.5 36
3.2.5 LTspice IV 37
3.2.6 MENTHOR Graphics 38
4 CAPITULO 4 39
4.1 Visión General. 39

4.2 Definición del formato de las instrucciones..40
4.2.1 Instrucciones tipo R. 40
4.2.2 Instrucciones tipo J. 40
4.2.3 Instrucciones de transferencia de datos. 41
4.3 Modos de direccionamiento. 42
4.4 Set de Instrucciones 42
4.4.1 Carga / Almacenamiento. 42
4.4.2 Aritméticas. 43
4.4.3 Salto 44
4.4.3.1 Salto condicional: Salto si no es igual (BNEQ). 44
4.4.3.2 Salto incondicional: Salto (B). 44
4.5 Definición del formato del tipo de dato (punto flotante). 45
4.5.1 Conversión de numero decimal a binario en punto flotante.................................45
4.5.2 Conversión de un número binario en punto flotante a número decimal. 47
4.5.3 Ejemplos de valores numéricos en formato de Punto Flotante según el estándar IEEE-754. 50
4.6 Componentes del procesador 51
4.6.1 Contador de Programa (CP) 51
4.6.2 Memoria de Programa 51
4.6.3 Banco de Registros 52
4.6.4 Memoria de Datos 53
4.6.5 Unidad de Control 53
4.6.6 Unidad de Punto Flotante 56
4.6.6.1 Suma en Punto Flotante 57
4.6.6.2 Resta en Punto Flotante 59
4.6.6.3 Multiplicación en Punto Flotante 61
4.6.6.4 División en Punto Flotante 63
4.6.6.5 Excepciones 65
4.7 Ruta de Datos. 66
4.7.1 Fases en la ejecución de una instrucción. 66
4.7.2 Instrucciones de trasferencia de datos: Carga (LW). 69
4.7.3 Instrucciones de trasferencia de datos: Almacenamiento (SW). 70
4.7.4 Instrucciones Aritmético-Lógicas: Tipo R 71

4.7.5 Instrucciones de Salto: BEQ 72
4.7.6 Instrucciones de Salto: B 73
5 CAPITULO 5 74
5.1 Aplicación. 74
5.1.1 Redes Neuronales [17] [18]. 74
5.1.1.1 Redes neuronales artificiales 75
5.1.1.2 Red ADALINE 78
5.1.1.3 Filtro Adaptativo. 79
5.1.1.4 Algoritmo de la red neuronal ADALINE sobre el procesador en punto flotante. 81
6 CAPITULO 6 82
6.1 Implementación de la Unidad de Punto Flotante 82
6.1.1 Módulo Check 82
6.1.1.1 Desnormalización Inicial. 85
6.1.1.2 Verificación por operandos erróneos. 85
6.1.1.3 Verificación por operandos en cero. 86
6.1.2 Módulo Ready 87
6.1.2.1 Resta del sesgo al exponente 88
6.1.2.2 Comparación entre exponentes. 88
6.1.2.3 Sumatoria al exponente 89
6.1.2.4 Corrimiento sobre la mantisa. 89
6.1.2.5 Extensor de signo. 90
6.1.2.6 Multiplexor entre mantisa y exponente. 90
6.1.3 Módulo COMPUTE 91
6.1.3.1 Multiplexor del módulo compute 95
6.1.3.2 Comparador entre mantisas 95
6.1.3.3 Selector de operación y signo 96
6.1.3.4 Sumador / Restador. 97
6.1.3.4.1 Ripple 102
6.1.3.4.2 Fast Carry-Ahead 105
6.1.3.5 Multiplicador. 108
6.1.3.5.1 Multiplicador (Unsigned Array Multiplier) 109
6.1.3.6 Divisor. 113

6.1.3.7 Selector de exponente. 113
6.1.3.8 Selector del resultado de la operación. 114
6.1.4 Módulo Set 114
6.1.4.1 Sumatoria del sesgo. 115
6.1.4.2 Multiplexor de salida. 116
6.1.4.3 Normalización final. 116
6.1.4.4 Verificador. 117
6.2 Implementación de la Memoria de Datos. 118
6.3 Implementación de la memoria de programa. 119
6.4 Implementación del banco de registros. 121
6.5 Implementación del contador de programa. 121
6.6 Implementación del sumador del contador 122
6.7 Implementación del contador de direcciones. 122
6.8 Implementación de la Unidad de Control 123
7 CAPITULO 7 125
7.1 Prueba y resultados. 125
7.1.1 Inicialización. 127
7.1.2 Ciclos 128
7.1.2.1 Primer muestra 128
7.1.2.2 Segunda muestra 130
7.1.2.3 Tercera muestra 132
7.1.2.4 Cuarta muestra 134
7.1.2.5 Quinta muestra 136
7.1.2.6 Sexta muestra 138
7.1.2.7 Séptima muestra 140
7.1.2.8 Octava muestra 142
7.1.2.9 Novena muestra 144
7.1.2.10 Decima muestra 146
7.1.3 Resultado. 148
CONCLUSIONES 149
CÓDIGOS 150
Códigos del Módulo CHECK 150

1. Verificación por operandos erróneos. 150
2. Desnormalizacion Inicial. 150
3. Verificación por operandos en cero. 151
4. Paquete del módulo Check....151
5. Top del módulo Check....152
Codigos Módulo READY....153
6. Resta del sesgo al exponente 153
7. Comparación....153
8. Resta....153
9. Paquete del comparador. 154
10. Top del Comparador....154
11. Sumatoria al exponente. 154
12. Corrimiento sobre la mantisa. 155
13. Extensor de signo. 156
14. Multiplexor entre mantisa y exponente....156
15. Paquete del Módulo Ready. 156
16. Top del Módulo Ready....157
Módulo Compute 159
17. Multiplexor del módulo compute....159
18. Comparador entre mantisas. 159
19. Selector de operación y signo....159
20. Sumador / Restador 160
21. Paqute Sumador – Restador. 161
22. Top del Sumador – Restador. 162
23. Multiplicador....166
24. Divisor....171
25. Selector de exponente....176
26. Selector del resultado de la operación. 176
27. Paquete del módulo Compue....176
28. Top del Módulo Compute. 178
Modulo Set....180
29. Sumatoria del sesgo. 180

30. Multiplexor de salida. 180
31. Normalización Final. 181
32. Verificador. 182
33. Paquete del módulo Set 183
34. Top del Módulo Set 184
Unidad de Punto Flotante (UPF) 185
35. Paquete de la UPF. 185
36. Top de la UPF 186
Banco de Registros 188
Memoria de Datos 189
Memoria de Programa 190
Contador de Programa 191
Sumador del Contador 192
Contador de Direcciones. 192
37. Sumador 192
38. Acumulador 193
39. Paquete del contador de direcciones 193
40. TOP del Contador de Direcciones 194
Unidad de Control 195
Paquete del Procesador de Punto Flotante 196
TOP del Procesador de Punto Flotante para el Analisis de Señales. 199
BIBLIOGRAFÍA 201

Índice de Figuras.

FIG 2.1. ARQUITECTURA VON NEUMANN....23
FIG 2.2 HARVARD ARCHITECTURE 24
FIG 2.3 FORMATO DE INSTRUCCIONES TIPO MIPS 26
FIG 2.4 SILICON LATTICE....28
FIG 2.5 DOPANT ATOM (AS) 29
FIG 2.6 DOPANT ATOM (B) 29
FIG 3.1 MAQUETA DIDÁCTICA GENÉRICA CON EL CPLD ISPMACH4256ZE....32
FIG 3.2 MAQUETA DIDÁCTICA GENÉRICA CON EL FPGA MACHXO3L....34
FIG 3.3 AMBIENTE DE DESARROLLO DE LATTICE DIAMOND....34
FIG 3.4 ENTORNO GRAFICO DE ISPLEVER CLASSIC PROJECT NAVIGATOR 35
FIG 3.5 ENTORNO GRAFICO DE ISE DESIGN SUITE 14.6....36
FIG 3.6 ENTORNO GRAFICO DE DSCH LITE 3.5....37
FIG 3.7 ENTORNO GRAFICO LTSPICE IV 37
FIG 4.1 DIAGRAMA DE BLOQUES DEL PROCESADOR EN PUNTO FLOTANTE PARA EL ANÁLISIS DE SEÑALES. 39
FIG 4.2 DIAGRAMA DEL CONTADOR DE PROGRAMA 51
FIG 4.3 DIAGRAMA DE LA MEMORIA DE PROGRAMA 51
FIG 4.4 DIAGRAMA DEL BANCO DE REGISTROS....52
FIG 4.5 DIAGRAMA DE LA MEMORIA DE DATOS....53
4.6 DIAGRAMA DE LA MÁQUINA DE ESTADOS FINITA PARA LA UNIDAD DE CONTROL 54
FIG 4.7. DIAGRAMA GENERAL DE LA UNIDAD DE PUNTO FLOTANTE 56
FIG 4.8 DIAGRAMA DEL ALGORITMO DE SUMA EN PUNTO FLOTANTE 58
FIG 4.9 DIAGRAMA DEL ALGORITMO DE RESTA EN PUNTO FLOTANTE 60
FIG 4.10 DIAGRAMA DEL ALGORITMO DE MULTIPLICACIÓN EN PUNTO FLOTANTE 62
FIG 4.11 DIAGRAMA DEL ALGORITMO DE DIVISIÓN EN PUNTO FLOTANTE 64
FIG 4.12. DIAGRAMA DE BLOQUES (BÚSQUEDA DE LA INSTRUCCIÓN) 66
FIG 4.13. DIAGRAMA DE BLOQUES (DECODIFICACIÓN DE LA INSTRUCCIÓN) 66
FIG 4.14. DIAGRAMA DE BLOQUES (OBTENCIÓN DE LOS OPERANDOS)....67
FIG 4.15. DIAGRAMA DE BLOQUES (EJECUCIÓN DE LA OPERACIÓN)....67
FIG 4.16. DIAGRAMA DE BLOQUES (ALMACENAMIENTO DEL RESULTADO) 68
FIG 4.17. RUTA DE DATOS (INSTRUCCIÓN DE CARGA) 69
FIG 4.18. RUTA DE DATOS (INSTRUCCIÓN DE ALMACENAMIENTO)....70
FIG 4.19. RUTA DE DATOS (INSTRUCCIOES TIPO R) 71
FIG 4.20. RUTA DE DATOS (SALTO CONDICIONAL) 72
FIG 4.21. RUTA DE DATOS (SALTO INCONDICIONAL) 73
5.1.1.1 UNA NEURONA (ARTIFICIAL) SIMPLE 75
5.1.1.2 NEURONA CON SESGO....76
5.1.1.3 NEURONA DE MÚLTIPLES ENTRADAS 77
5.1.1.4 FUNCIÓN DE TRANSFERENCIA LINEAR 78
5.1.1.5 RED ADALINE....78
5.1.1.6 BLOQUE LÍNEA DE RETARDO MUESTREADO 79
5.1.1.7 FILTRO ADAPTATIVO ADALINE 80
5.1.1.8 SISTEMA CANCELADOR DE RUIDO....80
FIG 6.1 SIMULACIÓN DE LA UPF 82
FIG 6.2 DIAGRAMA DEL MÓDULO CHECK....82
FIG 6.3 PRIMERA SIMULACIÓN DEL MÓDULO CHECK 83
FIG 6.4 SEGUNDA SIMULACIÓN DEL MÓDULO CHECK 84

FIG 6.5 TERCERA SIMULACIÓN DEL MÓDULO CHECK 85
FIG 6.6 SIMULACIÓN DEL MÓDULO DESNOMALIZACIÓN 85
FIG 6.7 SIMULACIÓN DEL MÓDULO VERIFICACIÓN DE OPERANDOS 86
FIG 6.8 PRIMERA SIMULACIÓN DEL MÓDULO OPERANDOS EN CERO 86
FIG 6.9 SEGUNDA SIMULACIÓN DEL MÓDULO OPERANDOS EN CERO 87
FIG 6.10 DIAGRAMA DEL MÓDULO READY 87
FIG 6.11 SIMULACIÓN DEL MÓDULO READY 87
FIG 6.12 SIMULACIÓN DEL MÓDULO RESTA DEL SESGO 88
FIG 6.13 SIMULACIÓN DEL MÓDULO COMPARACIÓN 89
FIG 6.14 SIMULACIÓN DE MÓDULO SUMATORIA DEL EXPONENTE 89
FIG 6.15 SIMULACIÓN DEL MÓDULO CORRIMIENTO DE MANTISA 90
FIG 6.16 SIMULACIÓN DEL MÓDULO EXTENSOR DE SIGNO 90
FIG 6.17 SIMULACIÓN DEL MULTIPLEXOR MANTISA-EXPONENTE 91
FIG 6.18 DIAGRAMA DEL MÓDULO COMPUTE 91
FIG 6.19 PRIMERA SIMULACIÓN DEL SUMADOR 92
FIG 6.20 SEGUNDA SIMULACIÓN DEL SUMADOR 92
FIG 6.21 PRIMERA SIMULACIÓN DEL RESTADOR 93
FIG 6.22 SEGUNDA SIMULACIÓN DEL RESTADOR 93
FIG 6.23 RESULTADOS DE LA MULTIPLICACIÓN 93
FIG 6.24 PRIMERA SIMULACIÓN DEL MULTIPLICADOR 94
FIG 6.25 SEGUNDA SIMULACIÓN DEL MULTIPLICADOR 94
FIG 6.26 PRIMERA SIMULACIÓN DEL DIVISOR 95
FIG 6.27 SIMULACIÓN DEL MULTIPLEXOR DEL MÓDULO COMPUTE 95
FIG 6.28 SIMULACIÓN DEL MÓDULO COMPARADOR DE MANTISAS 95
FIG 6.29 SIMULACIÓN DEL MÓDULO DE OPERACIÓN Y SIGNO - SUMA 96
FIG 6.30 SIMULACIÓN DEL MÓDULO DE OPERACIÓN Y SIGNO - RESTA 96
FIG 6.31 SIMULACIÓN DEL MÓDULO DE OPERACIÓN Y SIGNO – MULTIPLICACIÓN Y DIVISIÓN 97
FIG 6.32 PRIMERA SIMULACIÓN DE SUMADOR 97
FIG 6.33 SEGUNDA SIMULACIÓN DE SUMADOR 98
FIG 6.34 TERCERA SIMULACIÓN DE SUMADOR 98
FIG 6.35 CUARTA SIMULACIÓN DE SUMADOR 98
FIG 6.36 QUINTA SIMULACIÓN DE SUMADOR 99
FIG 6.37 SEXTA SIMULACIÓN DE SUMADOR 99
FIG 6.38 PRIMERA SIMULACIÓN DE RESTADOR 100
FIG 6.39 SEGUNDA SIMULACIÓN DE RESTADOR 100
FIG 6.40 TERCERA SIMULACIÓN DE RESTADOR 100
FIG 6.41 CUARTA SIMULACIÓN DE RESTADOR 101
FIG 6.42 QUINTA SIMULACIÓN DE RESTADOR 101
FIG 6.43 SEXTA SIMULACIÓN DE RESTADOR 101
FIG 6.44 DIAGRAMA DE BLOQUES DEL SUMADOR/RESTADOR DE 4 BITS POR EL MÉTODO RIPPLE. 102
FIG 6.45 REPORTE DE TIEMPO ESTÁTICO, UTILIZANDO HERRAMIENTAS DE ANÁLISIS DEL ENTORNO ISPLEVER CLASSIC. 103
FIG 6.46 RESUMEN DE RECURSOS DEL DISPOSITIVO 103
FIG 6.47 DIAGRAMA RTL SCHEMATIC DEL SUMADOR/RESTADOR DE 8 BITS POR MÉTODO DE RIPPLE. 104
FIG 6.48 RESUMEN DE LA DENSIDAD OCUPADA EN EL DISPOSITIVO. 104
FIG 6.49 RETARDOS MÁXIMOS DEL SUMADOR/RESTADOR POR MÉTODO DE RIPPLE. 105
FIG 6.50 DIAGRAMA A BLOQUES DEL SUMADOR RESTADOR DE 8 BITS POR EL MÉTODO DE ACARREO ANTICIPADO. 105
FIG 6.51 REPORTE DE TIEMPO ESTÁTICO, UTILIZANDO HERRAMIENTAS DE ANÁLISIS DEL ENTORNO ISPLEVER CLASSIC. 106
FIG 6.52 RESUMEN DE RECURSOS DEL DISPOSITIVO 106
FIG 6.53 DIAGRAMA RTL SCHEMATIC DEL SUMADOR/RESTADOR DE 8 BITS POR MÉTODO DE ACARREO ANTICIPADO. 107

FIG 6.54 RESUMEN DE LA DENSIDAD OCUPADA EN EL DISPOSITIVO. 107
FIG 6.55 RETARDOS MÁXIMOS DEL SUMADOR/RESTADOR CON ACARREO ANTICIPADO. 108
FIG 6.56 PRIMERA SIMULACIÓN DEL MULTIPLICADOR 108
FIG 6.57 SEGUNDA SIMULACIÓN DEL MULTIPLICADOR 109
FIG 6.58 TERCERA SIMULACIÓN DEL MULTIPLICADOR 109
FIG 6.59 CUARTA SIMULACIÓN DEL MULTIPLICADOR 109
FIG 6.60 DIAGRAMA A BLOQUES UNSIGNED ARRAY MULTIPLIER 110
FIG 6.61 DIAGRAMA RTL DEL MULTIPLICADOR DE 4 BITS UAM 110
FIG 6.62 REPORTE DE TIEMPO ESTÁTICO DEL MULTIPLICADOR, UTILIZANDO HERRAMIENTAS DE ANÁLISIS DEL ENTORNO ISPLEVER CLASSIC. 111
FIG 6.63 RESUMEN DE RECURSOS DE DISPOSITIVO 112
FIG 6.64 RESUMEN DE LA DENSIDAD OCUPADA EN EL DISPOSITIVO PARA EL DISEÑO DEL MULTIPLICADOR 112
FIG 6.65 RETARDOS MÁXIMOS DEL UNSIGNED ARRAY MULTIPLIER. 113
FIG 6.66 SIMULACIÓN DEL DIVISOR 113
FIG 6.67 SIMULACIÓN DEL MÓDULO SELECTOR DE EXPONENTES 113
FIG 6.68 SIMULACIÓN DEL MODULO SELECTOR DEL RESULTADO DE LA OPERACIÓN 114
FIG 6.69 DIAGRAMA DEL MÓSULO SET 114
FIG 6.70 SIMULACIÓN DEL MÓDULO SET 115
FIG 6.71 SIMULACIÓN DEL MÓDULO SUMATORIA DEL SESGO 115
FIG 6.72 SIMULACIÓN DEL MULTIPLEXOR DE SALIDA 116
FIG 6.73 PRIMERA SIMULACIÓN DEL MÓDULO NORMALIZACIÓN FINAL 117
FIG 6.74 SEGUNDA SIMULACIÓN DEL MÓDULO NORMALIZACION FINAL 117
FIG 6.75 SIMULACIÓN DEL MÓDULO VERIFICADOR 117
FIG 6.76 REPRESENTACIÓN DEL MAPA DE MEMORIA 118
FIG 6.77 SIMULACIÓN DE LA MEMORIA DE DATOS 119
FIG 6.78 SIMULACIÓN DE LA MEMORIA DE PROGRAMA EN INSTRUCCIONES DE TRANSFERENCIA DE DATOS LW 120
FIG 6.79 SIMULACIÓN DE LA MEMORIA DE PROGRAMA EN INSTRUCCIONES TIPO R 120
FIG 6.80 SIMULACIÓN DE LA MEMORIA DE PROGRAMA EN INSTRUCCIONES DE TRANFERENCIA DE DATOS SW 120
FIG 6.81 SIMULACIÓN DE LA MEMORIA DE PROGRAMA EN INSTRUCCIONES DE SALTO CONDICIONAL 120
FIG 6.82 PRIMERA SIMULACIÓN DEL BANCO DE REGISTROS 121
FIG 6.83 SEGUNDA SIMULACIÓN DEL BANCO DE REGISTROS 121
FIG 6.84 SIMULACIÓN DEL MÓDULO CONTADOR DE PROGRAMA 122
FIG 6.85 SIMULACIÓN DEL MÓDULO SUMADOR DEL CONTADOR 122
FIG 6.86 DIAGRAMA DEL CONTADOR DE DIRECCIONES 123
FIG 6.87 SIMULACIÓN DEL MÓDULO CONTADOR DE DIRECCIONES 123
FIG 6.88 PRIMERA SIMULACIÓN DE LA UNIDAD DE CONTROL 123
FIG 6.89 SEGUNDA SIMULACIÓN DE LA UNIDAD DE CONTROL 124
FIG 6.90 TERCERA SIMULACIÓN DE LA UNIDAD DE CONTROL 124
FIG 6.91 CARTA ASM DE LA UNIDAD DE CONTROL 124
FIG 7.1.1 NEURONA ARTIFICIAL CON VALORES Y RESULTADOS ESPERADOS 125
FIG 7.1.2 PROGRAMA EMBEBIDO EN LA MEMORIA DE PROGRAMA 126
FIG 7.1.1.1 SIMULACIÓN DE INICIALIZACIÓN 127
FIG 7.1.2.1.1 PRIMERA SIMULACIÓN DE CARGA 128
FIG 7.1.2.1.2 PRIMERA SIMULACIÓN DE OPERACIÓN Y BRINCO 129
FIG 7.1.2.2.1 SEGUNDA SIMULACIÓN DE CARGA 130
FIG 7.1.2.2.2 SEGUNDA SIMULACIÓN DE OPERACIÓN Y BRINCO 131
FIG 7.1.2.3.1 TERCERA SIMULACIÓN DE CARGA 132
FIG 7.1.2.3.2 TERCERA SIMULACIÓN DE OPERACIÓN Y BRINCO 133
FIG 7.1.2.4.1 CUARTA SIMULACIÓN DE CARGA 134
FIG 7.1.2.4.2 CUARTA SIMULACIÓN DE OPERACIÓN Y BRINCO 135
FIG 7.1.2.5.1 QUINTA SIMULACIÓN DE CARGA 136
FIG 7.1.2.5.2 QUINTA SIMULACIÓN DE OPERACIÓN Y BRINCO 137

FIG 7.1.2.6.1 SEXTA SIMULACIÓN DE CARGA 138
FIG 7.1.2.6.2 SEXTA SIMULACIÓN DE OPERACIÓN Y BRINCO 139
FIG 7.1.2.7.1 SÉPTIMA SIMULACIÓN DE CARGA 140
FIG 7.1.2.7.2 SÉPTIMA SIMULACIÓN DE OPERACIÓN Y BRINCO 141
FIG 7.1.2.8.1 OCTAVA SIMULACIÓN DE CARGA 142
FIG 7.1.2.8.2 OCTAVA SIMULACIÓN DE OPERACIÓN Y BRINCO 143
FIG 7.1.2.9.1 NOVENA SIMULACIÓN DE CARGA 144
FIG 7.1.2.9.2 NOVENA SIMULACIÓN DE OPERACIÓN Y BRINCO 145
FIG 7.1.2.10.1 DÉCIMA SIMULACIÓN DE CARGA 146
FIG 7.1.2.10.2 DÉCIMA SIMULACIÓN DE OPERACIÓN Y BRINCO 147
FIG 7.1.3.1 SIMULACIÓN DEL RESULTADO FINAL 148

Índice de Tablas

TABLA 1 RESUMEN DE PROYECTOS SIMILARES 20
TABLA 2 SET DE INSTRUCCIONES MIPS 26
TABLA 3 INSTRUCCIONES DE CARGA/ALMACENAMIENTO 42
TABLA 4 INSTRUCCIONES ARITMÉTICAS 43
TABLA 5 INSTRUCCIÓN SALTO CONDICIONAL 44
TABLA 6 INSTRUCCIÓN SALTO INCONDICIONAL 45
TABLA 7 FORMATO REPRESENTACIÓN DE UN NUMERO EN PUNTO FLOTANTE (IEEE-754) 45
TABLA 8 VALORES POR BIT DE LA MANTISA 49
TABLA 9. EJEMPLOS DE DATOS NUMÉRICOS CON SU REPRESENTACIÓN EN PUNTO FLOTANTE. 50
TABLA 10 TABLA DE VERDAD PARA LAS 12 SALIDAS DE CONTROL 54
TABLA 11 ENTRADAS / SALIDAS UNIDAD DE CONTROL 55
TABLA 12 RESULTADOS DE LA SUMA 91
TABLA 13 RESULTADOS DE LA RESTA 92
TABLA 14 RESULTADOS DE LA DIVISIÓN 94

1. CAPITULO 1

1.1. INTRODUCCIÓN.

El desarrollo de la computación se ha dado a partir del desarrollo de dos grandes áreas que son el software y el hardware. El desarrollo de hardware a nivel mundial en el sector industrial es considerablemente bajo siendo *ca.* 19%, en comparación al desarrollo de software que abarca *ca.* 81% del mercado [1], porcentajes que muestran las limitaciones de nuevas tecnologías por la falta de hardware. Estadísticamente, el continente Europeo tiene un mayor desarrollo en hardware en comparación al continente Americano [2], consecuentemente México no cuenta con el impulso adecuado desarrollando hardware.

En general, un procesador puede manejar la información utilizando números enteros con signo o sin signo, y además, números de punto flotante. Los números enteros pueden representar datos como son los números, texto o caracteres, es decir, dependiendo del contexto, este tipo de información se puede representar con este tipo de números. Con relación a la representación numérica utilizando números enteros, un procesador estaría limitado a representar números enteros con signo, lo cual limitaría el procesamiento de cantidades numéricas como son las fracciones, los números racionales, los irracionales, y en general, los números reales.

Es por esto que la presente propuesta de trabajo terminal es el desarrollo de un procesador dedicado para la ejecución de tareas específicas realizando operaciones en punto flotante.

El procesador es la parte más importante de un computador, se trata de la unidad encargada de ejecutar instrucciones y procesar datos necesarios para todas las funcionalidades del computador [3]; El desarrollo del procesador constará de cuatro etapas:

a) Para la primera etapa de diseño, se trabajará usando como herramientas dispositivos lógicos reconfigurables, entendiéndose por dispositivos lógicos reconfigurables el uso de los FPGA´s.

b) En la segunda etapa se diseñarán los elementos de hardware en esquemático, particularmente registros, contadores, y circuitos combinacionales de control como son los decodificadores y demultiplexores, que son componentes propios del procesador.

c) En la tercera etapa se realizará el *layout* del procesador, siendo el *layout* el diseño a nivel transistor, por capas, de los circuitos integrados. Las capas están constituidas por material de silicio, óxidos de silicio y nitruros de silicio [4].

d) En la cuarta etapa se realizarán pruebas directamente sobre el FPGA, ya que no se pretende mandar a fabricar el circuito y por consecuencia las pruebas no son realizables en el dispositivo.

1.2. DEFINICIÓN DE LA PROBLEMÁTICA.

El estado actual de desarrollo en hardware en México, por ejemplo, la referencia [6] es el único ejemplo de diseño de un procesador en la ESCOM, por lo que el diseño de procesadores es escaso y como consecuencia hace lento el desarrollo de nuevas tecnologías computacionales.

Con esta propuesta de trabajo terminal se impulsa un amplio panorama de nuevas tecnologías en diseño y desarrollo de hardware, y por consecuente a una ilimitada fuente de ideas para el desarrollo de software.

Por eso mismo es necesario que a nivel académico se desarrolle hardware para el funcionamiento eficiente de diversas aplicaciones, significando que las tareas se realicen de una manera más rápida, con un menor consumo de potencia y/o minimizando el espacio en cuanto a recursos, entendiéndose por hardware a la arquitectura física para el procesamiento de la información, es decir, un procesador.

La columna vertebral del procesador a desarrollar es que la unidad lógica aritmética realice operaciones en punto flotante, cuya ventaja es la precisión de los resultados al procesar operaciones aritméticas, en comparación de la unidad lógica aritmética simple.

Actualmente, las operaciones con punto flotante son implementadas en distintas aplicaciones, siendo estas de telecomunicaciones, control industrial, procesamiento de señales, entre otras. Por consecuencia el uso de las computadoras ha sido incorporado en estas mismas, pero a pesar de ello existe la falta de sistemas dedicados para la realización de tareas específicas de las distintas aplicaciones.

Es necesario que el área de desarrollo de procesadores de índole académica debido a que en las escuelas de ingeniería en sistemas computacionales y/o afines no se cuenta con un procesador en el cual los estudiantes puedan entrar más a detalle en la implementación de nuevos componentes. Es estrictamente necesario hacer el desarrollo de procesadores orientados hacia la academia, pues se considera existen tres beneficios importantes: a) los estudiantes sabrán cómo desarrollar un procesador, tanto de propósito general, como de propósito específico; b) los alumnos tendrán la capacidad de modificarlo en base a lo ya desarrollado y; c) los estudiantes desarrollarán más habilidades lógicas y de diseño de hardware.

La complejidad radica en el desarrollo del procesador para una aplicación específica. En forma general, consiste en la concepción de la idea, un diagrama general de

bloques, diseño con las herramientas de descripción de hardware, translación del diseño por descripción de hardware a diseño en esquemáticos, simulación, translación de nivel esquemático a *layout* y pruebas.

1.3. OBJETIVO.

1.3.1. Objetivo General.

Diseñar un procesador dedicado que realice operaciones lógicas y/o aritméticas en punto flotante mediante dispositivos lógicos reconfigurables (FPGA) para el análisis de señales digitales ya convertidas, proporcionadas por sensores comerciales.

Como parte fundamental de este objetivo, se busca incentivar académicamente el desarrollo de hardware.

1.3.2. Objetivos Específicos.

- Diseñar los componentes del procesador dedicado en punto flotante utilizando VHDL (lenguaje de descripción de hardware).
- Realizar las pruebas de los diseños de los componentes del procesador dedicado sobre el FPGA y/o el CPLD.
- Diseñar a nivel de transistor y de capas de semiconductores (Layout) las celdas básicas de los componentes del procesador.

1.4. ESTADO DEL ARTE.

En el siguiente apartado se describen algunos proyectos relacionados al trabajo que se pretende desarrollar. En la tabla 1 se muestran en forma general las características de los proyectos que encontré relevantes en apoyo a mi propuesta.

En primer lugar se presenta un proyecto publicado en 2013 por *International Conference on Advanced Electronic Systems* (ICAES) desarrollado en la universidad VIT en India [5], posteriormente un proyecto académico desarrollado en la Escuela Superior de Computo del Instituto Politécnico Nacional [6], a continuación un Trabajo Terminal de la Escuela Superior de Computo del Instituto Politécnico Nacional [7].

PROYECTOS DE HARDWARE	**CARACTERÍSTICAS**
Design and Implementation of Single Precision Pipelined Floating Point Co-Processor.	Desarrollo de un co-procesador con arquitectura segmentada para incrementar el rendimiento del diseño usando Verilog HDL. Unidad Lógica Aritmética segmentada de punto flotante para minimizar la potencia e incrementar la frecuencia de operación.
Microprocesador didáctico de arquitectura RISC implementado en un FPGA.	Implementación de un microprocesador en VHDL de 16 bits de arquitectura RISC tipo MIPS con arquitectura Harvard en un FPGA de la familia Spartan 3-A de Xilinx. Unidad Lógica aritmética con esquema de acarreo anticipado por propagación. Cada instrucción ejecutada se lleva a cabo en un ciclo de reloj.

Microprocesador RISC de 16 bits. MIR16B	Desarrollo de un procesador RISC en VHDL e implementado en un FPGA de Xilinx con arquitectura segmentada. Cuenta con un set de instrucciones generales para la implementación de diversos programas. Es un procesador de uso general, dejando abierta la posibilidad de adicionamiento de nuevos componentes.

Tabla 1 Resumen de proyectos similares

Aunque las ventajas del procesador aquí propuesto en este trabajo, respecto a los trabajos citados en la tabla 1, están mencionados de forma implícita (a) en el primer párrafo de la introducción (apartado 1.1) y (b) en los últimos tres renglones del objetivo (apartado 1.3), es importante mencionar, como ventaja y aportación, de este trabajo, la amplia descripción y explicación que se daría con un enfoque didáctico.

En el siguiente capítulo se presenta el marco teórico de este reporte técnico.

2. CAPITULO 2.

2.1 MARCO TEORICO.

2.1.1 Arquitectura de Computadoras.

Arquitectura de computadoras es el termino genérico que se utiliza por lo común para englobar y describir el funcionamiento de una computadora digital y los conceptos involucrados en su desarrollo; sin embargo, la arquitectura hace referencia a la planeación a nivel estructural, la cual considera diversos atributos lógicos y la interconexión de módulos de hardware. Por atributos lógicos entendemos, el número de bits, de una instrucción el número de campos, el número de bits de los operandos y tipos de instrucciones, registros de propósito específico entre otros atributos. El aspecto más importante es establecer los canales de comunicación y la forma en que se disponen para que la transferencia de datos sea exitosa (ver referencia [8]).

2.1.2 Procesador

La unidad central de proceso (CPU-*Central Process Unit* por sus siglas en ingles), viene a ser el cerebro del ordenador y tiene por misión efectuar las operaciones aritmético-lógicas y controlar las transferencias de información a realizar. La CPU está formada por los elementos más críticos de la arquitectura de Von Neumann, es decir, la ALU y la unidad de control, y en la actualidad estos dos componentes están construidos en un solo circuito integrado llamado microprocesador

El microprocesador es conocido metafóricamente como el "cerebro" del ordenador, ya que es el centro de todas las actividades que se producen en él y es el que controla todo el equipo. Es un circuito integrado o chip con la misión de interpretar y ejecutar las instrucciones del programa que se indique.

El microprocesador también es llamado CPU (Central Process Unit) o unidad central de proceso, y coincide con la CPU de la arquitectura de Von Neumann, incluyendo las unidades funcionales de la ALU (Unidad Aritmética-Lógica por sus siglas en ingles) y la CU (Unidad de Control).

El microprocesador o la CPU es, por tanto, la unidad que realiza las operaciones más importantes, además de sincronizar el funcionamiento del resto de unidades. Desarrollando el modelo de CPU del esquema de Von Neumann, un microprocesador moderno se divide en dos unidades funcionales básicas: la unidad de proceso (PU) también conocida como *datapath* (camino de datos) que ejecuta las instrucciones siguiendo una secuencia de operaciones preestablecidas y la unidad de control (UC) que se encarga de vigilar esta secuencia (ver referencia [9]).

2.1.3 Arquitecturas Harvard y Von Neumann

Existen dos clases de arquitecturas de computadoras, llamadas “Arquitecturas Harvard” y “Arquitecturas Von Neumann (Princeton)”, a continuación se definen a grandes rasgos, las características de estos tipos de arquitecturas, que en la actualidad se manejan.

2.1.3.1 Modelo de Von Neumann

La figura **.** muestra el modelo de Von Neumann, un sistema computacional uniprocesador típico consistente de la MU (Memory Unit), la ALU (Arithmetic-Logic Unit), la UC (Unit Control) y la IOU (Input/Output Unir). La MU es un dispositivo de puerto sencillo que consiste de un registro de dirección de memoria (MAR – Memory Address Register) y un registro de memoria de almacenamiento (MBR – Memory Buffer Register), también llamado registro de memoria de datos (MDR – Memory Data Register). Las celdas de memoria son arregladas en forma de varias palabras de memoria, donde cada palabra es una unidad de datos que puede ser escrita o leída. Todas las operaciones de lectura y escritura en memoria utilizan el puerto de memoria. La ALU realiza las operaciones lógicas y aritméticas sobre los elementos de datos en el acumulador (ACC) y/o en el MBR, y típicamente el ACC retiene el resultado de dichas operaciones. La CU consiste de un contador de programa (PC) el cual contiene la dirección de la instrucción extraída de la memoria para ser ejecutada. Dos registros son incluidos en la estructura. Estos pueden ser utilizados para retener datos o valores de direcciones durante el trabajo computacional. Por simplicidad, el subsistema de I/O es mostrado como entrada hacia y salida desde el subsistema de la ALU. En la práctica, la entrada/salida suele ocurrir directamente entre la memoria y los dispositivos de entrada/salida sin utilizar ningún registro del procesador. Los componentes del sistema son interconectados mediante la estructura de un bus múltiple en el cual los datos y las direcciones fluyen. La unidad de control maneja este flujo a través del uso apropiado de las señales de control. [8]

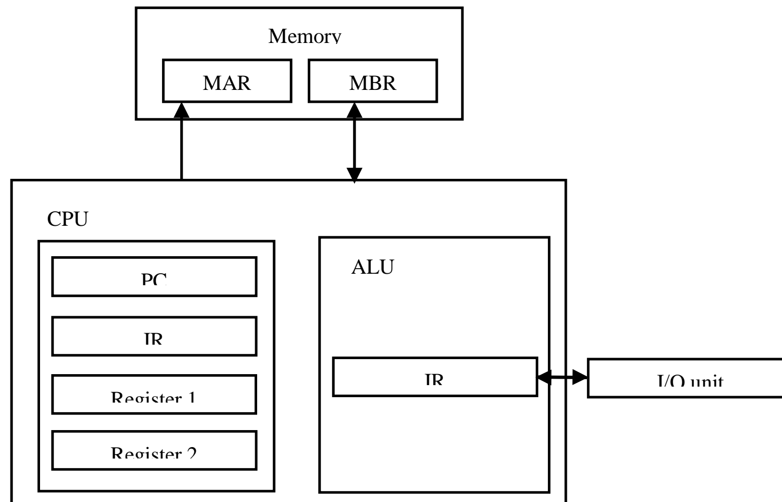


Fig 2.1. Arquitectura Von Neumann

Fuente: Sajjan G. Shiva, (2014), *"Computer Organization, Design, and Architecture"(p. 5)* Boca Raton, FL. CRC Press Taylor & Francis Group. 5th Edition.

2.1.3.2 Arquitectura Harvard

La arquitectura Harvard usa memorias separadas, una memoria para programa y otra memoria para datos, con sus direcciones y buses de datos independientes. Debido a los dos diferentes flujos de datos y de direcciones, no es necesario tener un divisor de tiempo multiplexando los buses de datos y de direcciones. La arquitectura no solo soporta buses paralelos para direcciones y datos, también permite una diferente organización interna tal que cada instrucción puede ser pre-obtenida y decodificada mientras los múltiples datos están siendo obtenidos y puestos en operación. Además, el bus de datos puede tener distinto tamaño en comparación del bus de direcciones. Esto permite optimizar los tamaños de los buses de datos y de direcciones para la rápida ejecución de la instrucción. [10]

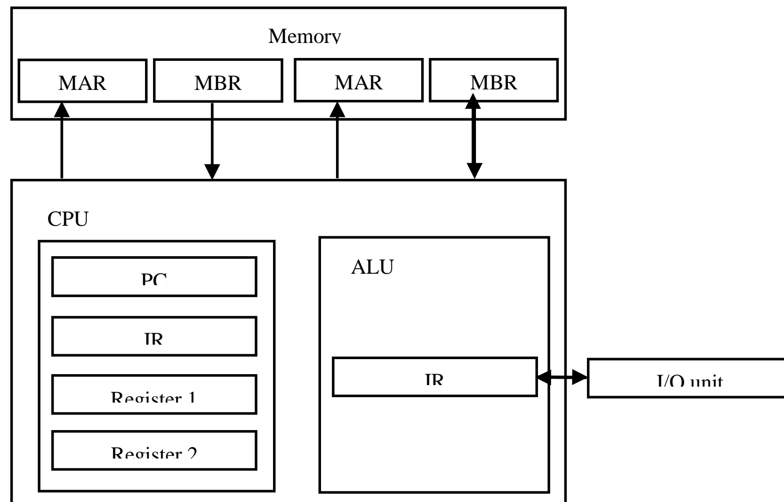


Fig 2.2 Harvard Architecture

Fuente: Sajjan G. Shiva, (2014), *"Computer Organization, Design, and Architecture",(p. 7)* Boca Raton, FL. CRC Press Taylor & Francis Group. 5th Edition.

2.1.4 Procesadores CISC y RISC

Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC) son terminologías comunes usadas mientras se habla acerca de microcontroladores o microprocesadores. [10]

2.1.4.1 Complex Instruction Set Computers

Los procesadores CISC tienen un gran número de instrucciones. Un set de instrucciones grande ayuda a los programadores de lenguaje ensamblador, proporcionando flexibilidad para escribir programas cortos y efectivos. El objetivo de la arquitectura CISC es escribir un programa en tan pocas líneas de código en lenguaje ensamblador como sea posible. Esto fue posible desarrollando en hardware un procesador que pueda entender y ejecutar un número de operaciones. Por ejemplo, en la arquitectura CISC la cual comprende únicamente operandos específicos en la instrucción, y la operación de multiplicación es realizada por hardware. En la mayoría de instrucciones, la construcción de instrucciones complejas directamente en hardware ayuda en dos caminos diferentes. No solamente la implementación en hardware es más rápida, también se guarda espacio en la memoria de programa en el sentido de que el código de instrucción es muy corto en comparación de aquel que para la operación de multiplicación necesite usar la instrucción ADD. [10]

2.1.4.2 Reduced Instruction Set Computers

Los procesadores RISC son más sencillo de diseñar en comparación de los procesadores CISC. Los arquitectos RISC desafiaron si esta creciente complejidad era el enfoque correcto. El principio fundamental del diseño de RISC es lograr el mayor desempeño del sistema posible, donde "sistema" incluye no solo el procesador y la memoria, sino también el compilador.

El desempeño del sistema, en término de tiempo requerido para llevar a cabo determinada tarea, puede ser visualizado como el producto de tres factores:

- Número de instrucciones requeridas para realizar una tarea.
- Número de ciclos de reloj requeridos por instrucción (*Clock Per Instruction*, CPI).
- La longitud de un ciclo de reloj (T). [11]

2.1.5 Procesador MIPS

El diseño del procesador MIPS fue originado a partir de un trabajo realizado en la universidad de Stanford a principio de los años ochenta bajo la dirección del profesor John Hennesy. Diferentes diseños han utilizado el nombre de MIPS, liderados por una confusión acerca de lo que el termino MIPS refiere, es decir, que mientras del termino MIPS refiere a Millones de Instrucciones Por Segundo, se le ha entendido en otro contexto como "*Microprocessor without Interlocked Pipelines Stages* ". [11]

2.1.6 Arquitectura MIPS

La arquitectura MIPS, basada de una arquitectura RISC, de un sistema computacional define tanto instrucciones para enteros como para punto flotante y conjuntos de registros.

La arquitectura MIPS32 es una simple arquitectura RISC de 32 bits. La arquitectura utiliza instrucciones codificadas de 32 bits pero solamente ocho registros de propósito general de 8 bits denominados \$0 - \$7. También utiliza un contador de programa de B bits (PC). El registro \$0 contiene por defecto solo ceros. Las instrucciones básicas son ADD, SUB, AND, OR, SLT, ADDI, BEG, J, LB y SB.

La función y la codificación de cada instrucción son mostradas en la tabla 2.

Instruction	Function	Encoding	Op	Funct
Add $1, $2, $3	Addition: $1 <- $2 + $3	R	000000	100000
Sub $1, $2, $3	substraction: $1 <- $2 - $3	R	000000	100010
And $1, $2, $3	Bitwise and: $1 <- $2 and $3	R	000000	100100
Or $1, $2, $3	Bitwise or: $1 <- $2 or $3	R	000000	100101
slt $1, $2, $3	Set less 26íd: $1 <- 1 if $2 < $3 $1 <- 0 otherwise	R	000000	101010
Addi $1, $2, imm	Add immediate: $1 <- $2 + imm	I	001000	n/a
Beq $1, $2, imm	Branch if equal: PC <- PC + imm x 4[a]	I	000100	n/a
J destination	Jump: PC <- destination[a]	J	000010	n/a
Lb $1, imm($2)	Load byte: $1 <- mem[$2 + imm]	I	100000	n/a
Sb $1, imm($2)	Store byte: mem[$2 + imm] <- $1	I	101000	n/a

Tabla 2 Set de Instrucciones MIPS

Nota. Fuente: Neil H. E. Weste, Money H. David, (2011), "CMOS VLSI DESIGN a circuits and systems perspective"(p. 33), Massachusetts, USA, Ed. Pearson.

Cada instrucción codificada usando una de las tres plantillas: R, I, J. Las instrucciones tipo R (basadas en registros) son usadas para la aritmética y específicamente dos registros fuente y un registro de destino. Las instrucciones tipo I son usadas cuando una constante de 16 bits (también conocida como un inmediato) y dos registros deben ser especificados. Las instrucciones tipo J (saltos) dedican la mayor parte del tamaño de palabra de la instrucción es decir 26 bits al destino. El formato de cada codificación está definido por:

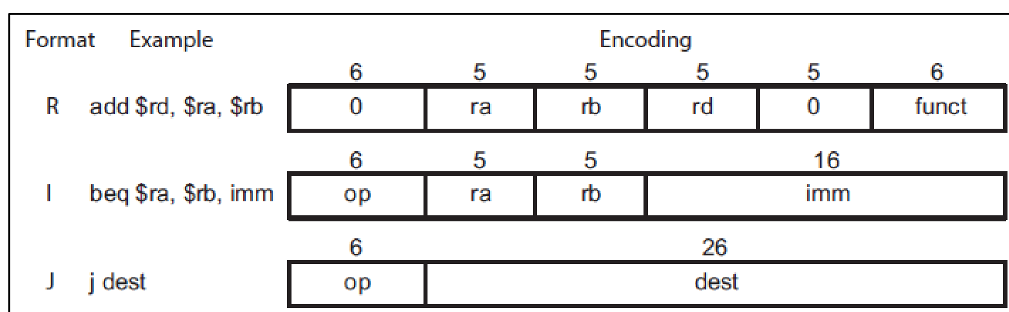


Fig 2.3 Formato de instrucciones tipo MIPS

Los seis bits más significativos de todos los formatos son para el código de operación (op). Todas las instrucciones tipo R comparten el mismo código de operación (OP = 0000) y usan seis bits más de funct para diferencias las operaciones. [12]

2.1.7 Microarquitecturas MIPS

Existen tres distintos tipos de microarquitectura MIPS:

- La microarquitectura de un solo ciclo ejecuta toda una instrucción en un ciclo. Es fácil de explicar y tiene una unidad de control simple. Debido a que se completa la operación en un ciclo, que no requiere de ningún estado non arquitectónico. Sin embargo, el tiempo de ciclo está limitado por la instrucción más lenta.

- La microarquitectura multiciclo ejecuta instrucciones en una serie de ciclos más cortos. Las instrucciones simples se ejecutan en menos ciclos requeridos. Por otra parte, la microarquitectura multiciclo reduce el costo de hardware mediante la reutilización de bloques de hardware costosos tales como sumadores y memorias. Por ejemplo, el sumador se puede utilizar en varios ciclos diferentes para varios propósitos mientras que se realiza una sola instrucción. El microprocesador multiciclo logra esto mediante la adición de varios registros non arquitectónicos para mantener los resultados intermedios. El procesador multiciclo ejecuta sólo una instrucción a la vez, pero cada instrucción se lleva a múltiples ciclos de reloj.

- La microarquitectura pipeline aplica la canalización para la microarquitectura de un solo ciclo. Por lo tanto, puede ejecutar varias instrucciones al mismo tiempo y mejorar el rendimiento significativamente. Para llevar a cabo el segmentado se debe añadir lógica para manejar dependencias al ejecutar instrucciones simultáneamente. También requiere registros de segmentado no arquitectónicos. La lógica y los registros añadidos valen la pena; todos los procesadores de alto rendimiento comerciales utilizar la segmentación hoy en día. [13]

2.1.8 Paralelismo

La velocidad de un sistema se mide en la latencia y el rendimiento de los *tokens* que se mueven a través de un sistema. Se define como *token* al grupo de entradas que se procesan para producir un grupo de salidas. El termino evoca la idea de colocar el fila a los *tokens* en un diagrama de circuito donde al moverlos alrededor se pueda visualizar los datos moviéndose a través del circuito. La latencia de un sistema es el tiempo requerido de un *token* para pasar a través el sistema desde principio a fin. El rendimiento es el número de *tokens* que se pueden producir por unidad de tiempo.

Como se puede imaginar, el rendimiento se puede mejorar mediante el procesamiento de varios *tokens* al mismo tiempo. A esto se le llama paralelismo, y se presenta en dos formas: espacial y temporal. Con el paralelismo espacial, se proporcionan múltiples copias del hardware, por lo que múltiples tareas se pueden hacer al mismo tiempo. Con el paralelismo temporal, una tarea se divide en etapas, como las cadenas de montaje. Las múltiples tareas pueden ser repartidas en las distintas etapas, aunque cada tarea debe pasar por todas las etapas, una tarea diferente estará en cada etapa en cualquier

momento dado, así que las múltiples tareas se pueden traslapar. El paralelismo temporal es comúnmente llamado como segmentado (*pipeline*). El paralelismo espacial es justamente llamado paralelismo.

La técnica de *Pipelining* (o paralelismo temporal) es particularmente atractivo porque acelera un circuito sin duplicar su hardware. En cambio, se colocan registro entre los bloques de lógica combinatoria para dividir la lógica en etapas más cortas que se pueden ejecutar con un reloj más rápido. Los registros previenen a un *token* en una etapa de segmentado de ponerse al corriente y corrompen el *token* en la siguiente etapa. [13]

2.1.9 Transistores MOS

Silicio (Si), un semiconductor, es el material básico inicial de la mayoría de los circuitos integrados. El silicio puro consiste en una celosía tridimensional de átomos. El silicio es un elemento del Grupo IV, por lo que forma enlaces covalentes con cuatro átomos adyacentes.

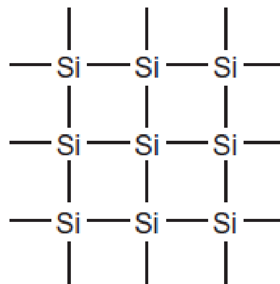


Fig 2.4 Silicon lattice

Neil H. E. Weste, Money H. David, (2011), “*CMOS VLSI DESIGN a circuits and systems perspective*”(p. 7), Massachusetts, USA, Ed. Pearson.

Como todos sus electrones de valencia están involucrados en los enlaces químicos, el silicio puro es un mal conductor. La conductividad se puede elevar mediante la introducción de pequeñas cantidades de impurezas, llamados agentes de dopado, en la red cristalina de silicio. A dopante del Grupo V de la tabla periódica, tales como arsénico, tiene cinco electrones de valencia. Sustituye a un átomo de silicio en la malla así como sus bondades hacia los cuatro vecinos, por lo que la valencia del quinto electrón es débilmente unida al átomo de arsénico.

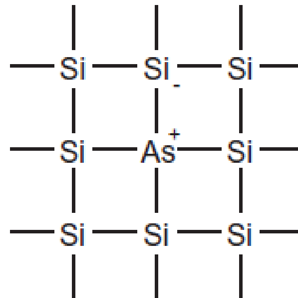


Fig 2.5 Dopant atom (As)

Neil H. E. Weste, Money H. David, (2011), “*CMOS VLSI DESIGN a circuits and systems perspective*”(p. 7), Massachusetts, USA, Ed. Pearson.

La vibración térmica de la red a temperatura ambiente es suficiente para establecer el electrón libre para moverse, dejando un ion con carga positiva As+ y un electrón libre. El electrón libre puede llevar la corriente por lo que la conductividad es más alta. A esto le llamamos un semiconductor Tipo-N porque los portadores libres, que en este caso son los electrones, tienen una carga eléctrica negativa. De manera similar, un dopante del grupo III, -tal como el Boro, el cual tiene tres electrones de valencia.

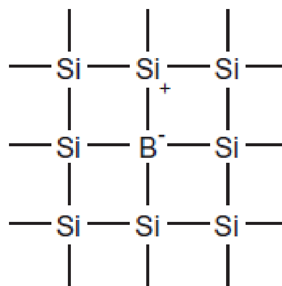


Fig 2.6 Dopant atom (B)

Neil H. E. Weste, Money H. David, (2011), “*CMOS VLSI DESIGN a circuits and systems perspective*”(p. 7), Massachusetts, USA, Ed. Pearson.

El átomo dopante puede tomar prestado un electrón de un átomo de silicio vecino, que a su vez se convierte en breve por un electrón. A su vez el átomo puede tomar prestado un electrón, y así sucesivamente, de modo que el electrón que falta, o agujero, se pueden propagar sobre la red. El agujero actúa como una carga positiva de modo tal que se le denomina a este como semiconductor tipo-P. [12]

2.1.10 Fabricación CMOS

Los diseñadores necesitan entender la implementación física de los circuitos porque tiene una mayor impacto en desempeño, poder, y costo.

Los transistores son fabricados sobre una delgada oblea de silicio que sirve como un soporte mecánico y uno eléctrico, comúnmente denominado sustrato. Se puede examinar el layout del transistor desde dos perspectivas. Una es la vista superior, obtenida mirando hacia abajo sobre la oblea. La otra es la sección transversal, obtenida de mirar completamente la sección transversal de un inversor CMOS. Después miramos la vista superior del mismo inversor y definimos un conjunto de máscaras usadas para la manufactura de diferentes partes del inversor. El tamaño de los transistores y cables es dado por las dimensiones de la máscara y está limitado por la resolución del proceso de manufacturación. Los continuos avances en esta resolución han alimentado el crecimiento exponencial de la industria de los semiconductores. [12]

3 CAPITULO 3

3.1 Dispositivos Programables

Se seleccionaron 2 diferentes dispositivos reconfigurables, con la intención de realizar distintas pruebas de simulación para el buen funcionamiento, así como para el buen entendimiento de los algoritmos aritméticos y generales utilizados en el diseño y la implementación del procesador de punto flotante para el análisis de señales. Anteriormente, el diseñador de hardware requería algo que le diera a él o a ella la flexibilidad y la complejidad de un ASIC, pero con el menor tiempo de vuelta de un dispositivo programable. La solución llego en forma de dos nuevos dispositivos: el Complex Programmable Logic Device (CPLD) y el Field Programmable Gate Array (FPGA). Los CPLDs y los FPGAs fueron el puente entre la brecha de los Arreglos de Puertas y los PALs, puesto que los CPLDs son tan rápidos como los PALs pero más complejos, mientras que los FPGAs se acercan a la complejidad de los arreglos de puertas, pero siguen siendo programables [14].

A continuación, se mencionan cada uno de ellos, con sus respectivas características y la debida justificación para su elección.

3.1.1 CPLD

3.1.1.1 CPLD ISPMACH4256ZE (ISPMACH400 FAMILY)

El alto rendimiento de la familia ispMACH 4000ZE de Lattice, ofrece una solución para el CPLD de ultra baja potencia. La nueva familia está basada en la arquitectura 31íder en la industria de Lattice ispMACH400. Se centra en innovaciones significativas para combinar un alto rendimiento en baja potencia con la Flexible familia CPLD.

El ispMACH 4000ZE combina alta velocidad y bajo consumo de energía con la flexibilidad necesaria para facilitar el diseño. Con su robusto Global Routing Pool y Output Routing Pool, esta familia ofrece una excelente First-Time-Fit, predictibilidad de tiempos, enrutamiento, retención de pin-out y la migración densidad.

La familia ispMACH 4000ZE ofrece densidades que van desde 32 hasta 256 macro celdas. Tiene múltiples combinaciones de densidad en cuanto a I/O en TQPF (Thin Quad Flat Pack). [15]

Se seleccionó este dispositivo en específico (ispMACH4256ZE) debido a la fácil implementación en físico con la que cuenta para la visualización de la simulación según el diseño elaborado y programado en el mismo, y específicamente por la familiarización que se tiene con la herramienta de diseño especializada para

el manejo del dispositivo. La Compañía de Latice Semiconductor maneja una gran variedad de CPLDs, pero se seleccionó este dispositivo gracias a que el costo de adquisición es el más bajo contando con un mayor número de macroceldas para el diseño.

Gracias a las características anteriormente mencionadas, se optó por elaborar una maqueta didáctica genérica para la realización de pruebas de simulación de diseños específicos pertenecientes al procesador de punto flotante para el análisis de señales.

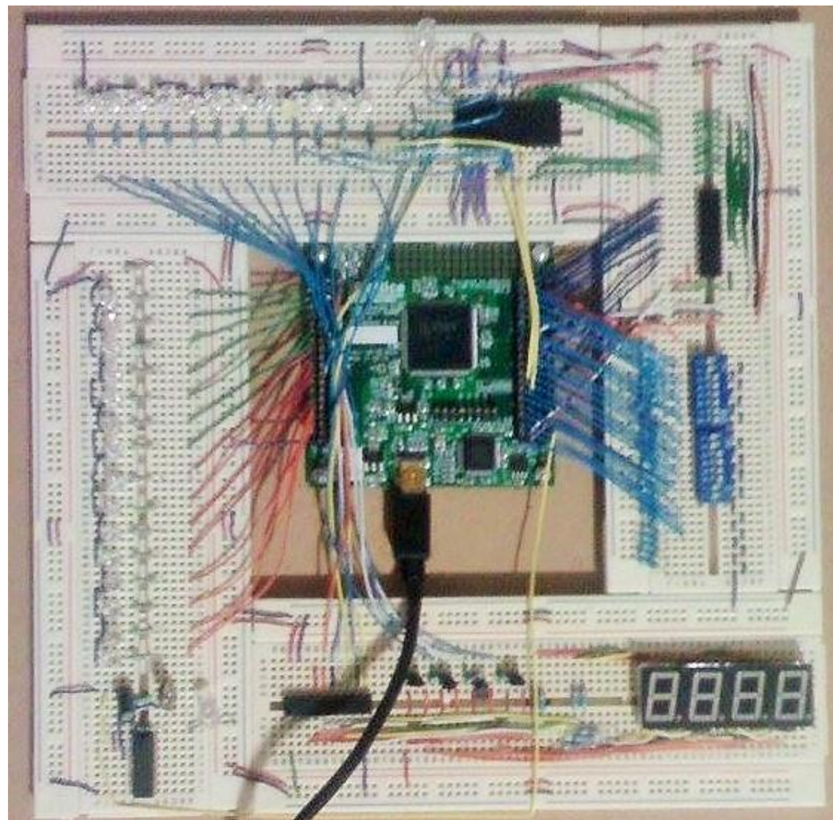


Fig 3.1 Maqueta didáctica genérica con el CPLD ispMACH4256ZE

3.1.2 FPGA

3.1.2.1 FPGA MachXO3L (MachXO Family).

La familia MachXO3L de Lattice Semiconductor cuenta con un consumo de energía extremadamente bajo, de encendido instantáneo, PLDs no volátiles que

tiene seis dispositivos con densidades que van desde 256 hasta 6864 Look Up Tables (LUT). Además de ser basado en LUTs, cuenta con un bajo costo de lógica programable, estos dispositivos cuentan con Embedded Blocked RAM (EBR), RAM distribuida, Usuario Memoria Flash (UFM), *Phase Locked Loops* (PLL), apoyo avanzado de configuración incluyendo capacidad *dual-boot* y versiones endurecidas de las funciones más utilizadas, como controlador de SPI, controlador I2C y temporizador / contador. Estas características permiten a estos dispositivos ser utilizados en bajo coste, consumo y aplicaciones de gran volumen.

Los dispositivos MachXO3L están diseñadas en un proceso de marcha de 65 nm no volátil bajo. La arquitectura del dispositivo tiene varias características como programable baja oscilación diferencial E / S y la posibilidad de desactivar I/O los bancos, en el chip PLL y osciladores dinámicamente. Estas características ayudan a administrar el consumo de energía estática y dinámica que resulta en baja estática de potencia para todos los miembros de la familia. [16]

3.1.2.1.1 Características.

El Starter Kit MachXO3L incluye:

- Tarjeta MachXO3L. La tarjeta es un tablero de 3x3 pulgadas que cuenta con los siguientes componentes y circuitos:
- MachXO3L FPGA - LCMXO3L - 6900C - 5BG256C
- Conector USB mini-B para la energía y programación
- 4 Mb de memoria flash en serie de imágenes de arranque y el apoyo para arranque dual.
- Ocho LEDs
- Interruptor DIP de 4 posiciones
- Interruptor de botón momentáneo
- Área de prototipo de 40 hoyos
- Cuatro 2 x 20 aterrizajes de cabecera para la expansión general I / O, JTAG , y fuente de alimentación externa
- Aterrizaje de cabecera 1 x 8 expansión para JTAG
- Aterrizaje de cabecera 1 x 6 de expansión para SPI / I2C
- 3,3 V y 1,2 V carriles de alimentación

Como anteriormente se mencionó, este FPGA pertenece a una de las familias de FPGAs que la compañía Lattice Semiconducotr ofrece, se seleccionó esta herramienta en específico, debido a la gran capacidad en cuanto a densidad se refiere, es evidente que la compañía maneja otro tipo de FPGAs con las características anteriormente mencionadas, pero aquí la diferencia radica en el costo de adquisición del dispositivos, pues en comparación a otros, este dispositivo en específico es de bajo costo.

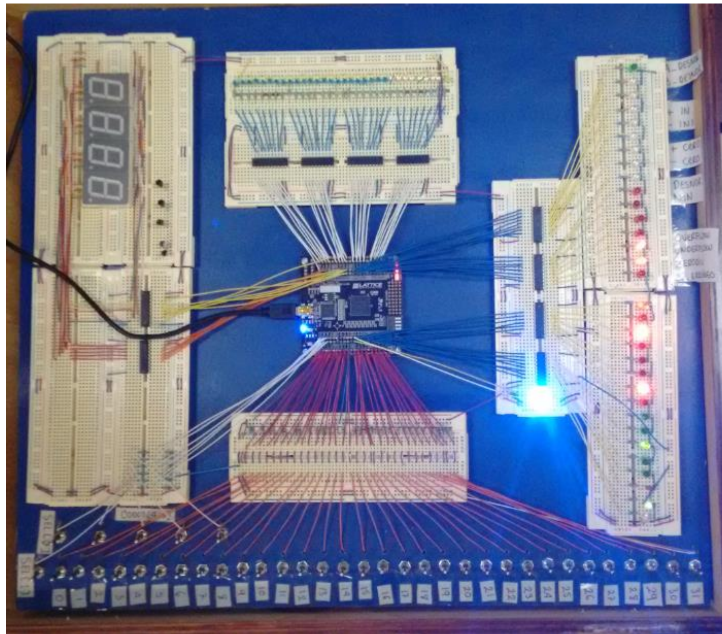


Fig 3.2 Maqueta didáctica genérica con el FPGA MachXO3L

3.2 Tecnologías de Desarrollo.

3.2.1 Lattice Diamond

El ambiente de diseño ispLever Classic se utiliza para trabajar con los CPLDs de Lattice y de los productos programables maduros. Puede ser utilizado para realizar el completo diseño de un dispositivo de Lattice a través el proceso de diseño, desde el concepto, hasta la generación del JEDEC del dispositivo o el archivo de salida bitstream para programarlo.

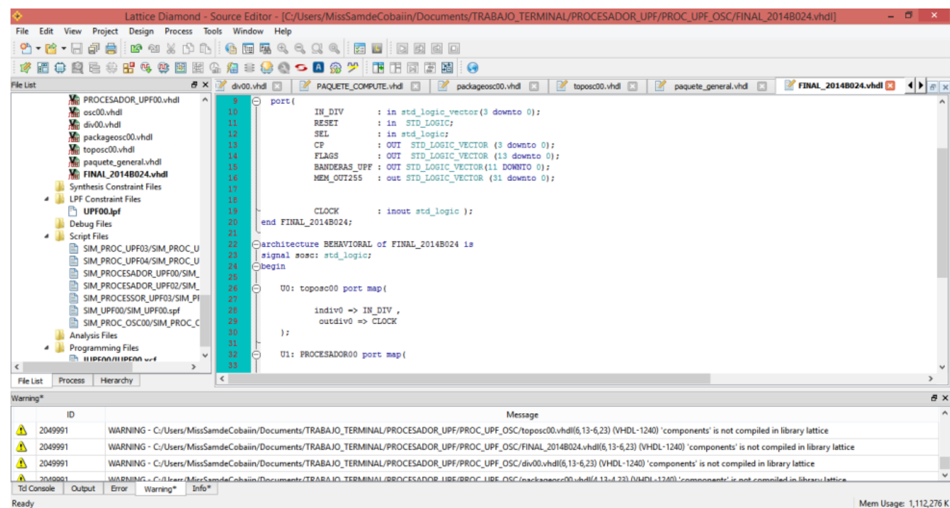


Fig 3.3 Ambiente de desarrollo de Lattice Diamond

3.2.2 ispLEVER Classic Project Navigator

El ambiente de diseño ispLever Classic se utiliza para trabajar con los CPLDs de Lattice y de los productos programables maduros. Puede ser utilizado para realizar el completo diseño de un dispositivo de Lattice a través el proceso de diseño, desde el concepto, hasta la generación del JEDEC del dispositivo o el archivo de salida bitstream para programarlo.

La versión actualizada empleada es ispLEVER Classic 1.8 (este software fue publicado el 14 de Agosto de 2014).

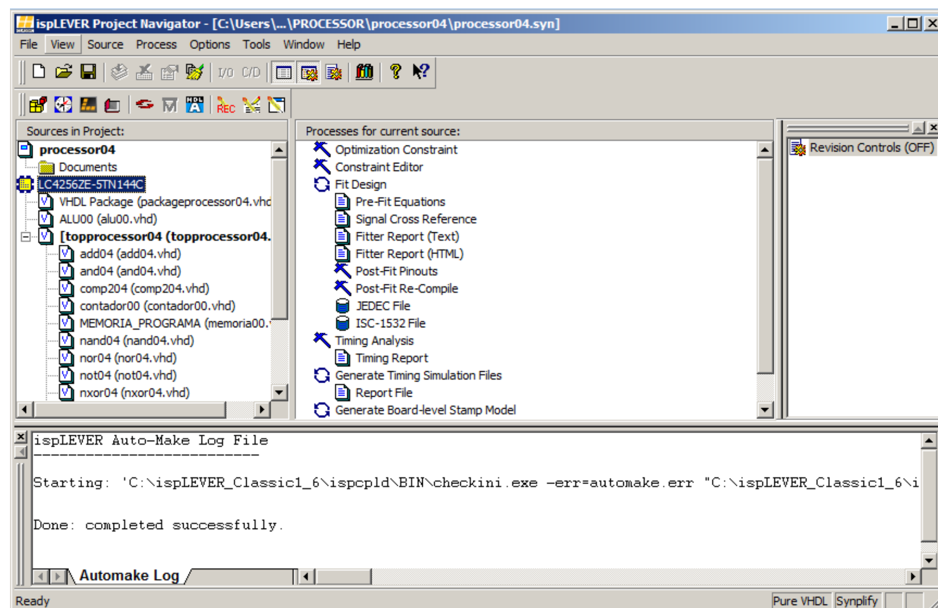


Fig 3.4 Entorno Grafico de ispLEVER Classic Project Navigator

3.2.3 ISE Design Suite 14.6

Esta tecnología de diseño desarrollada por Xilinx ISE ofrece un completo flujo de diseño front-to-end proporcionando acceso instantáneo a las características y funcionalidades de la tecnología de diseño ISE sin costo alguno.

El ISE Design Suite también ofrece herramientas a la carta para mejorar la productividad del diseñador y para proporcionar configuraciones flexibles de la suite de edición de diseño.

Sintetizador de alto nivel: Vivado High-Level Synthesis acelera la creación de IP permitiendo C, C++ y especificaciones del Sistema C para ser dirigido

directamente a todos los dispositivos programables de Xilinx sin la necesidad de crear manualmente el RTL.

Reconfiguración Parcial: La tecnología de reconfiguración parcial de Xilinx permite a los diseñadores cambiar la funcionalidad sobre la marcha, lo que elimina la necesidad de reconfigurar totalmente y volver a establecer conexiones, mejorando dramáticamente la flexibilidad que los FPGAs ofrecen.

ChipScope: El ChipScope Pro Serial I/O Toolkit proporciona una configuración y depuración de los canales de alta velocidad de entrada y salida en los diseños sobre FPGA para usar con la edición Webpack.

Kit de desarrollo Integrado: El kit de desarrollo integrado (EDK) es un entorno de desarrollo integrado para el diseño de sistemas de procesamiento integrados para su uso con ISE Webpack.

Generador de Sistema DSP: herramienta de alto nivel de líderes en la industria para el diseño de sistemas DSP de alto rendimiento utilizando dispositivos FPGA.

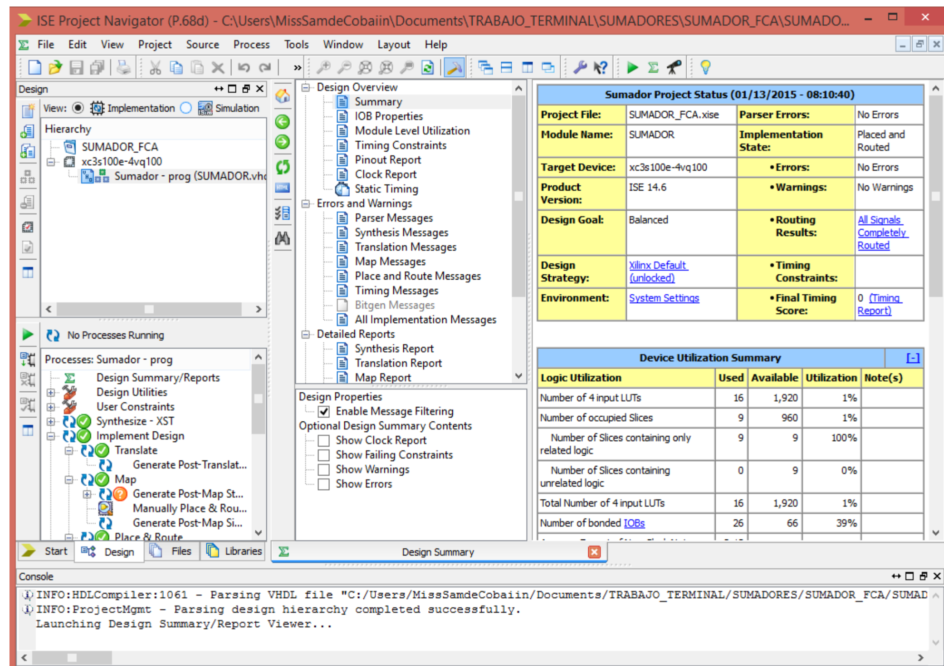


Fig 3.5 Entorno grafico de ISE Design Suite 14.6.

3.2.4 Dsch Lite 3.5

DSCH 3.5 es el software acompañante de Microwind Software para el diseño de lógica combinatoria y secuencial. Basándose en las primitivas, es decir, compuertas AND,

OR, NOT, entre otras, un circuito jerárquico es posible de construir y de ser simulado, Los símbolos interactivos son implementados para una simulación fácil de usar, la cual incluye retardos y consumo de energía.

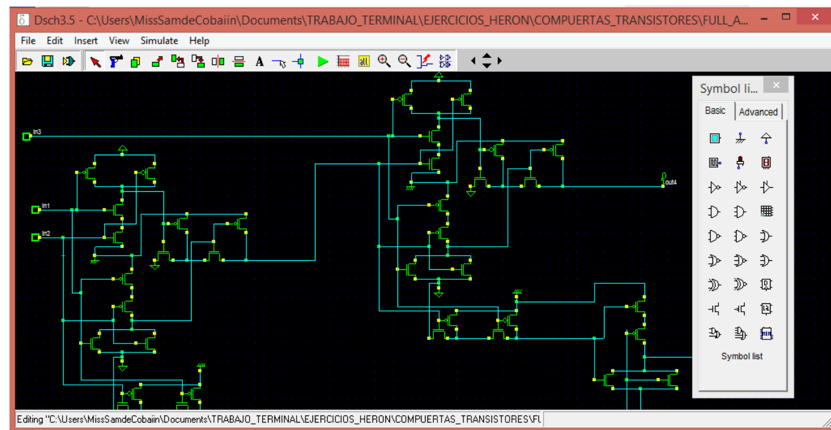


Fig 3.6 Entorno grafico de Dsch Lite 3.5

3.2.5 LTspice IV

LTSpice IV es un simulador de alto rendimiento SPICE, captura esquemática y visor de forma de onda con mejoras y modelos para facilitar la simulación de reguladores de conmutación. Las mejoras en SPICE han sido hechas simulando reguladores de conmutación muy rápido en comparación con simuladores SPICE normales, lo que permite visualizar formas de onda para la mayoría de los reguladores de conmutación en tan sólo unos minutos. Se incluye Modelos Macro para el 80% de los reguladores de conmutación de Linear Technology, más de 200 modelos de amplificadores operacionales, así como resistencias, transistores y modelos MOSFET.

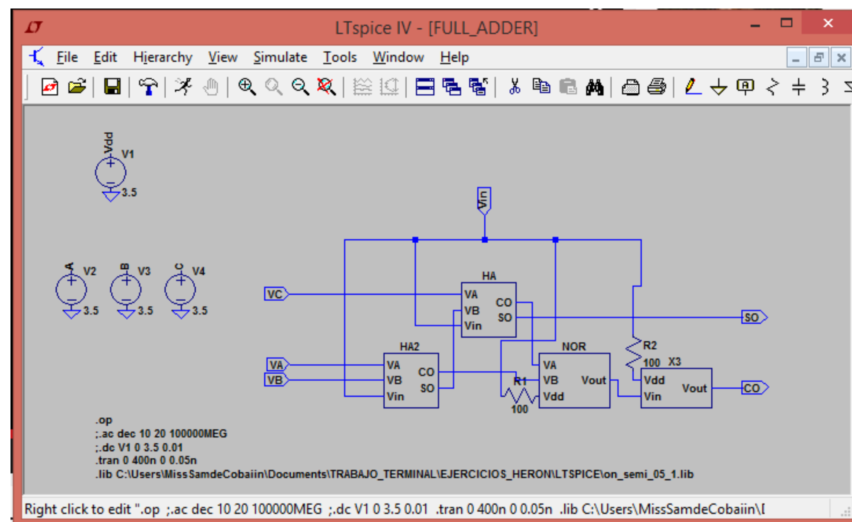


Fig 3.7 Entorno grafico LTspice IV

3.2.6 MENTHOR Graphics

Nota: No he tenido acceso a la herramienta Menthor Graphics, pues ESCOM no cuenta con este software, para adquirir este software, se necesita de una licencia, la cual tiene un costo elevado, no apto a mis posibilidades, así pues, el Centro de Investigación en Computación, me brindará el acceso al servidor donde se encuentra instalada dicha tecnología de desarrollo.

4 CAPITULO 4

Diseño del Procesador de punto flotante para el análisis de señales.

4.1 Visión General.

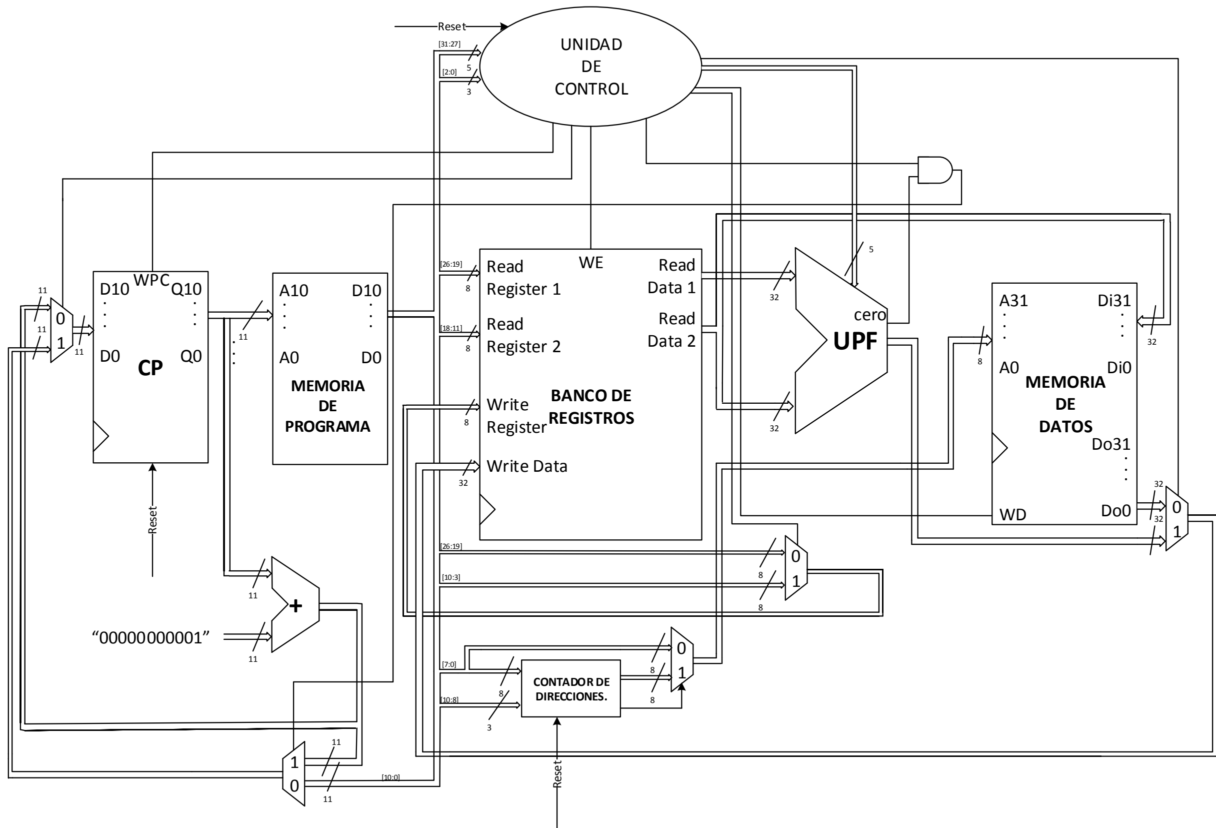


Fig 4.1 Diagrama de Bloques del Procesador en Punto Flotante para el análisis de señales.

La Figura **¡Error! No se encuentra el origen de la referencia.** Muestra el Diagrama de Bloques del Procesador Dedicado en Punto Flotante para Análisis de Señales. Se presenta un procesador dedicado de 32 bits con arquitectura tipo Harvard. El diseño se realiza implementando una filosofía orientada a procesadores RISC basados en MIPS, el modelado de la arquitectura fue llevado a cabo mediante VHDL (lenguaje de descripción de Hardware). Las principales características del procesador son: Los datos procesados únicamente serán del tipo Punto Flotante, únicamente se realizaran operaciones sobre registros, cuenta con una memoria de programa y una memoria para datos, así como con un banco de registros, solo contiene la Unidad de Punto Flotante (UPF) para realizar operaciones, un Contador de Direcciones, el cual indica el conteo en la dirección al ejecutarse un programa y un Contador de Programa (CP) para indicar direcciones de memoria. La unidad de control es la encargada de controlar las señales destinadas al buen

funcionamiento de los componentes de acuerdo al tipo de instrucción implementada. El procesador maneja una arquitectura multiciclo.

4.2 Definición del formato de las instrucciones.

Para la definición del formato las instrucciones, se debe tener en cuenta que todas las instrucciones del procesador en punto flotante para análisis de señales vienen representadas por una cadena de 32 bits y que se establecen distintos campos para cada tipo de instrucción.

4.2.1 Instrucciones tipo R.

Todas las instrucciones de este tipo leen dos registros, realizan una operación en la UPF sobre los contenidos de los registros y escriben el resultado en otro registro. Se le conoce como instrucciones tipo R o bien, instrucciones aritmético-lógicas (ya que realizan operaciones aritmético-lógicas).

El formato para este tipo de instrucción corresponde con:

op	rs	rt	rd	cr
5 bits	8 bits	8 bits	8 bits	3 bits

op: código de operación. Indica el tipo de operación a realizar.
rs: primer registro del operando fuente
rt: segundo registro del operando fuente.
rd: registro del operando destino. Obtiene el resultado de la operación.
cr: campo resultante. Definido para opciones de control.

4.2.2 Instrucciones tipo J.

Las instrucciones tipo J o mejor conocidas instrucciones de salto, consta de 5 bits para el código de operación y los 11 bits menos significativos son para el campo de dirección.

El formato de la instrucción tipo J corresponde con el siguiente:

op	s/u	s/u	dirección
5 bits	8 bits	8 bits	11 bits

Cabe mencionar que existe otro formato de instrucción para específicamente la instrucción de salto condicional, en la cual se debe especificar dos campos para los operandos, además de la dirección de salto. El formato para este tipo de instrucción de salto condicional viene dado por:

op	rs	rt	dirección
5 bits	8 bits	8 bits	11 bits

4.2.3 Instrucciones de transferencia de datos.

La instrucción de transferencia de datos que desplaza un dato desde memoria hasta un registro se denomina carga. El formato de la instrucción es el nombre de la operación seguido por el registro que se va a cargar, después la dirección de comienzo del *array* (arreglo), y finalmente un registro que contiene el índice del elemento del array que se va a cargar. Así, la dirección de memoria del elemento del array estará formada por la suma de la parte constante de la instrucción y un registro.

El formato para este tipo de instrucción corresponde con:

op	rd	s/u	dirección
5 bits	8 bits	8 bits	11 bits

op: código de operación. Indica el tipo de operación a realizar, ya sea de carga o almacenamiento
rd: registro al que se va a cargar el dato.
s/u: bits de la instrucción sin uso.
dirección: registro que contiene el índice del elemento del array que se va a almacenar.

La instrucción complementaria a la de cargar se denomina almacenar, esta transfiere datos de un registro a memoria. El formato de una instrucción de almacenamiento es similar al de una de carga: el código de operación, seguido por el registro que se va a almacenar, después la dirección del comienzo del array, y finalmente un registro que contiene el índice del elemento del array que se va a almacenar.

El formato para este tipo de instrucción corresponde con:

op	s/u	rs	dirección
5 bits	8 bits	8 bits	11 bits

op: código de operación. Indica el tipo de operación a realizar, ya sea de carga o almacenamiento
s/u: bits de la instrucción sin uso.
rs: registro que contiene el dato a almacenar.
dirección: registro que contiene el índice del elemento del array que se va a almacenar.

4.3 Modos de direccionamiento.

Los modos de direccionamiento que podemos encontrar en el procesador de punto flotante para el análisis de señales son:

- Direccionamiento de registros, donde el operando es un registro.
- Direccionamiento base o desplazamiento, donde el operando está en la posición de memoria cuya dirección es la suma de un registro y una dirección de la instrucción.
- Direccionamiento relativo al PC, donde la dirección es la suma del PC y una constante de la instrucción.

4.4 Set de Instrucciones

4.4.1 Carga / Almacenamiento.

Se describirán las instrucciones de carga y almacenamiento, así como la tabla representativa:

- **Carga:** esta instrucción tiene asociado el nemónico LW (Load-Word), y su función consiste en cargar un dato (32 bits) desde memoria hacia un registro.

- **Almacenamiento:** esta instrucción tiene asociado el nemónico SW (Store-Word), su función consiste en almacenar un dato (32 bits) de un registro en memoria.

TIPO	CODOP	INSTR.	NEMÓNICO	EJEMPLO	SIGNIFICADO
Transf. datos	00000	CARGA	LW	LW rd, dir	rd = Mem [dir]
Transf. datos	10000	ALMACE-NAMIENTO	SW	SW rs, dir	Mem[dir] = rs

Tabla 3 Instrucciones de carga/almacenamiento

4.4.2 Aritméticas.

En la tabla posterior, se enlistaran las instrucciones aritméticas que serán ejecutadas por el procesador de punto flotante para el análisis de señales, con su respectivo formato de instrucción, así como su respectivo código de operación.

TIPO	CODOP	INSTR.	NEMÓNICO	EJEMPLO	SIGNIFICADO
INSTRUCCIONES ARIMÉTICAS					
R	00001	SUMA	ADD	add rd,rs,rt	rd = rs + rt
R	00010	RESTA	SUB	sub rd,rs,rt	rd = rs - rt
R	00011	MULTIPLICACIÓN	MULT	Mult rd,rs,rt	rd = rs * rt
R	00100	DIVISIÓN	DIV	div rd,rs,rt	rd = rs / rt

Tabla 4 Instrucciones aritméticas

4.4.2.1 Instrucciones aritméticas.

Inicialmente se enunciaran las operaciones matemáticas que el procesador atenderá:

- Suma
- Resta
- Multiplicación
- División

A partir de las definiciones en los incisos anteriores, a continuación se describirán algunas de las operaciones de menor nivel:

- **Comparación.** Se considera que la operación de comparación es una operación elemental definida en una tabla de verdad, las comparacion será si dos datos son iguales.

- **Suma.** Si bien la suma se encuentra por definición en la librería unsigned de VHDL mediante el operador más (+), dicha operación será implementada mediante el sumador con acarreo anticipado a nivel de puertas lógicas.

- **Resta.** De modo similar a la operación de suma, la resta se encuentra definida en la librería unsigned de VHDL mediante el operador (-) menos, pero dicha operación será implementada mediante el restador con acarreo anticipado a nivel de puertas lógicas.

- **Multiplicación**. Debido a que se trata de un procesador dedicado, la operación de multiplicación estará definida por una sola instrucción asociada al nemónico MULT. Dicha instrucción será ejecutada por un núcleo dedicado a este fin.

- **División**. Asociada al nemónico DIV, esta operación será implementada por un núcleo específico diseñado en hardware para la obtención del resultado.

Se sobre entiende que las operaciones con números enteros son un caso particular de las operaciones en punto flotante, es decir, que directamente los números enteros tienen su representación en punto flotante, y esta representación será la única representación, de acuerdo al estándar IEEE-754 para números en punto flotantes, que manejará el procesador.

4.4.3 Salto

Existen diferentes tipos de instrucciones de salto (branch), para este procesador en particular, solo se implementarán dos tipos de instrucciones de salto, un salto incondicional (B) y un salto condicional (BNEQ). Se describen los dos tipos de instrucciones:

4.4.3.1 Salto condicional: Salto si no es igual (BNEQ).

La instrucción de salto si no es igual (BNEQ), compara dos datos almacenados en los registros fuente (rs, rt), si la diferencia realizada por la UPF es igual a cero, significa que los datos son iguales, por lo tanto el salto a la dirección indicada, no es realizado, en caso contrario, es decir, que los datos almacenados en los registros no son iguales, salta a la dirección indicada. La siguiente tabla describe la instrucción:

TIPO	CODOP	INSTRUCCIÓN	NEMÓNICO	EJEMPLO	SIGNIFICADO
J	10010	Salto si es igual	BNEQ	BNEQ dir	If(rs!=rt) go to Dir

Tabla 5 Instrucción salto condicional

4.4.3.2 Salto incondicional: Salto (B).

La instrucción de Salto incondicional (B), simplemente salta a la dirección de memoria indicada para ejecutar la instrucción deseada. La tabla contigua describe la instrucción:

TIPO	CODOP	INSTRUCCIÓN	NEMÓNICO	EJEMPLO	SIGNIFICADO
J	10001	Salto	B	B dir	PC = Dir

Tabla 6 Instrucción salto incondicional

4.5 Definición del formato del tipo de dato (punto flotante).

4.5.1 Conversión de numero decimal a binario en punto flotante.

1.- Representar el número decimal **-118.625** usando el formato IEEE 754.

El formato para le representación de punto flotante en el estándar IEEE 754 viene dado por:

Bit de Signo (S)	**Exponente** (E)	**Mantisa** (M)
b_{31}	$b_{30}\, b_{29} \ldots \ldots b_{24}\, b_{23}$	$B_{22}\, b_{21} \ldots \ldots b_{1}\, b_{0}$

Tabla 7 Formato representación de un numero en Punto Flotante (IEEE-754)

1.- Si el número es positivo S = 0, en caso contrario si el número es negativo S= 1.

S = 1

2.- Convertir la parte entera en binario.

128	64	32	16	8	4	2	1
0	1	1	1	0	1	1	0

118 – 128 = x	0
118 – 64 = 54	1
54 – 32 = 22	1
22 – 16 = 6	1
6 – 8 = x	0
6 – 4 = 2	1
2 – 2 = 0	1
0 – 1 = x	0

3.- Convertir la parte fraccionaria a su respectiva representación en binario para formar la mantisa:

3.1 El número 0.625 se **multiplica** por 2.

0.625 x 2 = 1.25

3.2 Si el resultado **es mayor** a 1, el bit más significativo de la mantisa adquiere el valor de 1, en caso contrario el valor es igual a cero 0

22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

3.3 La nueva parte fraccionaria (0.25) será el siguiente número a multiplicar por dos y se repite el proceso hasta completar.

22	0.625 x 2	1.25	1
21	0.25 x 2	0.5	0
20	0.5 x 2	1.0	1
19	0 x 2	0	0
…	….	…	…
0	0 x 2	0	0

3.4 Así la mantisa tiene el siguiente valor:

22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.- Juntar la parte entera con la parte fraccionaria para posteriormente normalizar. **Normalizar:** consiste en mover el punto a la izquierda/derecha del número en binario, dejando únicamente un 1 a la izquierda del mismo, es decir, hacer un **corrimiento** de n bits a la izquierda o derecha.

	Exponente
01110110 . 10100000000000000000000 x 2^0	0000 0000
0111011 . 010100000000000000000000 x 2^1	0000 0001
011101 . 1010100000000000000000000 x 2^2	0000 0010
01110 . 11010100000000000000000000 x 2^3	0000 0011
0111 . 011010100000000000000000000 x 2^4	0000 0100
011 . 1011010100000000000000000000 x 2^5	0000 0101
01 . 11011010100000000000000000000 x 2^6	0000 0110

** El cero a la izquierda del punto se elimina.

** Es entendido que al normalizar, los bits menos significativos se perderán, puesto que el total de bits con los que cuenta la mantisa es igual a 23

M = 11011010100000000000000

5.- Es necesario que al exponente en cuestión se le **sume** el uno implícito en el formato de punto flotante IEEE 754 denominado BIAS = $2^{8-1} - 1 = 127$, por lo tanto:

0000 0110	6
+ 0111 1111	+ 127
1000 0101	133

E = 1000 0101

6.- Finalmente, el número -118.625 es representado así:

Bit de Signo (**S**)	Exponente (**E**)	Mantisa (**M**)
1	**1000 0101**	**11011010100000000000000**

Bit de Signo (**S**)	Exponente (**E**)	Mantisa (**M**)	Representación
0	**0000 0000**	**00000000000000000000000**	**+ 0**
1	**0000 0000**	**00000000000000000000000**	**- 0**
0	**1111 1111**	**00000000000000000000000**	**+ Infinito**
1	**1111 1111**	**00000000000000000000000**	**- Infinito**
0	**1111 1111**	**00000100000001000000000**	**NaN**
1	**1111 1111**	**00100111000001000010000**	**NaN**
1	**1000 0101**	**11011010100000000000000**	**Numero Normalizado**
1	**0000 0000**	**11011010100000000000000**	**Numero desnormalizado**

4.5.2 Conversión de un número binario en punto flotante a número decimal.

1.- Hallar el número en decimal representado por el siguiente número en formato IEEE 754 de punto flotante:

1 1000 0101 11011010100000000000000

El formato general de la representación de un número N en punto flotante es:

$$N = F \times r^e$$

Donde:

N = número en punto flotante
F = Fracción o mantisa

r = base
e = exponente.

1.- Separar el número binario de acuerdo al formato IEEE 754, es decir, 1 bit para el signo, 8 bits para el exponente y 23 bits para la mantisa.

S = 1
E = 1000 0101
M = 11011010100000000000000

2.- Una vez hallado el exponente E = 1000 0101, se le tendrá que restar el BIAS = $2^{8-1} - 1 = 127$:

1000 0101	133
- 1000 0001	- 127
0000 0110	6

Por lo tanto **e = 6**

3.- Una vez hallada la mantisa, se coloca un 1 a la izquierda del punto, y el resto de la mantisa a la derecha del punto.

1 . 11011010100000000000000

4.- Cada bit perteneciente a la parte fraccionaria, se multiplica por su valor respectivo. (Observar tabla de referencia en el anexo). Y finalmente se suma el conjunto de valores obtenidos, para conocer en decimal el valor resultante **(F)**.

Posición bit.	Valor.	Bit	Operación	Resultado
22	1/2	1	1 * 1/2	0.5
21	1/4	1	1 * 1/4	0.25
20	1/8	0	0 * 1/8	0
19	1/16	1	1 * 1/16	0.0625
18	1/32	1	1 * 1/32	0.03125
17	1/64	0	0 * 1/64	0
16	1/128	1	1 * 1/128	0.0078125
15	1/256	0	0 * 1/256	0
14	1/512	1	1 * 1/512	0.001953125
13	1/1024	0	0 * 1/1024	0
12	1/2048	0	0 * 1/2048	0
…	…	…	…	…
1	1/4194304	0	0 * 1/4194304	0
0	1/8388608	0	0 * 1/8388608	0
TOTAL	**8388607/8388608**	**-**	**-**	**0.853515625**

Así:

1.11011010100000000000000 = **1.853515625 = F**

5.- Una vez obtenido el valor decimal resultante de la mantisa más el uno implícito, se procede a colocar el exponente encontrado previamente **(e = 6)** a la base = 2 **(r = 2),** Es decir:

$$\mathbf{N = 1.853515625 \times 2^{6}}$$

6.- Ya conocido el valor de N, es necesario multiplicar ese valor por ±1, Si el bit de signo es igual a 1, N es multiplicado por -1, en caso contrario, si el bit de signo es igual a 0, N será multiplicado por +1, para este ejemplo en específico, como el bit de signo es igual a 1:

$$\mathbf{(N) \times (-1) = (1.853515625 \times 2^{6}) \times (-1)}$$
$$\mathbf{= -118.625}$$

Posición bit.	Valor.	Posición bit.	Valor.
22	1/2	10	1/8192
21	1/4	9	1/16384
20	1/8	8	1/32768
19	1/16	7	1/65536
18	1/32	6	1/131072
17	1/64	5	1/262144
16	1/128	4	1/524288
15	1/256	3	1/1048576
14	1/512	2	1/2097152
13	1/1024	1	1/4194304
12	1/2048	0	1/8388608
11	1/4096		

Tabla 8 Valores por bit de la mantisa

4.5.3 Ejemplos de valores numéricos en formato de Punto Flotante según el estándar IEEE-754.

	DECIMAL	PUNTO FLOTANTE		
		SIGNO	EXP.	MANTISA
1	1.8875	0	01111111	11100011001100110011001
2	25.7261	0	10000011	10011011100111100001101
3	6.8174	0	10000001	10110100010100000100100
4	-30031993	1	10010111	11001010011001111000100
5	12012.20111817	0	10001001	00101100100011001101111
6	535.24451	0	10001000	00001011100111110100110
7	-3.55261698761	1	10000000	11100110101111000010011
8	5626.4155116677	0	10001011	01011111101001101010010
9	-5363.369922	1	10001011	01001111001101011110101
10	2.45562	0	10000000	00111010010100011100000
11	0.0000876342	0	01110001	01101111100100001000000
12	-0.0763429	1	01111011	00111000101100110101010
13	0.32456124	0	01111101	01001100010110011100100
14	0.000356168	0	01110011	01110101011110000001000
15	-0.00000000235	1	01100010	01000000000000000000000
16	0.000000096046447753906	1	01111111	00000000000000000000101
17	0	0	00000000	00000000000000000000000
18	1	0	01111111	00000000000000000000000
19	10	0	10000010	01000000000000000000000
20	3	0	10000000	10000000000000000000000
21	5	0	10000001	01000000000000000000000
22	1.202211	0	01111111	00110011110001000001100

Tabla 9. Ejemplos de datos numéricos con su representación en punto flotante.

4.6 Componentes del procesador

4.6.1 Contador de Programa (CP)

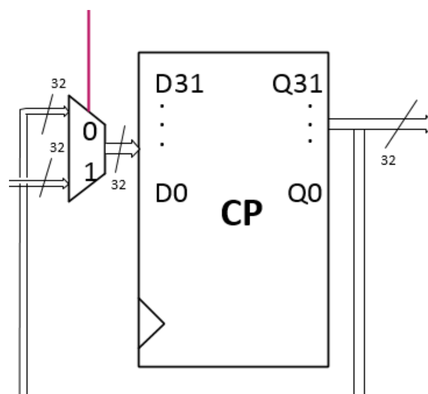


Fig 4.2 Diagrama del Contador de Programa

El contador de Programa del procesador es un registro que contiene la dirección de la siguiente instrucción a ejecutar, dicho registro será incrementado automáticamente por cada ciclo de instrucción, es decir, que gracias al PC las instrucciones serán ejecutadas consecutivamente desde memoria de programa, las instrucciones de salto, interrumpen la secuencia al colocar una nueva dirección sobre el registro.

4.6.2 Memoria de Programa

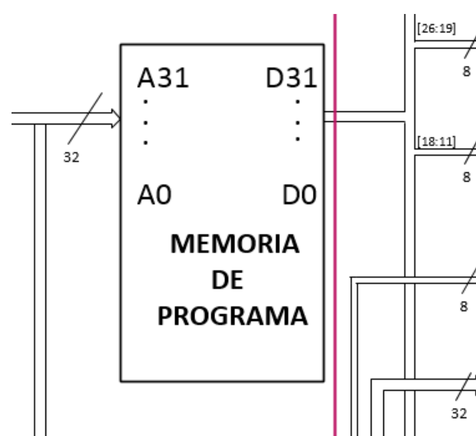


Fig 4.3 Diagrama de la Memoria de Programa

La memoria de programa de este procesador es una memoria ROM, es decir, de solo lectura, las instrucciones del programa a procesar serán almacenadas en esta memoria. Su

capacidad de memoria es de 4096M x 32, la instrucción leída de esta memoria es dirigida hacia la unidad de control, hacia el banco de registros y hacia el modulo extensor de signo. Cuenta con un bus de direcciones y un bus de datos. El bus de direcciones (32 bits) es un bus de entrada a dicha memoria, el cual indica la instrucción a ser ejecutada así como la localidad de memoria a leer. El bus de datos (32 bits) es el bus de salida de la memoria, dicho bus contiene la instrucción almacenada en la memoria la cual fue previamente direccionada por el bus de direcciones.

4.6.3 Banco de Registros

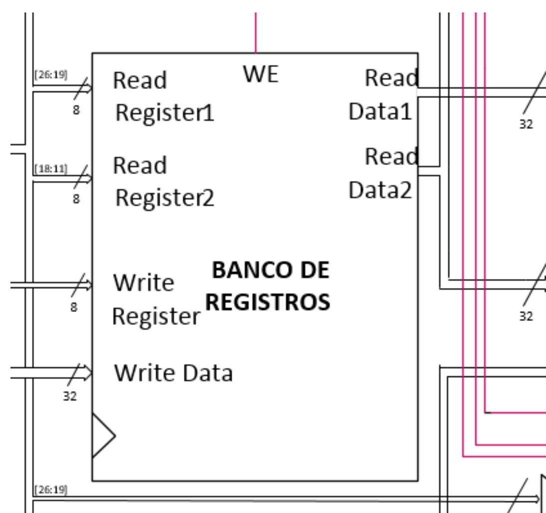


Fig 4.4 Diagrama del Banco de Registros

A este componente del procesador se le denomina banco de registros, pues está formado por 256 registros de 32 bits cada uno. Cuenta con dos puertos de lectura y un solo puerto de escritura, los puertos de lectura tienen 8 bits a la entrada, Read Register1 y Read Register2, cada uno específicamente de $2^8 = 256$ registros como operandos fuente, dichos registros almacenan los datos en punto flotante de 32 bits, los cuales son leídos a través de las salidas Read Data1 y Read Data2, respectivamente. El puerto de escritura toma 8 bits de dirección a la entrada de Write Register, así como 32 bits de datos para la entrada de Write Data. Cuenta con un bit de control denominado WE, el cual si está habilitado en 1, el banco de registros es capaz de escribir los datos recibidos (Write Data) en el registro especificado (Write Register), en caso contrario es de solo lectura sobre los registros fuente.

4.6.4 Memoria de Datos

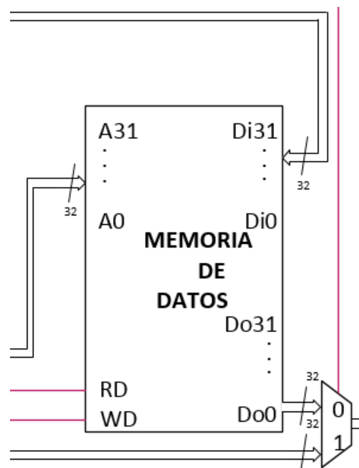
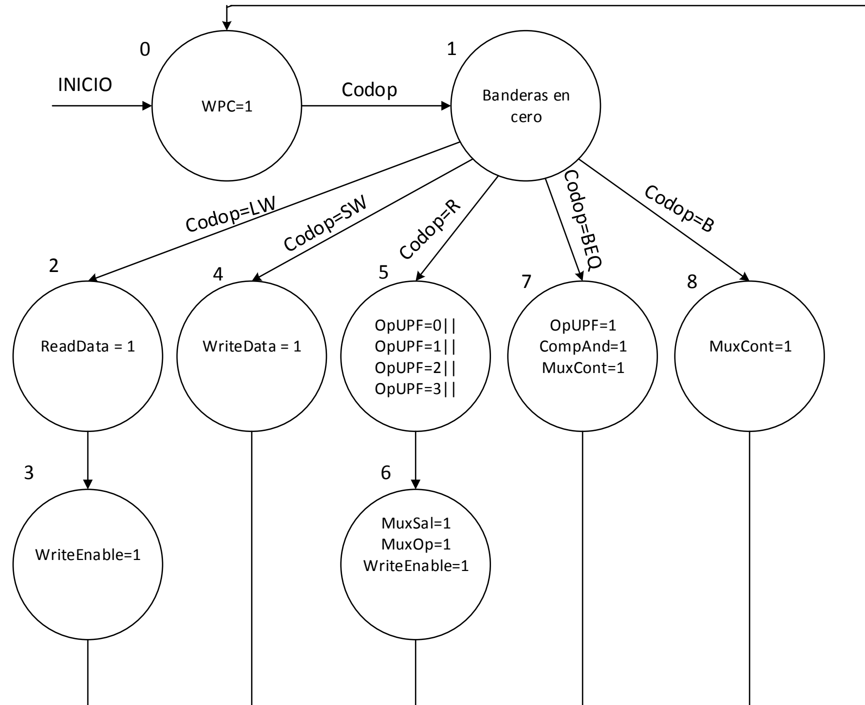


Fig 4.5 Diagrama de la Memoria de Datos

La memoria de programa de este procesador es una memoria RAM, es decir de lectura /escritura, su capacidad de memoria es de 4096M x 32, la cual permite almacenar datos en punto flotante que el procesador requiera utilizar. Dicha memoria de datos maneja dos buses de datos así como un bus de dirección, sabe mencionar que todos los buses son de 32 bits, cuenta con un bus de datos de entrada y un bus de datos de salida. Si el procesador requiere que la memoria sea de escritura los datos de entrada se almacenan en la dirección indicada por el bus de dirección, siempre y cuando el bit de control de escritura (WR) este activado en alto. En caso contrario, si el procesador requiere que la memoria realice una lectura, los datos obtenidos a partir de la localidad de memoria especificada por el bus de dirección serán la salida del bus de datos, cuando el bit de control de lectura (RD) este activado en alto. En caso contrario, no se realiza ninguna acción.

4.6.5 Unidad de Control

Para implementar la unidad de control como una máquina de estados finita, primero se asignó un número a cada uno de los 9 estados, en este caso, se usó un enumerado simple secuencial, la siguiente figura muestra el diagrama de la máquina de estado finita implementada. Con nueve estados, se necesitaron de 4 bits para codificar el número de estado, es decir, S3, S2, S1 Y S0.



4.6 Diagrama de la máquina de estados finita para la unidad de control

La manera más sencilla de implementar las funciones de la unidad de control, es codificar la tabla de verdad que se observa a continuación, en una memoria de solo lectura, es decir una memoria ROM. El número de entradas en la memoria será el estado actual donde se encuentre el proceso, mientras la salida será un bus de 12 bits, los cuales indican las banderas activas para cada estado respectivo.

SALIDAS	ESTADOS								
	0	1	2	3	4	5	6	7	8
MuxCont	0	0	0	0	0	0	0	1	1
WriteEnable	0	0	0	1	0	0	1	0	0
MuxOp	0	0	0	0	0	0	1	0	0
ReadData	0	0	1	0	0	0	0	0	0
WriteData	0	0	0	0	1	0	0	0	0
MuxSal	0	0	0	0	0	0	1	0	0
CompAnd	0	0	0	0	0	0	0	1	0
MuxBranch	0	0	0	0	0	0	0	0	0
WPC	1	0	0	0	0	0	0	0	0
OpUPF(2)	0	0	0	0	0	x	0	0	0
OpUPF(1)	0	0	0	0	0	X	0	1	0
OpUPF(0)	0	0	0	0	0	X	0	0	0

Tabla 10 Tabla de verdad para las 12 salidas de control

La siguiente tabla muestra de manera general las banderas utilizadas por cada camino de datos respectivo por cada instrucción implementada en el procesador.

CONTROL	NOMBRE DE LA SEÑAL	CARGA	ALMAC	TIPO R				BEQ	B
				ADD	SUB	MULT	DIV		
ENTRADAS	Codop(4)	0	1	0	0	0	0	1	1
	Codop(3)	0	0	0	0	0	0	0	0
	Codop(2)	0	0	0	0	0	1	0	0
	Codop(1)	0	0	0	1	1	0	1	0
	Codop(0)	0	0	1	0	1	0	0	1
SALIDAS	MuxCont	0	0	0	0	0	0	1	0
	WriteEnable	1	0	1	1	1	1	0	0
	MuxOp	1	0	1	1	1	1	0	0
	ReadData	0	0	0	0	0	0	0	0
	WriteData	1	1	0	0	0	0	0	0
	MuxSal	0	0	1	1	1	1	0	0
	CompAnd	0	0	0	0	0	0	1	0
	MuxBranch	0	0	0	0	0	0	1	1
	WPC	1	1	1	1	1	1	1	1
	OpUPF(2)	0	0	0	0	0	0	0	0
	OpUPF(1)	0	0	0	0	1	1	0	0
	OpUPF(0)	0	0	0	1	0	1	0	0

Tabla 11 Entradas / Salidas Unidad de Control

4.6.6 Unidad de Punto Flotante

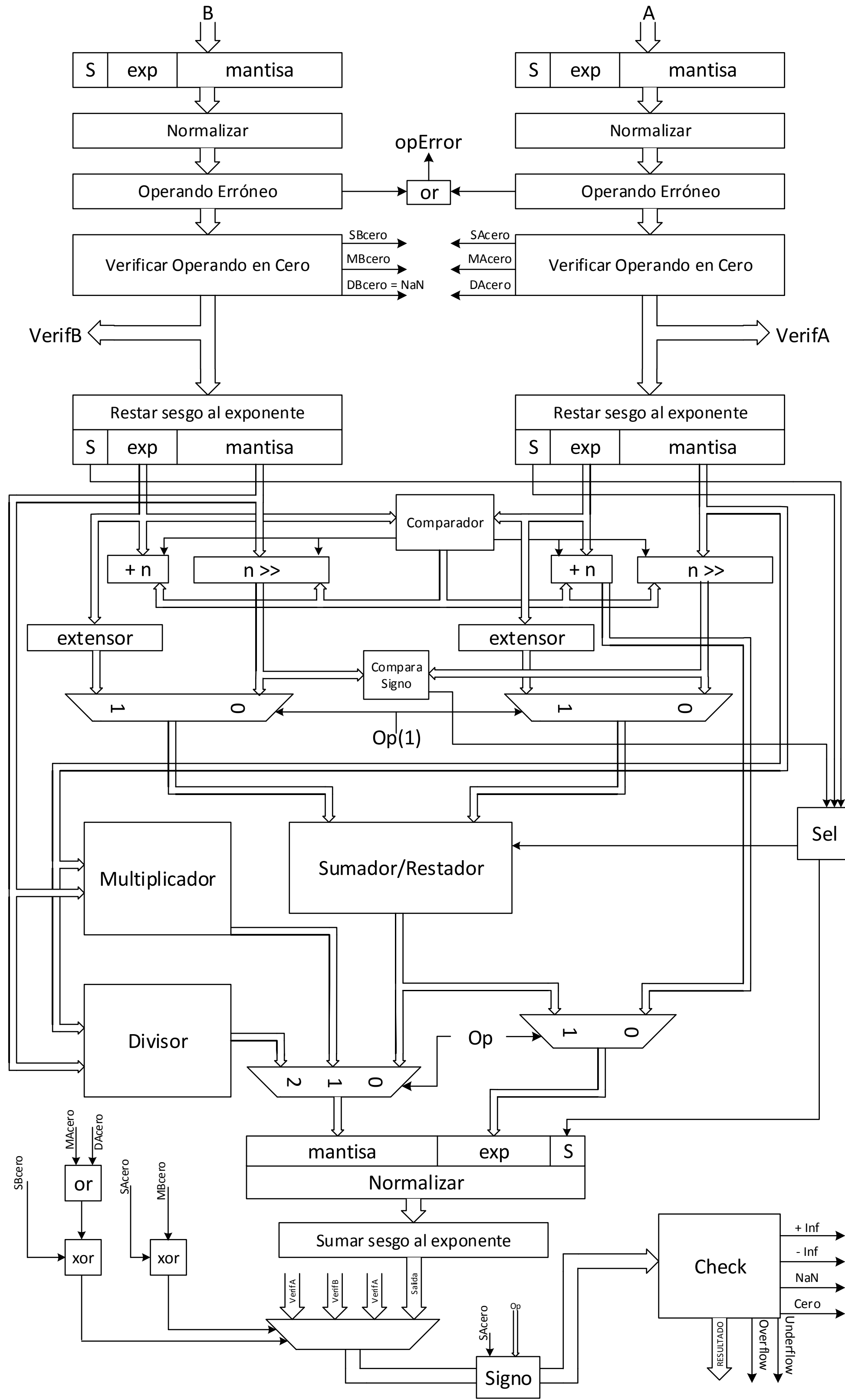


Fig 4.7. Diagrama General de la Unidad de Punto Flotante

4.6.6.1 Suma en Punto Flotante

La suma en punto flotante es definida como se muestra a continuación para dos números A y B, donde $A = f_A \times r^{e_A}$ y $B = f_B \times r^{e_B}$

$$A + B = (f_A \times r^{e_A}) + (f_B \times r^{e_B})$$

El algoritmo de suma, puede ser particionado en los siguientes pasos como se muestra a continuación:

1. Normalizar los operandos
2. Detectar excepción al inicio de algún operando erróneo.
3. Verificar por operandos en cero.
4. Restar el sesgo a los exponentes.
5. Alinear las fracciones seleccionando el número con el menor exponente y realizar el corrimiento sobre la fracción hacia la derecha por una cantidad igual a la diferencia de los dos exponentes.
6. Verificar la operación a realizar
    a. Si los signos de los operandos son iguales, realizar una suma, de lo contrario ejecutar una resta
7. Sumar/restar las fracciones.
8. Colocar el exponente del resultado igual al mayor exponente.
9. Normalizar el resultado si es necesario.
    a. Si el acarreo a la salida es igual a 1, entonces realizar un corrimiento a la derecha sobre la mantisa, y reemplazar el acarreo por el bit más significativo de la misma, es decir, {Cout, mantisa}.
    b. Incrementar en uno al exponente
    c. No cambiar el signo
10. Sumar el sesgo al exponente
11. Verificar el valor obtenido en el resultado.

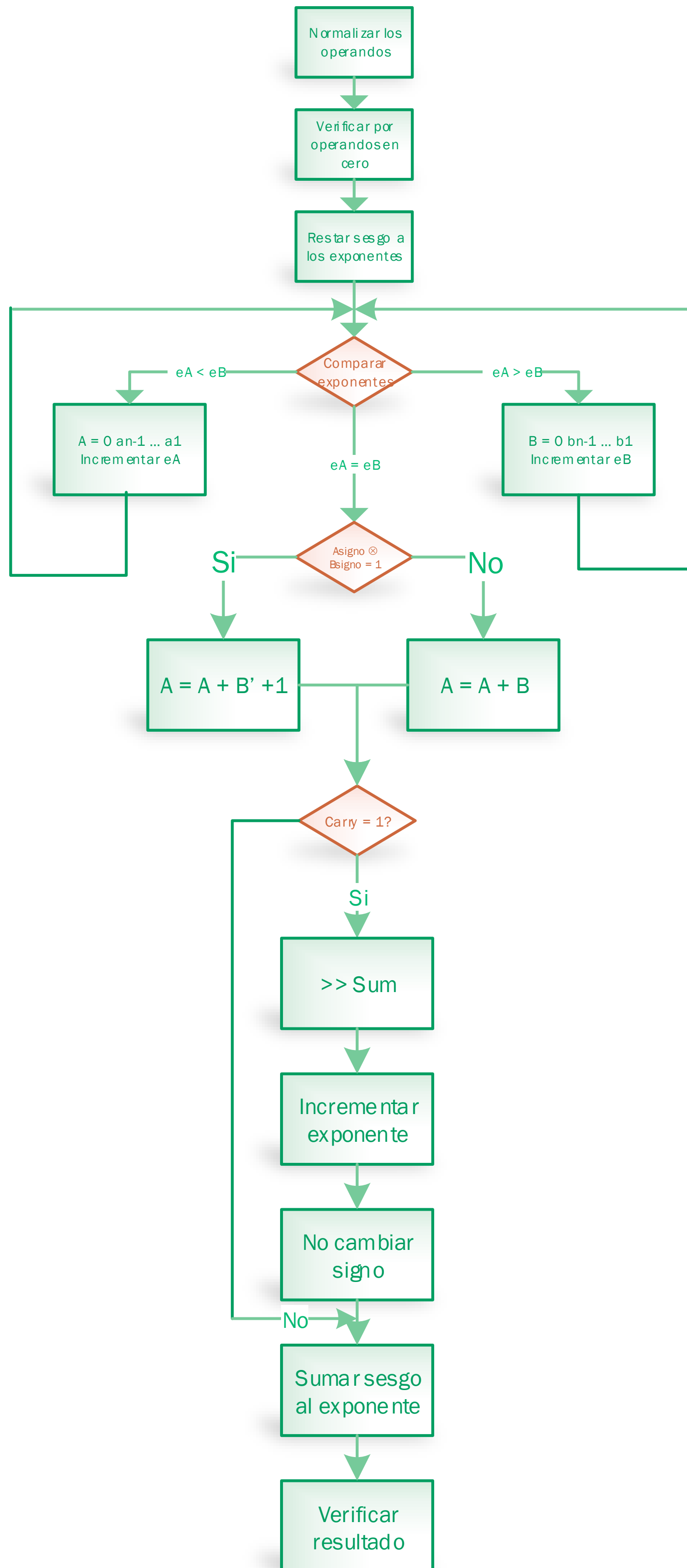


Fig 4.8 Diagrama del algoritmo de suma en punto flotante

4.6.6.2 Resta en Punto Flotante

La resta en punto flotante es definida como se muestra a continuación para dos números A y B, donde $A = f_A \times r^{e_A}$ y $B = f_B \times r^{e_B}$

$$A + B = (f_A \times r^{e_A}) - (f_B \times r^{e_B})$$

El algoritmo de resta, puede ser particionado en los siguientes pasos como se muestra a continuación:

1. Normalizar los operandos
2. Detectar excepción al inicio de algún operando erróneo.
3. Verificar por operandos en cero.
4. Restar el sesgo a los exponentes.
5. Alinear las fracciones seleccionando el número con el menor exponente y realizar el corrimiento sobre la fracción hacia la derecha por una cantidad igual a la diferencia de los dos exponentes.
6. Verificar la operación a realizar
    a. Si los signos de los operandos son diferentes, realizar una suma, de lo contrario ejecutar una resta
7. Sumar/restar las fracciones.
8. Colocar el exponente del resultado igual al mayor exponente.
9. Normalizar el resultado si es necesario.
    a. Si el acarreo a la salida es igual a 1, realizar el complemento a dos sobre la mantisa así como colocar el bit de digno igual al complemento del bit de signo de A, en caso contrario, mantener el signo.
    b. Verificar si el bit más significativo de la mantisa del resultado es igual a uno
        i. Si el bit más significativo es igual a uno, continuar con el algoritmo, en caso contrario, realizar un corrimiento a la izquierda y decrementar el exponente, repetir el paso 9.b.
10. Sumar el sesgo al exponente
11. Verificar el valor obtenido en el resultado.

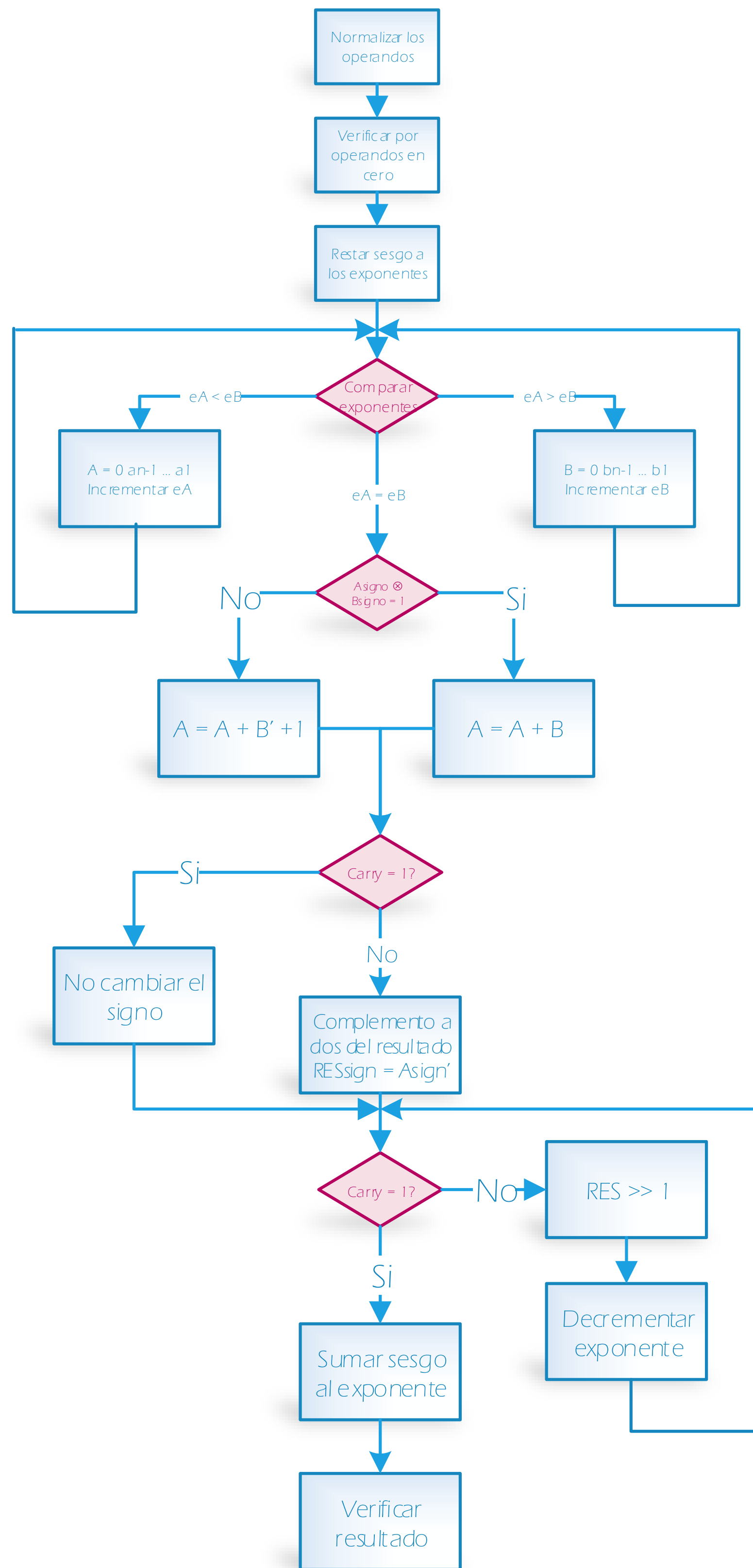


Fig 4.9 Diagrama del algoritmo de resta en punto flotante

4.6.6.3 Multiplicación en Punto Flotante

La multiplicación en punto flotante es definida como se muestra a continuación para dos números A y B, donde $A = f_A \times r^{e_A}$ y $B = f_B \times r^{e_B}$

$$A + B = (f_A \times r^{e_A}) * (f_B \times r^{e_B})$$

El algoritmo de multiplicación, puede ser particionado en los siguientes pasos como se muestra a continuación:

1. Normalizar los operandos
2. Detectar excepción al inicio de algún operando erróneo.
3. Verificar por operandos en cero.
4. Determinar el signo del producto, es decir, si los signos de los operandos son iguales el signo del producto será cero, en caso contrario tomará el valor de uno.
5. Restar el sesgo a los exponentes.
6. Sumar exponentes.
7. Multiplicar las fracciones.
8. Sumar sesgo al exponente.
9. Normalizar el resultado si es necesario.
    a. Verificar si el bit más significativo de la mantisa del resultado es igual a uno.
        i. Si el bit más significativo es igual a uno, continuar con el algoritmo, en caso contrario, realizar un corrimiento a la izquierda y decrementar el exponente, repetir el paso 9.a.
10. Verificar el valor obtenido en el resultado.

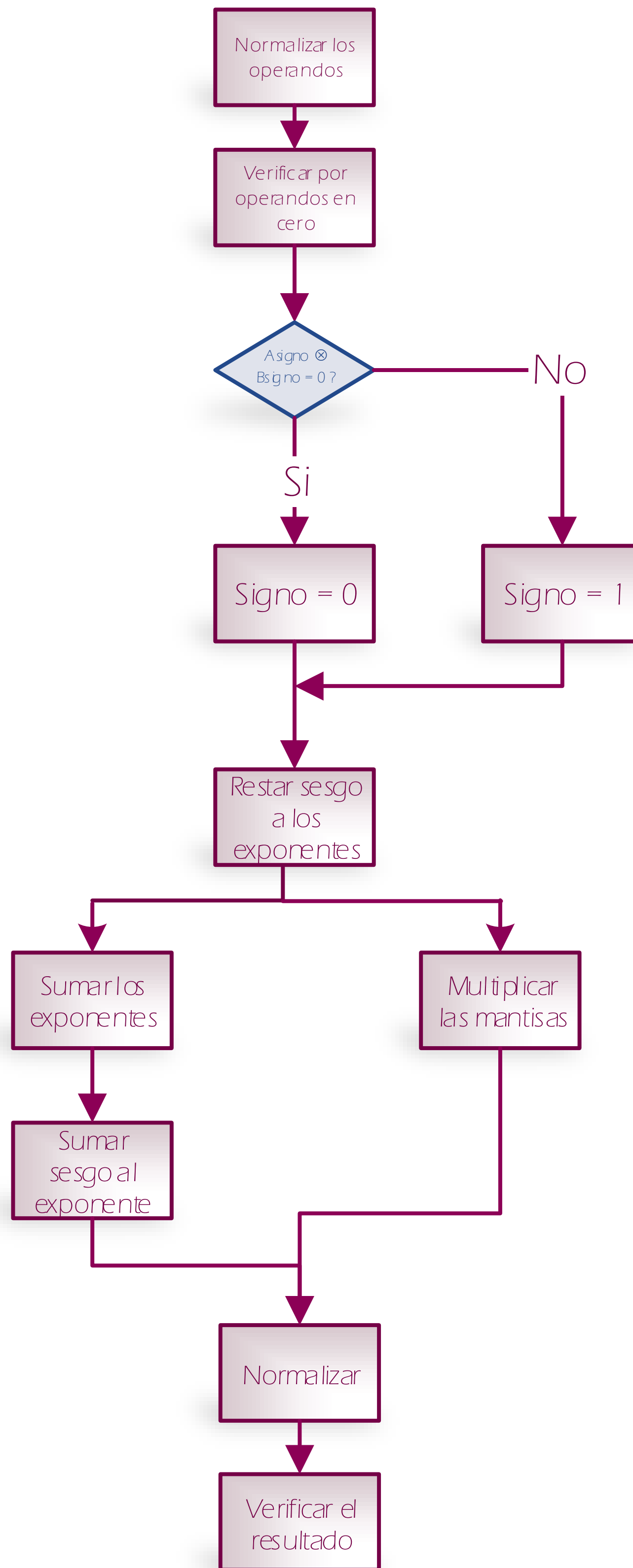


Fig 4.10 Diagrama del algoritmo de multiplicación en punto flotante

4.6.6.4 División en Punto Flotante

La división en punto flotante es definida como se muestra a continuación para dos números A y B, donde $A = f_A \times r^{e_A}$ y $B = f_B \times r^{e_B}$

$$A + B = (f_A \times r^{e_A})/(f_B \times r^{e_B})$$

El algoritmo de división, puede ser particionado en los siguientes pasos como se muestra a continuación:

1. Normalizar los operandos
2. Detectar excepción al inicio de algún operando erróneo.
3. Verificar por operandos en cero.
4. Determinar el signo del producto, es decir, si los signos de los operandos son iguales el signo del producto será cero, en caso contrario tomara el valor de uno.
5. Restar el sesgo a los exponentes.
6. Restar exponentes.
7. Dividir las fracciones.
8. Sumar sesgo al exponente.
9. Normalizar el resultado si es necesario.
    a. Verificar si el bit más significativo de la mantisa del resultado es igual a uno.
        i. Si el bit más significativo es igual a uno, continuar con el algoritmo, en caso contrario, realizar un corrimiento a la izquierda y decrementar el exponente, repetir el paso 9.a
10. Verificar el valor obtenido en el resultado.

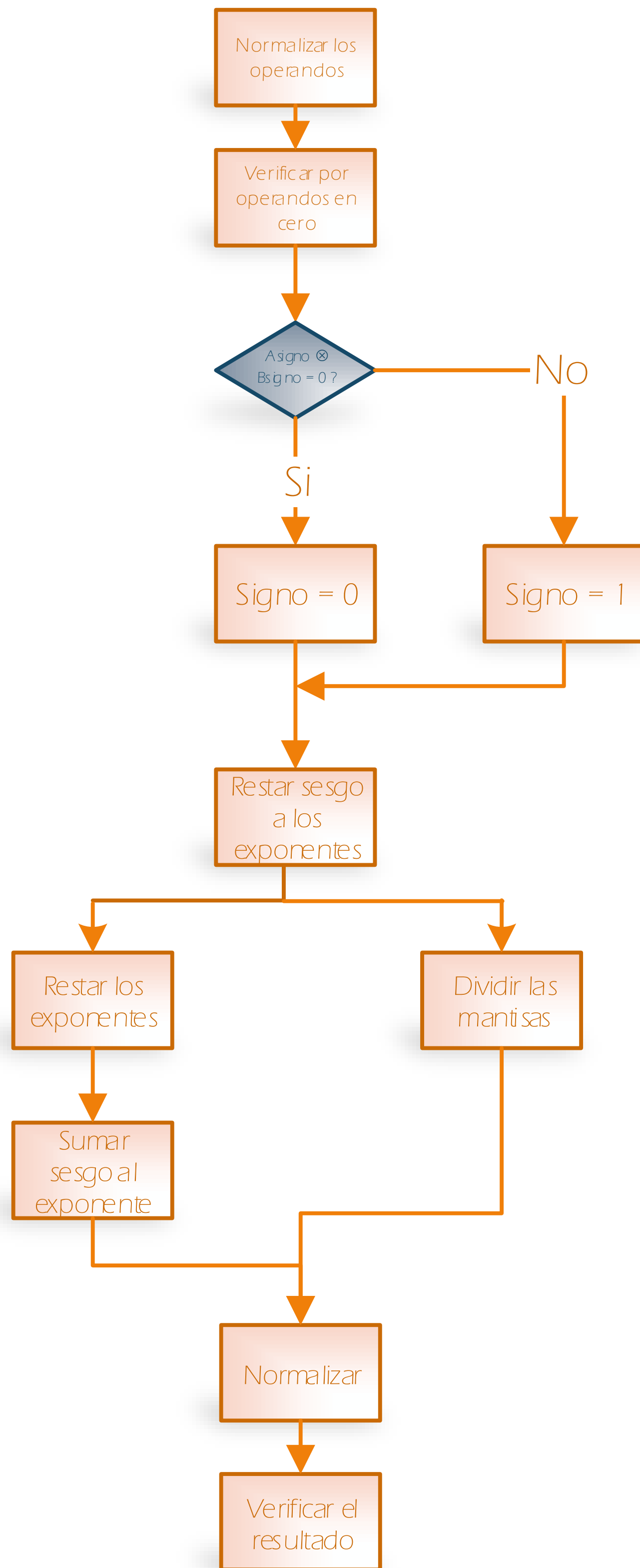


Fig 4.11 Diagrama del algoritmo de división en punto flotante

4.6.6.5 Excepciones

Además de los valores infinitos que son producidos cuando un sobre flujo ocurre, existe un valor especial, un Not a Number (NaN) el cual es producido por ciertas operaciones, por ejemplo, realizar la raíz cuadrada de un numero negativo. Un NaN es codificado con el exponente reservado de 128 y un campo significativo que lo distingue del infinito. La intención de que existan valores producidos como un NaN o el infinito (ya sea positivo o negativo) es que, bajo las circunstancias más comunes, ellos puedes ser propagados desde una operación hacia la siguiente (cualquier operación con un NaN como operando, produce un NaN como resultado), lo que puede producir que ciertos eventos sucedan, tales como:

- **Overflow:** ocurre cuando se produce un infinito.
- **Underflow:** ocurre cuando se produce un número desnormalizado.
- **Zerodivide:** ocurre si el divisor es cero, produciendo así un infinito con su apropiado signo, se debe tomar en cuenta que un divisor muy pequeño pero no cero, puede causar un overflow y producir un infinito.
- **Operand error:** ocurre cuando uno de los operandos está definido como un NaN

4.7 Ruta de Datos.

4.7.1 Fases en la ejecución de una instrucción.

La ejecución de una instrucción en el procesador, conlleva las siguientes fases:

- **Búsqueda de la instrucción (Instruction Fetch):** el procesador buscará en la memoria la instrucción a ejecutar, en el caso de este procesador, como la palabra es de 32 bits, esta se cargará en el registro de instrucción.

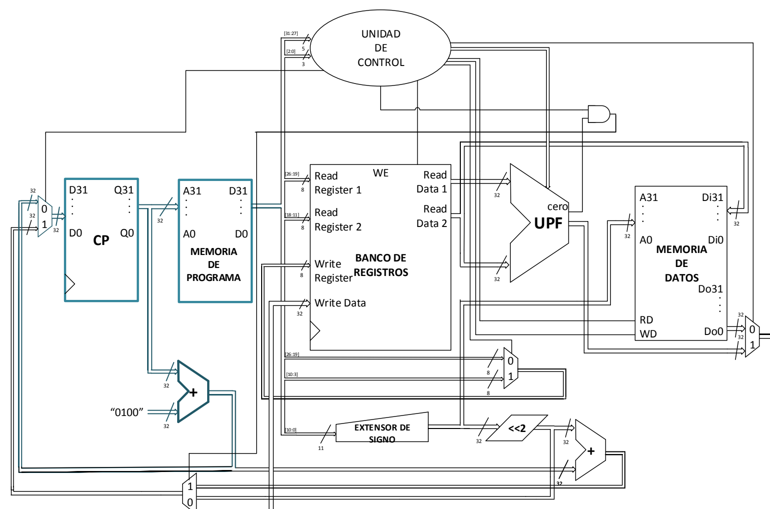


Fig 4.12. Diagrama de Bloques (Búsqueda de la Instrucción)

- **Decodificación de la instrucción (Instruction Decode):** se interpreta cada uno de los campos que forman la palabra de instrucción. El primer campo a interpretar siempre será el código de operación, pues nos indicará que instrucción se ejecutará.

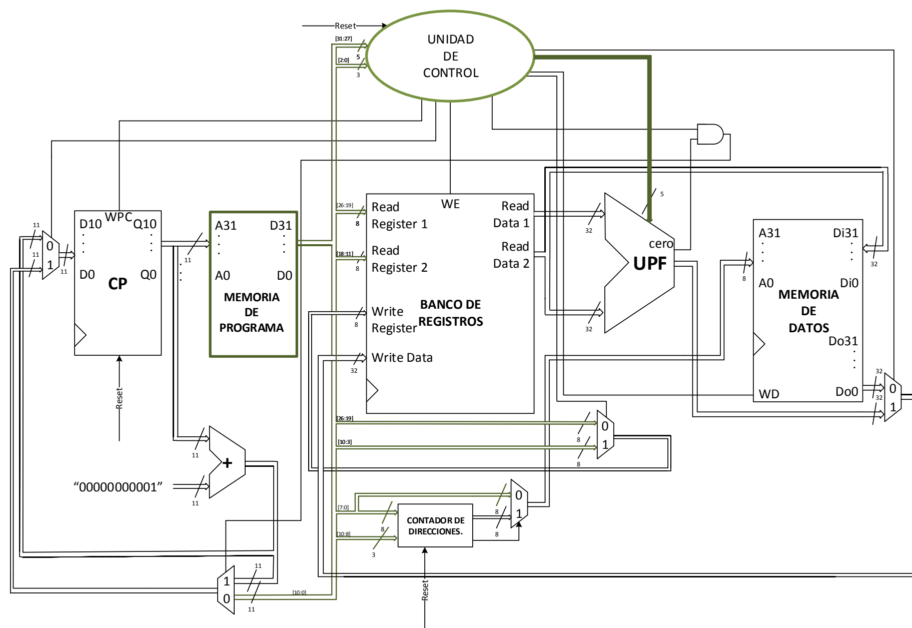


Fig 4.13. Diagrama de Bloques (Decodificación de la Instrucción)

- **Obtención de los operandos:** se localizan y se obtienen los operandos de la instrucción correspondiente y se almacenan en los registros respectivos.

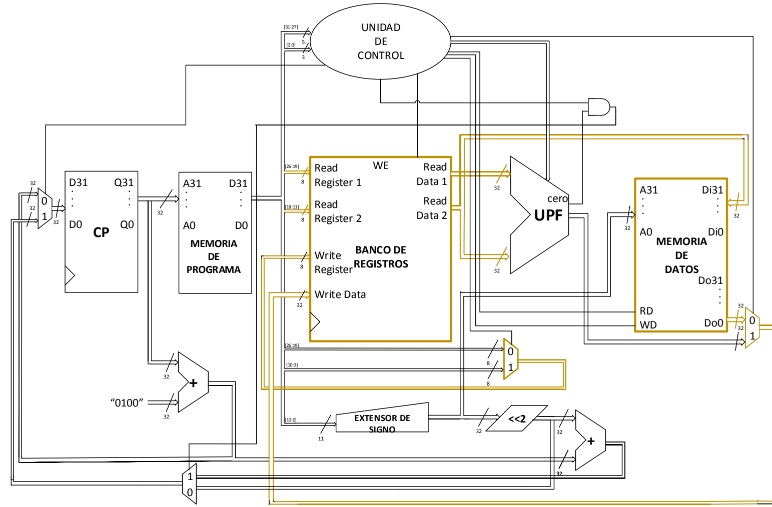


Fig 4.14. Diagrama de Bloques (Obtención de los operandos)

- **Ejecución de la operación** indicada por la instrucción.

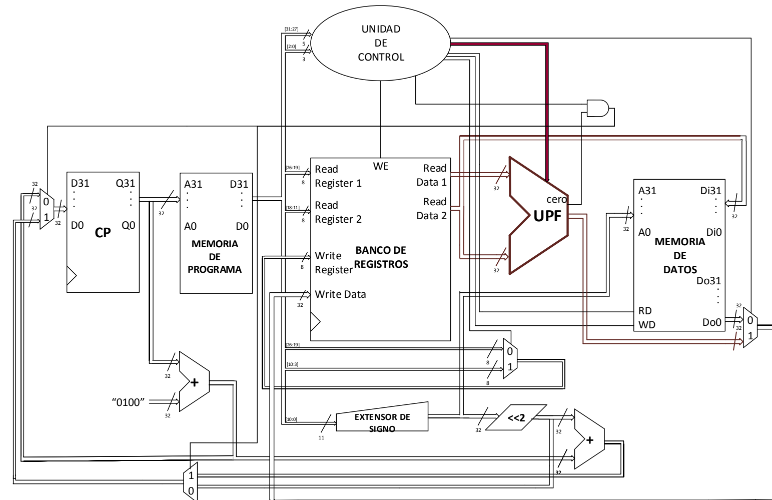


Fig 4.15. Diagrama de Bloques (Ejecución de la operación)

- **Almacenamiento del resultado** en caso de ser una operación aritmético-lógica.

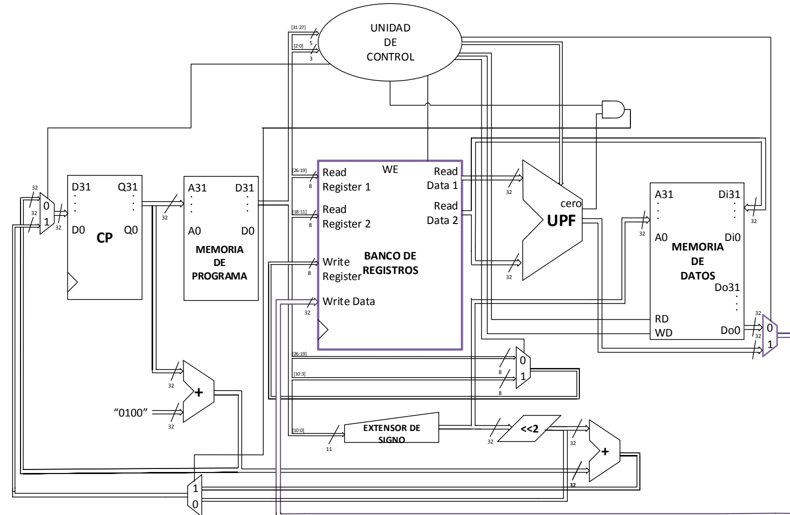


Fig 4.16. Diagrama de Bloques (Almacenamiento del Resultado)

- **Re-dirección** a la próxima dirección de memoria y ejecutar de nuevo las fases anteriores.

4.7.2 Instrucciones de trasferencia de datos: Carga (LW).

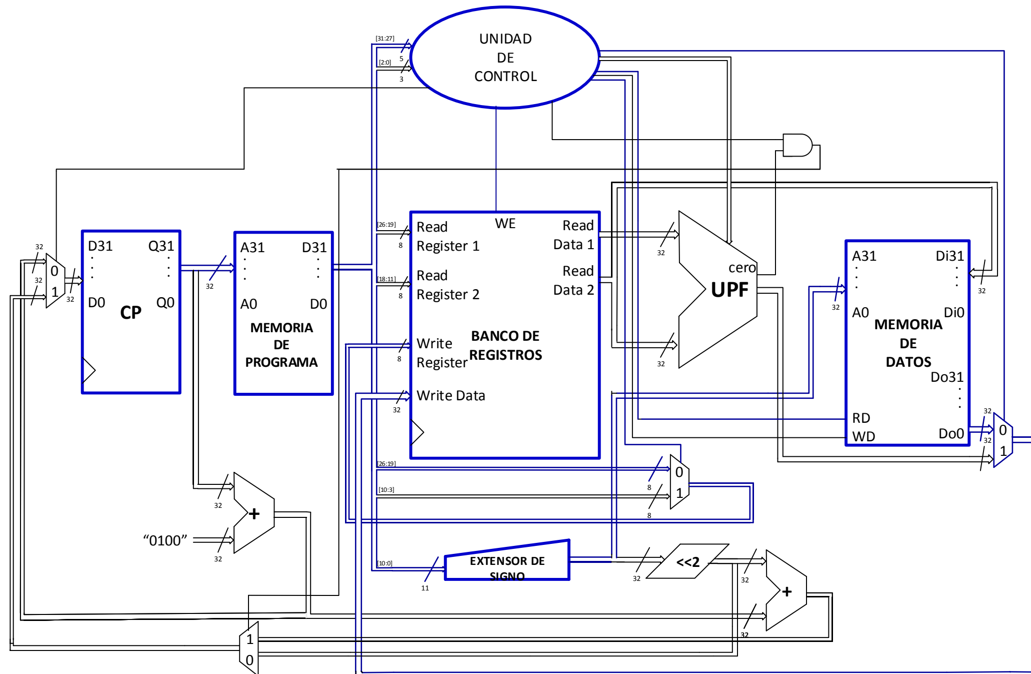


Fig 4.17. Ruta de Datos (Instrucción de Carga)

La ruta de datos de la instrucción de transferencia de datos, para este caso la instrucción de carga, inicia cuando el contador de programa (PC) indica a la memoria de programa la siguiente instrucción a leer, si la siguiente instrucción a ejecutarse es la instrucción de carga (LW), la memoria de programa obtiene el formato de la instrucción para poder mandar el bus correspondiente a cada unidad respectiva, es decir, los 5 bits correspondientes a la unidad de control, los 8 bits indicando el registro del operando a utilizar y los 11 bits restantes que indican la dirección de memoria de datos donde se obtendrá el dato en punto flotante. Una vez realizada la decodificación de la instrucción, los 11 bits de la dirección pasan por la unidad de extensor de signo, para obtener los 32 bits correspondientes al tamaño del bus de datos para la asignación de la dirección de memoria de datos a leer, la cual recibe 32 bits a la entrada. La unidad de control, por su parte, debe dejar en bajo al multiplexor que controla el paso de datos de los registros, así como poner en alto la bandera de solo lectura (RD) de la memoria de datos y dejar en bajo el bit de selección del multiplexor que controla el paso del dato en punto flotante ya sea proveniente de la memoria o de la Unidad de Punto Flotante, específicamente se deja el bit de selección inhabilitado puesto que el dato en punto flotante proviene de la memoria de datos. Así, el dato a ser escrito en el banco de registros se encuentra en la entrada correspondiente (Write Data) y el registro ocupado para almacenar dicho dato también (Write Register), por lo que la unidad de control debe activar la bandera de escritura sobre el banco de registros (WE) para poder cargar el dato proveniente de la memoria de datos al registro indicado.

4.7.3 Instrucciones de trasferencia de datos: Almacenamiento (SW).

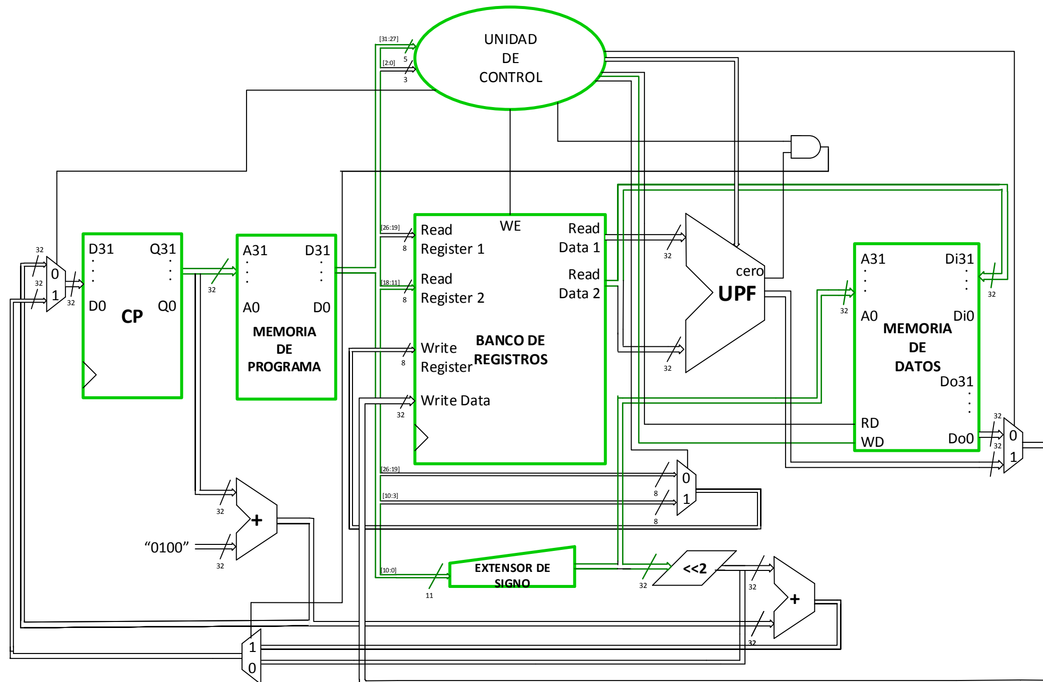


Fig 4.18. Ruta de Datos (Instrucción de Almacenamiento)

La ruta de datos que sigue la instrucción de almacenamiento (SW) comienza cuando el contador de programa (CP) una vez incrementado le indica a la memoria de programa la siguiente instrucción a leer, siendo esta instrucción, la instrucción de almacenamiento. Cuando la memoria de programa obtiene el formato de la instrucción, puede mandar el bus correspondiente a cada unidad implicada en la ruta, es decir, 5 bits para el código de operación direccionados a la unidad de control, 8 bits del registro del operando y 11 bits para la dirección de la memoria de datos donde se almacenará la información que haya en el operando anteriormente indicado. El banco de registros obtiene los bits del operando donde se hará la lectura del dato para posteriormente liberar el dato, una vez obtenido el dato a ser almacenado en la memoria de programa, esté se direcciona a la entra de datos de la memoria, mientras que la dirección de 11 bits obtenida del formato de instrucción para por el extensor de signo para asi poder obtener los 32 bits requeridos a la entrada de la dirección de la memoria de datos. Cuando los respectivos buses requeridos por la memoria de datos se encuentran en sus respectivas entradas y con sus datos correspondientes, la unidad de control habilita la instrucción de escritura de datos sobre la memoria de programa (WD), de tal modo que de esta manera el dato obtenido de un registro en el banco de registro, queda almacenado en la memoria de datos.

4.7.4 Instrucciones Aritmético-Lógicas: Tipo R

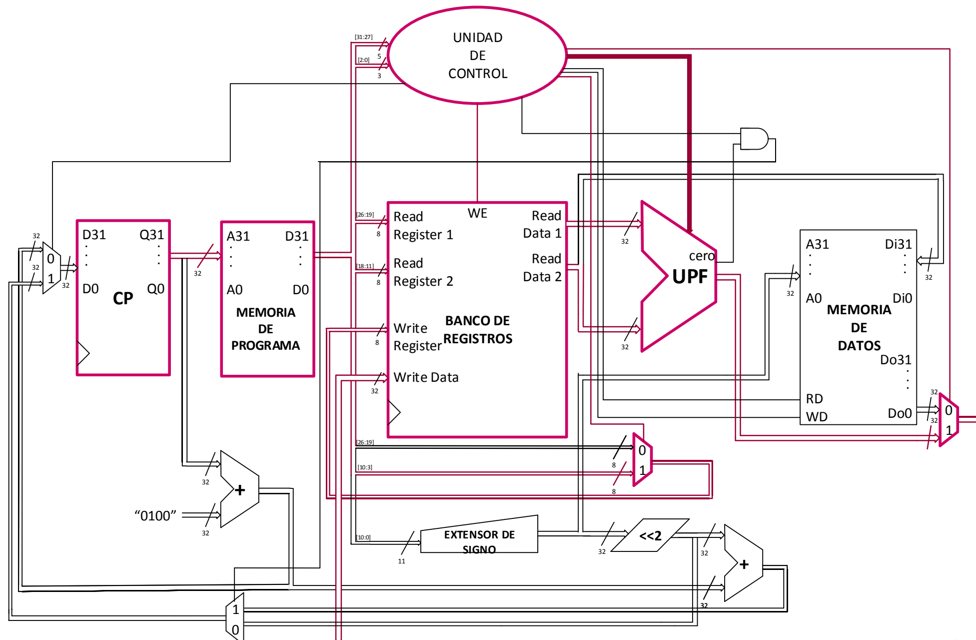


Fig 4.19. Ruta de Datos (Instruccioes Tipo R)

Todas las instrucciones aritmético-lógicas de tipo R siguen una misma ruta de datos, dicha ruta de datos inicia cuando el contador de programa (PC) apunta a la memoria de programa la siguiente instrucción a ser leída, cuando la memoria de programa obtiene los datos indicados por el formato de instrucción, manda el bus correspondiente para cada unidad ocupada por la ruta de datos, es decir 5 bits para el código de operación y 3 bits de control direccionados a la unidad de control, 8 bits para el primer registro del operando fuente, 8 bits para el segundo registro del operando fuente y 8 bits para el registro destino donde será almacena el resultado de la operación indicada, una vez decodificada la instrucción, el bando de registros da lectura a los datos almacenados en los registros indicados por el formato de instrucción, los datos obtenidos por el banco de registros ingresan a la Unidad de Punto Flotante para ser operados por dicha unidad cuando la unidad de control indica la operación a ejecutarse mediante el código de operación de la Unidad de Punto Flotante. Una vez ejecutada la operación de la Unidad de Punto Flotante el resultado obtenido es direccionado hacia el multiplexor que indica el paso de datos provenientes de la UPF, es decir que la unidad de control activa en alto el bit de selección de dicho multiplexor, para que la entrada de datos del banco de registros (Write Data) sea el resultado de la UPF, así mismo la unidad de control debe poner en alto el multiplexor que controla si se ocupará el primer registro del operando fuente o el registro destino, por lo que al colocar en alto el bit de selección de este multiplexor, indica que el dato a ser escrito será sobre el registro destino, finalmente la unidad de control debe habilitar la bandera de escritura del banco de registros (WE), para que el resultado obtenido sea almacenado sobre el registro indicado.

4.7.5 Instrucciones de Salto: BEQ

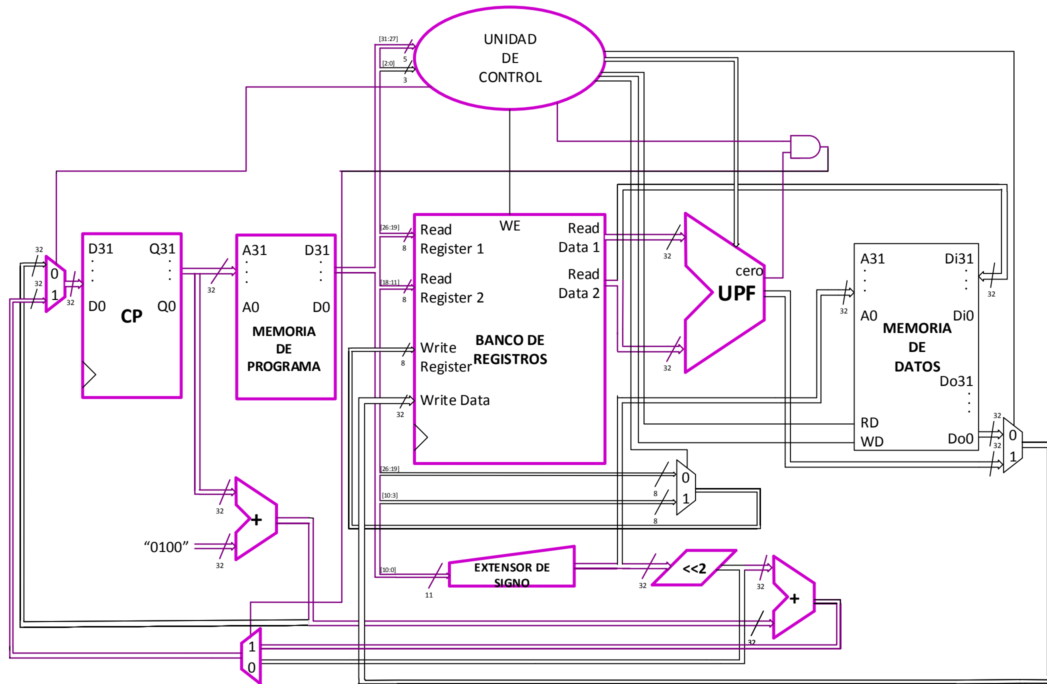


Fig 4.20. Ruta de Datos (Salto Condicional)

La ruta de datos seguida por la instrucción de salto condicional (BEQ), da comienzo cuando el contador de programa (CP), una vez incrementado, indica la siguiente dirección de la memoria de programa donde se encuentra la instrucción a ejecutarse, si la instrucción a ejecutarse es una instrucción de brinco condicional, el formato de la instrucción es obtenido de la memoria de programa, al mismo tiempo la dirección emitida por el contador de programa, pasa por un sumador de enteros, el cual le suma el número 4, y el resultado de este sumador es enviado a la entrada de un segundo sumador. Los datos obtenidos por la memoria de programa son decodificados, es decir, los 5 bits del código de operación son direccionados a la entrada de la unidad de control, los 8 bits del primer registro del operando fuente direccionados a la entrada read register 1 del banco de registros, los 8 bits del segundo registro del operando fuente direccionados a la entrada read register 2 del banco de registros y los 11 bits que contienen la dirección para realizar el salto son dirigidos a la unidad de extensor de signo. El banco de registros se encarga de leer los datos almacenados en los registros indicados, obtiene los datos y los envía a las respectivas entradas de la Unidad de Punto Flotante, la cual realiza la operación de resta, indicada por la unidad de control, cuando la UPF realiza la resta, si los datos son distintos, la bandera de cero de la UPF se mantiene en bajo, en cambio si los datos son iguales, la bandera se mantendrá en alto, es decir que al momento de que la unidad de control coloque en alto la bandera que llega a la entra de la compuerta AND, dicha compuerta activará en alto el bit de selección del multiplexor que indica el paso del resultado obtenido del sumador de

enteros, quien se encarga de sumar los 32 bits que el extensor de signo envía, más el resultado del sumador anteriormente mencionado. Este resultado es la nueva dirección de salto la cual llegara al contador de programa mediante la activación en alto del bit de selección del último multiplexor enviado por la unidad de control.

4.7.6 Instrucciones de Salto: B

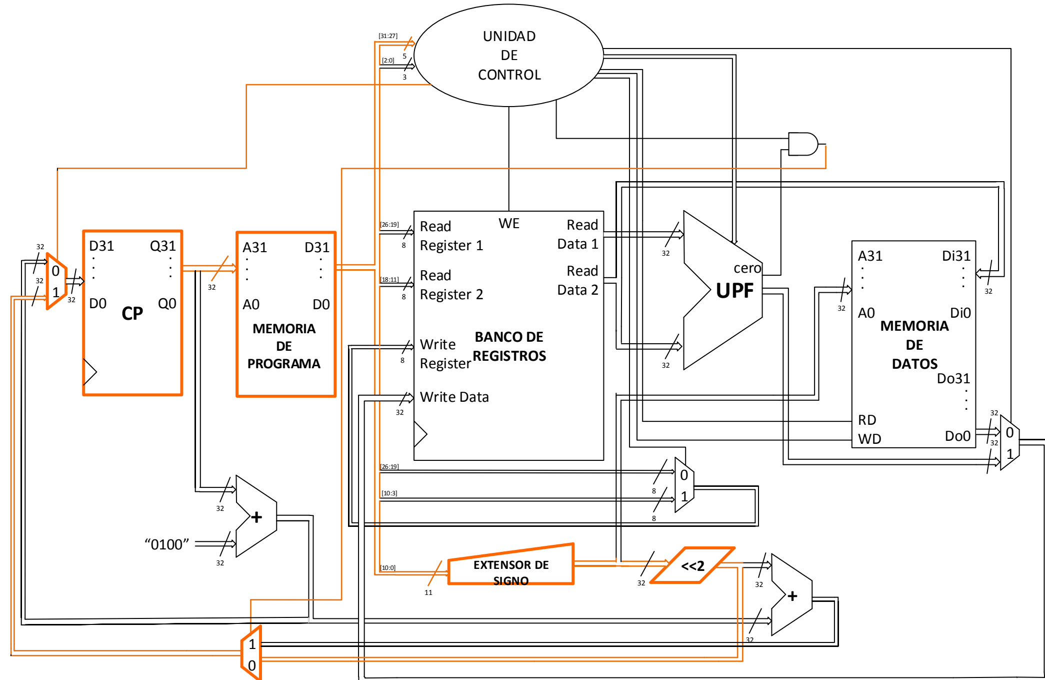


Fig 4.21. Ruta de Datos (Salto Incondicional)

La ruta de datos que sigue la instrucción de salto incondicional es la menos extensa, pues inicia cuando el contador de programa le indica a la memoria de programa la siguiente instrucción a ejecutarse, cuando la memoria de programa obtiene los datos pertenecientes al formato de instrucción, los cuales solo son dos, los 5 bits para el código de operación direccionados a la unidad de control y los 11 bits de la dirección a saltar, la cual pasa por el extensor de signo, se realiza un corrimiento de dos bits y llega a la entrada del multiplexor, el cual cuando la unidad de control mantiene en bajo el bit de selección del mismo, la nueva dirección obtenida, pasa a la entra del ultimo multiplexor, donde la unidad de control deberá colocar en alto el bit de selección para dar paso de la nueva dirección a la entrada del contador de programa.

5 CAPITULO 5

5.1 Aplicación.

5.1.1 Redes Neuronales [17] [18].

Las computadoras modernas comienzan a ser cada vez más poderosas, los científicos continúan con el reto de utilizar maquinas efectivas para la realización de tareas simples para los humanos, es decir, los humanos con una correcta retroalimentación proveniente de un “maestro”, aprendemos fácilmente a reconocer una letra A, o distinguir a un gato de un pájaro, a tomar decisiones, tratar de lograr una meta que consiste en maximizar los recursos mientras se satisfacen ciertas restricciones, cada uno de estos problemas ilustra las tareas para las cuales se buscaban soluciones computables.

El renovado interés reciente puede ser atribuido a distintos factores, uno de ellos es a la alta velocidad en que las computadoras digitales realizan la simulación de un proceso neuronal más factible. La tecnología de hoy en día, está disponible para producir hardware especializado para redes neuronales. El nivel de éxito alcanzado por la computación tradicional se acerca a muchos tipos de problemas dejando espacio para la consideración de alternativas, por ejemplo, los ingenieros en computación están intrigados por el potencial de hardware para implementar redes neuronales eficientemente y por las aplicaciones de las redes neuronales en la robótica.

El estudio de las redes neuronales es un campo extremadamente interdisciplinario, tanto en su desarrollo como en sus aplicaciones. Un claro ejemplo es que las redes neuronales son implementadas para el procesamiento de señales.

Existen bastantes aplicaciones de las redes neuronales en el área general de procesamiento de señales. Una de las primeras aplicaciones más comerciales fue y sigue siendo la supresión de ruido en una línea telefónica. La red neuronal para este propósito es una forma de ADALINE, la necesidad de adaptar los canceladores de ruido ha crecido exponencialmente con el desarrollo de los satélites transcontinentales para comunicar largas distancias vía telefónica, entre otras aplicaciones.

Como conclusión las aplicaciones de las redes neuronales, el dinero que ha sido invertido en software y hardware para redes neuronales, la profundidad y amplitud del interés en estos dispositivos ha estado creciendo rápidamente.

5.1.1.1 Redes neuronales artificiales

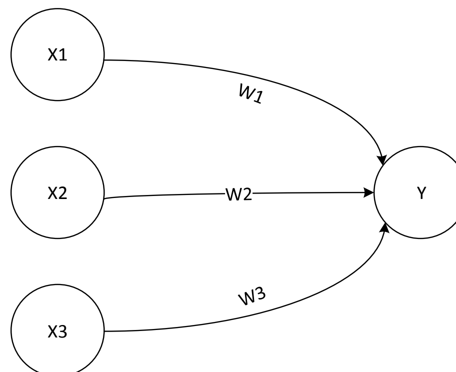
Una red neuronal artificial es un sistema de procesamiento de información que cuenta con ciertas características de desempeño en común con una red neuronal biológica, es decir:

1 El elemento de procesamiento recibe muchas señales.
2 Las señales pueden ser modificadas por un valor ponderado adquirido de la sinapsis
3 El elemento de procesamiento suma las entradas ponderadas.
4 Bajo ciertas circunstancias (suficientes entradas), la neurona transmite una sola salida.
5 La salida de una neurona en particular, suele ir a otras neuronas.

Por ejemplo, considerar una neurona Y, mostrada en la figura posterior, que recibe entradas desde otras neuronas $X_1, X_2\ y\ X_3$. Las activaciones (señales de salida) de estas neuronas son $x_1, x_2\ y\ x_3$ respectivamente. Los pesos sobre las conexiones de $X_1, X_2\ y\ X_3$ hacia la neurona Y son $w_1, w_2\ y\ w_3$ respectivamente. La entrada de la red,y_in hacia la neurona Y, es la suma de las señales ponderadas de las neuronas $X_1, X_2\ y\ X_3$.

$$y_{in} = w_1x_1 + w_2x_2 + w_3x_3$$

La activación de una neurona Y, está dada por alguna función de su red de entrada.



5.1.1.1 Una neurona (artificial) simple

La operación básica de una neurona artificial implica sumar sus señales de entrada ponderadas y aplicar una salida, o activación, o función. Para las unidades de entrada, esta función es la función de identidad, la función de identidad está representada por: $f(x) = x,\ para\ toda\ x$.

Método de multiplicación matricial para calcular los valores de entrada

Si las conexiones ponderadas de una red neuronal son almacenadas en una matriz $W = (w_{ij})$, la red de entrada a la unidad Y_J (con ningún sesgo sobre la unidad j), es

simplemente el producto punto de los vectores $x = (x_1, \dots, x_i, \dots, x_n)$ y $w_j(la\ j - esima\ \ columna\ del\ peso\ de\ la\ matriz)$.

$$y_in_j = x \,.\, w_j$$

$$= \sum_{i=0}^{n} x_i w_{ij}$$

Sesgo

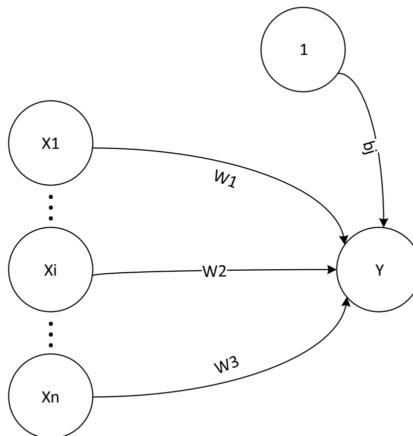
Un sesgo puede ser incluido sumando un componente $X_0 = 1$ al vector x, por ejemplo: $x = (1, x_1, \dots, x_i, \dots, x_n)$. El sesgo es tratado exactamente como cualquier otro peso, suponiendo que $w_{0j} = b_j$. La entrada de la red hacia la unidad Y_j está dada por:

$$y_in_j = x \,.\, w_j$$

$$= \sum_{i=0}^{n} x_i w_{ij}$$

$$= w_{0j} + \sum_{i=0}^{n} x_i w_{ij}$$

$$= b_j + \sum_{i=0}^{n} x_i w_{ij}$$

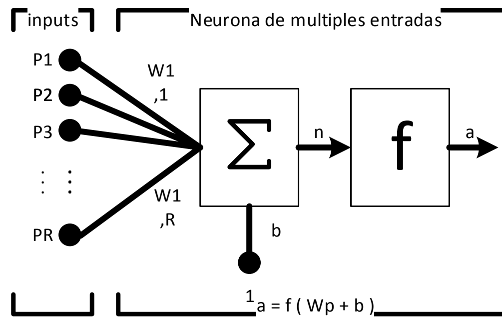


5.1.1.2 Neurona con sesgo

Modelo Neuronal

Neurona de múltiples entradas.

La neurona de múltiples entradas (R) es como se muestra en la siguiente figura. Las entradas individuales $P_1, P_2, \dots, P_R$ son ponderadas por sus correspondientes elementos $w_{1,1}, w_{1,2}, \dots, w_{1,R}$ de los pesos de la matriz W.



5.1.1.3 Neurona de múltiples entradas

La neurona anterior tiene un sesgo b, el cual es sumado junto con las entradas ponderadas para formar la red de entrada n.

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b$$

La expresión anterior puede ser descrita en una matriz como:

$$n = Wp + b$$

Donde la matriz W de una neurona simple, tiene solo una celda., asi la salida de la neurona puede ser escrita como:

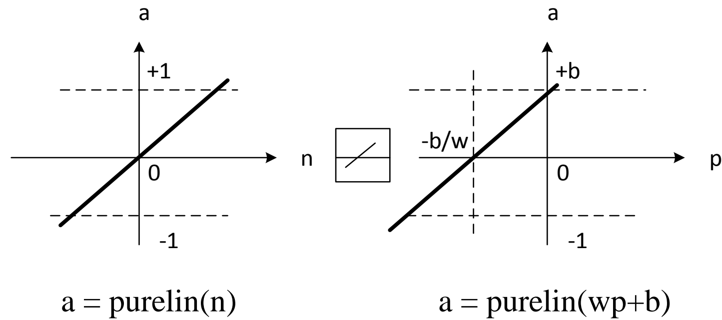
$$a = f(Wp + b)$$

Función de transferencia linear (f)

La salida de una función de transferencia linear es igual a su entrada:

$$a = n$$

Neuronas con esta función de transferencia, son usadas en las redes ADALINE.



5.1.1.4 Función de transferencia linear

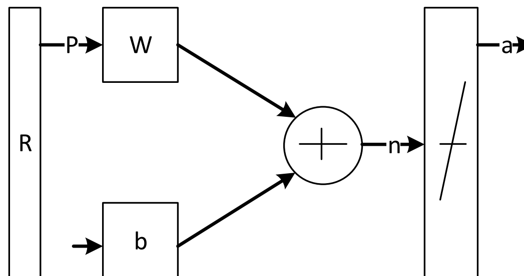
La salida (a) contra la salida (p) característica de una neurona simple lineal con sesgo se mostró anteriormente.

5.1.1.2 Red ADALINE

La neurona ADALINE (adaptative linear neuron) típicamente emplea pesos en las conexiones de los valores de entrada para que sean ajustables. ADALINE también cuenta con un sesgo, el cual actúa como un peso ajustable en una conexione desde una unidad la cual su activación siempre estará dada en uno. En general, una neurona ADALINE puede ser entrenada usando una regla delta, mejor conocida como LMS (least mean squares). La activación de la unidad es su propia red de entrada, es decir, que la función de activación es la función de identidad. La regla de aprendizaje, minimiza el error entre la activación el valor objetivo.

Arquitectura.

Una neurona ADALINE es una neurona simple de una unidad, la cual recibe entradas de varias unidades. También recibe una entrada de una "unidad" la cual siempre tiene el valor de +1, para que el peso del sesgo sea entrenado por el mismo proceso (regla delta), el cual es usado para poder entrenar a los otros pesos.



5.1.1.5 Red ADALINE

La salida de la red esta dada por:

$$a = purelin(Wp + b)$$

Algoritmo.

El algoritmo de entrenamiento de una neurona ADALINE es el siguiente:

1 Paso 0. Inicializar los pesos (pequeños valores aleatorios)
Colocar el valor de aprendizaje a (el valor de aprendizaje es usado para controlar la cantidad de peso ajustable para cada paso del entrenamiento, generalmente los valores utilizados son $0.1 < a < 1$)

2 Paso 1. Mientras la condición de paro sea falsa, realiza los pasos 2-6.

3 Paso 2. Colocar la activación de las unidades de entrada

4 Paso 3. Calcular la entrada de la red hacia la salida de la unidad

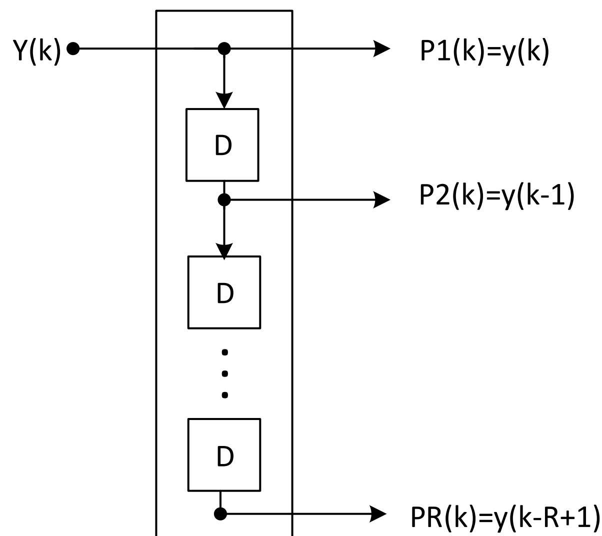
$$y_in = b + \sum_{1} x_i w_i$$

5 Paso 4. Actualizas sesgo y pesos

6 Paso 5. Verificar la condición de paro.

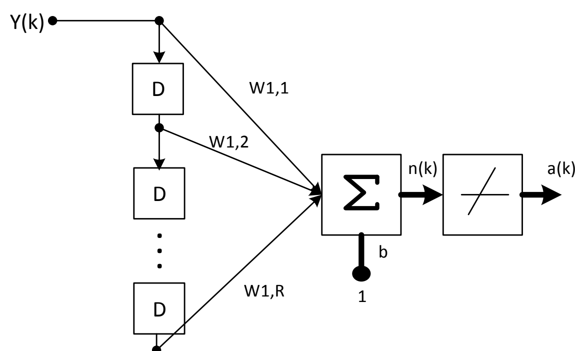
5.1.1.3 Filtro Adaptativo.

Para usar la red neuronal ADALINE como un filtro adaptativo, se necesita implementar un bloque nuevo, denominado línea de retardo muestreado, como se muestra a continuación:



5.1.1.6 Bloque línea de retardo muestreado

Si se combina el bloque línea de retardo muestreado con la red neuronal ADALINE, se crea un filtro adaptativo como se muestra en la siguiente figura:



5.1.1.7 Filtro Adaptativo ADALINE

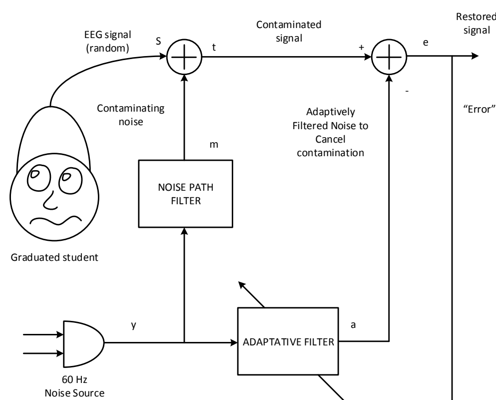
La salida de este filtro está dado por:

$$a(k) = purelin(Wp + b) = \sum_{I=1}^{R} W_{1,i}\, y(k - i + 1) + b$$

$$a(k) = purelin(Wp(k) + b)$$

Ejemplo

Supongamos que un doctor, está tratando de revisar el encefalograma (EEG) de un estudiante graduado distraído, encuentra que la señal que a él le gustaría ver ha sido contaminada por una fuente de ruido a 60 Hz. Él está examinando al paciente conectado y quiere revisar la mejor señal posible a obtener. La siguiente figura muestra como un filtro adaptativo puede ser usado para remover el ruido de la señal contaminada.



5.1.1.8 Sistema cancelador de ruido.

5.1.1.4 Algoritmo de la red neuronal ADALINE sobre el procesador en punto flotante.

LW R0, x”00” **; R0= MEM[0]** – Carga del número de muestras al registro 0
LW R1, x”01” **; R1= MEM[1]** – Carga el número uno al registro 1
LW R2, x”02” **; R2= MEM[2]** – Carga el número cero al registro 2
LW R6, x”02” **; R6= MEM[2]** – Carga el número cero al registro 6

CICLO:
LW R4, x”0D” **; R4= MEM[13**] – Carga el valor del peso
LW R5, x”03” **; R5= MEM[3]** -- Carga el valor de la señal
MULT R5, R4, R5 **; R5 = R4 * R5** – Multiplica el peso por la señal
ADD R6, R5, R6 **; R6 = R5 + R6** – Realiza la suma acumulatoria.
SUB R0, R1, R0 **; R0 = R1 – R0** -- Resta el número de muestras menos uno
BNEQ R0, R2, x”04” **; if (R2!=R0) go to CICLO** – Si el número de iteraciones es distinto al número de muestras, salta a la etiqueta CICLO, en caso contrario, continúa.

ADD R6, R1, R6 **; R6 = R1 + R6** – Suma el sesgo a la suma acumulatoria.
SW R6, x”FF” **; MEM[255] = R6** – Almacena en memoria el valor del registro 6.
LW R7, x”FF” **; R7 = MEM[255]** – Carga el resultado final sobre el registro 7.
B x”0C” **; PC = 12** – Brinca a la dirección 12

6 CAPITULO 6

6.1 Implementación de la Unidad de Punto Flotante

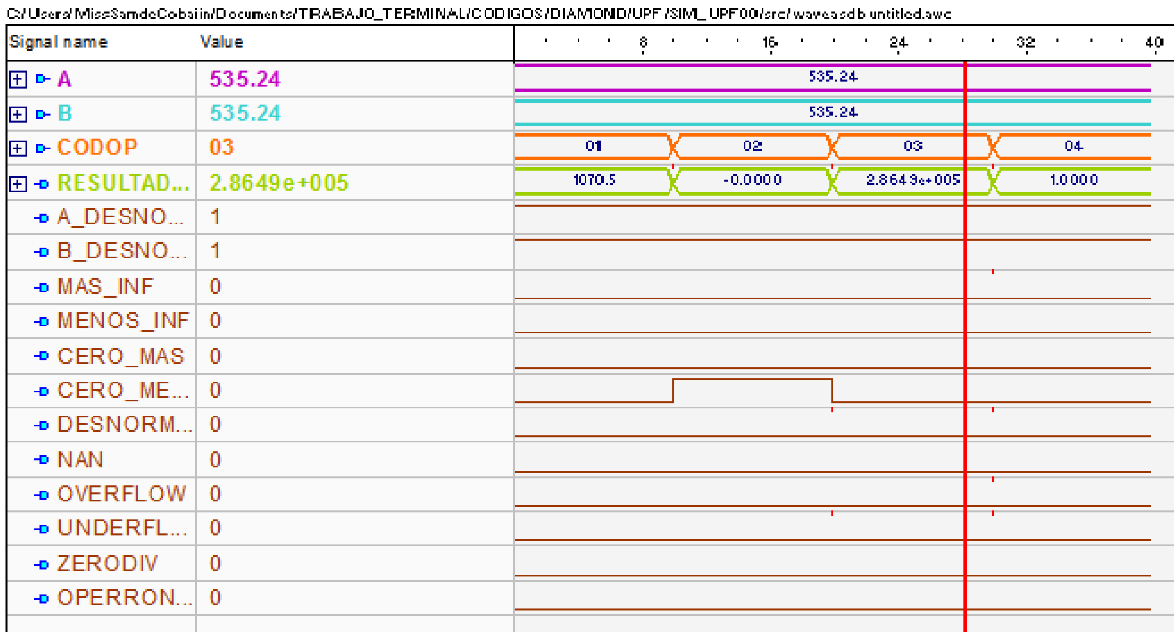


Fig 6.1 Simulación de la UPF

6.1.1 Módulo Check

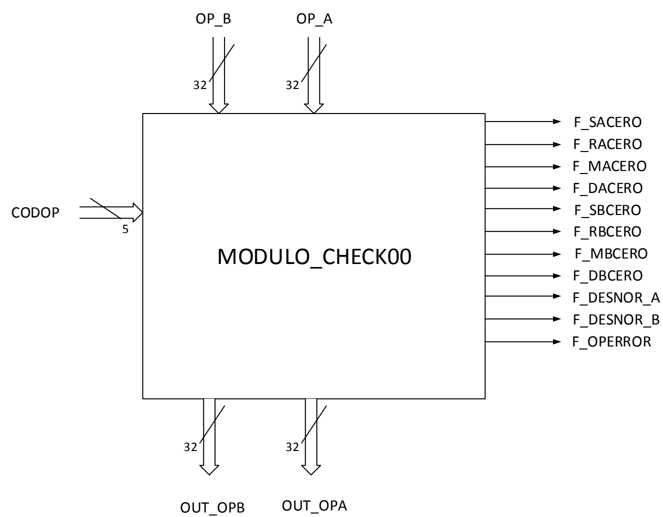


Fig 6.2 Diagrama del módulo CHECK

El módulo Check únicamente incluye tres fases de la Unidad de Punto flotante, las cuales son la desnormalizacion inicial del operando, la verificación por operandos erróneos y la verificación por operandos en cero. Cuenta con respectivos buses de entrada así como banderas que se mostraran en las simulaciones de las siguientes pruebas realizadas.

Pruebas realizadas sobre el Módulo Check:

1. OP_A normalizado => OUT_OPA desnormalizado

OP_A				OUT_OPA			
s	exp	mantisa	>	s	exp	mantisa	F_DESNOR_A
1	00000000	11100000111110000011111		1	00000001	11110000011111000001111	1

2. OP_B normalizado => OUT_OPB desnormalizado

OP_B				OUT_OPB			
s	exp	mantisa	>	s	exp	mantisa	F_DESNOR_B
1	00011100	00011111000001111100000		1	00011101	10001111100000111110000	1

3. OP_A & OP_B normalizado => OUT_OPA & OUT_OPB desnormalizado

ENTRADA					SALIDA			
	s	exp	mantisa	>	s	exp	mantisa	F_DESNOR_x
OP_A	1	00000000	11100000111110000011111		1	00000001	11110000011111000001111	1
OP_B	1	00011100	00011111000001111100000		1	00011101	10001111100000111110000	1

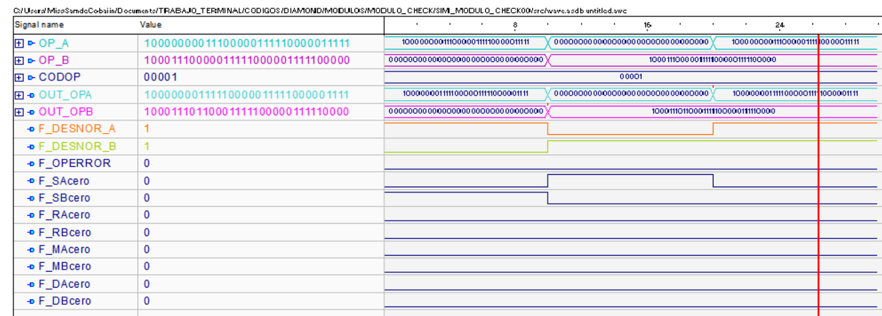


Fig 6.3 Primera simulación del módulo CHECK

4. OP_A es un NaN (Not a Number) => OUT_OPA = “x00…0”

OP_A				OUT_OPA			
s	exp	mantisa	>	s	exp	mantisa	F_OPERROR
0	11111111	11100000111110000011111		0	00000000	00000000000000000000000	1

5. OP_B es un NaN (Not a Number) => OUT_OPB = “x00…0”

OP_B				OUT_OPB			
s	exp	mantisa	>	s	exp	mantisa	F_OPERROR
0	11111111	00011111000001111100000		0	00000000	00000000000000000000000	1

6. OP_A & OP_B son un NaN (Not a Number) => OUT_OPB = “x00…0” & OUT_OPA = “x00…0”

ENTRADA				>	SALIDA			
	s	exp	mantisa		s	exp	mantisa	F_OPERROR
OP_A	0	11111111	11100000111110000011111		0	00000000	00000000000000000000000	1
OP_B	0	11111111	00011111000001111100000		0	00000000	00000000000000000000000	1

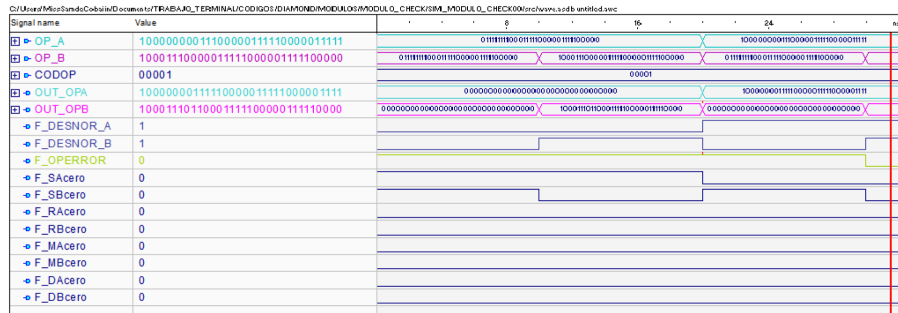


Fig 6.4 Segunda simulación del módulo CHECK

7. OP_A = “x000…0” & CODOP=”00001” | CODOP=”00010” | CODOP=”00011”| CODOP=”00100” => F_SAcero=’1’| F_RAcero=’1’| F_MAcero=’1’| F_DAcero=’1’.

CODOP	OP_A			>	BANDERAS CERO (A)			
	s	exp	mantisa		F_SAcero	F_RAcero	F_MAcero	F_DAcero
00001	0	00000000	00000000000000000000000		1	0	0	0
00010	0	00000000	00000000000000000000000		0	1	0	0
00011	0	00000000	00000000000000000000000		0	0	1	0
00100	0	00000000	00000000000000000000000		0	0	0	1

8. OP_B = “x000…0” & CODOP=”00001”| CODOP=”00010”| CODOP=”00011”| CODOP=”00100” => F_SBcero=’1’| F_RBcero=’1’| F_MBcero=’1’| F_DBcero=’1’.

CODOP		OP_B			>	BANDERAS CERO (A)			
		S	exp	mantisa		F_SBcero	F_RBcero	F_MBcero	F_DBcero
00001		0	00000000	00000000000000000000000		1	0	0	0
00010		0	00000000	00000000000000000000000		0	1	0	0
00011		0	00000000	00000000000000000000000		0	0	1	0
00100		0	00000000	00000000000000000000000		0	0	0	1

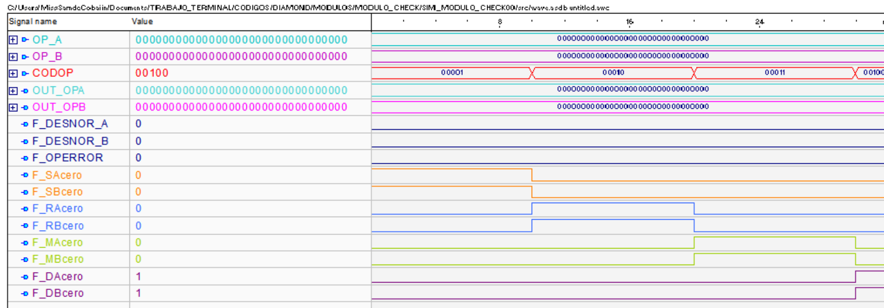


Fig 6.5 Tercera simulación del módulo CHECK

6.1.1.1 Desnormalización Inicial.

El módulo de desnormalizar cuenta con un bus de entrada de 32 bits (OP_A) y un bus de salida del mismo tamaño que el bus de entrada (S_OPA), internamente cuenta con una bandera denominada DESNOR, la cual indica si el numero ingresado en punto flotante es un número a desnormalizar. Un número desnormalizado en punto flotante, tiene la característica de que el valor del exponente del mismo es igual a cero y la mantisa es un valor distinto de cero, mientras que el valor del signo de dicho número es indistinto, de modo que se verifican dichas condiciones para activar la bandera en alto en dado caso que sea un número desnormalizado. Cuando la bandera DESNOR se encuentra en alto, el módulo desnormalizar se encarga de desnormalizar el numero en punto flotante; para normalizar un numero en punto flotante a nivel de diseño de hardware, basta con hacer el corrimiento de un bit hacia la derecha sobre la mantisa e incrementar el exponente, en otros términos, sumar un ‘1’ al exponente. Dichos cambios se ven reflejados en la siguiente simulación:

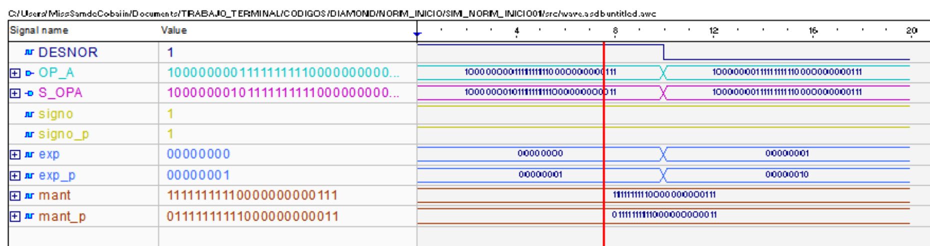


Fig 6.6 Simulación del módulo desnomalización

6.1.1.2 Verificación por operandos erróneos.

La bandera F_OPERROR se coloca en alto cuando el exponente del número en punto flotante es igual a “11111111”, el valor del signo del número es indistinto y cuando el valor de la mantisa del mismo número en punto flotante es distinto

de cero, por lo tanto cuando la bandera esta activada a ‘1’ indica que se trata de un Not a Number (NaN). El valor de la salida OUT_OPA, que es el bus de datos indicativo del número en punto flotante adquiere el valor de cero. En caso contrario cuando el valor de la Bandera F_OPERROR es igual a ‘0’ indica que es un número en punto flotante, pues el exponente del número en punto flotante es distinto de “11111111”. Tal y como se muestra en la siguiente figura:

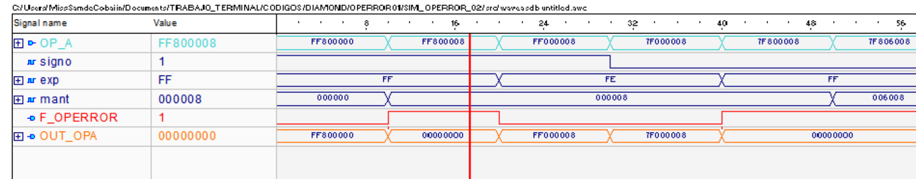


Fig 6.7 Simulación del módulo verificación de operandos

6.1.1.3 Verificación por operandos en cero.

El presente modulo cuenta con dos buses de entrada, CODOP y OP_A, el primero indica el código de operación a realizar y el segundo es el valor del número en punto flotante a verificar, cuenta con un bus (S_OPA) y 4 banderas de salida (F_SAcero, F_RAcero, F_MAcero, F_DAcero). Básicamente el modulo se encarga de verificar si el numero en punto flotante es un cero, ya sea cero positivo o cero negativo). La bandera F_SAcero únicamente se activa cuando el código de operación es igual a 1, indicando que se trata de una operación de suma, y el número en punto flotante es igual a cero. F_RAcero únicamente se activa cuando el código de operación es igual a 2, indicando que se trata de una operación de resta, y el número en punto flotante es igual a cero. F_MAcero únicamente se activa cuando el código de operación es igual a 3, indicando que se trata de una operación de multiplicación, y el número en punto flotante es igual a cero. F_DAcero únicamente se activa cuando el código de operación es igual a 4, indicando que se trata de una operación de división, y el número en punto flotante es igual a cero. El comportamiento de las características de dicho módulo puede verse reflejadas en las siguientes simulaciones:

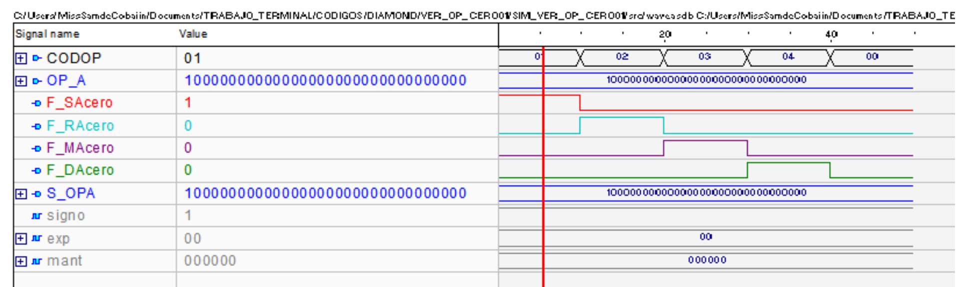


Fig 6.8 Primera simulación del módulo operandos en cero

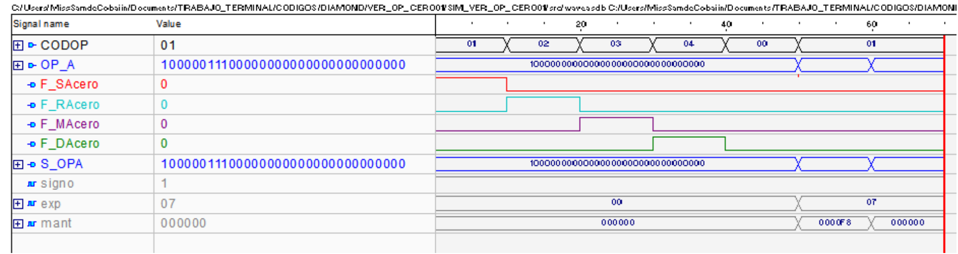


Fig 6.9 Segunda simulación del módulo operandos en cero

6.1.2 Módulo Ready

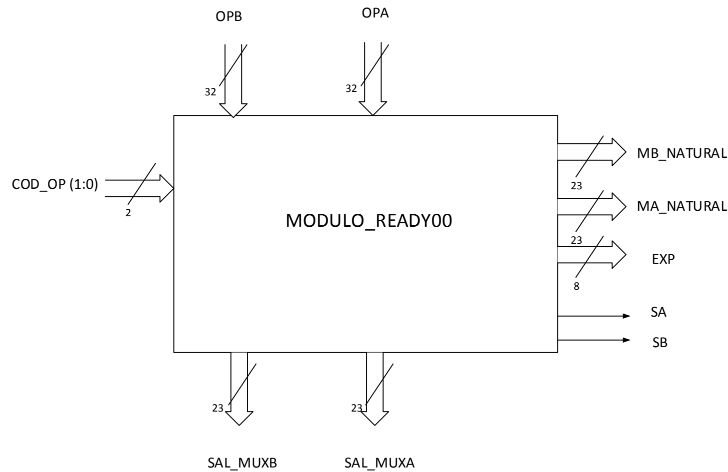


Fig 6.10 Diagrama del módulo READY

El módulo ready cuenta con diez entidades diferentes que preparan a los operandos para poder ser operados, las siguiente simulación ejemplifica su funcionamiento.

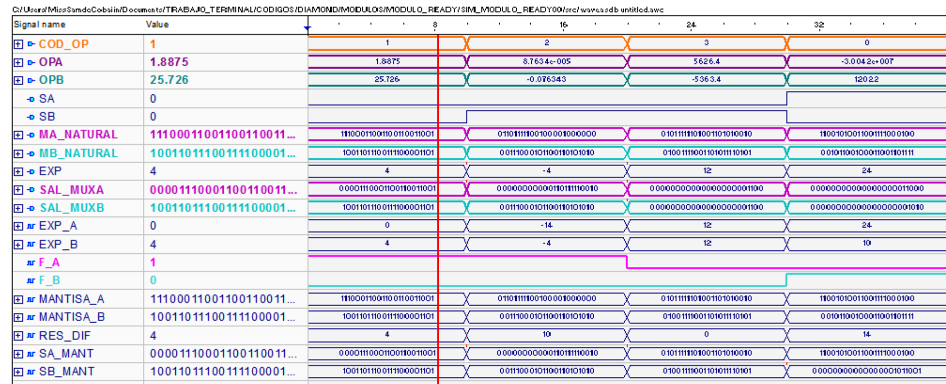


Fig 6.11 Simulación del módulo READY

6.1.2.1 Resta del sesgo al exponente

Este módulo cuenta con un bus de 32 bits de entrada denominado OP, y 1 bit de salida indicando el signo del número en punto flotante (signo), un bus de salida de 8 bits indicando el exponente del número en punto flotante (exp) y finalmente un bus de salida de 23 bits indicando el valor de la mantisa del número en punto flotante (mantisa).

ENTRADA					SALIDA			
	s	exp	mantisa		s	exp	mantisa	F_OPERROR
-118.625	1	10000101	11011010100000000000000	→	1	00000110	11011010100000000000000	1
	-	133	DA8000		-	6	DA8000	
53	0	10000100	10101000000000000000000		0	00000101	10101000000000000000000	1
	+	132	540000		+	5	540000	
-6.5	0	10000001	10100000000000000000000		0	00000010	10100000000000000000000	1
	+	129	500000		+	2	500000	

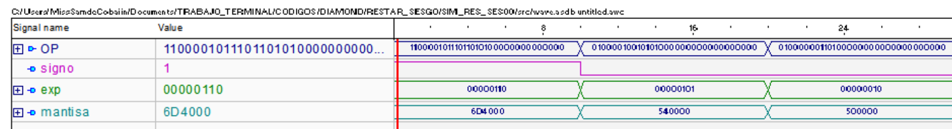


Fig 6.12 Simulación del módulo resta del sesgo

6.1.2.2 Comparación entre exponentes.

El modulo comparador cuenta de dos módulos, el primero encargado de detectar cual exponente es mayor que otro, o si es igual; el segundo se encarga de realizar la diferencia entre dichos números para conocer cuántos bits se deben recorrer sobre la mantisa y así igualar los exponentes para poder realizar operaciones de suma/resta. En sí, el modulo en general, recibe dos buses de entrada, A y B, los cuales son los respectivos exponentes de los números en punto flotante a operar, realiza la comparación de cual exponente es mayor, menor o si ambos son iguales y sobre ello, enciende las bandera de F_COMP_A, en caso de que B sea mayor que A y el corrimiento deba realizar sobre la mantisa de A e incrementar el ponente de A, o F_COMP_B, en caso de que A sea mayor que B y el corrimiento deba realizarse sobre la mantisa de B e incrementar el exponente de B, en dado caso que los exponentes sean iguales, ninguna de las dos banderas será activada, finalmente arroja como resultado la diferencia de dichos valores en los buses, la cual indica cuantos lugares se verá afectada el corrimiento sobre la mantisa de cual sea el numero en punto flotante a modificar.

ENTRADAS					SALIDAS			
exp_A		exp_B			DIF		F_COMP_A	F_COMP_B
12	00001100	2	00000010		10	00001010	0	1
-12	11110100	-2	11111110		10	00001010	1	0
10	00001010	-10	11110110		20	00010100	0	1
-5	11111011	5	00000101		10	00001010	1	0
127	01111111	-128	10000000		255	11111111	0	1
85	10101010	85	10101010		0	00000000	0	0

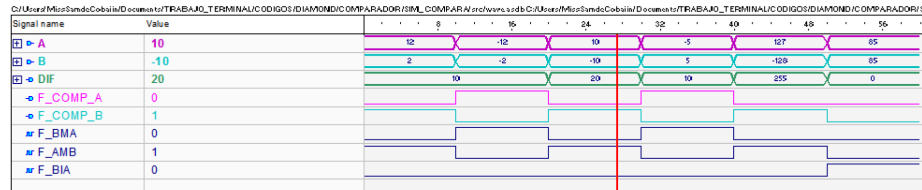


Fig 6.13 Simulación del módulo comparación

6.1.2.3 Sumatoria al exponente

Dicho modulo recibe a la entrada un bus de 8 bits denominado EXP_A y un bus de 8 bits denominado DIFF, el primero indica el valor actual del exponente, y el segundo indica la diferencia que debe ser sumada al primer bus de datos para obtener un bus de salida denominado SAL siempre y cuando la bandera F_COMP_A este activada en alto, en caso contrario, solamente a la salida se mostrara el valor actual de EXP_A. Tal y como se muestra en la siguiente simulación:

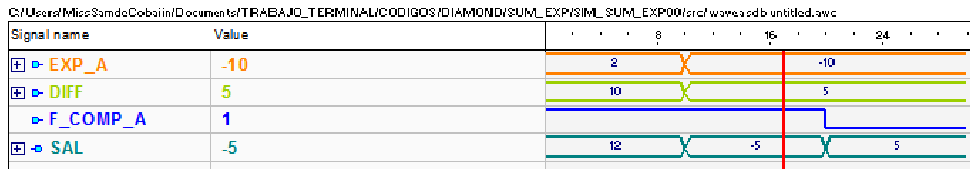


Fig 6.14 Simulación de módulo sumatoria del exponente

6.1.2.4 Corrimiento sobre la mantisa.

El módulo shift_mant cuenta con dos buses de entrada; DIF, el cual indica la diferencia como resultado de la resta entre los exponentes, es decir, el número de lugares a recorrer sobre la mantisa y MANTISA, bus indicador de la mantisa a operar en caso de ser necesario, el modulo en general se encarga de realizar el corrimiento hacia la derecha sobre la mantisa siempre y cuando la bandera de entrada F_COMP_X se encuentre activa, en caso contrario la mantisa no deberá

sufrir ningún cambio, de modo tal que, la mantisa, una vez operada o no, se muestra como resultado en el bus de salida S_MANT.

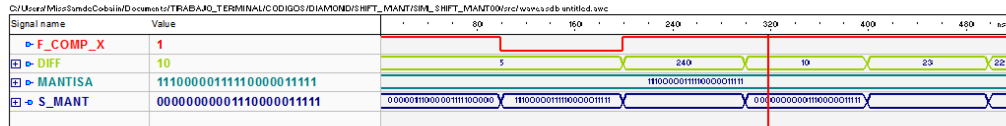


Fig 6.15 Simulación del módulo corrimiento de mantisa

6.1.2.5 Extensor de signo.

El modulo extensor de signo únicamente se encarga de aumentar el tamaño de bus del exponente a 23 bits para que pueda ser operado por el modulo sumador directamente sin sufrir ningún cambio, es decir, cuenta con un bus de entrada de 8 bits que corresponden al exponente a operar y yn bus de salida de 23 bits que contiene el mismo valor que el exponente de entrada, a diferencia del tamaño de buses. A continuación se muestran los resultados simulados con dos distintas representaciones. Cabe mencionar que dicho exponente maneja números signados.

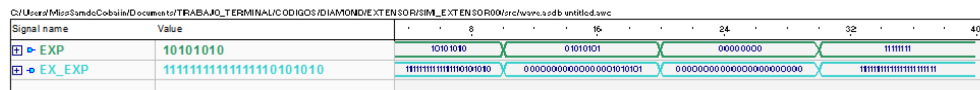


Fig 6.16 Simulación del módulo extensor de signo

6.1.2.6 Multiplexor entre mantisa y exponente.

El modulo en sí, es un multiplexor, cuenta con tres buses de entrada uno denominados EXTENSOR, el cual indica que se trata del bus proveniente del módulo Extensor de signo, otro denominado MANTISA proveniente del módulo que realiza o no el corrimiento sobre la mantisa y finalmente el bus selector del multiplexor SEL, dicho bus es el más importante dentro de este módulo, pues selecciona el resultado a mostrar en el bus de salida denominado SAL_MUX: si el valor del bus de selección es igual a 1 o a 2, indica que la operación a realizar por la UPF se trata de una suma o una resta respectivamente, en caso contrario si el valor del bus SEL es igual a 3 o a 0 indica que se trata de una multiplicación o de una división, por lo tanto si se trata de una suma o de una resta el valor a la salida será el del bus MANTISA, en cambio, si se trata de una multiplicación o de una división el valor de la salida será el bus EXTENSOR, tal y como se muestra en la siguiente simulación.

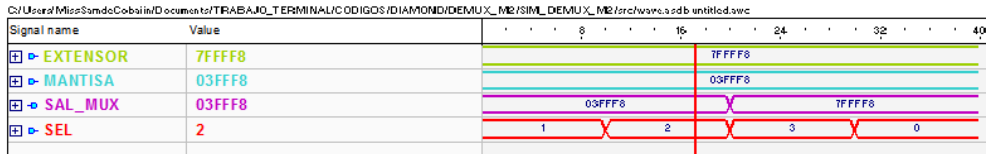


Fig 6.17 Simulación del multiplexor mantisa-exponente

6.1.3 Módulo COMPUTE

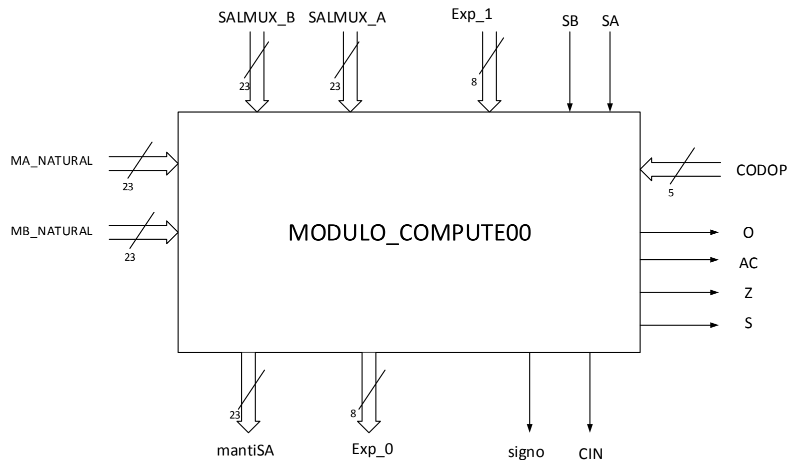


Fig 6.18 Diagrama del módulo COMPUTE

El módulo compute es el encargado de realizar las operaciones respectivas a cada número en punto flotante ya preparado previamente para su operatividad, cuenta con nueve módulos mencionados a continuación. Las pruebas realizadas de este módulo se tuvieron que dividir en cuatro fases: Suma, Resta, Multiplicación y División.

	A				B						R
	S	EXP	MANTISA		S	EXP	MANTISA		S	EXP	MANTISA
1	+	10	0.17402446269989	+	-	10	0.30941641330719	=	-	10	-0.1353919506073
	0	00001010	00101100100011001101111		1	00001010	01001111001101011110101		1	00001010	0.00100010101010010000110
2	-	16	0.901308417320251	+	-	16	0.704349994659424	=	-	16	1.605658411979675
	1	00010000	11100110101111000010011		1	00010000	10110100010100000100100		1	00010000	1.10011011000011000110110
3	-	12	0.221486330032349	+	+	12	0.704349994659424	=	+	12	0.482863664627075
	1	00001100	00111000101100110101010		0	00001100	10110100010100000100100		0	00001100	0.01111011100111001111001
4	+	6	0.0453994274139404	+	+	6	0.887499928474426	=	+	6	0.9328993558883664
	0	00000110	00001011100111110100110		0	00000110	11100011001100110011001		0	00000110	0. 11101110110100100111111

Tabla 12 Resultados de la SUMA

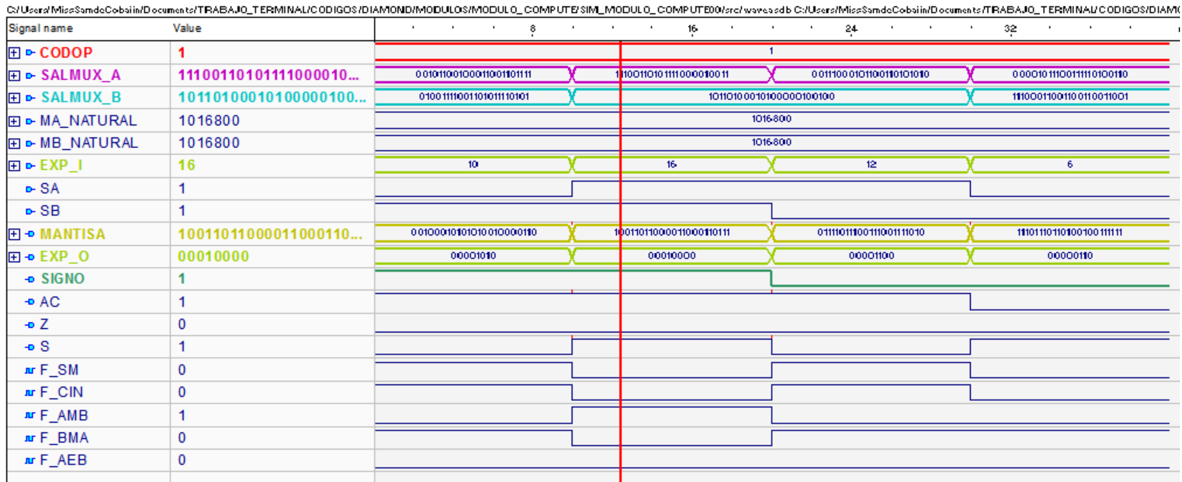


Fig 6.19 Primera simulación del sumador

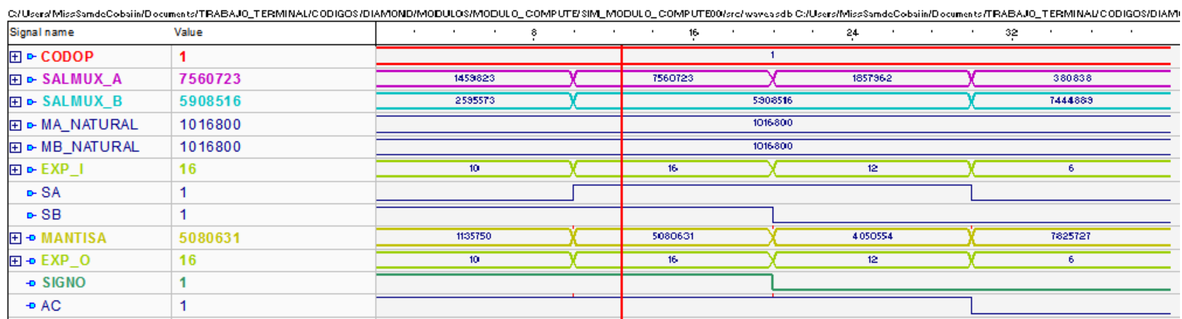


Fig 6.20 Segunda simulación del sumador

	A				B				R		
	S	EXP	MANTISA		S	EXP	MANTISA		S	EXP	MANTISA
1	+	10	0.17402446269989	-	-	10	0.30941641330719	=	+	10	0.48344087600708
	0	00001010	0010110010001100110111 1		1	00001010	0100111100110101111 0101		0	00001010	0.0111101111000010110 0100
2	-	16	0.901308417320251	-	-	16	0.704349994659424	=	-	16	0.196958422660827
	1	00010000	1110011010111100001 0011		1	00010000	1011010001010000010 0100		1	00010000	0.0011001001101011110 1110
3	-	12	0.221486330032349	-	+	12	0.704349994659424	=	-	12	0.925836324691773
	1	00001100	0011100010110011010 1010		0	00001100	1011010001010000010 0100		1	00001100	0.1110110100000011100 1110
4	+	6	0.0453994274139404	-	+	6	0.887499928474426	=	-	6	0.8421005010604856
	0	00000110	0000101110011111010 0110		0	00000110	1110001100110011001 1001		1	00000110	0.1101011110010011111 0010

Tabla 13 Resultados de la RESTA

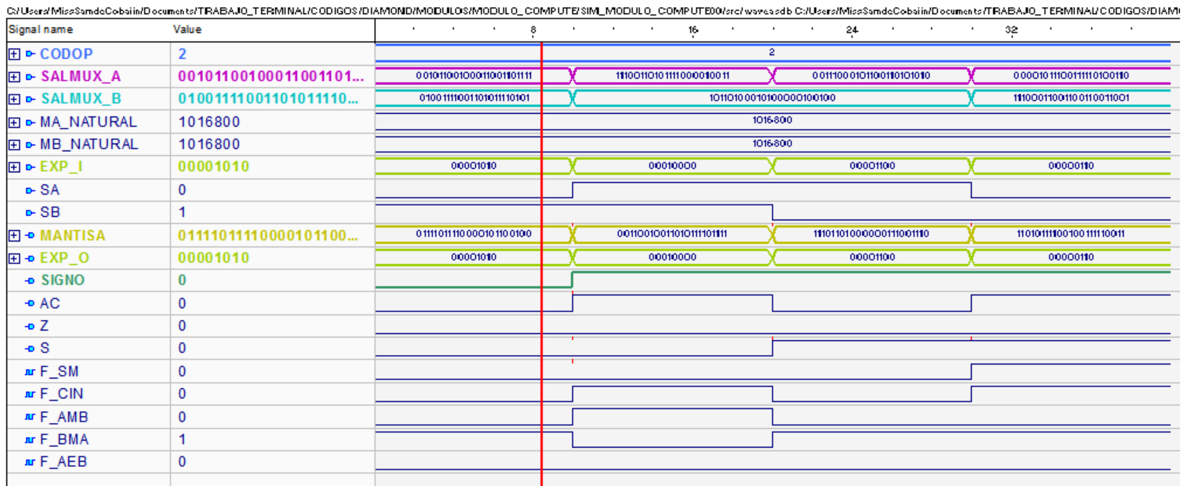


Fig 6.21 Primera simulación del Restador

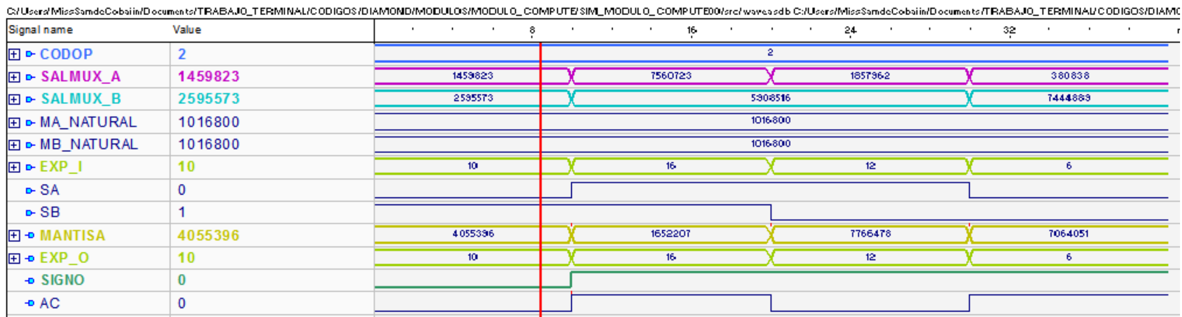


Fig 6.22 Segunda simulación del Restador

	A				B					R
	S	EXP	MANTISA		S	EXP	MANTISA		S	EXP	MANTISA
1	+	6	0.887499928474426	x	+	6	0.887499928474426	=	+	12	0.7876561230
	0	00000110	11100011001100110011001		0	00000110	11100011001100110011001		0	00001100	0.11001001101000111101010
2	-	16	0.901308417320251	x	-	-12	0.221486330032349	=	+	4	0.19962749358
	1	00010000	11100110101111000010011		1	11110100	00111000101100110101010		0	00000100	0.00110011000110101100100
3	+	8	0	x	-	12	0.704349994659424	=	-	20	0
	0	00001000	00000000000000000000000		1	00001100	10110100010100000100100		1	00010100	0.00000000000000000000000
4	-	-6	0.0453994274139404	x	+	1	0	=	-	-5	0
	1	00000110	00001011100111110100110		0	00000001	00000000000000000000000		1	11111011	0.00000000000000000000000

Fig 6.23 Resultados de la MULTIPLICACIÓN

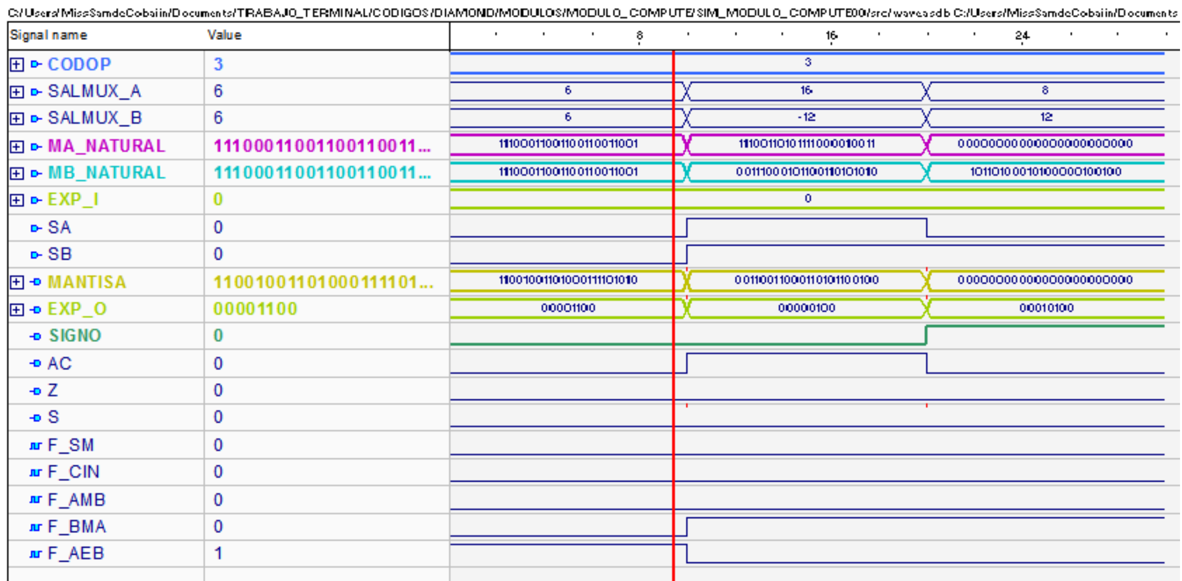


Fig 6.24 Primera simulación del Multiplicador

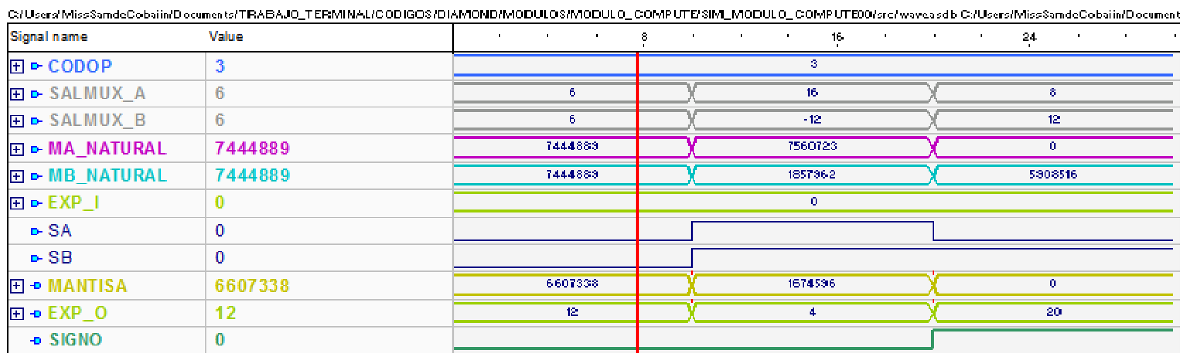


Fig 6.25 Segunda simulación del Multiplicador

	A				B				R		
	S	EXP	MANTISA		S	EXP	MANTISA		S	EXP	MANTISA
1	+	6	0.887499928474426	/	+	6	0.887499928474426	=	+	0	0
	0	00000110	11100011001100110011001		0	00000110	11100011001100110011001		0	00000000	00000000000000000000000
2	-	16	0.000000096046447753906	/	-	-12	0.901308417320251	=	+	4	
	1	00010000	00000000000000000000101		1	11110100	11100110101111000010011		0	00000100	
3	+	8	0	/	-	12	0.704349994659424	=	-	-4	0
	0	00001000	00000000000000000000000		1	00001100	10110100010100000100100		1	11111100	00000000000000000000000
4	-	-6	0.0453994274139404	/	+	1	0	=	-	-7	0
	1	11111010	00001011100111110100110		0	00000001	00000000000000000000000		1	11111001	11111111111111111111111

Tabla 14 Resultados de la DIVISIÓN

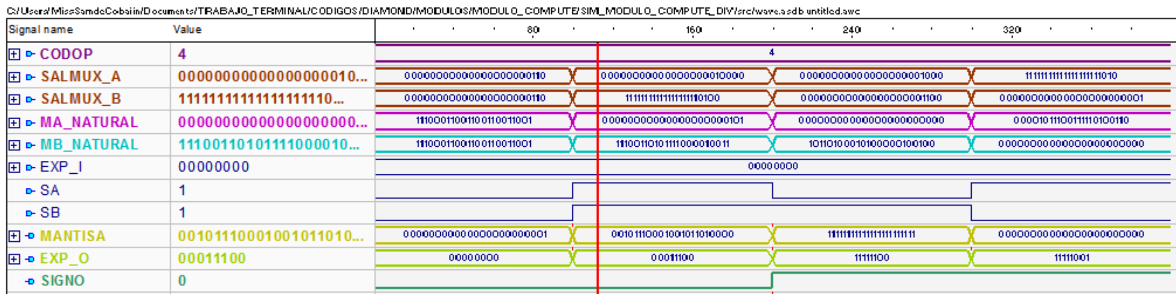


Fig 6.26 Primera simulación del Divisor

6.1.3.1 Multiplexor del módulo compute

Modulo encargado únicamente de llevar a cabo la permutación de las mantisas entrantes al módulo Sumador/Restador, dicho modulo únicamente cuenta con dos buses de entrada A y B los cuales indican las mantisas de cada operando respectivamente, asi mismo con un bit selector ya sea 0 para obtener a la salida SAL_MUXX el valor de A o 1 para obtener el valor de B.

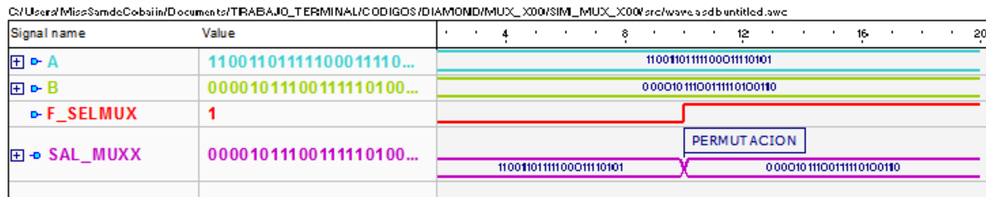


Fig 6.27 Simulación del multiplexor del módulo COMPUTE

6.1.3.2 Comparador entre mantisas

Este módulo se encarga de emitir tres banderas (AMB, BMA, AEB) De salida hacia el modulo selector indicando si las mantisas recibidas (MA,MB) son mayores, menores o iguales.

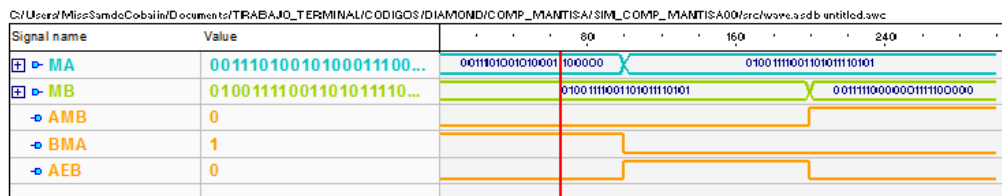


Fig 6.28 Simulación del módulo comparador de mantisas

6.1.3.3 Selector de operación y signo

El modulo selector se encarga de indicarle al sumador/restador, el tipo de operación a realizar, pues debemos tomar en cuenta los signos del número en punto flotante así como el código de operación manejado para poder obtener un resultado correcto. El modulo cuenta con tres banderas de entrada (AMB, BMA, AEB) que indican si el número a operar (mantisa) es mayor, menor o igual, del mismo modo tiene otras dos banderas de entrada (SA, SB) las cuales son los signos del número a y b respectivamente, así mismo cuenta con un bus de entrada el cual es el código de operación, pues se deben tener en cuenta todos estos valores para poder indicar una salida correcta. Para el caso de las salidas emitidas por este módulo, cabe mencionar que se tuvo que tomar en cuenta la permutación de valores recibidos a la entrada del sumador y la bandera de salida SELMUX es la encargada de deducir cuando se debe realizar esa permutación de acuerdo a ciertos criterios evaluados con anterioridad, es evidente que el signo resultante (SR) es la combinación de varias condiciones a la entrada para poder ser el adecuado, finalmente la bandera CIN nos indica el acarreo de entrada del sumador restador. A continuación se muestra la serie de condicionantes tomadas para emitir las banderas de salida adecuadas.

CODOP = 00001 - SUMA.

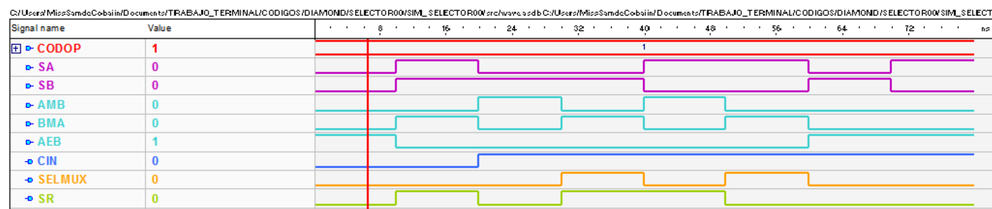


Fig 6.29 Simulación del módulo de operación y signo - SUMA

CODOP = 00010 - RESTA.

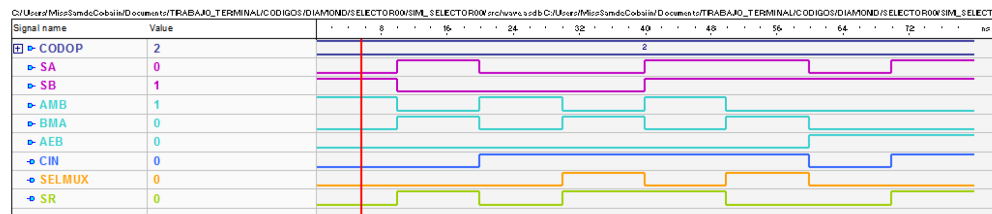


Fig 6.30 Simulación del módulo de operación y signo - RESTA

CODOP = 00011 & 00100 - MULTIPLICACION Y DIVISION

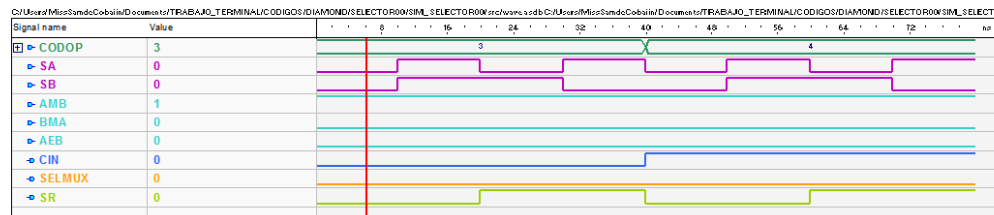


Fig 6.31 Simulación del módulo de operación y signo – MULTIPLICACIÓN Y DIVISIÓN

6.1.3.4 Sumador / Restador.

Modulo encargado de realizar la suma o la resta de las mantisas según sea el caso, o bien, la suma o resta de los exponentes que se toman en cuenta para la multiplicación o la división. Recibe dos buses de entrada (A, Bb) y un bit de acarreo (Cin) el cual indica si se realizara suma o resta de los buses recibidos, para finalmente emitir un bus de salida (SAL) el cual muestra el resultado obtenido de las operaciones. Dicho modulo muestra cuatro banderas de salida: S, el cual indica el signo del resultado (para el caso de la suma/resta de exponentes), Z en caso de que el resultado sea cero, AC cuando la suma genere un acarreo a la salida, cabe mencionar que dicha bandera es vital para la normalización del resultado general obtenido en el formato de punto flotante y O en caso de que el sumador/restador produzca un resultado erróneo (overflow). A continuación se muestran las simulaciones realizadas en caso de que se realice una suma o una resta:

- Simulaciones del sumador.

SA		SB		AC	S_GEN	Z	0	S
8388607	+	8388607	=		16777214			
11111111111111111111111	+	11111111111111111111111	=	1	11111111111111111111110	0	0	1

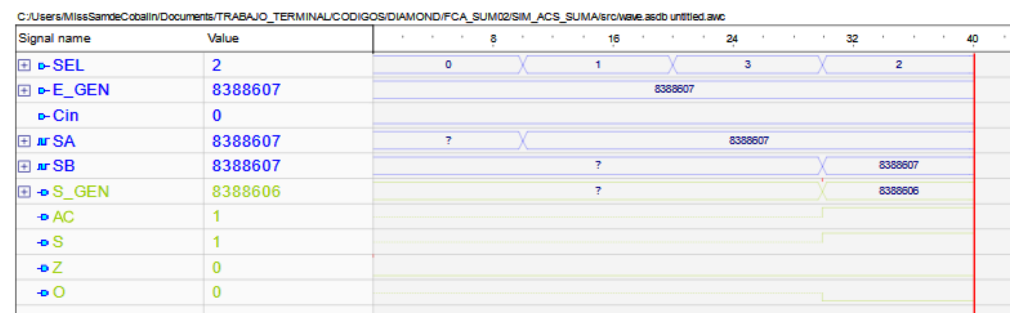


Fig 6.32 Primera simulación de sumador

SA		SB		AC	S_GEN	Z	0	S
0	+	0	=	0				
000000000000000000000000	+	000000000000000000000000	=	0	000000000000000000000000	1	0	0

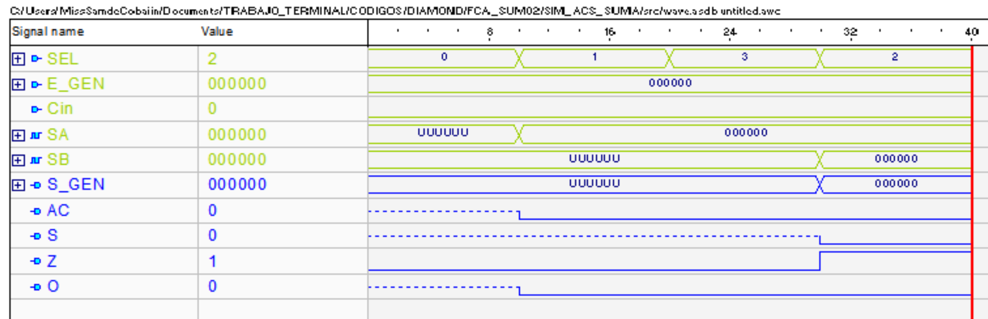


Fig 6.33 Segunda simulación de sumador

SA		SB		AC	S_GEN	Z	0	S
1222011	+	0	=	1222011				
001001010100101011111011	+	000000000000000000000000	=	0	001001010100101011111011	0	0	0

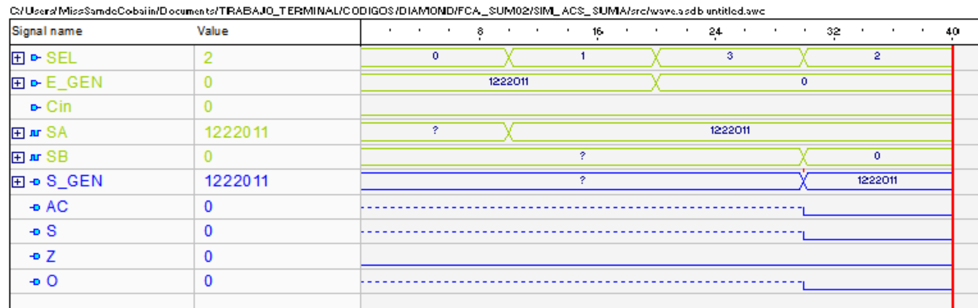


Fig 6.34 Tercera simulación de sumador

SA		SB		AC	S_GEN	Z	0	S
0	+	1222011	=	1222011				
000000000000000000000000	+	001001010100101011111011	=	0	001001010100101011111011	0	0	0

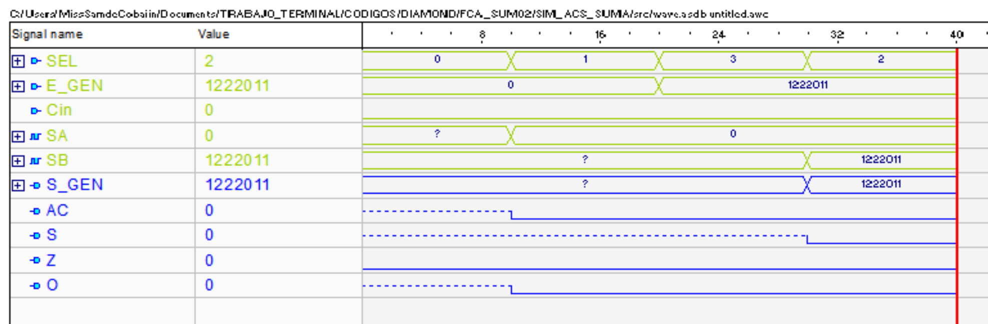


Fig 6.35 Cuarta simulación de sumador

SA		SB		AC	S_GEN	Z	0	S
1222011	+	1222011	=	2444022				
00100101010010101111011	+	00100101010010101111011	=	0	01001010100101011110110	0	0	0

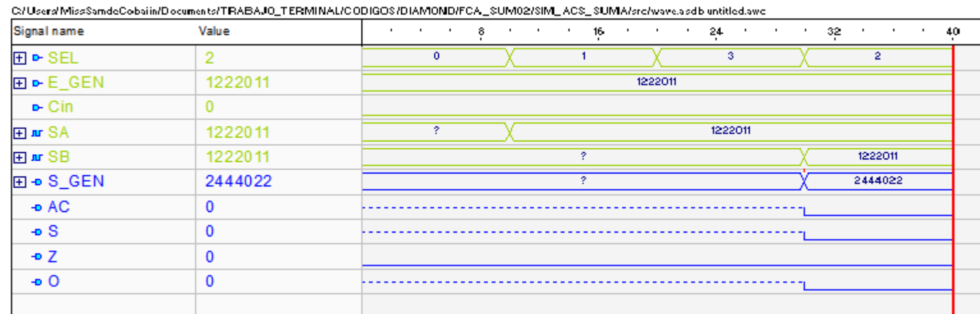


Fig 6.36 Quinta simulación de sumador

SA		SB		AC	S_GEN	Z	0	S
8388607	+	1	=	0				
11111111111111111111111	+	00000000000000000000001	=	1	00000000000000000000000	0	0	1

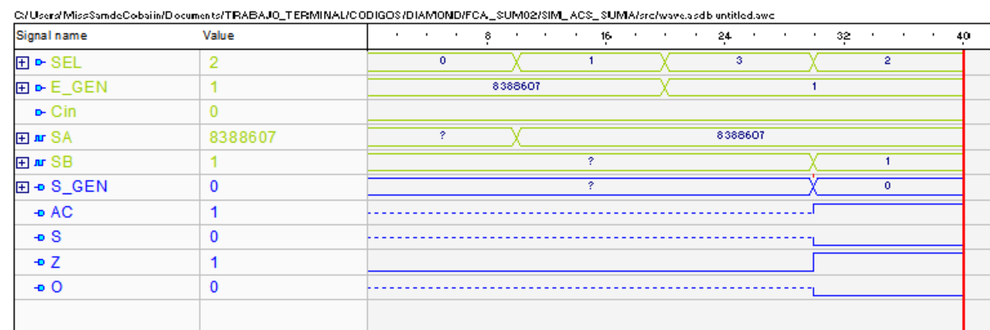


Fig 6.37 Sexta simulación de sumador

Nota: cuando el bit de acarreo (AC=1) se encuentra en alto, se debe tomar en cuenta el resultado obtenido para proceder con el algoritmo de normalización.

- Simulaciones del restador

SA		SB		AC	S_GEN	Z	0	S
8388607	-	8388607	=	0				
11111111111111111111111	-	11111111111111111111111	=	1	00000000000000000000000	1	0	0

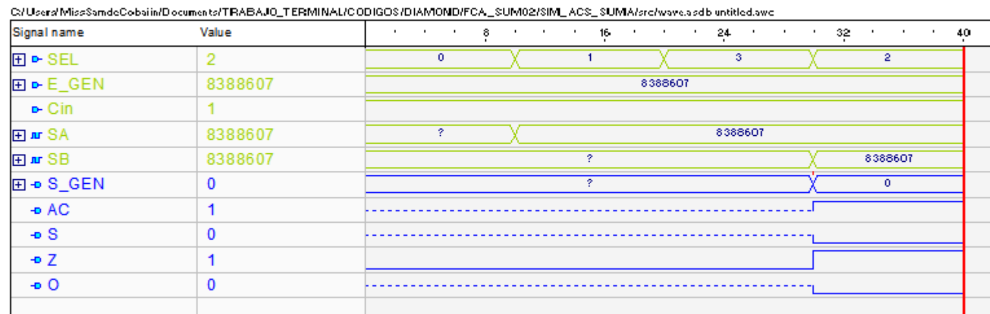


Fig 6.38 Primera simulación de restador

SA		SB		AC	S_GEN	Z	0	S
0	-	0	=	0				
000000000000000000000000	-	000000000000000000000000	=	1	000000000000000000000000	1	0	0

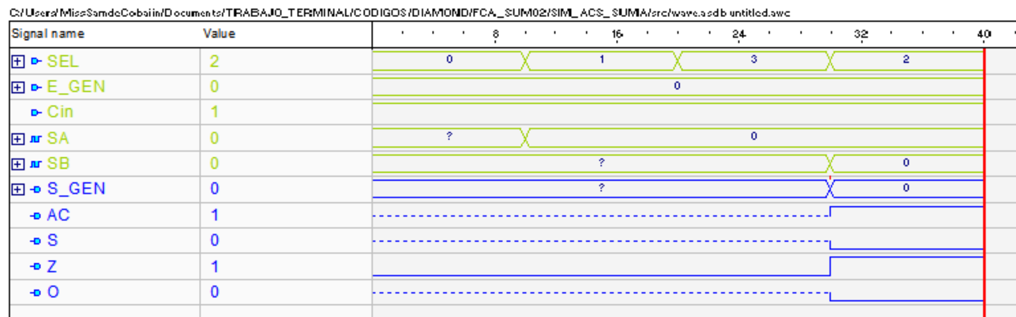


Fig 6.39 Segunda simulación de restador

SA		SB		AC	S_GEN	Z	0	S
1222011	-	0	=	1222011				
00100101010010101111011	-	000000000000000000000000	=	1	00100101010010101111011	0	0	0

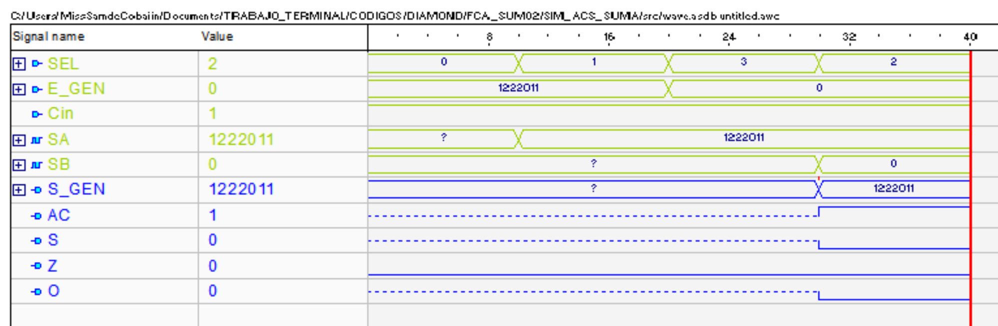


Fig 6.40 Tercera simulación de restador

SA		SB		AC	S_GEN	Z	0	S
0	-	1222011	=	7166597				
000000000000000000000000	-	00100101010010101111011	=	0	11011010101101010000101	0	0	1

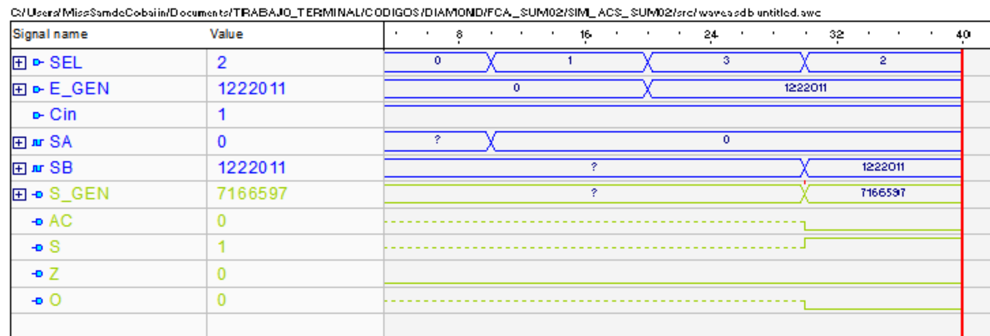


Fig 6.41 Cuarta simulación de restador

SA		SB		AC	S_GEN	Z	0	S
1222011	-	2912992	=	6697627				
0010010101001010111011	-	0101100011100101110000	=	0	11001100011001010011011	0	0	1

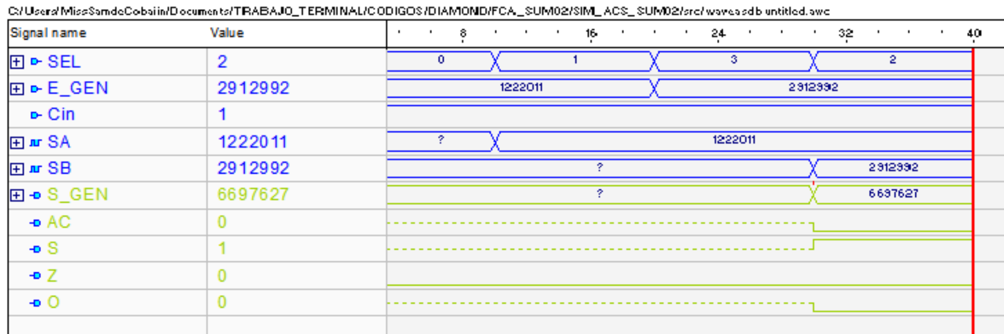


Fig 6.42 Quinta simulación de restador

SA		SB		AC	S_GEN	Z	0	S
2912992	-	1222011	=	1690981				
0101100011100101110000	-	0010010101001010111011	=	1	0011001110011010110010 1	0	0	0

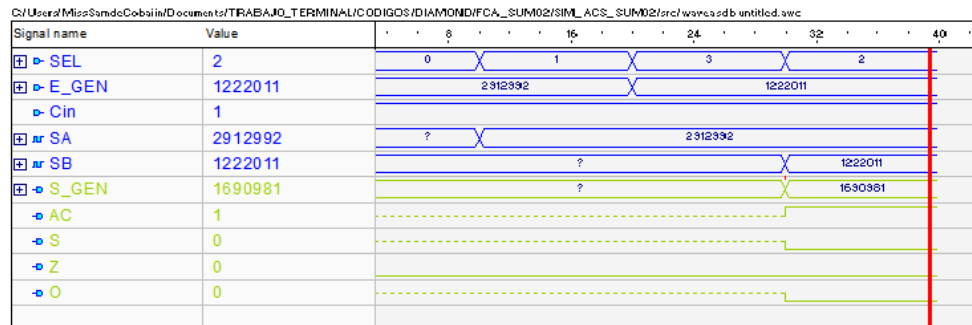


Fig 6.43 Sexta simulación de restador

Cabe mencionar que a partir de una serie de comparaciones entre sumadores/restadores se decidió por el sumador de acarreo anticipado. A continuación se muestra el análisis llevado a cabo con anterioridad:

Existen diversos diseños para la implementación de un sumador/restador, entre ellos se encuentran el sumador/restador por el método de Ripple, así como el sumador restador por el método de acarreo anticipado (FCA). A continuación se muestra la comparación entre estos dos diseños, se analizan los retardos de propagación generados, así como la densidad ocupada para ambos diseños.

6.1.3.4.1 Ripple

La siguiente figura muestra el diagrama a bloques del diseño de un sumador/restador de ocho bits mediante el método de ripple, para poder llevar un análisis en menor escala que pueda permitir la fácil comparación entre algoritmos.

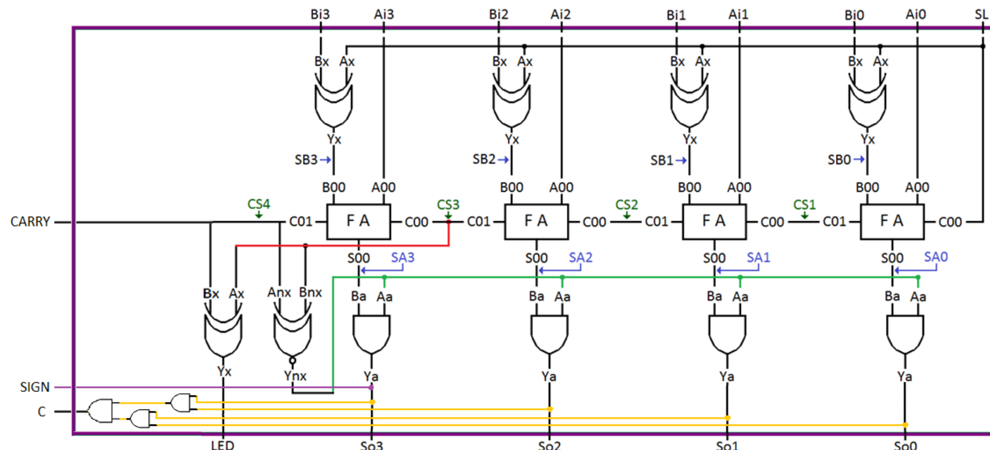


Fig 6.44 Diagrama de bloques del sumador/restador de 4 bits por el método Ripple.

Los siguientes datos mostrados, presentan el tiempo de retardo generado desde una locación a otra, siendo el retardo mayor de 39.75 desde el pin 100 hasta el pin 30, es decir desde SL, bit de signo, hasta C, bit para la bandera indicadora de cero. Este análisis pudo ser visualizado mediante el reporte de análisis de tiempos arrojado por la herramienta ispLever Classic.

```
Timing Report

//  Project = adder4bit00
//  Family  = lc4k
//  Device  = LC4256ZE
//  Speed   = -5.8
//  Voltage = 1.8
//  Operating Condition = COM
//  Data sheet version  = 0.9

Section tPD

   Delay         Level   Location(From => To)    Source                          Destination
  =====         =====   ====================    ======                          ==========
  39.75            9    p100      => p30        SL                              C
  39.75            9    p100      => p12        SL                              S[1]
  39.70            9    p100      => p31        SL                              SIGN
  39.40            9    p110      => p30        Ai[0]                           C
  39.40            9    p110      => p12        Ai[0]                           S[1]
  39.40            9    p104      => p30        Bi[0]                           C
  39.40            9    p104      => p12        Bi[0]                           S[1]
  39.35            9    p110      => p31        Ai[0]                           SIGN
  39.35            9    p104      => p31        Bi[0]                           SIGN
  39.30            9    p117      => p30        Ai[1]                           C
  39.30            9    p117      => p12        Ai[1]                           S[1]
  39.30            9    p105      => p30        Bi[1]                           C
  39.30            9    p105      => p12        Bi[1]                           S[1]
  39.25            9    p117      => p31        Ai[1]                           SIGN
  39.25            9    p105      => p31        Bi[1]                           SIGN
  35.60            8    p100      => p13        SL                              S[0]
  35.60            8    p100      => p11        SL                              S[2]
  35.60            8    p100      => p9         SL                              S[3]
  35.60            8    p100      => p8         SL                              S[4]
  35.60            8    p100      => p7         SL                              S[5]
```

Fig 6.45 Reporte de tiempo estático, utilizando herramientas de análisis del entorno ispLever Classic.

La figura posterior nos indica la densidad ocupada e implementada para el dispositivo ispMach4256ZE. Para este diseño en específico se ocuparon únicamente 22 macroceldas de 256 disponibles, siendo un 8% del total.

```
Device_Resource_Summary


                                 Device
                                 Total    Used   Not Used   Utilization
-----------------------------------------------------------------------
Dedicated Pins
  Clock/Input Pins                  4        0        4   -->     0
  Input-Only Pins                  10        2        8   -->    20
  I/O / Enable Pins                 2        1        1   -->    50
I/O Pins                           94       26       68   -->    27
Logic Functions                   256       22      234   -->     8
  Input Registers                  96        0       96   -->     0

GLB Inputs                        576       76      500   -->    13
Logical Product Terms            1280       90     1190   -->     7
Occupied GLBs                      16       12        4   -->    75
Macrocells                        256       22      234   -->     8
```

Fig 6.46 Resumen de recursos del dispositivo

El diagrama de bloques presentado fue obtenido a partir de la sintetización del sumador/restador en la herramienta ISE Project Navigator,

y a partir de la opción permitida para la visualización del diagrama RTL en esquemático.

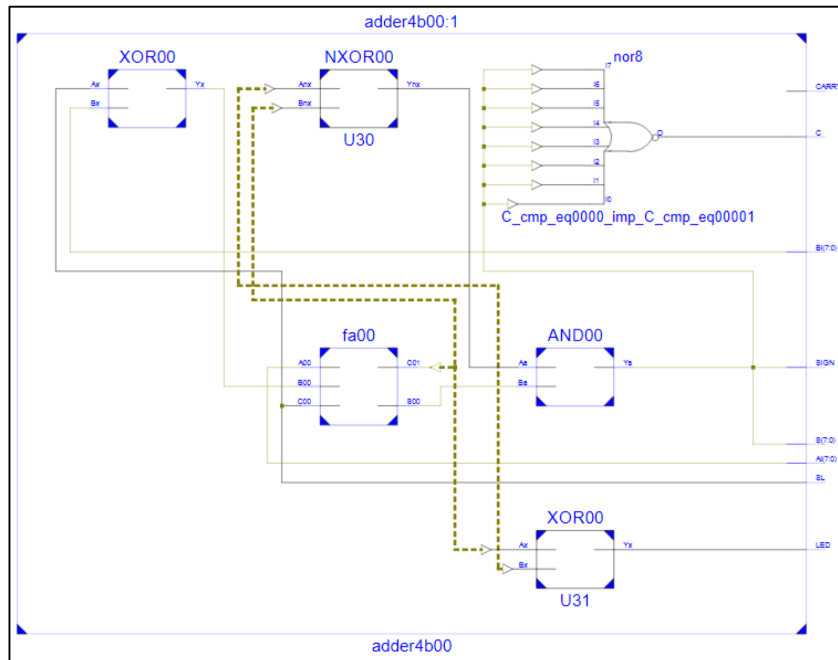


Fig 6.47 Diagrama RTL schematic del sumador/restador de 8 bits por método de Ripple.

El resumen obtenido a partir del análisis llevado a cabo por la herramienta ISE Project Navigator, es el que se muestra en la tabla posterior, la cual nos indica el número de LUTs utilizadas, así como los Slices que utiliza el dispositivo para la implementación del sumador/restador esquemático por el método de Ripple.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	26	1,920	1%	
Number of occupied Slices	13	960	1%	
Number of Slices containing only related logic	13	13	100%	
Number of Slices containing unrelated logic	0	13	0%	
Total Number of 4 input LUTs	26	1,920	1%	
Number of bonded IOBs	29	66	43%	
Average Fanout of Non-Clock Nets	2.63			

Fig 6.48 Resumen de la densidad ocupada en el dispositivo.

La herramienta de ISE Project Navigator nos permite observar los tiempos estáticos calculados según el dispositivo implementado, para este caso los tiempos estáticos son mostrados para el dispositivo XC3S100E Spartan 3E.

	Source Pad	Destination Pad	Delay
1	SL	C	20.321
2	Bi<0>	C	19.726
3	Ai<0>	C	19.627
4	SL	S<4>	19.464
5	Bi<1>	C	19.036
6	Ai<2>	C	18.987
7	Bi<0>	S<4>	18.869
8	Ai<0>	S<4>	18.770
9	Ai<1>	C	18.477
10	Bi<1>	S<4>	18.179

Fig 6.49 Retardos máximos del sumador/restador por método de Ripple.

6.1.3.4.2 Fast Carry-Ahead

La siguiente figura muestra el diagrama a bloques del sumador/restador por el método de acarreo anticipado, o mejor conocido como Fast Carry-Ahead. Del mismo modo que el sumador/restador anteriormente analizado, este sumador es de ocho bits para realizar un mejor análisis comparativo.

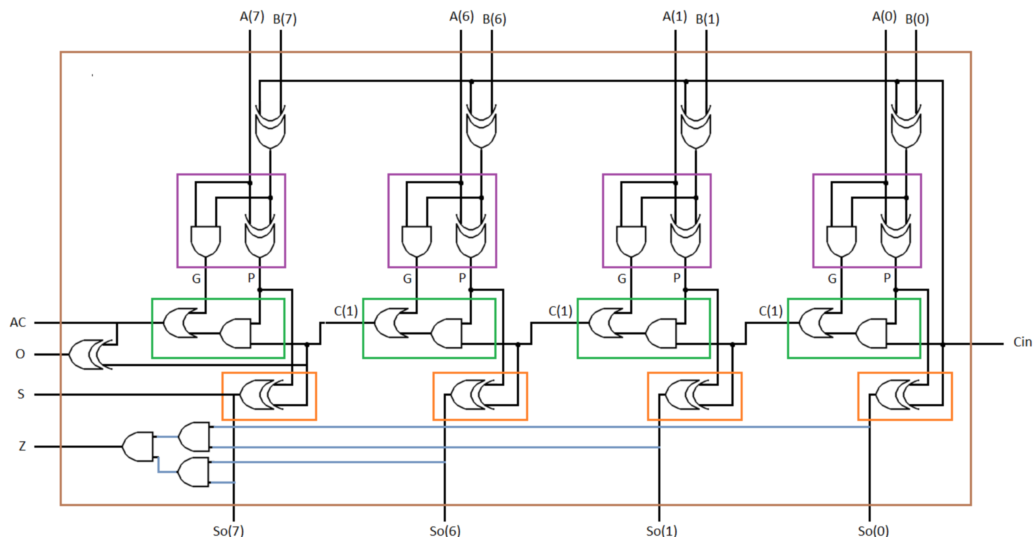


Fig 6.50 Diagrama a bloques del sumador restador de 8 bits por el método de acarreo anticipado.

Los siguientes datos mostrados, presentan el tiempo de retardo generado desde una locación a otra, siendo el retardo mayor de 39.60 desde el pin 100 hasta el pin 28, es decir desde Cin, acarreo de entrada, hasta S, bit para la

bandera indicadora de signo. Este análisis pudo ser visualizado mediante el reporte de análisis de tiempos arrojado por la herramienta ispLever Classic.

```
Timing Report

//  Project = untitled
//  Family  = lc4k
//  Device  = LC4256ZE
//  Speed   = -5.8
//  Voltage = 1.8
//  Operating Condition = COM
//  Data sheet version  = 0.9

Section tPD

   Delay         Level   Location(From => To)    Source                  Destination
  =====          =====   ====================    ======                  ===========
  39.60            9     p100      => p28        Cin                     S
  39.60            9     p100      => p29        Cin                     Z
  39.20            9     p117      => p28        A[1]                    S
  39.20            9     p117      => p29        A[1]                    Z
  39.20            9     p105      => p28        Bb[1]                   S
  39.20            9     p105      => p29        Bb[1]                   Z
  35.50            8     p100      => p30        Cin                     AC
  35.50            8     p100      => p31        Cin                     O
  35.50            8     p100      => p5         Cin                     So[7]
  35.10            8     p110      => p28        A[0]                    S
  35.10            8     p110      => p29        A[0]                    Z
  35.10            8     p117      => p30        A[1]                    AC
  35.10            8     p117      => p31        A[1]                    O
  35.10            8     p117      => p5         A[1]                    So[7]
  35.10            8     p104      => p28        Bb[0]                   S
  35.10            8     p104      => p29        Bb[0]                   Z
  35.10            8     p105      => p30        Bb[1]                   AC
  35.10            8     p105      => p31        Bb[1]                   O
  35.10            8     p105      => p5         Bb[1]                   So[7]
  35.00            8     p120      => p28        A[2]                    S
```

Fig 6.51 Reporte de tiempo estático, utilizando herramientas de análisis del entorno ispLever Classic.

Los datos mostrados en la figura siguiente, son obtenidos del reporte ajustado arrojado por la herramienta ispLever Classic, el cual nos muestra la cantidad de macroceldas ocupadas por el diseño en el dispositivo ispMACH4256ZE, siendo un total de 15 macroceldas de 256 propias del CPLD, es decir, solamente el 5% del total.

```
Device_Resource_Summary


                                 Device
                                 Total    Used   Not Used   Utilization
-----------------------------------------------------------------------
Dedicated Pins
  Clock/Input Pins                  4       4       0     -->    100
  Input-Only Pins                  10      10       0     -->    100
  I/O / Enable Pins                 2       2       0     -->    100
I/O Pins                           94      10      84     -->     10
Logic Functions                   256      15     241     -->      5
  Input Registers                  96       0      96     -->      0

GLB Inputs                        576      36     540     -->      6
Logical Product Terms            1280      75    1205     -->      5
Occupied GLBs                      16      10       6     -->     62
Macrocells                        256      15     241     -->      5
```

Fig 6.52 Resumen de recursos del dispositivo

El diagrama de bloques presentado fue obtenido a partir de la sintetización del sumador/restador en la herramienta ISE Project Navigator, y a partir de la opción permitida para la visualización del diagrama RTL en

esquemático. Donde se puede apreciar, que el diseño generado previamente en esquemático, es similar, al mostrado por la herramienta de diseño.

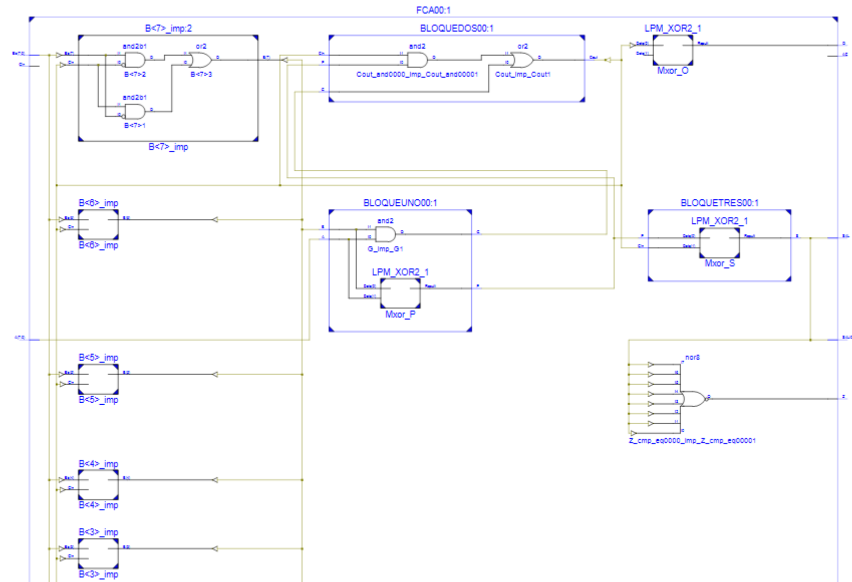


Fig 6.53 Diagrama RTL schematic del sumador/restador de 8 bits por método de acarreo anticipado.

El resumen obtenido a partir del análisis llevado a cabo por la herramienta ISE Project Navigator, es mostrado en la figura siguiente, indica el número de LUTs utilizadas, así como los Slices que utiliza el dispositivo para la implementación del sumador/restador con acarreo anticipado en esquemático, cabe señalar, que el número de LUTs y el número de Slices utilizados, es menor en comparación con el sumador/restador por el método de Ripple.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	20	1,920	1%	
Number of occupied Slices	10	960	1%	
Number of Slices containing only related logic	10	10	100%	
Number of Slices containing unrelated logic	0	10	0%	
Total Number of 4 input LUTs	20	1,920	1%	
Number of bonded IOBs	29	66	43%	
Average Fanout of Non-Clock Nets	2.38			

Fig 6.54 Resumen de la densidad ocupada en el dispositivo.

La herramienta de ISE Project Navigator nos permite observar los tiempos estáticos calculados según el dispositivo implementado, para este caso los tiempos estáticos son mostrados para el dispositivo XC3S100E Spartan 3E, evidentemente, los tiempos de retardo, con menores en

comparación con los tiempos estáticos arrojados por la misma herramienta con el diseño del sumador/restador por el método de Ripple.

	Source Pad	Destination Pad	Delay
1	Cin	Z	16.131
2	Bb<0>	Z	15.756
3	Cin	O	15.637
4	Cin	AC	15.618
5	A<0>	Z	15.460
6	Cin	SAL	15.439
7	Bb<0>	O	15.262
8	Bb<0>	AC	15.243
9	Cin	SALIDA<7>	15.187
10	Bb<1>	Z	15.132

Fig 6.55 Retardos máximos del sumador/restador con acarreo anticipado.

6.1.3.5 Multiplicador.

El modulo multiplicador recibe las mantisas a la entrada (A y B) de 23 bits cada bus y únicamente arroja un resultado de 46 bits en el bus de salida denominado M, se debe tomar en cuenta únicamente los 23 bits más significativos, pues es el resultado más aproximado al resultado esperado de la operación debido al redondeo por precisión. A continuación se muestra un serie de ejemplo simulados para este módulo. Para poder elaborar este módulo se debió tomar en cuenta el sumador anterior y a partir de ahí hacer las respectivas modificaciones en los siguientes módulos para obtener el multiplicador aquí señalado.

SA		SB		S_GEN
8,388,607	x	8,388,607	=	70,368,727,400,449
11111111111111111111111	x	11111111111111111111111	=	1111111111111111111111000000000000000000000001

C:/Users/MissSamdeCoballn/Documents/TRABAJO_TERMINAL/CODIGOS/DIAMOND/MULT01/SIM_ACS_MULT/src/wave.asdb untitled.awc

Signal name	Value
E_GEN	8388607
SEL	10
S_GEN	70368727400449
SA	8388607
SB	8388607

Fig 6.56 Primera simulación del multiplicador

SA		SB		S_GEN
8,388,607	x	500	=	4194303500
11111111111111111111111	x	00000000000000111110100	=	0000000000000011111001111111111111111000001100

C:/Users/MissSamdeCoballn/Documents/TRABAJO_TERMINAL/CODIGOS/DIAMOND/MULT01/SIM_ACS_MULT/src/wave.asdb untitled.awc

Signal name	Value
E_GEN	8388607
SEL	01
S_GEN	4194303500
SA	8388607
SB	500

Fig 6.57 Segunda simulación del multiplicador

SA		SB		S_GEN
8,388,607	x	0	=	0
11111111111111111111111	x	00000000000000000000000	=	0000000000000000000000000000000000000000000000

C:/Users/MissSamdeCoballn/Documents/TRABAJO_TERMINAL/CODIGOS/DIAMOND/MULT01/SIM_ACS_MULT/src/wave.asdb untitled.awc

Signal name	Value
E_GEN	0
SEL	10
S_GEN	0
SA	8388607
SB	0

Fig 6.58 Tercera simulación del multiplicador

SA		SB		S_GEN
0	x	8,388,607	=	0
00000000000000000000000	x	11111111111111111111111	=	0000000000000000000000000000000000000000000000

C:/Users/MissSamdeCoballn/Documents/TRABAJO_TERMINAL/CODIGOS/DIAMOND/MULT01/SIM_ACS_MULT/src/wave.asdb untitled.awc

Signal name	Value
E_GEN	8388607
SEL	01
S_GEN	0
SA	8388607
SB	0

Fig 6.59 Cuarta simulación del multiplicador

A continuación se muestra el diseño implementado para el respectivo multiplicador de la unidad de Punto Flotante:

6.1.3.5.1 Multiplicador (Unsigned Array Multiplier)

El siguiente diseño de multiplicador implementado, se conoce como Unsigned Array Multiplier, se caracteriza de ser un circuito combinacional, a diferencia de los circuitos secuenciales, la cuestión aquí es fundamental, debido a que un multiplicador secuencial, tal es el caso de la implementación del algoritmo de Booth en hardware, donde se utiliza una operación por cada flanco de subida del reloj, es decir, por cada suma y corrimiento realizado, para llevar a cabo la multiplicación, lo cual indica que consume demasiados ciclos de reloj para mostrar el resultado, en cambio, un multiplicador secuencial, como el Unsigned Array Multiplier, realiza la operación de multiplicación al momento, con pequeños retardos, traducidos en nanosegundos.

Debido a esta ventaja se optó por implementar el multiplicador combinacional con uno de los sumadores/restadores previamente analizados, siendo el sumador/restador Fast Carry-Ahead el seleccionado para la implementación de este multiplicador, como se muestra en el siguiente diagrama de bloques.

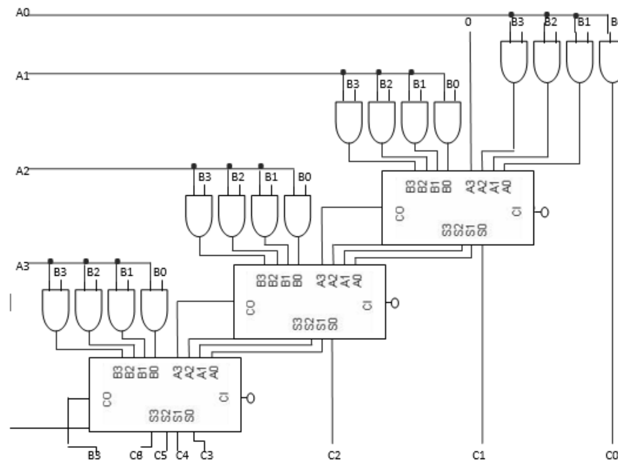


Fig 6.60 Diagrama a bloques Unsigned Array Multiplier

El siguiente diagrama RTL sintetizado por la herramienta de diseño de Xilinx, muestra el diseño similar al diagrama de bloques mostrado con anterioridad. Cuenta con cuatro sumadores/restadores por acarreo anticipado, así como sus respectivas compuertas AND.

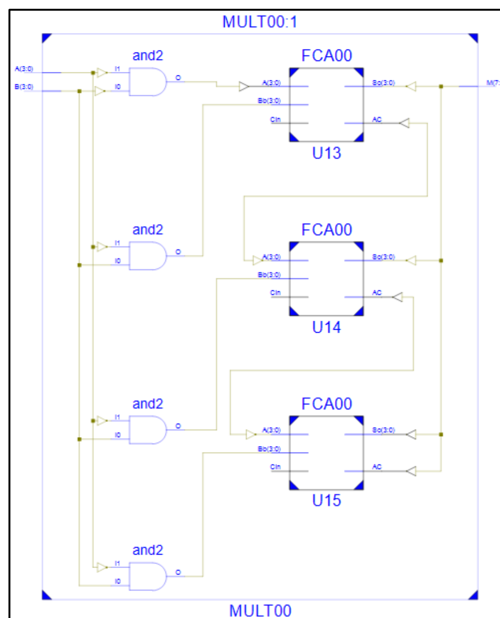


Fig 6.61 Diagrama RTL del multiplicador de 4 bits UAM

La figura posterior muestra el tiempo de retardo generado desde una locación a otra en el diseño del multiplicador, siendo el retardo mayor de 31.05 desde el pin 114 hasta el pin 6, es decir desde B[1], el bit uno del bus B de entrada, hasta M[6], bit seis del resultado de la multiplicación. Este análisis pudo ser visualizado mediante el reporte de análisis de tiempos arrojado por la herramienta ispLever Classic.

```
Timing Report

//  Project = untitled
//  Family  = lc4k
//  Device  = LC4256ZE
//  Speed   = -5.8
//  Voltage = 1.8
//  Operating Condition = COM
//  Data sheet version  = 0.9

Section tPD

   Delay        Level  Location(From => To)   Source     Destination
  =====         =====  ====================   ======     ===========
  31.05           7    p114      => p6        B[1]          M[6]
  31.05           7    p114      => p5        B[1]          M[7]
  31.05           7    p115      => p6        B[2]          M[6]
  31.05           7    p115      => p5        B[2]          M[7]
  31.00           7    p123      => p6        A[1]          M[6]
  31.00           7    p123      => p5        A[1]          M[7]
  30.95           7    p122      => p6        A[0]          M[6]
  30.95           7    p122      => p5        A[0]          M[7]
  30.90           7    p116      => p6        B[3]          M[6]
  30.90           7    p116      => p5        B[3]          M[7]
  26.95           6    p114      => p7        B[1]          M[5]
  26.95           6    p115      => p7        B[2]          M[5]
  26.90           6    p123      => p7        A[1]          M[5]
  26.85           6    p122      => p7        A[0]          M[5]
```

Fig 6.62 Reporte de tiempo estático del multiplicador, utilizando herramientas de análisis del entorno ispLever Classic.

Los datos obtenidos del reporte ajustado arrojado por la herramienta ispLever Classic, se muestran en la siguiente figura y representa la cantidad de macroceldas ocupadas por el diseño del multiplicador en el dispositivo ispMACH4256ZE, siendo un total de 20 macroceldas de 256 propias del CPLD, es decir, solamente el 7% del total.

```
Device_Resource_Summary


                                 Device
                                 Total    Used   Not Used   Utilization
-----------------------------------------------------------------------
Dedicated Pins
  Clock/Input Pins                  4        0        4   -->    0
  Input-Only Pins                  10        0       10   -->    0
  I/O / Enable Pins                 2        1        1   -->   50
I/O Pins                           94       15       79   -->   15
Logic Functions                   256       20      236   -->    7
  Input Registers                  96        0       96   -->    0

GLB Inputs                        576       71      505   -->   12
Logical Product Terms            1280       87     1193   -->    6
Occupied GLBs                      16       12        4   -->   75
Macrocells                        256       20      236   -->    7
```

Fig 6.63 Resumen de Recursos de Dispositivo

El análisis que realiza la herramienta ISE Project Navigator para la visualización de la utilización general del dispositivo para el diseño del multiplicador es el siguiente, donde se indica que es muy poco el recurso utilizado para la implementación del diseño sintetizado.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	30	1,920	1%	
Number of occupied Slices	16	960	1%	
Number of Slices containing only related logic	16	16	100%	
Number of Slices containing unrelated logic	0	16	0%	
Total Number of 4 input LUTs	30	1,920	1%	
Number of bonded IOBs	16	66	24%	
Average Fanout of Non-Clock Nets	2.89			

Fig 6.64 Resumen de la densidad ocupada en el dispositivo para el diseño del multiplicador.

El resumen estático de tiempo que muestra la herramienta de diseño de Xilinx, nos indica los retardos máximos generados desde un bit fuente a un bit de destino, la siguiente figura nos proporciona la información requerida al respecto, donde los retardos vienen dados en nanosegundos.

	Source Pad	Destination Pad	Delay
1	B<2>	M<6>	15.084
2	B<2>	M<7>	14.929
3	A<2>	M<6>	14.501
4	A<0>	M<6>	14.457
5	A<2>	M<7>	14.346
6	A<0>	M<7>	14.302
7	B<1>	M<6>	14.294
8	B<1>	M<7>	14.139
9	B<0>	M<6>	14.135
10	A<1>	M<6>	14.098

Fig 6.65 Retardos máximos del Unsigned Array Multiplier.

6.1.3.6 Divisor.

El modulo divisor se encarga de dividir los valores de la mantisa, dicho modulo recibe dos buses de entrada (DIVISOR y DIVIDENDO) para poder operarlos y obtener un solo bus de salida como resultado (RES). A continuación se muestra una pequeña simulación de la operatividad de dicho modulo.

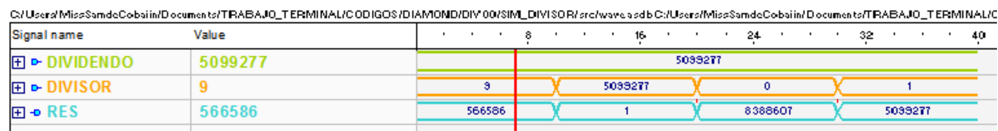


Fig 6.66 Simulación del DIVISOR

6.1.3.7 Selector de exponente.

El modulo selecciona exponente es un multiplexor de dos bits de selección, pues dependiendo de la operación que se deba realiza, ya sea suma y resta o multiplicación y división es el exponente que el multiplexor dejara pasar, tal y como se muestra en la continua simulación:

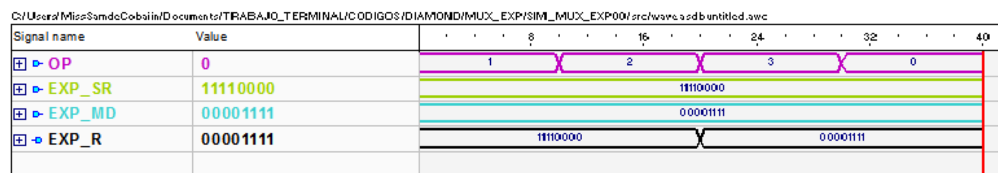


Fig 6.67 Simulación del módulo selector de exponentes

6.1.3.8 Selector del resultado de la operación.

El modulo seleccionador del resultado de la operación, se encarga de seleccionar que resultado de la operación correspondiente dejara pasar, es decir, e caso de ser suma (01) o resta (10) será el bus proveniente del módulo sumador/restador, en caso de ser multiplicación (11) la salida del multiplexor será la entrada proveniente del módulo multiplicador, del mismo modo si se trata de una división (00) el resultado de este módulo será la entrada proveniente del módulo divisor. Dicha operatividad se puede ver reflejada en la siguiente simulación.

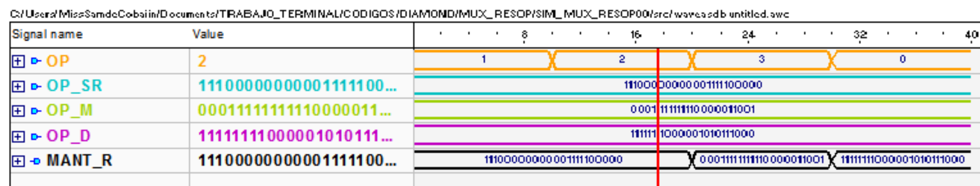


Fig 6.68 Simulación del modulo selector del resultado de la operación

6.1.4 Módulo Set

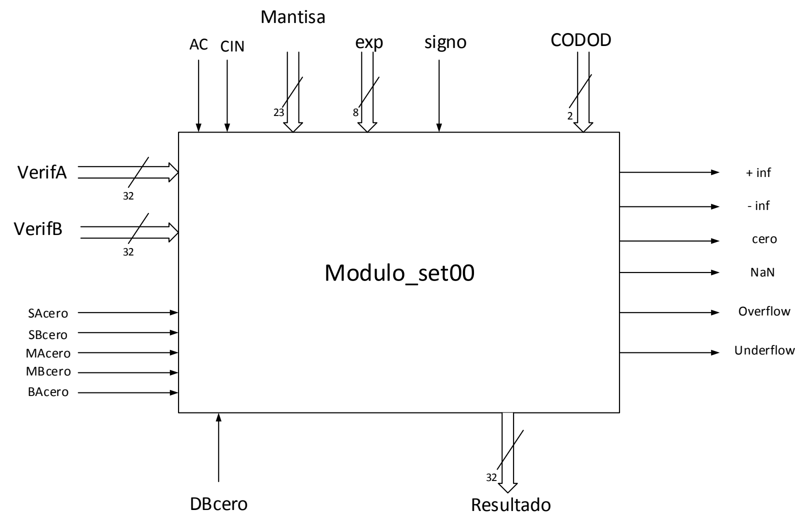


Fig 6.69 Diagrama del mósulo SET

El módulo set es el encargado de arrojar un resultado correcto después de haber sido operado los dos números en punto flotante, para ello lleva a cabo la sumatoria del sesgo para posteriormente normalizar, dentro de este módulo se encuentra un componente importante denominado como verificador, pues dicho módulo arroja

las banderas resultantes del resultado obtenido, tal y como se aprecia a continuación:

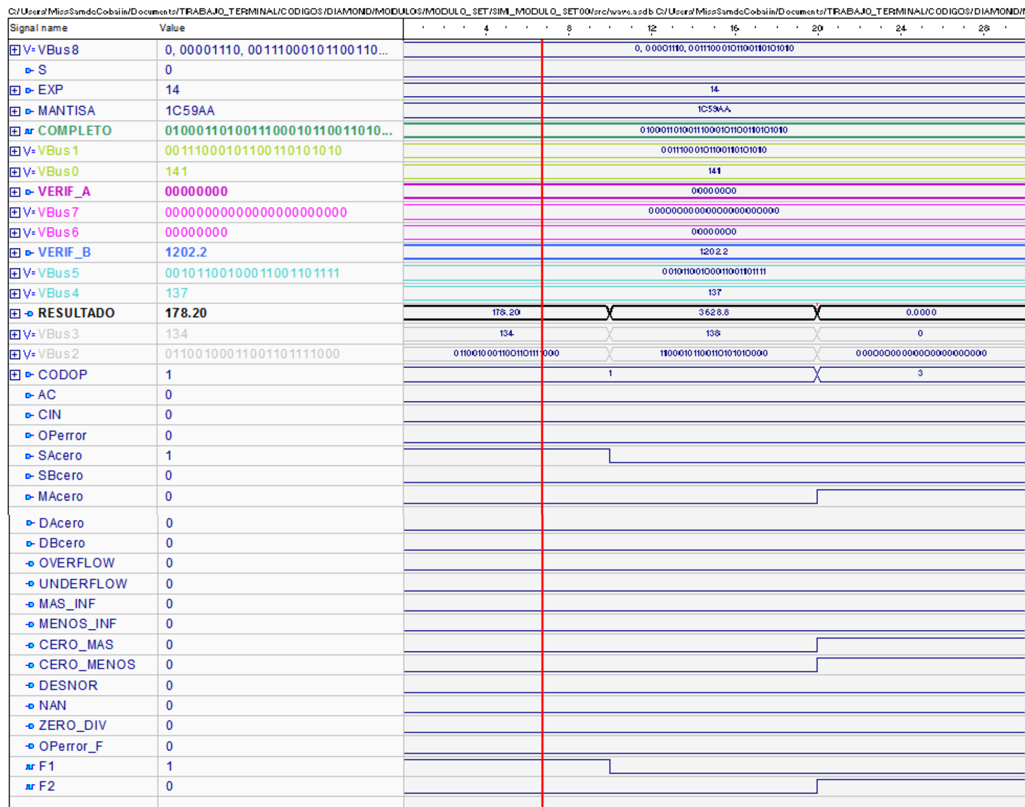


Fig 6.70 Simulación del módulo SET

6.1.4.1 Sumatoria del sesgo.

El modulo sumar sesgo cuenta con un bus de entrada de 8 bits llamado EXP_IN y expide un bus de salida denominado EXP_OUT con una respectiva bandera FSS_DESNOR, la cual indica si se trata de un underflow. El modulo únicamente se encarga de realizar la suma del sesgo que con anterioridad se le resto al exponente de un numero normalizado en el formato de punto flotante para poder operarlo. Dicha simulación se muestra a continuación:

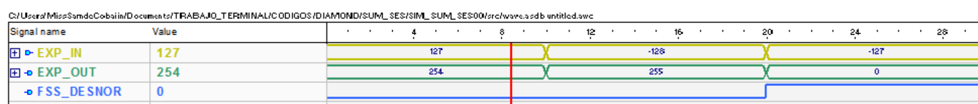


Fig 6.71 Simulación del módulo sumatoria del sesgo

6.1.4.2 Multiplexor de salida.

EL ultimo multiplexor que se encuentra en esta arquitectura se denomina MUX_RESULT00 , dicho multiplexor se encarga de permitir la salida de un solo resultado esperado, puesto que recibe tres buses a la entrada (OPERADO, VERIF_A, VERIF_B), indicando que uno de los valores aceptados podría ser cero en caso de permitir la salida de los buses VERIF_A o VERIF_B, o ser un resultado correcto proveniente del bus OPERADO, la selección de la salida viene dada por dos banderas a la entrada, FLAG1 y FLAG2, quien a partir de una serie de condicionantes indicaran el valor a la salida de dicho multiplexor, cabe mencionar que tanto los buses de entrada como el de salida son de 32 bits representados ya en punto flotante.. La simulación perteneciente a este módulo se aprecia en la siguiente imagen:

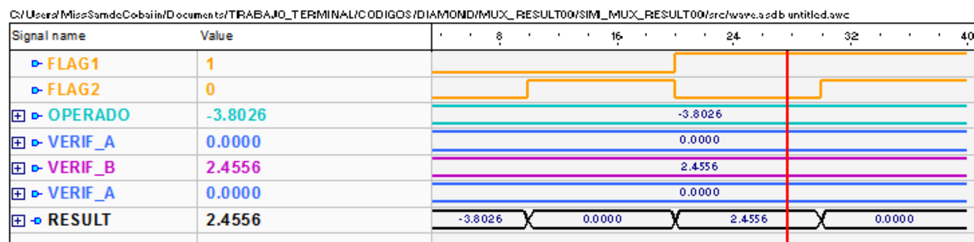


Fig 6.72 Simulación del multiplexor de salida

6.1.4.3 Normalización final.

El modulo normalizar final de esta Unidad de Punto Flotante es el encargado, como su nombre lo indica, de normalizar el resultado esperado, cuenta con una serie de condicionantes o restricciones para llevar a cabo la normalización y para ello intervienen dos banderas de entrada, AC quien es el acarreo resultante del sumador/restador y CIN el cual es el acarreo a la entrada del sumador/restador indicativo de la realización de una suma o una resta según sea el caso, también influye el código de le operación que se trata (CODOP), así como el valor del numero en punto flotante a normalizar, si bien sabemos en el caso de la suma o resta, dependiendo los valores de los signos se deben sumar las mantisas, si dicha suma genera un acarreo, no será necesario normalizar el numero recibido, pues el acarreo del sumador restador se toma como el uno implícito indicado en el formato de punto flotante, siendo esta una de las restricciones a considerar para normalizar el número. Cuando se trata de una multiplicación, no existe restricción alguna, simplemente se lleva a cabo la normalización correspondiente y cuando se opere una división se condiciona el resultado final de la mantisa, pues de modo que si es igual a 1, la normalización debe quedar como un uno implícito. Es preciso observar en la siguiente simulación todas las condicionantes anteriores, aunado que si el resultado final de la mantisa a la entrada es igual a cero, la bandera de cero (ZE) queda habilitada en alto, tal y como se muestra en la simulación siguiente:

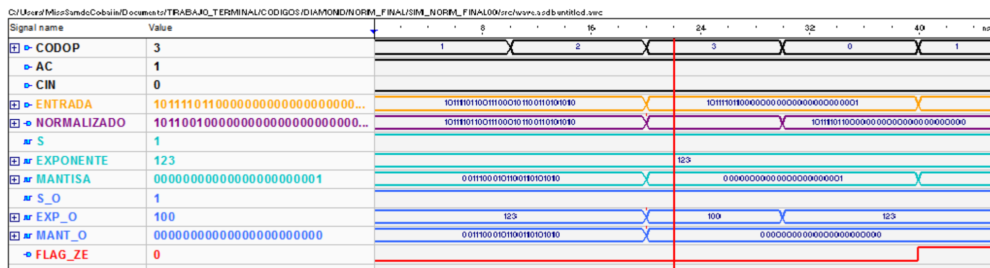


Fig 6.73 Primera simulación del módulo normalización final

Para una mejor observación de la normalización del número en punto flotante se ingresaron diversos valores en punto flotante a normalizar:

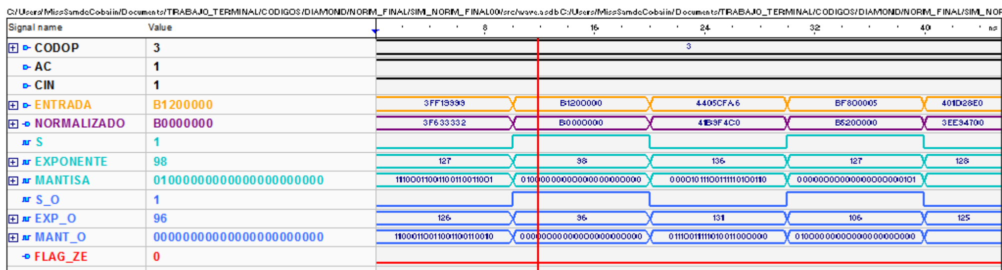


Fig 6.74 Segunda simulación del módulo normalizacion final

6.1.4.4 Verificador.

Este módulo es el encargado de verificar que los resultados obtenidos después de haber operado dos números en punto flotante sean correctos, o en caso contrario, mostrar las respectivas banderas de señalización cuando algún resultado obtenido llega a caer en alguna de las excepciones vistas del formato IEEE-754. La siguiente simulación presenta cada uno de los posibles valores obtenidos por las excepciones resultantes:

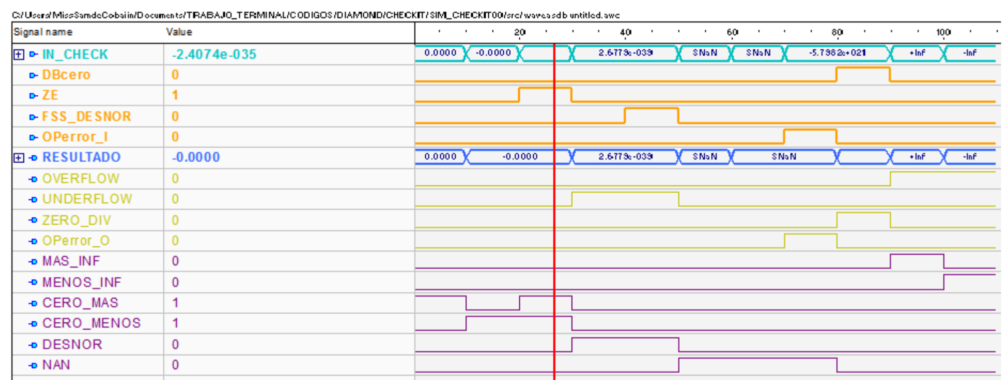


Fig 6.75 Simulación del módulo verificador

6.2 Implementación de la Memoria de Datos.

Si bien, la memoria de datos es fundamental así como al igual que los otros componentes del procesador de punto flotante para el análisis de señales, pues es aquí donde se tienen almacenados los datos a ser operados por el procesador. La memoria de datos fue dividida de la siguiente manera como se muestra en el siguiente mapa de memoria.

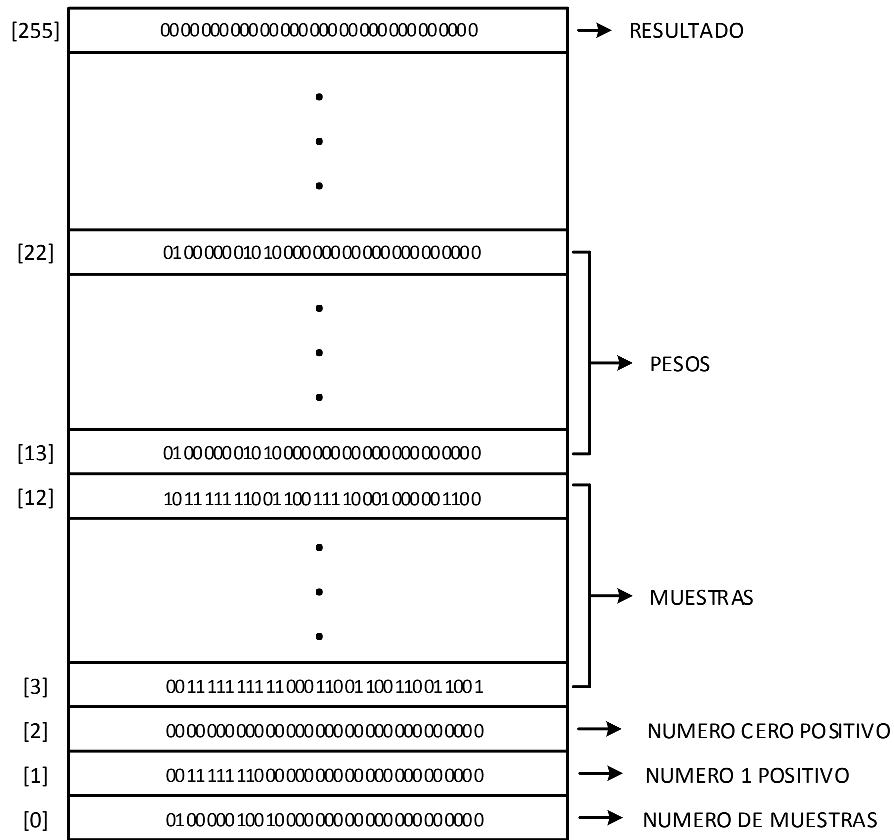


Fig 6.76 Representación del mapa de memoria

El mapa de memoria anterior nos muestra la forma en que se tienen almacenados los datos a procesar, es decir, en la dirección cero se tiene almacenado el número de muestras a ser procesadas, en específico al número 10.0, mientras que la dirección uno de la memoria de datos se almacena el número 1.0 y en la dirección dos al 0.0, posteriormente se divide la memoria desde la dirección tres hasta la dirección doce para almacenar las muestras de las señales a ser evaluadas y de la dirección trece a la veintidós se almacenan los respectivos pesos con las muestras de señales ya consideradas, finalmente se reserva la dirección doscientos cincuenta y cinco para el almacenamiento del resultado. La siguiente simulación muestra el funcionamiento de este bloque.

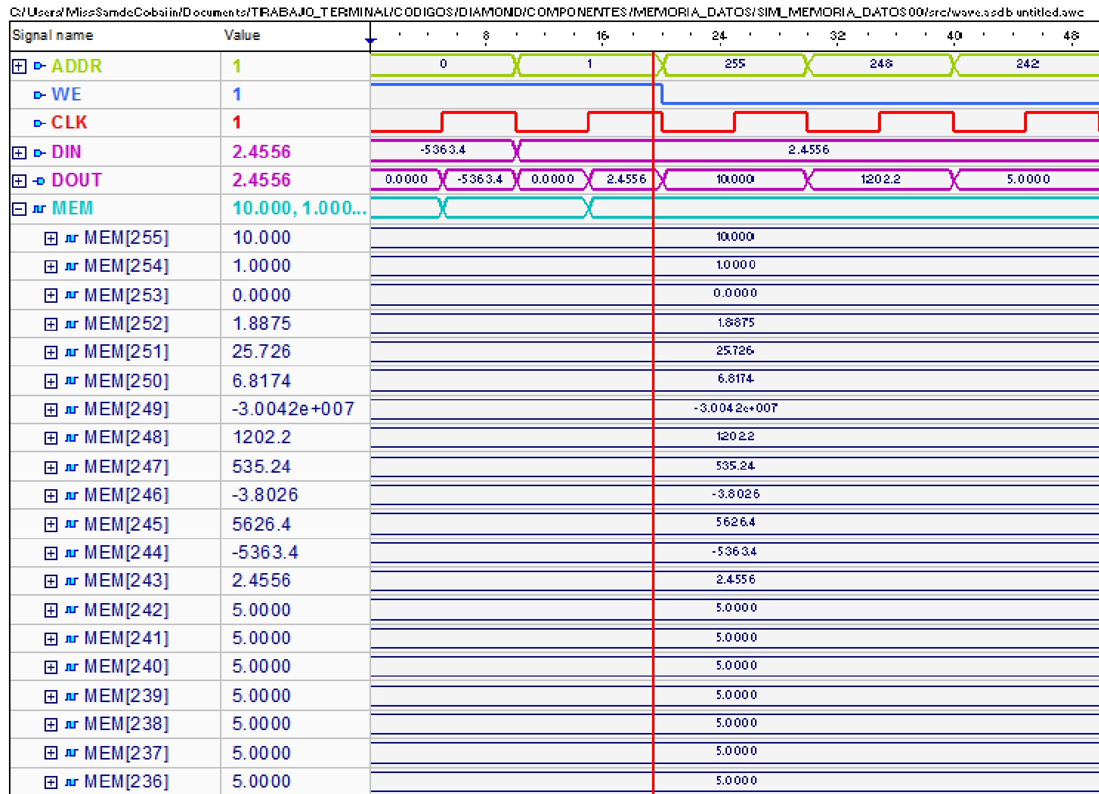


Fig 6.77 Simulación de la memoria de datos

6.3 Implementación de la memoria de programa.

A diferencia de la memoria de datos que es de lectura y escritura, la memoria de programa es de solo lectura, en ella se encuentra almacenada la lista de instrucciones a ser ejecutadas por el procesador, es decir, el programa que se desee ejecutar, para este caso en específico, la memoria de programa tiene almacenado el programa que simula la primera fase de entrenamiento del filtro adaptativo ADALINE, tal y como se muestra a continuación:

```
OPCODE_LW & R0 & SU & INI & x"00", -- CARGA EL NUMERO DE MUESTRAS AL REGISTRO 0
OPCODE_LW & R1 & SU & INI & x"01", -- CARGA EL NUMERO 1 POSITIVO AL REGISTRO 1
OPCODE_LW & R2 & SU & INI & x"02", -- CARGA  EL NUMERO 0 POSITIVO AL REGISTRO 2
OPCODE_LW & R6 & SU & INI & x"02", -- CARGA EL NUMERO 0 POSITIVO AL REGISTRO 6

OPCODE_LW & R4 & SU & "101" & x"0D", -- SUMA, PASA     --- CARGA EL DATO DE LA MEMORIA DE DATOS (DIR[3])
OPCODE_LW & R3 & SU & "111" & x"03", -- SUMA, INCREMENTA, PASA  --- CARGA EL PESO DE LA MEMORIA DE DATOS (DIN[13])
OPCODE_MULT & R4 & R3 & R5 & "000",  --- MULTIPLICA R4 Y R3  --- MULTIPLICA EL PESO POR LA SEÑAL
OPCODE_SUMA & R6 & R5 & R6 & "000", -- SUMA R6  Y R5 --  SUMA ACUMULATORIA DE PONDERACION
OPCODE_RESTA & R0 & R1 & R0 & "000", -- DECREMENTA UNA ITERACION AL NUMERO DE MUESTRAS


OPCODE_BEQ & R0 & R2 & INI & x"04", -- PREGUNTA SI LA ITERACION ES DISTINTA AL NUMERO DE MUESTRAS
                                     -- SI NO ES IGUAL REGRESA EL CONTADOR DEL PROGRAMA A LA DIRECCION 4
                                    -- SI ES IGUAL CONTINUA CON EL PROGRAMA.

OPCODE_SUMA & R6 & R1 & R6 & "000", -- SUMA EL SESGO A LA SUMA ACUMULATORIA.
OPCODE_SW & SU & R6 & INI & x"FF", -- ALMACENA EL RESULTADO EN LA ULTIMA DIRECCION DE MEMORIA.
OPCODE_LW & R7 & SU & INI & x"FF", -- MUESTRA EN EL REGISTRO 7 EL VALOR ALMACENADO DE
                                    --  LA ÚLTIMA DIRECCION DE MEMORIA DE DATOS
OPCODE_B & SU & SU & INI & x"0C", -- BRINCA A SW DEL RESULTADO
OPCODE_NOP & SU & SU & SU & "000",              -- NOP
```

Las siguientes simulaciones muestran la forma en que a partir de la dirección de memoria obtenida a la entrada, se van ejecutando las instrucciones almacenadas:

TRANSFERENCIA DE DATOS (LW)

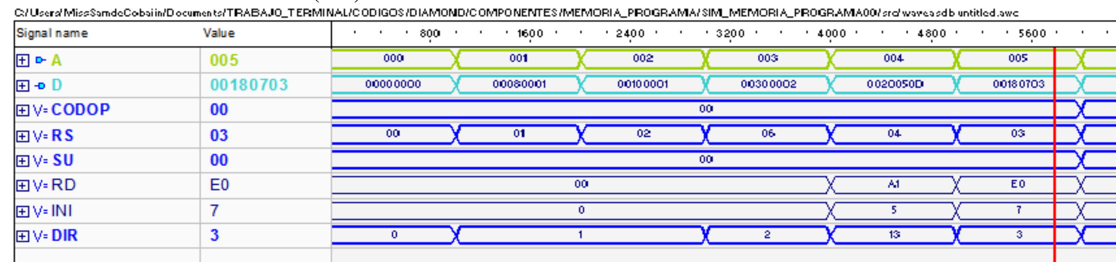


Fig 6.78 Simulación de la memoria de programa en instrucciones de transferencia de datos LW

TIPO R

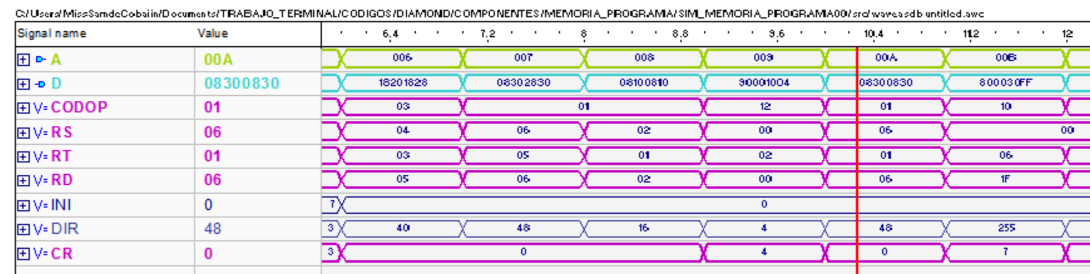


Fig 6.79 simulación de la memoria de programa en instrucciones tipo R

TRANSFERENCIA DE DATOS (SW)

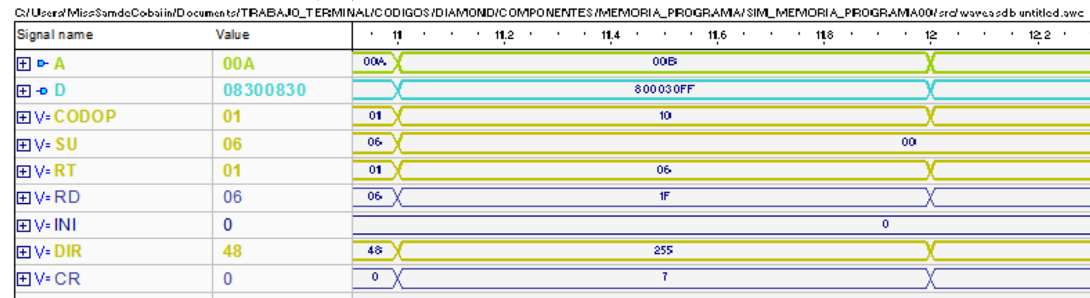


Fig 6.80 Simulación de la memoria de programa en instrucciones de tranferencia de datos SW

SALTO CONDICIONAL (BEQ)

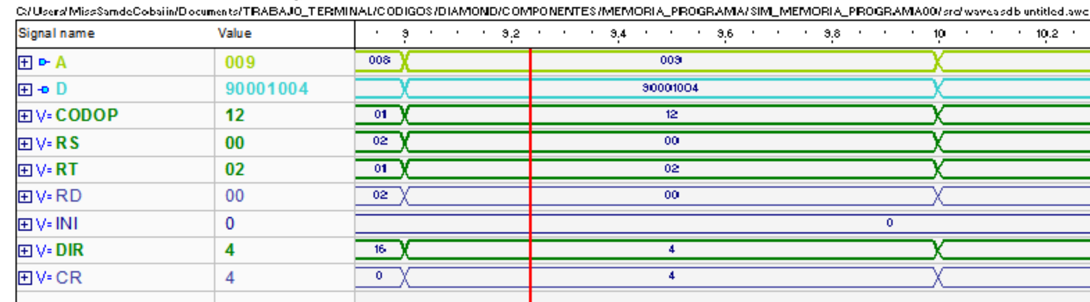


Fig 6.81 Simulación de la memoria de programa en instrucciones de salto condicional

6.4 Implementación del banco de registros.

Es en este módulo, donde se almacenan los registros a ser operados directamente a partir del formato de instrucción tipo R, dichos registros contienen valores de números en punto flotante para ser operados inmediatamente por la Unidad de Punto Flotante, mismos que son recuperados a partir de las lecturas realizadas sobre la memoria de datos. Las siguientes simulaciones muestran el funcionamiento de este módulo en específico cuando la bandera WE esta activada y cuando esta desactivada.

WRITE ENABLE ACTIVADO

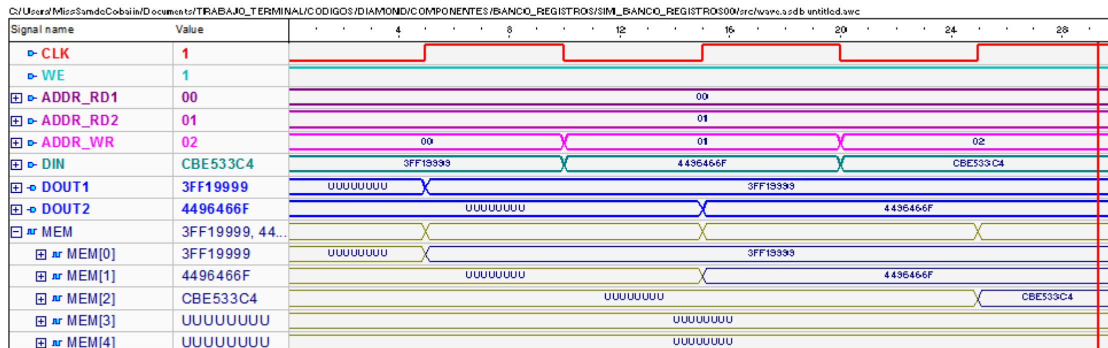


Fig 6.82 Primera simulación del banco de registros

WRITE ENABLE DESACTIVADO

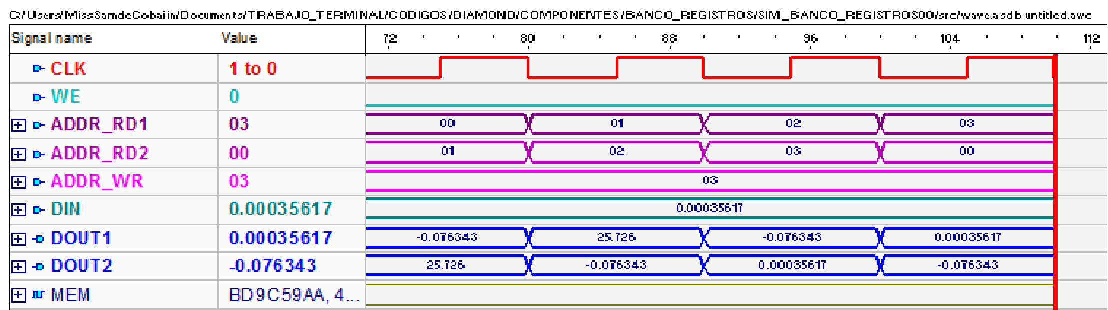


Fig 6.83 Segunda simulación del banco de registros

6.5 Implementación del contador de programa.

El contador de programa, para esta arquitectura, es la segunda parte principal de la misma, pues a partir de este módulo, se van realizando las lecturas de la memoria de programa donde se encuentran las instrucciones a procesar, por lo tanto, es evidente que sin el contador de programa, no se puede llevar a cabo el procesamiento secuencial de un programa almacenado en memoria. La siguiente simulación hace visible la manera en que dicho modulo opera.

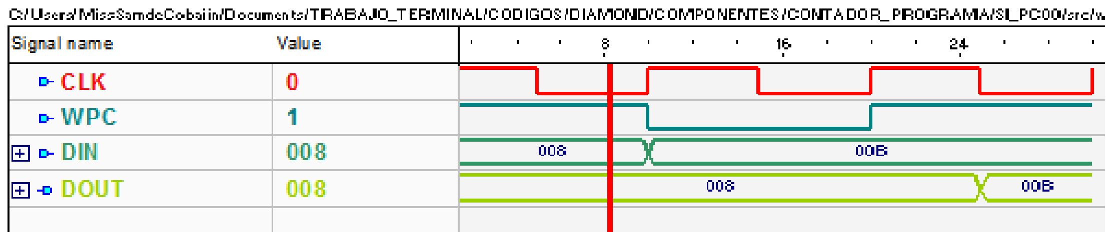


Fig 6.84 Simulación del módulo contador de programa

6.6 Implementación del sumador del contador

El modulo del sumador contador únicamente incrementa en uno la dirección recibida a la entrada desde el contador de programa, es decir, se trata de un sumador combinacional que suma el número uno a la dirección emitida, su funcionamiento se muestra en la siguiente simulación:

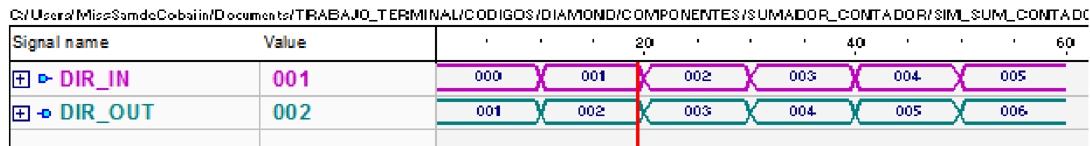


Fig 6.85 Simulación del módulo sumador del contador

6.7 Implementación del contador de direcciones.

El modulo contador de direcciones, es un módulo independiente, adaptado a la arquitectura para realizar la cuenta sucesiva y secuencial del direccionamiento de la memoria de datos. Es un módulo dedicado, específicamente para esta arquitectura pues a partir de la decodificación del formato de instrucción recibido realiza las operaciones necesarias para su funcionamiento. El contador de direcciones cuenta con dos módulos dependientes directamente entre sí, es decir, cuenta con un acumulador y un sumador. El acumulador, al principio es inicializado con cero, posteriormente almacena el dato y lo trasfiere a una de las dos entradas del sumador, si la bandera de incremento esta en alto, el acumulador va incrementando de uno en uno su valor, siempre y cuando este activa su respectiva bandera, en caso contrario solo transfiere el dato acumulado con anterioridad, dichas operaciones son realizadas durante el flanco de bajada de la señal de reloj. En cuanto al sumador refiere, es un sumador secuencial, pues realiza la suma de la dirección recibida más el resultado emitido por el acumulador durante el flanco de subida de la señal de reloj siempre y cuando la bandera de suma este activada en alto, en caso contrario no realizada dicha operación. Cabe mencionar que este módulo también recibe una bandera denominada de paso, pues en caso de ser activa, le indica al multiplexor

inmediato al resultado emitido que deje pasar el valor obtenido en vez de la dirección inmediata. En los siguientes diagramas se tiene una mejor visualización del bloque internamente.

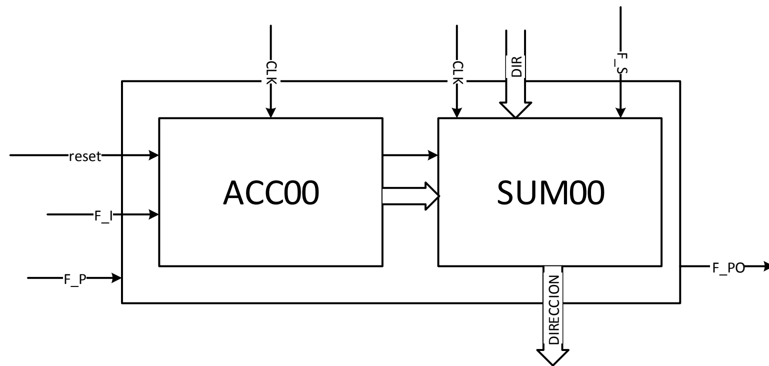


Fig 6.86 Diagrama del contador de direcciones

Para poder observar de una mejor forma la funcionalidad de este módulo se obtuvo la siguiente simulación, que especifica cada ciclo de reloj la manera en que se incrementa el acumulador al flanco de bajada y se va procesando la suma al flanco de subida de la señal de reloj:

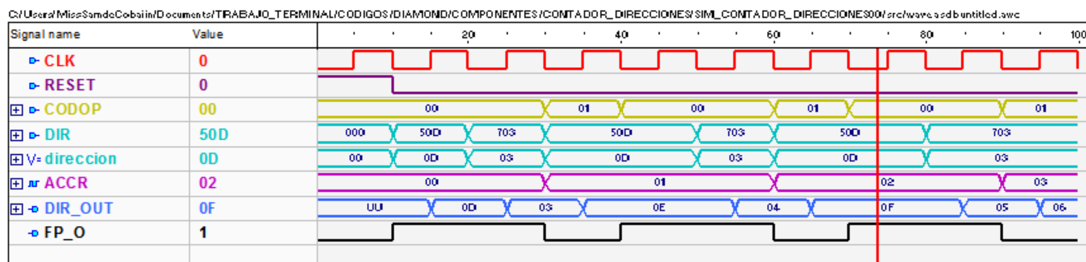


Fig 6.87 Simulación del módulo contador de direcciones

6.8 Implementación de la Unidad de Control

Las siguientes simulaciones muestran el funcionamiento de dicha unidad de control:

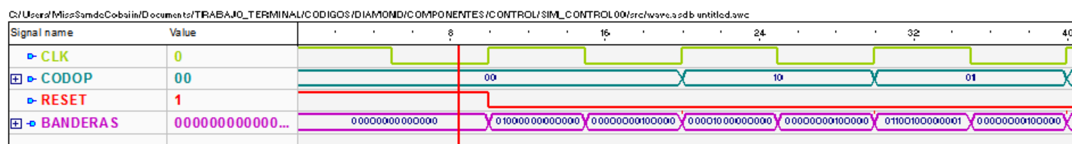


Fig 6.88 Primera simulación de la unidad de control

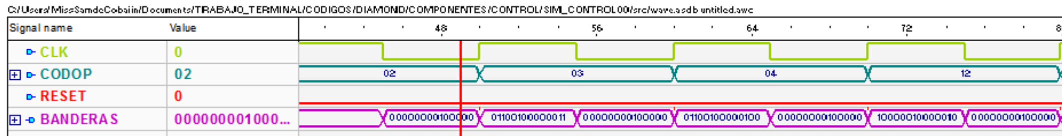


Fig 6.89 Segunda simulación de la unidad de control

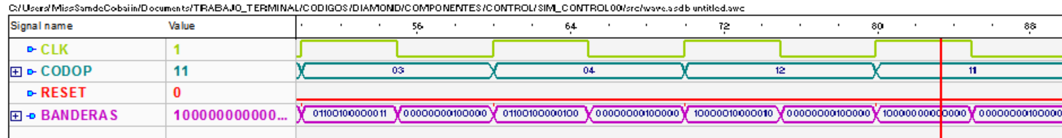


Fig 6.90 Tercera simulación de la unidad de control

La implementación de la unidad de control se lleva a cabo de acuerdo al análisis realizado sobre el siguiente diagrama:

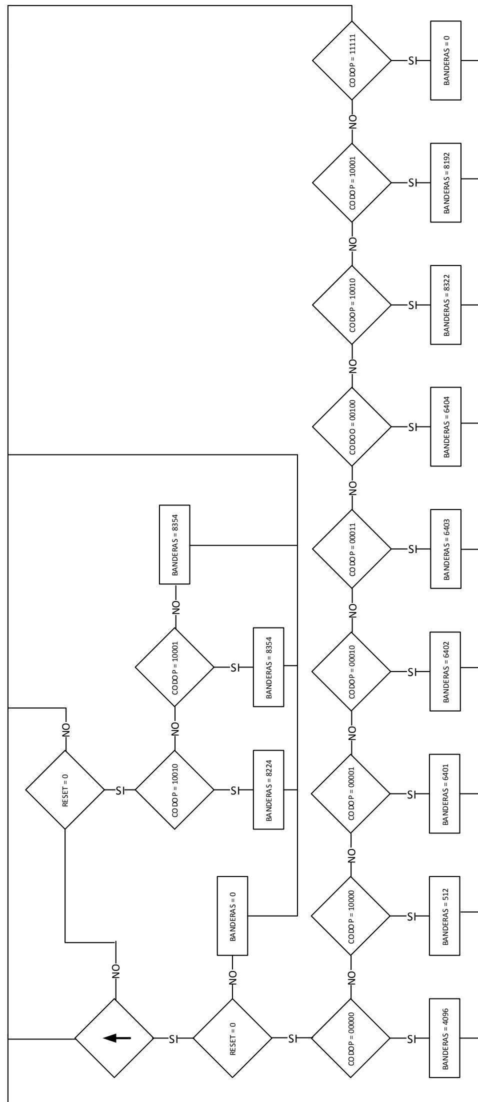


Fig 6.91 Carta ASM de la Unidad de Control

7 CAPITULO 7

7.1 Prueba y resultados.

La prueba realizada sobre el Procesador de Punto Flotante para el Análisis de Señales fue la simulación de la primera etapa de entrenamiento de la red neuronal ADALINE. El algoritmo implementado fue el algoritmo descrito en el Capítulo 5. A continuación se mostrarán los valores utilizados en muestras (señales), pesos y sesgo, así como el resultado esperado de la red neuronal.

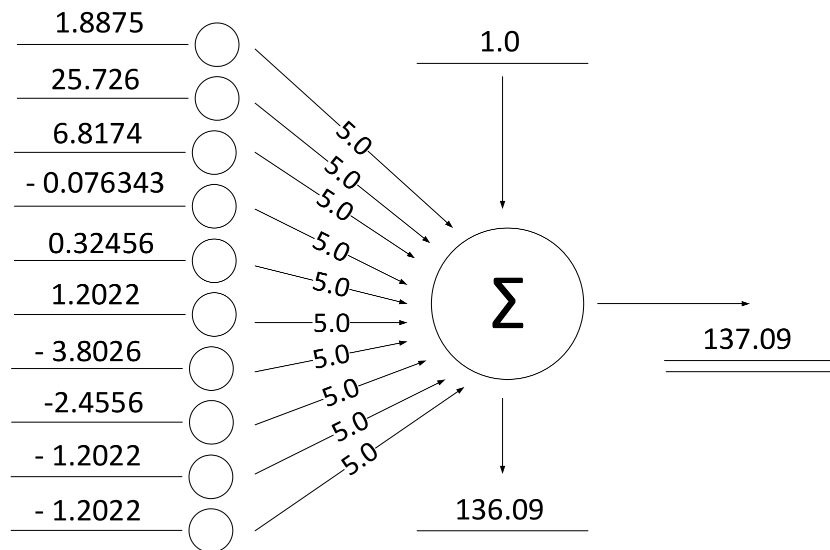


Fig 7.1.1 Neurona artificial con valores y resultados esperados

A continuación se muestra la ponderación de las señales por los respectivos pesos:

10	1.8875 x 5 =	9.4375
9	25.726 x 5 =	128.63
8	6.8174 x 5 =	34.087
7	-0.076343 x 5 =	-0.381715
6	0.32456 x 5 =	1.6228
5	1.2022 x 5 =	6.011055
4	-3.8026 x 5 =	-19.013
3	-2.4556 x 5 =	-12.278
2	-1.2022 x 5 =	-6.01155
1	-1.2022 x 5 =	-6.01155
	Total	**136.09353**

Finalmente se muestra las etapas junto con los valores obtenidos de la suma acumulatoria:

	0
	+ 9.4375
10	**= 9.4375**
	+ 128.63
9	**=138.0675**
	+ 34.087
8	**=172.1545**
	- 0.381715
7	**= 171.772785**
	+ 1.6228
6	**=173.395585**
	+ 6.011055
5	**= 179.403964**
	- 19.013
4	**=160.39364**
	- 12.278
3	**= 148.11564**
	- 6.011055
2	**=142.104585**
	- 6.011055
1	**= 136.09353**

Recordando la lista de instrucciones almacenada en la memoria de programa del procesador, se procederá a mostrar la simulación obtenida a partir de la misma. Cabe mencionar que los valores mostrados en las tablas anteriores son procesados y observados de forma iterativa mediante el procesador de punto flotante para el análisis de señales.

```
LW R0, x"00"          ; R0= MEM[0] – Carga del número de muestras al registro 0
LW R1, x"01"          ; R1= MEM[1] – Carga el número uno al registro 1
LW R2, x"02"          ; R2= MEM[2] – Carga el número cero al registro 2
LW R6, x"02"          ; R6= MEM[2] – Carga el número cero al registro 6

CICLO:
      LW R4, x"0D"          ; R4= MEM[13] – Carga el valor del peso
      LW R5, x"03"          ; R5= MEM[3]  -- Carga el valor de la señal
      MULT R5, R4, R5       ; R5 = R4 * R5 – Multiplica el peso por la señal
      ADD R6, R5, R6        ; R6 = R5 + R6 – Realiza la suma acumulatoria.
      SUB R0, R1, R0        ; R0 = R1 – R0 -- Resta el número de muestras menos uno
      BNEQ R0, R2, x"04"    ; if (R2!=R0) go to CICLO – Si el número de iteraciones
                                                        es distinto al número de
                                                        muestras, salta a la etiqueta
                                                        CICLO, en caso contrario,
                                                        continúa.

ADD R6, R1, R6      ; R6 = R1 + R6 – Suma el sesgo a la suma acumulatoria.
SW R6, x"FF"        ; MEM[255] = R6 – Almacena en memoria el valor del registro 6.
LW R7, x"FF"        ; R7 = MEM[255] – Carga el resultado final sobre el registro 7.
B x"0C"             ; PC = 12 – Brinca a la dirección 12
```

Fig 7.1.2 Programa embebido en la memoria de programa

7.1.1 Inicialización.

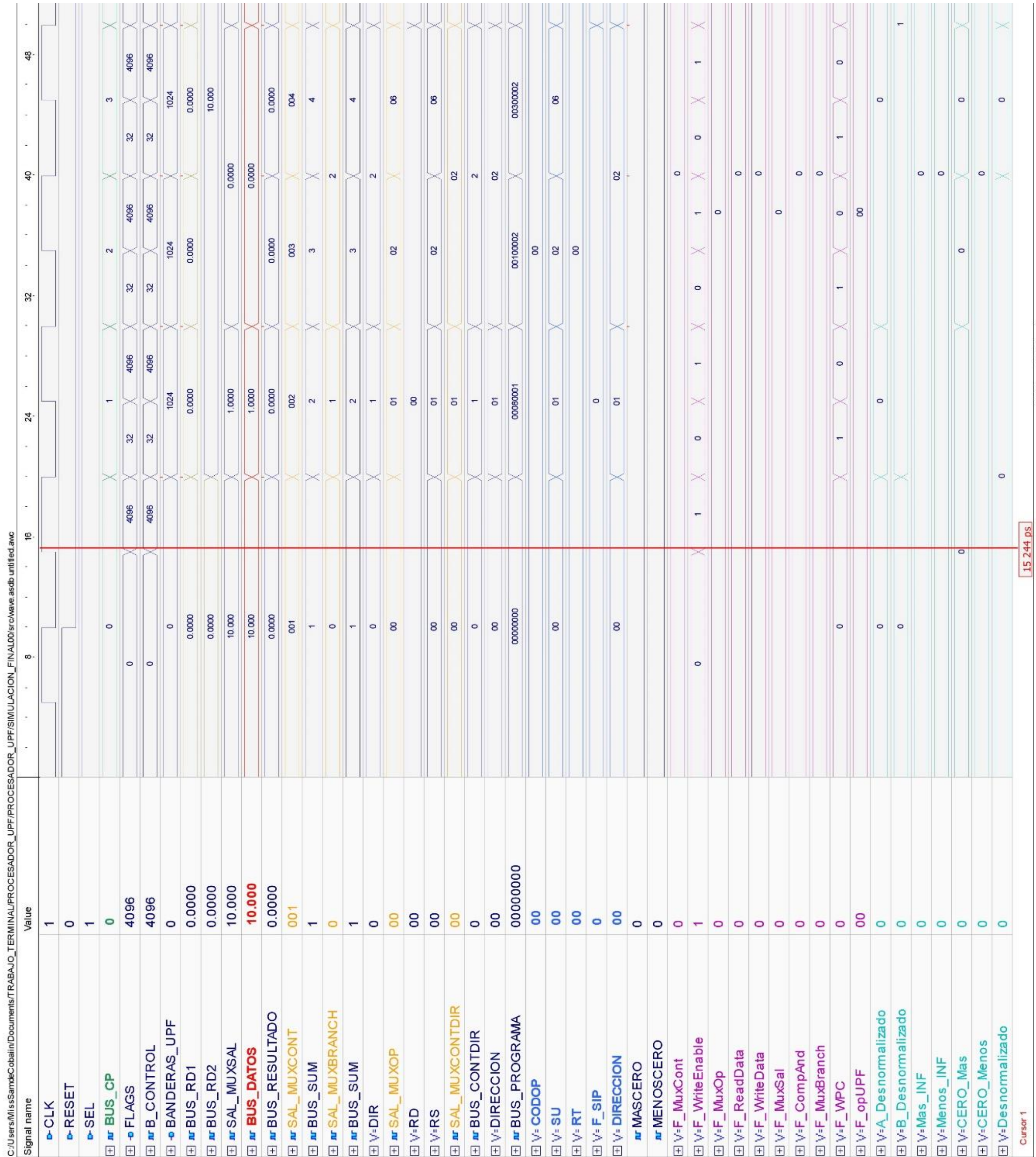


Fig 7.1.1.1 Simulación de inicialización

7.1.2 Ciclos

7.1.2.1 Primer muestra

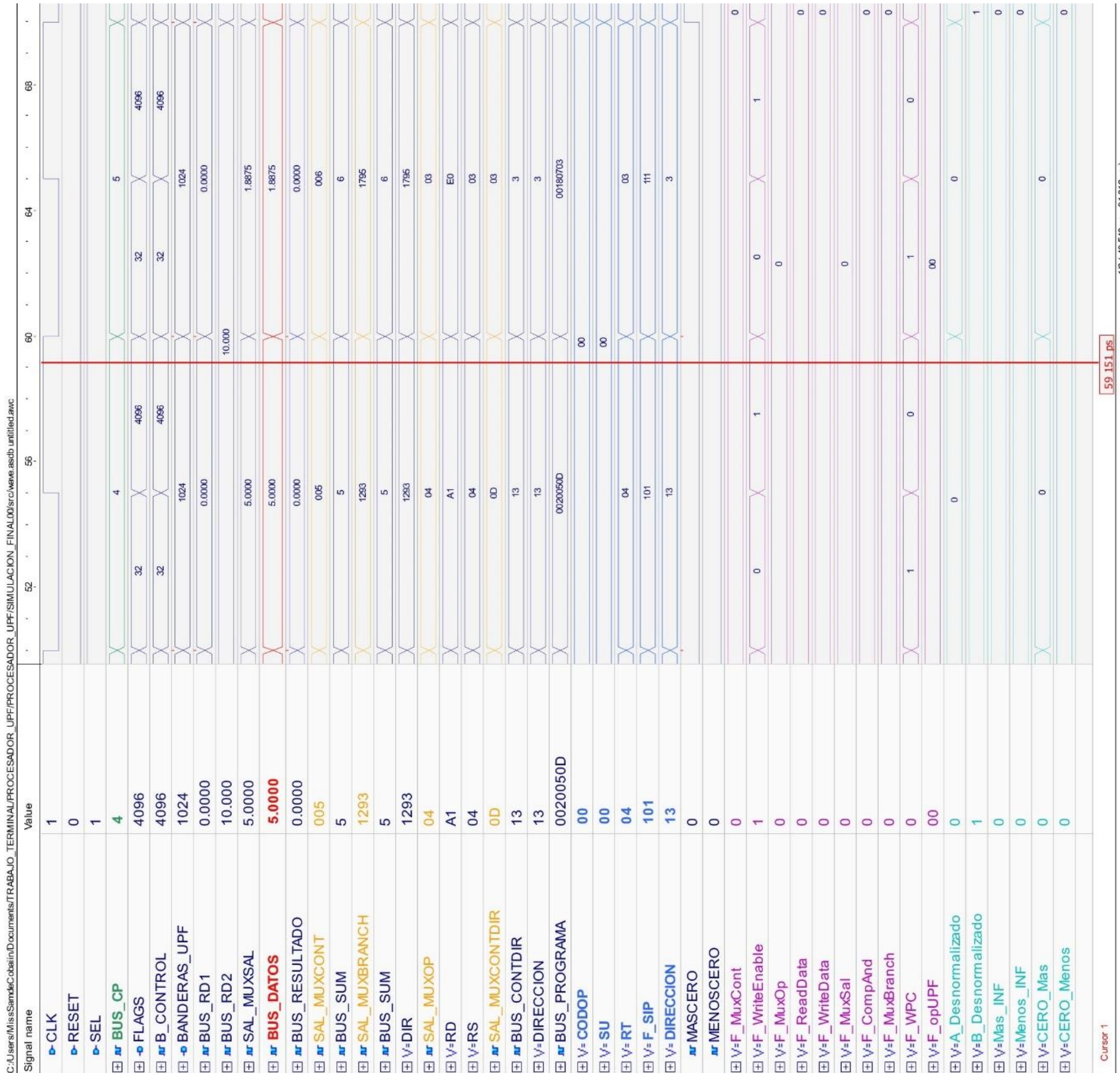


Fig 7.1.2.1.1 Primera simulación de carga

Fig 7.1.2.1.2 Primera simulación de operación y brinco

7.1.2.2 Segunda muestra

Fig 7.1.2.2.1 Segunda simulación de carga

Fig 7.1.2.2.2 Segunda simulación de operación y brinco

7.1.2.3 Tercera muestra

Fig 7.1.2.3.1 Tercera simulación de carga

Fig 7.1.2.3.2 Tercera simulación de operación y brinco

7.1.2.4 Cuarta muestra

Fig 7.1.2.4.1 Cuarta simulación de carga

Fig 7.1.2.4.2 Cuarta simulación de operación y brinco

7.1.2.5 Quinta muestra

Fig 7.1.2.5.1 Quinta simulación de carga

Fig 7.1.2.5.2 Quinta simulación de operación y brinco

7.1.2.6 Sexta muestra

Fig 7.1.2.6.1 Sexta simulación de carga

Fig 7.1.2.6.2 Sexta simulación de operación y brinco

7.1.2.7 Séptima muestra

Fig 7.1.2.7.1 Séptima simulación de carga

Fig 7.1.2.7.2 Séptima simulación de operación y brinco

7.1.2.8 Octava muestra

Fig 7.1.2.8.1 Octava simulación de carga

Fig 7.1.2.8.2 Octava simulación de operación y brinco

7.1.2.9 Novena muestra

Fig 7.1.2.9.1 Novena simulación de carga

Fig 7.1.2.9.2 Novena simulación de operación y brinco

7.1.2.10 Decima muestra

Fig 7.1.2.10.1 Décima simulación de carga

Fig 7.1.2.10.2 Décima simulación de operación y brinco

7.1.3 Resultado.

Fig 7.1.3.1 Simulación del resultado final

CONCLUSIONES

Los resultados obtenidos en la simulación fueron los mismos resultados que los esperados, cabe mencionar que siguiendo el formato de punto flotante IEEE-754 para el cálculo de operaciones fue posible implementar en hardware una arquitectura dedicada capaz de procesar cualquier valor numérico existente en el formato mismo. El procesador de punto flotante para el análisis de señales estaba basado en una arquitectura MIPS, lo cual permitió poder ejecutar cada instrucción del mismo set de instrucciones en un solo ciclo de reloj, es decir, que el procesado de datos en mayor cantidad, se puede realizar con un mejor tiempo de respuesta. Si bien, la filosofía de la arquitectura de computadoras nos indica una proporcionalidad en relación densidad-tiempo, de modo que al reducir el tiempo de procesamiento, se incrementa la densidad de nuestro diseño. El procesador únicamente maneja datos en punto flotante debido a que a los números enteros también los opera como datos en punto flotante.

Hoy en día existen diversas arquitecturas dedicadas al procesamiento de datos y al análisis de señales, pero el procesador presentado es un procesador dedicado para diversas aplicaciones específicas, tal y como puede ser implementado en aparatos de tamaño mínimo que permitan la cancelación de ruido, el promedio de diversos valores para el diagnóstico y/o control de la hipertensión o la diabetes, la selección de una señal con mayor intensidad, entre otras.

CÓDIGOS

Códigos del Módulo CHECK

1. Verificación por operandos erróneos.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MOD_OP_ERR_02 is

port(
        OP_A: in std_logic_vector(31 downto 0);

        F_OPERROR: out std_logic;
        OUT_OPA: out std_logic_vector(31 downto 0)  );
end;

architecture MOD_OP_ERROR2 of MOD_OP_ERR_02 is

        signal signo, COMP: std_logic;
        signal exp: std_logic_vector(7 downto 0);
        signal mant: std_logic_vector(22 downto 0);

begin
        signo  <= OP_A(31);
        exp <= OP_A(30 downto 23);
        mant <= OP_A(22 downto 0);

        COMP <= '1' WHEN ((exp = "11111111") and (mant /=
        "00000000000000000000000") and ((signo='1')or(signo = '0'))) ELSE '0';

        OUT_OPA <= (signo & exp & mant) WHEN COMP='0' ELSE
        ("00000000000000000000000000000000");

        F_OPERROR <= COMP;

end MOD_OP_ERROR2;
```

2. Desnormalizacion Inicial.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity NORM_INICIO01 is

port(
        OP_A: in std_logic_vector(31 downto 0) ;

        S_OPA: out std_logic_VECTOR(31 DOWNTO 0)  );
end;

architecture NORM_INICIO0 of NORM_INICIO01 is

        signal signo, signo_p, DESNOR: std_logic;
        signal exp, exp_p: std_logic_vector(7 downto 0);
        signal mant, mant_p: std_logic_vector(22 downto 0);

begin

        signo  <= OP_A(31);
        exp <= OP_A(30 downto 23);
        mant <= OP_A(22 downto 0);

        signo_p <= signo;
        exp_p <= (exp + '1');
        mant_p <= ('0' & mant(22 downto 1));

        DESNOR <= '1' WHEN ((exp = "00000000") and (mant /=
        "00000000000000000000000") and ((signo='1')or(signo = '0'))) ELSE '0';

        S_OPA <= (signo & exp & mant) WHEN DESNOR='0' ELSE (signo_p & exp_p &
        mant_p);

end NORM_INICIO0;
```

3. Verificación por operandos en cero.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity VER_OP_CERO01 is
port(
        OP_A: in std_logic_vector(31 downto 0);
        CODOP: in std_logic_vector(4 downto 0);

        F_SAcero: out std_logic;
        F_RAcero: out std_logic;
        F_MAcero: out std_logic;
        F_DAcero: out std_logic;
        S_OPA: out std_logic_vector(31 downto 0)  );
end;

architecture VER_OP_CERO0 of VER_OP_CERO01 is
signal signo: std_logic;
signal exp: std_logic_vector(7 downto 0);
signal mant: std_logic_vector(22 downto 0);
begin
        signo  <= OP_A(31);
        exp <= OP_A(30 downto 23);
        mant <= OP_A(22 downto 0);

F_SAcero <= '1' WHEN ((exp = "00000000") and (mant =
"00000000000000000000000") and ((signo='1')or(signo = '0')) and (CODOP =
"00001")) ELSE '0';
F_RAcero <= '1' WHEN ((exp = "00000000") and (mant =
"00000000000000000000000") and ((signo='1')or(signo = '0')) and (CODOP =
"00010")) ELSE '0';
F_MAcero <= '1' WHEN ((exp = "00000000") and (mant =
"00000000000000000000000") and ((signo='1')or(signo = '0')) and (CODOP =
"00011")) ELSE '0';
F_DAcero <= '1' WHEN ((exp = "00000000") and (mant =
"00000000000000000000000") and ((signo='1')or(signo = '0')) and (CODOP =
"00100")) ELSE '0';

        S_OPA <= (signo & exp & mant);

end VER_OP_CERO0;
```

4. Paquete del módulo Check.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_NORM00 is

component NORM_INICIO01
port(
        OP_A: in std_logic_vector(31 downto 0) ;
        DESNOR: out std_logic;
        S_OPA: out std_logic_VECTOR(31 DOWNTO 0)  ); end component;

component MOD_OP_ERR_02
port(
        OP_A: in std_logic_vector(31 downto 0);
        F_OPERROR: out std_logic;
        OUT_OPA: out std_logic_vector(31 downto 0)  ); end component;

component VER_OP_CERO01
port(
        OP_A: in std_logic_vector(31 downto 0);
        CODOP: in std_logic_vector(4 downto 0);

        F_SAcero: out std_logic;
        F_RAcero: out std_logic;
        F_MAcero: out std_logic;
        F_DAcero: out std_logic;

        S_OPA: out std_logic_vector(31 downto 0)  );
end component;


end PAQUETE_NORM00;
```

5. Top del módulo Check

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use PAQUETE_NORM00.all;

entity MODULO_CHECK00 is
port(
        OP_A: in std_logic_vector(31 downto 0);
        OP_B: in std_logic_vector(31 downto 0);
        CODOP: in std_logic_vector(4 downto 0);
        OUT_OPA: out std_logic_vector(31 downto 0);
        OUT_OPB: out std_logic_vector(31 downto 0);

        F_DESNOR_A: out std_logic;
        F_DESNOR_B: out std_logic;

        F_OPERROR: out std_logic;

        F_SAcero: out std_logic;
        F_RAcero: out std_logic;
        F_MAcero: out std_logic;
        F_DAcero: out std_logic;

        F_SBcero: out std_logic;
        F_RBcero: out std_logic;
        F_MBcero: out std_logic;
        F_DBcero: out std_logic  );

end;

architecture MODULO_CHECK0 of MODULO_CHECK00 is
signal F_ERROR_A, F_ERROR_B: std_logic;
signal S_NORM_A, S_NORM_B: std_logic_vector(31 downto 0);
signal S_ERROR_A, S_ERROR_B: std_logic_vector(31 downto 0);
--signal mant: std_logic_vector(22 downto 0);
begin

-- NORMALIZAR A
U1: NORM_INICIO01 port map(
        OP_A => OP_A,
        DESNOR => F_DESNOR_A,
S_OPA => S_NORM_A  );

-- NORMALIZAR B
U2: NORM_INICIO01 port map(
        OP_A => OP_B,
        DESNOR => F_DESNOR_B,
        S_OPA => S_NORM_B  );

-- OPERANDO ERRONEO A
U3: MOD_OP_ERR_02 port map(
        OP_A => S_NORM_A,
        F_OPERROR => F_ERROR_A,
        OUT_OPA => S_ERROR_A  );

-- OPERANDO ERRONEO B
U4: MOD_OP_ERR_02 port map(
        OP_A => S_NORM_B,
        F_OPERROR => F_ERROR_B,
        OUT_OPA => S_ERROR_B  );

-- VERIFICAR OPERANDO EN CERO A
U5: VER_OP_CERO01 port map(
        OP_A => S_ERROR_A,
        CODOP => CODOP,
        F_SAcero => F_SAcero,
        F_RAcero => F_RAcero,
        F_MAcero => F_MAcero,
        F_DAcero => F_DAcero,
        S_OPA => OUT_OPA  );

-- VERIFICAR OPERANDO EN CERO A
U6: VER_OP_CERO01 port map(
        OP_A => S_ERROR_B,
        CODOP => CODOP,
        F_SAcero => F_SBcero,
        F_RAcero => F_RBcero,
        F_MAcero => F_MBcero,
        F_DAcero => F_DBcero,
        S_OPA => OUT_OPB  );

  -- XOR BANDERA opERROR
F_OPERROR <= F_ERROR_A OR F_ERROR_B;

end MODULO_CHECK0;
```

Codigos Módulo READY

6. Resta del sesgo al exponente

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity RES_SES00 is
port(
        OP: in std_logic_vector(31 downto 0) ;
        signo: out std_logic;
        exp: out std_logic_VECTOR(7 DOWNTO 0);
        mantisa: out std_logic_VECTOR(22 DOWNTO 0));
end;

architecture RES_SES0 of RES_SES00 is
signal exp_p: std_logic_vector(7 downto 0);
begin
        signo  <= OP(31);
        exp_p <= OP(30 downto 23);
        mantisa <= OP(22 downto 0);
        exp <= (exp_p - "01111111");

end RES_SES0;
```

Comparación entre exponentes

7. Comparación

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity COMP00 is
port(
        A: in std_logic_vector(7 downto 0);
        B: in std_logic_vector(7 downto 0);
        BMA: out std_logic;
        AMB: out std_logic;
        BIA: out std_logic   );
end;
architecture COMP0 of COMP00 is
begin
        AMB <= '1' when ( ((A(7) = '0') AND (B(7) = '1')) OR (((A(6 DOWNTO 0)) > (B(6
DOWNTO 0))) AND ((((A(7)='0') AND (B(7)='0')) OR ((A(7)='1') AND (B(7)='1'))))) ) else
'0';
        BMA <= '1' when ( ((A(7) = '1') AND (B(7) = '0')) OR (((A(6 DOWNTO 0)) < (B(6
DOWNTO 0))) AND ((((A(7)='0') AND (B(7)='0')) OR ((A(7)='1') AND (B(7)='1'))))) ) else
'0';
        BIA <= '1' when ( A = B ) else '0';
end COMP0;
```

8. Resta

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity RESTA00 is
port(
        A: in std_logic_vector(7 downto 0);
        B: in std_logic_vector(7 downto 0);
        BMA: in std_logic;
        AMB: in std_logic;
        BIA: in std_logic;
        DIF: out std_logic_vector(7 downto 0);
        F_COMP_A: out std_logic;
        F_COMP_B: out std_logic );
end;

architecture RESTA0 of RESTA00 is
signal OPA, OPB: std_logic_vector(7 downto 0);
begin
        OPA <= (A - B) ;
        OPB <= (B - A) ;
        DIF <= OPA WHEN AMB = '1' ELSE
                        OPB WHEN BMA = '1'
                        ELSE "00000000";
        F_COMP_A <= '1' when BMA = '1' else '0';
        F_COMP_B <= '1' when AMB= '1' else '0';
end RESTA0;
```

9. Paquete del comparador.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_COMPARADOR00 is

component COMP00
        port(
        A: in std_logic_vector(7 downto 0);
        B: in std_logic_vector(7 downto 0);
        BMA: out std_logic;
        AMB: out std_logic;
        BIA: out std_logic   );
end component;

component RESTA00
port(
        A: in std_logic_vector(7 downto 0);
        B: in std_logic_vector(7 downto 0);
        BMA: in std_logic;
        AMB: in std_logic;
        BIA: in std_logic;
        DIF: out std_logic_vector(7 downto 0);
        F_COMP_A: out std_logic;
        F_COMP_B: out std_logic
);
end component;

end PAQUETE_COMPARADOR00;
```

10. Top del Comparador.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use paquete_comparador00.all;

entity COMPARA00 is
port(
        A: in std_logic_vector(7 downto 0);
        B: in std_logic_vector(7 downto 0);
        DIF: out std_logic_vector(7 downto 0);
        F_COMP_A: out std_logic;
        F_COMP_B: out std_logic   );
end;

architecture COMPARA0 of COMPARA00 is
signal F_BMA, F_AMB, F_BIA: std_logic;
begin

U7: COMP00 port map(
        A => A,
        B => B,
        BMA => F_BMA,
        AMB => F_AMB,
        BIA => F_BIA  );

 U8: RESTA00 port map(
        A => A,
        B => B,
        BMA => F_BMA,
        AMB => F_AMB,
        BIA => F_BIA,
        DIF => DIF,
        F_COMP_A => F_COMP_A,
        F_COMP_B => F_COMP_B  );

end COMPARA0;
```

11. Sumatoria al exponente.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity SUM_EXP00 is
port(
        EXP_A: in std_logic_vector(7 downto 0);
        DIFF: in std_logic_vector(7 downto 0);

        F_COMP_A: in std_logic;
```

```
		SAL: out std_logic_vector(7 downto 0) );
end;

architecture SUM_EXP0 of SUM_EXP00 is
begin
		SAL <= (EXP_A + DIFF) WHEN F_COMP_A = '1' ELSE
					EXP_A ;
end SUM_EXP0;
```

12. Corrimiento sobre la mantisa.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity SHIFT_MANT00 is
port(
		MANTISA: in std_logic_vector(22 downto 0);
		DIFF: in std_logic_vector(7 downto 0);

		F_COMP_X: in std_logic;

		S_MANT: out std_logic_vector(22 downto 0) );
end;

architecture SHIFT_MANT0 of SHIFT_MANT00 is
begin

   S_MANT <= MANTISA WHEN (DIFF = "00000000" AND
(F_COMP_X='1')) ELSE
					"0" & MANTISA(22 downto 1) WHEN
(DIFF = "00000001" AND (F_COMP_X='1')) ELSE
					"00" & MANTISA(22 downto 2)
WHEN (DIFF = "00000010" AND (F_COMP_X='1')) ELSE
					"000" & MANTISA(22 downto 3)
WHEN (DIFF = "00000011" AND (F_COMP_X='1')) ELSE
					"0000" & MANTISA(22 downto 4)
WHEN (DIFF = "00000100" AND (F_COMP_X='1')) ELSE
					"00000" & MANTISA(22 downto 5)
WHEN (DIFF = "00000101" AND (F_COMP_X='1')) ELSE
					"000000" & MANTISA(22 downto 6)
WHEN (DIFF = "00000110" AND (F_COMP_X='1')) ELSE
					"0000000" & MANTISA(22 downto 7)
WHEN (DIFF = "00000111" AND (F_COMP_X='1')) ELSE
					"00000000" & MANTISA(22 downto
8) WHEN (DIFF = "00001000" AND (F_COMP_X='1')) ELSE
					"000000000" & MANTISA(22 downto
9) WHEN (DIFF = "00001001" AND (F_COMP_X='1')) ELSE
					"0000000000" & MANTISA(22
downto 10) WHEN (DIFF = "00001010" AND (F_COMP_X='1')) ELSE
					"00000000000" & MANTISA(22
downto 11) WHEN (DIFF = "00001011" AND (F_COMP_X='1')) ELSE
					"000000000000" & MANTISA(22
downto 12) WHEN (DIFF = "00001100" AND (F_COMP_X='1')) ELSE
					"0000000000000" & MANTISA(22
downto 13) WHEN (DIFF = "00001101" AND (F_COMP_X='1')) ELSE
					"00000000000000" & MANTISA(22
downto 14) WHEN (DIFF = "00001110" AND (F_COMP_X='1')) ELSE
					"000000000000000" & MANTISA(22
downto 15) WHEN (DIFF = "00001111" AND (F_COMP_X='1')) ELSE
					"0000000000000000" & MANTISA(22
downto 16) WHEN (DIFF = "00010000" AND (F_COMP_X='1')) ELSE
					"00000000000000000" &
MANTISA(22 downto 17) WHEN (DIFF = "00010001" AND
(F_COMP_X='1')) ELSE
					"000000000000000000" &
MANTISA(22 downto 18) WHEN (DIFF = "00010010" AND
(F_COMP_X='1')) ELSE
					"0000000000000000000" &
MANTISA(22 downto 19) WHEN (DIFF = "00010011" AND
(F_COMP_X='1')) ELSE
					"00000000000000000000" &
MANTISA(22 downto 20) WHEN (DIFF = "00010100" AND
(F_COMP_X='1')) ELSE
					"000000000000000000000" &
MANTISA(22 downto 21) WHEN (DIFF = "00010101" AND
(F_COMP_X='1')) ELSE
					"0000000000000000000000" &
MANTISA(22) WHEN (DIFF = "00010110" AND (F_COMP_X='1'))
ELSE
					"00000000000000000000000" WHEN
(DIFF > "00010110" AND (F_COMP_X='1')) ELSE
					MANTISA;

end SHIFT_MANT0;
```

13. Extensor de signo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity EXTENSOR00 is
port(
        EXP: in std_logic_vector(7 downto 0);
        EX_EXP: out std_logic_vector(22 downto 0) );
end;

architecture EXTENSOR0 of EXTENSOR00 is
begin
        EX_EXP <= "000000000000000"&EXP WHEN EXP(7)='0' ELSE
"111111111111111"&EXP;

end EXTENSOR0;
```

14. Multiplexor entre mantisa y exponente

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity DEMUX_M2 is
port(
        EXTENSOR: in std_logic_vector(22 downto 0);
        MANTISA: in std_logic_vector(22 downto 0);
        SEL: in std_logic_vector(1 downto 0);
        SAL_MUX: out std_logic_vector(22 downto 0)
        );
end;

architecture DEMUX_M20 of DEMUX_M2 is
begin

SAL_MUX <= MANTISA WHEN ((SEL = "01") OR (SEL = "10")) ELSE EXTENSOR;

end DEMUX_M20;
```

15. Paquete del Módulo Ready.

```
library ieee
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_MODULO_READY is

-- MODULO RESTAR SESGO
component RES_SES00
        port(
        OP: in std_logic_vector(31 downto 0) ;

        signo: out std_logic;
        exp: out std_logic_VECTOR(7 DOWNTO 0);
        mantisa: out std_logic_VECTOR(22 DOWNTO 0)
);
end component;


-- MODULO COMPARAR EXPONENTES
component COMPARA00
        port(
        A: in std_logic_vector(7 downto 0);
        B: in std_logic_vector(7 downto 0);

        DIF: out std_logic_vector(7 downto 0);
        F_COMP_A: out std_logic;
        F_COMP_B: out std_logic
);
end component;


-- MODULO SUMA DE EXPONENTES (+N)
component SUM_EXP00
        port(
        EXP_A: in std_logic_vector(7 downto 0);
        DIFF: in std_logic_vector(7 downto 0);
        F_COMP_A: in std_logic;

        SAL: out std_logic_vector(7 downto 0)
);
end component;


-- MODULO CORRIMIENTO DE MANTISA (N>>)
```

```
component SHIFT_MANT00
        port(
        MANTISA: in std_logic_vector(22 downto 0);
        DIFF: in std_logic_vector(7 downto 0);
        F_COMP_X: in std_logic;

        S_MANT: out std_logic_vector(22 downto 0)
);
end component;


-- MODULO EXTENSOR DE SIGNO
component EXTENSOR00
        port(
        EXP: in std_logic_vector(7 downto 0);
        EX_EXP: out std_logic_vector(22 downto 0)
);
end component;


-- MODULO MULTIPLEXOR
component MUX_M2
        port(
        EXTENSOR: in std_logic_vector(22 downto 0);
        MANTISA: in std_logic_vector(22 downto 0);
        SEL: in std_logic_vector(1 downto 0);

        SAL_MUX: out std_logic_vector(22 downto 0)
);
end component;


end PAQUETE_MODULO_READY;
```

16. Top del Módulo Ready

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use paquete_modulo_ready.all;

entity MODULO_READY00 is

port(
OPA: in std_logic_vector(31 downto 0);
OPB: in std_logic_vector(31 downto 0);
COD_OP: in std_logic_vector(1 downto 0);


MA_NATURAL: out std_logic_vector(22 downto 0);
MB_NATURAL: out std_logic_vector(22 downto 0);
SAL_MUXA: out std_logic_vector(22 downto 0);
SAL_MUXB: out std_logic_vector(22 downto 0);
EXP: out std_logic_vector(7 downto 0);
SA: out std_logic;
SB: out std_logic   );

end;

architecture MODULO_READY0 of MODULO_READY00 is
signal F_A, F_B: std_logic;
signal EXP_A, EXP_B, RES_DIF: std_logic_vector(7 downto 0);
signal MANTISA_A, MANTISA_B, SA_MANT, SB_MANT, SX_A, SX_B:
std_logic_vector(22 downto 0);
begin

--- RESTAR SESGO OPERANDO A
U9: RES_SES00 port map(

        OP => OPA,

        signo => SA,
        exp => EXP_A,
        mantisa => MANTISA_A

  );

--- RESTAR SESGO OPERANDO B
U10: RES_SES00 port map(

        OP => OPB,

        signo => SB,
        exp => EXP_B,
        mantisa => MANTISA_B

  );



-- COMPARADOR DE EXPONENTES
U11: COMPARA00 port map(
```

```
        A => EXP_A,
        B => EXP_B,

        DIF => RES_DIF,
        F_COMP_A => F_A,
        F_COMP_B => F_B

  );


-- SUMA DE EXPONENTES
U12: SUM_EXP00 port map(

        EXP_A => EXP_A,
        DIFF => RES_DIF,
        F_COMP_A => F_A,

        SAL => EXP

  );


-- MODULO CORRIMIENTO DE MANTISA DE A
U13: SHIFT_MANT00 port map(

        MANTISA => MANTISA_A,
        DIFF => RES_DIF,
        F_COMP_X => F_A,

        S_MANT => SA_MANT

  );

-- MODULO CORRIMIENTO DE MANTISA DE B
U14: SHIFT_MANT00 port map(

        MANTISA => MANTISA_B,
        DIFF => RES_DIF,
        F_COMP_X => F_B,

        S_MANT => SB_MANT

  );


-- MODULO EXTENSOR DE SIGNO DEL EXPONENTE DE A
U15: EXTENSOR00 port map(

        EXP => EXP_A,
        EX_EXP => SX_A

  );


-- MODULO EXTENSOR DE SIGNO DEL EXPONENTE DE B
U16: EXTENSOR00 port map(

        EXP => EXP_B,
        EX_EXP => SX_B

  );

-- MULTIPLEXOR ENTRE EXTENSOR Y MANTISA DE A
U17: MUX_M2 port map(

        EXTENSOR => SX_A,
        MANTISA => SA_MANT,
        SEL => COD_OP,

        SAL_MUX => SAL_MUXA

  );

-- MULTIPLEXOR ENTRE EXTENSOR Y MANTISA DE B
U18: MUX_M2 port map(

        EXTENSOR => SX_B,
        MANTISA => SB_MANT,
        SEL => COD_OP,

        SAL_MUX => SAL_MUXB

  );


MA_NATURAL <= MANTISA_A;
MB_NATURAL <= MANTISA_B;

end MODULO_READY0;
```

17. Multiplexor del módulo compute

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity MUX_X00 is

port(
A: in std_logic_vector(22 downto 0);
B: in std_logic_vector(22 downto 0);
F_SELMUX: in std_logic;

SAL_MUXX: out std_logic_vector(22 downto 0)

);

end;

architecture MUX_X0 of MUX_X00 is
begin
        SAL_MUXX <= A when F_SELMUX = '0' else B;
end MUX_X0;
```

18. Comparador entre mantisas.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity COMP_MANTISA00 is

port(
MA: in std_logic_vector(22 downto 0);
MB: in std_logic_vector(22 downto 0);

AMB: out std_logic;
BMA: out std_logic;
AEB: out std_logic
);

end;

architecture COMP_MANTISA0 of COMP_MANTISA00 is
begin


        AMB <= '1' when MA > MB  else '0';
        BMA <= '1' when MA < MB  else '0';
        AEB <= '1' when MA = MB  else '0';


end COMP_MANTISA0;
```

19. Selector de operación y signo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity SELECTOR00 is
port(
        CODOP: in std_logic_vector(2 downto 0);
        BMA: in std_logic;
        AMB: in std_logic;
        AEB: in std_logic;
        SA: in std_logic;
        SB: in std_logic;
        SELMUX: out std_logic;
        CIN: out std_logic;
        SR: out std_logic
);

end;
```

```
architecture SELECTOR0 of SELECTOR00 is
signal JUNTO: std_logic_vector(2 downto 0);
begin
                                        ---- EN CASO DE SUMA
        JUNTO <= "00"&SA WHEN ((CODOP = "001") AND
(SA=SB)) ELSE
                                        "01"&SA WHEN ((CODOP = "001")
AND (SA/=SB) AND (AMB = '1')) ELSE
                                        "11"&SB WHEN ((CODOP = "001")
AND (SA/=SB) AND (BMA = '1')) ELSE
                                        "010" WHEN   ((CODOP = "001")
AND (SA/=SB) AND (AEB = '1')) ELSE

                                        ---- EN CASO DE RESTA
                                        "00"&SA WHEN ((CODOP = "010")
AND (SA/=SB)) ELSE--
                                        "01"&SA WHEN ((CODOP = "010")
AND (SA=SB) AND (AMB = '1')) ELSE
                                        "11"&(NOT(SA)) WHEN ((CODOP =
"010") AND (SA=SB) AND (BMA = '1')) ELSE
                                        "011" WHEN   ((CODOP = "010")
AND (SA=SB) AND (AEB = '1')) ELSE

                                        ---- EN CASO DE
MULTIPLICACION
                                        "00"&(SA XOR SB) WHEN (CODOP
= "011") ELSE

                                        ---- EN CASO DE DIVISION
                                        "01"&(SA XOR SB) WHEN(CODOP
= "100") ELSE "000";

SELMUX <= JUNTO(2);
CIN <= JUNTO(1);
SR <= JUNTO(0);

end SELECTOR0;
```

20. Sumador / Restador

Bloque uno.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity BLOQUEUNO00 is

port(
        A: in std_logic ;
        B: in std_logic ;
        G: out std_logic ;
        P: out std_logic  );

end;

architecture BLOQUEUNO0 of BLOQUEUNO00 is
begin


G <= (A AND B);
P <= (A XOR B);


end BLOQUEUNO0;
```

Bloque Dos

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity BLOQUEDOS00 is

port(
        G: in std_logic ;
        P: in std_logic ;
        Cin: in std_logic ;
        Cout: out std_logic  );
```

```
end;

architecture BLOQUEDOS0 of BLOQUEDOS00 is
begin

Cout <= (G OR(P AND Cin));

end BLOQUEDOS0;
```

Bloque Tres

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity BLOQUETRES00 is

port(
        P: in std_logic ;
        Cin: in std_logic ;
        S: out std_logic
          );

end;

architecture BLOQUETRES0 of BLOQUETRES00 is

begin



S <= Cin XOR P;



end BLOQUETRES0;
```

21. Paqute Sumador – Restador.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package paquete_sum is

component BLOQUEDOS00

port(
        G: in std_logic ;
        P: in std_logic ;
        Cin: in std_logic ;
        Cout: out std_logic  );

end component;


component BLOQUETRES00

port(
        P: in std_logic ;
        Cin: in std_logic ;
        S: out std_logic  );

end component;

component BLOQUEUNO00

port(
        A: in std_logic ;
        B: in std_logic ;
        G: out std_logic ;
        P: out std_logic  );

end component;

end paquete_sum;
```

22. Top del Sumador – Restador.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use paquete_sum.all;

entity SUM_RES00 is

port(
        A: in std_logic_vector ( 22 downto 0 );
        Bb: in std_logic_vector ( 22 downto 0 );
        Cin: in std_logic ;
        SAL: out std_logic_vector ( 22 downto 0 );


        S: out std_logic;
        Z: out std_logic;
        AC: out std_logic;
        O: out std_logic



 );




end;

architecture SUM_RES0 of SUM_RES00 is
signal B, G, P, C, So: std_logic_vector(22 downto 0);

begin



B <= (NOT(Bb)) WHEN Cin = '1' ELSE Bb;

--------------------------------------BIT0
        U1: BLOQUEUNO00 port map(
                A => A(0),
                B => B(0),
                G => G(0),
                P => P(0)  );

        U2: BLOQUEDOS00 port map(
                G => G(0),
                P => P(0),
                Cin => Cin,
                Cout => C(0)            );

        U3: BLOQUETRES00 port map(
                P => P(0),
                Cin => Cin,
                S => So(0));


--------------------------------------BIT1
        U4: BLOQUEUNO00 port map(
                A => A(1),
                B => B(1),
                G => G(1),
                P => P(1)  );

        U5: BLOQUEDOS00 port map(
                G => G(1),
                P => P(1),
                Cin => C(0),
                Cout => C(1)            );

        U6: BLOQUETRES00 port map(
                P => P(1),
                Cin => C(0),
                S => So(1));



--------------------------------------BIT2
        U7: BLOQUEUNO00 port map(
                A => A(2),
                B => B(2),
                G => G(2),
                P => P(2)  );

        U8: BLOQUEDOS00 port map(
                G => G(2),
                P => P(2),
                Cin => C(1),
                Cout => C(2)            );

        U9: BLOQUETRES00 port map(
                P => P(2),
                Cin => C(1),
                S => So(2));



--------------------------------------BIT3
        U10: BLOQUEUNO00 port map(
                A => A(3),
                B => B(3),
                G => G(3),
                P => P(3)  );

        U11: BLOQUEDOS00 port map(
                G => G(3),
                P => P(3),
                Cin => C(2),
                Cout => C(3)            );

        U12: BLOQUETRES00 port map(
                P => P(3),
                Cin => C(2),
                S => So(3));




--------------------------------------BIT4
        U13: BLOQUEUNO00 port map(
                A => A(4),
                B => B(4),
                G => G(4),
                P => P(4)  );

        U14: BLOQUEDOS00 port map(
                G => G(4),
                P => P(4),
                Cin => C(3),
                Cout => C(4)            );

        U15: BLOQUETRES00 port map(
                P => P(4),
                Cin => C(3),
                S => So(4));


--------------------------------------BIT5
        U16: BLOQUEUNO00 port map(
```

```
            A => A(5),
            B => B(5),
            G => G(5),
            P => P(5)  );

    U17: BLOQUEDOS00 port map(
            G => G(5),
            P => P(5),
            Cin => C(4),
            Cout => C(5)          );

    U18: BLOQUETRES00 port map(
            P => P(5),
            Cin => C(4),
            S => So(5));


--------------------------------------BIT6
    U19: BLOQUEUNO00 port map(
            A => A(6),
            B => B(6),
            G => G(6),
            P => P(6)  );

    U20: BLOQUEDOS00 port map(
            G => G(6),
            P => P(6),
            Cin => C(5),
            Cout => C(6)          );

    U21: BLOQUETRES00 port map(
            P => P(6),
            Cin => C(5),
            S => So(6));


--------------------------------------BIT7
    U22: BLOQUEUNO00 port map(
            A => A(7),
            B => B(7),
            G => G(7),
            P => P(7)  );

    U23: BLOQUEDOS00 port map(
            G => G(7),
            P => P(7),
            Cin => C(6),
            Cout => C(7)          );

    U24: BLOQUETRES00 port map(
            P => P(7),
            Cin => C(6),
            S => So(7));


--------------------------------------BIT8
    U25: BLOQUEUNO00 port map(
            A => A(8),
            B => B(8),
            G => G(8),
            P => P(8)  );

    U26: BLOQUEDOS00 port map(
            G => G(8),
            P => P(8),
            Cin => C(7),
            Cout => C(8)          );

    U27: BLOQUETRES00 port map(
            P => P(8),
            Cin => C(7),
            S => So(8));



--------------------------------------BIT9
    U28: BLOQUEUNO00 port map(
            A => A(9),
            B => B(9),
            G => G(9),
            P => P(9)  );

    U29: BLOQUEDOS00 port map(
            G => G(9),
            P => P(9),
            Cin => C(8),
            Cout => C(9)          );

    U30: BLOQUETRES00 port map(
            P => P(9),
            Cin => C(8),
            S => So(9));


--------------------------------------BIT10
    U31: BLOQUEUNO00 port map(
            A => A(10),
            B => B(10),
            G => G(10),
            P => P(10));

    U32: BLOQUEDOS00 port map(
            G => G(10),
            P => P(10),
            Cin => C(9),
            Cout => C(10)         );

    U33: BLOQUETRES00 port map(
            P => P(10),
            Cin => C(9),
            S => So(10)           );



--------------------------------------BIT11
    U34: BLOQUEUNO00 port map(
            A => A(11),
            B => B(11),
            G => G(11),
            P => P(11));

    U35: BLOQUEDOS00 port map(
            G => G(11),
            P => P(11),
            Cin => C(10),
            Cout => C(11)         );

    U36: BLOQUETRES00 port map(
            P => P(11),
            Cin => C(10),
            S => So(11)           );



--------------------------------------BIT12
    U37: BLOQUEUNO00 port map(
            A => A(12),
            B => B(12),
            G => G(12),
            P => P(12));

    U38: BLOQUEDOS00 port map(
            G => G(12),
            P => P(12),
```

```
		Cin => C(11),
		Cout => C(12)		);

	U39: BLOQUETRES00 port map(
		P => P(12),
		Cin => C(11),
		S => So(12)		);

--------------------------------------BIT13
	U40: BLOQUEUNO00 port map(
		A => A(13),
		B => B(13),
		G => G(13),
		P => P(13));

	U41: BLOQUEDOS00 port map(
		G => G(13),
		P => P(13),
		Cin => C(12),
		Cout => C(13)		);

	U42: BLOQUETRES00 port map(
		P => P(13),
		Cin => C(12),
		S => So(13)		);


--------------------------------------BIT14
	U43: BLOQUEUNO00 port map(
		A => A(14),
		B => B(14),
		G => G(14),
		P => P(14));

	U44: BLOQUEDOS00 port map(
		G => G(14),
		P => P(14),
		Cin => C(13),
		Cout => C(14)		);

	U45: BLOQUETRES00 port map(
		P => P(14),
		Cin => C(13),
		S => So(14)		);


--------------------------------------BIT15
	U46: BLOQUEUNO00 port map(
		A => A(15),
		B => B(15),
		G => G(15),
		P => P(15));

	U47: BLOQUEDOS00 port map(
		G => G(15),
		P => P(15),
		Cin => C(14),
		Cout => C(15)		);

	U48: BLOQUETRES00 port map(
		P => P(15),
		Cin => C(14),
		S => So(15)		);


--------------------------------------BIT16
	U49: BLOQUEUNO00 port map(
		A => A(16),
		B => B(16),
		G => G(16),
		P => P(16));

	U50: BLOQUEDOS00 port map(
		G => G(16),
		P => P(16),
		Cin => C(15),
		Cout => C(16)		);

	U51: BLOQUETRES00 port map(
		P => P(16),
		Cin => C(15),
		S => So(16)		);


--------------------------------------BIT17
	U52: BLOQUEUNO00 port map(
		A => A(17),
		B => B(17),
		G => G(17),
		P => P(17));

	U53: BLOQUEDOS00 port map(
		G => G(17),
		P => P(17),
		Cin => C(16),
		Cout => C(17)		);

	U54: BLOQUETRES00 port map(
		P => P(17),
		Cin => C(16),
		S => So(17)		);



--------------------------------------BIT18
	U55: BLOQUEUNO00 port map(
		A => A(18),
		B => B(18),
		G => G(18),
		P => P(18));

	U56: BLOQUEDOS00 port map(
		G => G(18),
		P => P(18),
		Cin => C(17),
		Cout => C(18)		);

	U57: BLOQUETRES00 port map(
		P => P(18),
		Cin => C(17),
		S => So(18)		);

--------------------------------------BIT19
	U58: BLOQUEUNO00 port map(
		A => A(19),
		B => B(19),
		G => G(19),
		P => P(19));

	U59: BLOQUEDOS00 port map(
		G => G(19),
		P => P(19),
		Cin => C(18),
		Cout => C(19)		);

	U60: BLOQUETRES00 port map(
		P => P(19),
		Cin => C(18),
		S => So(19)		);


--------------------------------------BIT20
	U61: BLOQUEUNO00 port map(
```

```
                A => A(20),
                B => B(20),
                G => G(20),
                P => P(20));

        U62: BLOQUEDOS00 port map(
                G => G(20),
                P => P(20),
                Cin => C(19),
                Cout => C(20)           );

        U63: BLOQUETRES00 port map(
                P => P(20),
                Cin => C(19),
                S => So(20)             );

--------------------------------------BIT21
        U64: BLOQUEUNO00 port map(
                A => A(21),
                B => B(21),
                G => G(21),
                P => P(21));

        U65: BLOQUEDOS00 port map(
                G => G(21),
                P => P(21),
                Cin => C(20),
                Cout => C(21)           );

        U66: BLOQUETRES00 port map(
                P => P(21),
                Cin => C(20),
                S => So(21)             );

--------------------------------------BIT22
        U67: BLOQUEUNO00 port map(
                A => A(22),
                B => B(22),
                G => G(22),
                P => P(22));

        U68: BLOQUEDOS00 port map(
                G => G(22),
                P => P(22),
                Cin => C(21),
                Cout => C(22)           );

        U69: BLOQUETRES00 port map(
                P => P(22),
                Cin => C(21),
                S => So(22)             );


SAL <= So;

 S <= So(22);


 Z <= '1' when (So = "00000000000000000000000") else '0';
 AC <= C(22);
 O <= (C(22) XOR C(21));

end SUM_RES0;
```

23. Multiplicador

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

package packtotal00 is


component FCA00

port(
            A: in
std_logic_vector ( 22 downto 0
);
            Bb: in
std_logic_vector ( 22 downto 0
);
            Cin: in std_logic ;
            So: out
std_logic_vector ( 22 downto 0
);

            AC: out std_logic
  );

end component;

end packtotal00;


---------------------------------------
--


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;
use packtotal00.all;

entity MULT00 is

port(
            A: in
std_logic_vector ( 22 downto 0
);
            B: in
std_logic_vector ( 22 downto 0
);
            M: out
std_logic_vector ( 45 downto 0
)

);


end;

architecture MULT0 of
MULT00 is

signal Y1, Y2,    Y3, Y4,    Y5,
Y6,   Y7, Y8,   Y9, Y10, Y100,
Y11, Y12,    Y13, Y14,    Y15,
Y16,   Y17, Y18,   Y19, Y20,
Y21, Y22,    Y23, Y24,    Y25,
Y26,   Y27, Y28,   Y29, Y30,
Y31, Y32,    Y33, Y34,    Y35,
Y36,   Y37, Y38,   Y39, Y40,
Y41, Y42, Y43 :
std_logic_vector(22 downto 0);
signal Z1, Z2, Z3, Z4, ZZ5, Z6,
Z7, Z8, Z9, Z10, Z11, Z12,
Z13, Z14, Z15, Z16, Z17, Z18,
Z19, Z20, Z21 :
std_logic_vector(22 downto 0);

signal C1, C2, c3, c4, c5, c6,
c7, c8, c9, C10, C11, C12,
C13, C14, C15, C16, C17,
C18, C19, C20, C21  :
std_logic;
begin

M(0) <= (B(0) and A(0));

Y1(0) <= (B(1) and A(0));
Y1(1) <= (B(2) and A(0));
Y1(2) <= (B(3) and A(0));
Y1(3) <= (B(4) and A(0));
Y1(4) <= (B(5) and A(0));

Y1(5) <= (B(6) and A(0));
Y1(6) <= (B(7) and A(0));
Y1(7) <= (B(8) and A(0));
Y1(8) <= (B(9) and A(0));
Y1(9) <= (B(10) and A(0));

Y1(10) <= (B(11) and A(0));
Y1(11) <= (B(12) and A(0));
Y1(12) <= (B(13) and A(0));
Y1(13) <= (B(14) and A(0));
Y1(14) <= (B(15) and A(0));

Y1(15) <= (B(16) and A(0));
Y1(16) <= (B(17) and A(0));
Y1(17) <= (B(18) and A(0));
Y1(18) <= (B(19) and A(0));
Y1(19) <= (B(20) and A(0));

Y1(20) <= (B(21) and A(0));
Y1(21) <= (B(22) and A(0));
Y1(22) <= '0';
-------------------------

Y2(0) <= (B(0) and A(1));
Y2(1) <= (B(1) and A(1));
Y2(2) <= (B(2) and A(1));
Y2(3) <= (B(3) and A(1));
Y2(4) <= (B(4) and A(1));

Y2(5) <= (B(5) and A(1));
Y2(6) <= (B(6) and A(1));
Y2(7) <= (B(7) and A(1));
Y2(8) <= (B(8) and A(1));
Y2(9) <= (B(9) and A(1));

Y2(10) <= (B(10) and A(1));
Y2(11) <= (B(11) and A(1));
Y2(12) <= (B(12) and A(1));
Y2(13) <= (B(13) and A(1));
Y2(14) <= (B(14) and A(1));

Y2(15) <= (B(15) and A(1));
Y2(16) <= (B(16) and A(1));
Y2(17) <= (B(17) and A(1));
Y2(18) <= (B(18) and A(1));
Y2(19) <= (B(19) and A(1));

Y2(20) <= (B(20) and A(1));
Y2(21) <= (B(21) and A(1));
Y2(22) <= (B(22) and A(1));
---------------------------------


U100: FCA00 port map(

            A => Y1,
            Bb => Y2,
            Cin => '0',
            So => Z1,

            AC => C1

);

M(1) <= Z1(0);

--------------------------------


Y3 <= (C1 & Z1(22
DOWNTO 1));

Y4(0) <= (B(0) and A(2));
Y4(1) <= (B(1) and A(2));
Y4(2) <= (B(2) and A(2));
Y4(3) <= (B(3) and A(2));
Y4(4) <= (B(4) and A(2));

Y4(5) <= (B(5) and A(2));
Y4(6) <= (B(6) and A(2));
Y4(7) <= (B(7) and A(2));
Y4(8) <= (B(8) and A(2));
Y4(9) <= (B(9) and A(2));

Y4(10) <= (B(10) and A(2));
Y4(11) <= (B(11) and A(2));
Y4(12) <= (B(12) and A(2));
Y4(13) <= (B(13) and A(2));
Y4(14) <= (B(14) and A(2));

Y4(15) <= (B(15) and A(2));
Y4(16) <= (B(16) and A(2));
Y4(17) <= (B(17) and A(2));
Y4(18) <= (B(18) and A(2));
Y4(19) <= (B(19) and A(2));

Y4(20) <= (B(20) and A(2));
Y4(21) <= (B(21) and A(2));
Y4(22) <= (B(22) and A(2));

---------------------------



U101: FCA00 port map(

            A => Y3,
            Bb => Y4,
            Cin => '0',
            So => Z2,

            AC => C2

);


M(2) <= Z2(0);

-----------------------------

Y5 <= (C2 & Z2(22
DOWNTO 1));

Y6(0) <= (B(0) and A(3));
Y6(1) <= (B(1) and A(3));
Y6(2) <= (B(2) and A(3));
Y6(3) <= (B(3) and A(3));
Y6(4) <= (B(4) and A(3));

Y6(5) <= (B(5) and A(3));
Y6(6) <= (B(6) and A(3));
Y6(7) <= (B(7) and A(3));
Y6(8) <= (B(8) and A(3));
Y6(9) <= (B(9) and A(3));

Y6(10) <= (B(10) and A(3));
Y6(11) <= (B(11) and A(3));
Y6(12) <= (B(12) and A(3));
Y6(13) <= (B(13) and A(3));
Y6(14) <= (B(14) and A(3));

Y6(15) <= (B(15) and A(3));
Y6(16) <= (B(16) and A(3));
Y6(17) <= (B(17) and A(3));
Y6(18) <= (B(18) and A(3));
Y6(19) <= (B(19) and A(3));

Y6(20) <= (B(20) and A(3));
Y6(21) <= (B(21) and A(3));
Y6(22) <= (B(22) and A(3));

-----------------------------


U102: FCA00 port map(

            A => Y5,
            Bb => Y6,
            Cin => '0',
            So => Z3,

            AC => C3

);


M(3) <= Z3(0);

-----------------------------

Y7 <= (C3 & Z3(22
DOWNTO 1));

Y8(0) <= (B(0) and A(4));
Y8(1) <= (B(1) and A(4));
```

```
Y8(2) <= (B(2) and A(4));
Y8(3) <= (B(3) and A(4));
Y8(4) <= (B(4) and A(4));

Y8(5) <= (B(5) and A(4));
Y8(6) <= (B(6) and A(4));
Y8(7) <= (B(7) and A(4));
Y8(8) <= (B(8) and A(4));
Y8(9) <= (B(9) and A(4));

Y8(10) <= (B(10) and A(4));
Y8(11) <= (B(11) and A(4));
Y8(12) <= (B(12) and A(4));
Y8(13) <= (B(13) and A(4));
Y8(14) <= (B(14) and A(4));

Y8(15) <= (B(15) and A(4));
Y8(16) <= (B(16) and A(4));
Y8(17) <= (B(17) and A(4));
Y8(18) <= (B(18) and A(4));
Y8(19) <= (B(19) and A(4));

Y8(20) <= (B(20) and A(4));
Y8(21) <= (B(21) and A(4));
Y8(22) <= (B(22) and A(4));

-----------------------------

U103: FCA00 port map(

        A => Y7,
        Bb => Y8,
        Cin => '0',
        So => Z4,

        AC => C4

);


M(4) <= Z4(0);

-------------------------------



Y9 <= (C4 & Z4(22
DOWNTO 1));

Y10(0) <= (B(0) and A(5));
Y10(1) <= (B(1) and A(5));
Y10(2) <= (B(2) and A(5));
Y10(3) <= (B(3) and A(5));
Y10(4) <= (B(4) and A(5));

Y10(5) <= (B(5) and A(5));
Y10(6) <= (B(6) and A(5));
Y10(7) <= (B(7) and A(5));
Y10(8) <= (B(8) and A(5));
Y10(9) <= (B(9) and A(5));

Y10(10) <= (B(10) and A(5));
Y10(11) <= (B(11) and A(5));
Y10(12) <= (B(12) and A(5));
Y10(13) <= (B(13) and A(5));
Y10(14) <= (B(14) and A(5));

Y10(15) <= (B(15) and A(5));
Y10(16) <= (B(16) and A(5));
Y10(17) <= (B(17) and A(5));
Y10(18) <= (B(18) and A(5));
Y10(19) <= (B(19) and A(5));

Y10(20) <= (B(20) and A(5));
Y10(21) <= (B(21) and A(5));
Y10(22) <= (B(22) and A(5));

-----------------------------

U104: FCA00 port map(

        A => Y9,
        Bb => Y10,
        Cin => '0',
        So => ZZ5,

        AC => C5

);


M(5) <= ZZ5(0);

-------------------------------



Y100 <= (C5 & ZZ5(22
DOWNTO 1));

Y11(0) <= (B(0) and A(6));
Y11(1) <= (B(1) and A(6));
Y11(2) <= (B(2) and A(6));
Y11(3) <= (B(3) and A(6));
Y11(4) <= (B(4) and A(6));

Y11(5) <= (B(5) and A(6));
Y11(6) <= (B(6) and A(6));
Y11(7) <= (B(7) and A(6));
Y11(8) <= (B(8) and A(6));
Y11(9) <= (B(9) and A(6));

Y11(10) <= (B(10) and A(6));
Y11(11) <= (B(11) and A(6));
Y11(12) <= (B(12) and A(6));
Y11(13) <= (B(13) and A(6));
Y11(14) <= (B(14) and A(6));

Y11(15) <= (B(15) and A(6));
Y11(16) <= (B(16) and A(6));
Y11(17) <= (B(17) and A(6));
Y11(18) <= (B(18) and A(6));
Y11(19) <= (B(19) and A(6));

Y11(20) <= (B(20) and A(6));
Y11(21) <= (B(21) and A(6));
Y11(22) <= (B(22) and A(6));

-----------------------------

U105: FCA00 port map(

        A => Y100,
        Bb => Y11,
        Cin => '0',
        So => Z6,

        AC => C6

);


M(6) <= Z6(0);

----------------------------


Y12 <= (C6 & Z6(22
DOWNTO 1));

Y13(0) <= (B(0) and A(7));
Y13(1) <= (B(1) and A(7));
Y13(2) <= (B(2) and A(7));
Y13(3) <= (B(3) and A(7));
Y13(4) <= (B(4) and A(7));

Y13(5) <= (B(5) and A(7));
Y13(6) <= (B(6) and A(7));
Y13(7) <= (B(7) and A(7));
Y13(8) <= (B(8) and A(7));
Y13(9) <= (B(9) and A(7));
Y13(10) <= (B(10) and A(7));
Y13(11) <= (B(11) and A(7));
Y13(12) <= (B(12) and A(7));
Y13(13) <= (B(13) and A(7));
Y13(14) <= (B(14) and A(7));

Y13(15) <= (B(15) and A(7));
Y13(16) <= (B(16) and A(7));
Y13(17) <= (B(17) and A(7));
Y13(18) <= (B(18) and A(7));
Y13(19) <= (B(19) and A(7));

Y13(20) <= (B(20) and A(7));
Y13(21) <= (B(21) and A(7));
Y13(22) <= (B(22) and A(7));

-----------------------------

U106: FCA00 port map(

        A => Y12,
        Bb => Y13,
        Cin => '0',
        So => Z7,

        AC => C7

);


M(7) <= Z7(0);

----------------------------


Y14 <= (C7 & Z7(22
DOWNTO 1));

Y15(0) <= (B(0) and A(8));
Y15(1) <= (B(1) and A(8));
Y15(2) <= (B(2) and A(8));
Y15(3) <= (B(3) and A(8));
Y15(4) <= (B(4) and A(8));

Y15(5) <= (B(5) and A(8));
Y15(6) <= (B(6) and A(8));
Y15(7) <= (B(7) and A(8));
Y15(8) <= (B(8) and A(8));
Y15(9) <= (B(9) and A(8));

Y15(10) <= (B(10) and A(8));
Y15(11) <= (B(11) and A(8));
Y15(12) <= (B(12) and A(8));
Y15(13) <= (B(13) and A(8));
Y15(14) <= (B(14) and A(8));

Y15(15) <= (B(15) and A(8));
Y15(16) <= (B(16) and A(8));
Y15(17) <= (B(17) and A(8));
Y15(18) <= (B(18) and A(8));
Y15(19) <= (B(19) and A(8));

Y15(20) <= (B(20) and A(8));
Y15(21) <= (B(21) and A(8));
Y15(22) <= (B(22) and A(8));

-----------------------------

U107: FCA00 port map(

        A => Y14,
        Bb => Y15,
        Cin => '0',
        So => Z8,

        AC => C8

);


M(8) <= Z8(0);

----------------------------


Y16 <= (C8 & Z8(22
DOWNTO 1));

Y17(0) <= (B(0) and A(9));
Y17(1) <= (B(1) and A(9));
Y17(2) <= (B(2) and A(9));
Y17(3) <= (B(3) and A(9));
Y17(4) <= (B(4) and A(9));

Y17(5) <= (B(5) and A(9));
Y17(6) <= (B(6) and A(9));
Y17(7) <= (B(7) and A(9));
Y17(8) <= (B(8) and A(9));
Y17(9) <= (B(9) and A(9));

Y17(10) <= (B(10) and A(9));
Y17(11) <= (B(11) and A(9));
Y17(12) <= (B(12) and A(9));
Y17(13) <= (B(13) and A(9));
Y17(14) <= (B(14) and A(9));

Y17(15) <= (B(15) and A(9));
Y17(16) <= (B(16) and A(9));
```

```
Y17(17) <= (B(17) and A(9));
Y17(18) <= (B(18) and A(9));
Y17(19) <= (B(19) and A(9));

Y17(20) <= (B(20) and A(9));
Y17(21) <= (B(21) and A(9));
Y17(22) <= (B(22) and A(9));

-----------------------------

U108: FCA00 port map(

        A => Y16,
        Bb => Y17,
        Cin => '0',
        So => Z9,

        AC => C9

);

M(9) <= Z9(0);

----------------------------

Y18 <= (C9 & Z9(22
DOWNTO 1));

Y19(0) <= (B(0) and A(10));
Y19(1) <= (B(1) and A(10));
Y19(2) <= (B(2) and A(10));
Y19(3) <= (B(3) and A(10));
Y19(4) <= (B(4) and A(10));

Y19(5) <= (B(5) and A(10));
Y19(6) <= (B(6) and A(10));
Y19(7) <= (B(7) and A(10));
Y19(8) <= (B(8) and A(10));
Y19(9) <= (B(9) and A(10));

Y19(10) <= (B(10) and A(10));
Y19(11) <= (B(11) and A(10));
Y19(12) <= (B(12) and A(10));
Y19(13) <= (B(13) and A(10));
Y19(14) <= (B(14) and A(10));

Y19(15) <= (B(15) and A(10));
Y19(16) <= (B(16) and A(10));
Y19(17) <= (B(17) and A(10));
Y19(18) <= (B(18) and A(10));
Y19(19) <= (B(19) and A(10));
Y19(20) <= (B(20) and A(10));
Y19(21) <= (B(21) and A(10));
Y19(22) <= (B(22) and A(10));

-----------------------------

U109: FCA00 port map(

        A => Y18,
        Bb => Y19,
        Cin => '0',
        So => Z10,

        AC => C10

);

M(10) <= Z10(0);

----------------------------

Y20 <= (C10 & Z10(22
DOWNTO 1));

Y21(0) <= (B(0) and A(11));
Y21(1) <= (B(1) and A(11));
Y21(2) <= (B(2) and A(11));
Y21(3) <= (B(3) and A(11));
Y21(4) <= (B(4) and A(11));

Y21(5) <= (B(5) and A(11));
Y21(6) <= (B(6) and A(11));
Y21(7) <= (B(7) and A(11));
Y21(8) <= (B(8) and A(11));
Y21(9) <= (B(9) and A(11));

Y21(10) <= (B(10) and A(11));
Y21(11) <= (B(11) and A(11));
Y21(12) <= (B(12) and A(11));
Y21(13) <= (B(13) and A(11));
Y21(14) <= (B(14) and A(11));

Y21(15) <= (B(15) and A(11));
Y21(16) <= (B(16) and A(11));
Y21(17) <= (B(17) and A(11));
Y21(18) <= (B(18) and A(11));
Y21(19) <= (B(19) and A(11));

Y21(20) <= (B(20) and A(11));
Y21(21) <= (B(21) and A(11));
Y21(22) <= (B(22) and A(11));

-----------------------------

U110: FCA00 port map(

        A => Y20,
        Bb => Y21,
        Cin => '0',
        So => Z11,

        AC => C11

);

M(11) <= Z11(0);

----------------------------

Y22 <= (C11 & Z11(22
DOWNTO 1));

Y23(0) <= (B(0) and A(12));
Y23(1) <= (B(1) and A(12));
Y23(2) <= (B(2) and A(12));
Y23(3) <= (B(3) and A(12));
Y23(4) <= (B(4) and A(12));

Y23(5) <= (B(5) and A(12));
Y23(6) <= (B(6) and A(12));
Y23(7) <= (B(7) and A(12));
Y23(8) <= (B(8) and A(12));
Y23(9) <= (B(9) and A(12));

Y23(10) <= (B(10) and A(12));
Y23(11) <= (B(11) and A(12));
Y23(12) <= (B(12) and A(12));
Y23(13) <= (B(13) and A(12));
Y23(14) <= (B(14) and A(12));

Y23(15) <= (B(15) and A(12));
Y23(16) <= (B(16) and A(12));
Y23(17) <= (B(17) and A(12));
Y23(18) <= (B(18) and A(12));
Y23(19) <= (B(19) and A(12));

Y23(20) <= (B(20) and A(12));
Y23(21) <= (B(21) and A(12));
Y23(22) <= (B(22) and A(12));

-----------------------------

U111: FCA00 port map(

        A => Y22,
        Bb => Y23,
        Cin => '0',
        So => Z12,

        AC => C12

);

M(12) <= Z12(0);

----------------------------

Y24 <= (C12 & Z12(22
DOWNTO 1));

Y25(0) <= (B(0) and A(13));
Y25(1) <= (B(1) and A(13));
Y25(2) <= (B(2) and A(13));
Y25(3) <= (B(3) and A(13));
Y25(4) <= (B(4) and A(13));

Y25(5) <= (B(5) and A(13));
Y25(6) <= (B(6) and A(13));
Y25(7) <= (B(7) and A(13));
Y25(8) <= (B(8) and A(13));
Y25(9) <= (B(9) and A(13));

Y25(10) <= (B(10) and A(13));
Y25(11) <= (B(11) and A(13));
Y25(12) <= (B(12) and A(13));
Y25(13) <= (B(13) and A(13));
Y25(14) <= (B(14) and A(13));

Y25(15) <= (B(15) and A(13));
Y25(16) <= (B(16) and A(13));
Y25(17) <= (B(17) and A(13));
Y25(18) <= (B(18) and A(13));
Y25(19) <= (B(19) and A(13));

Y25(20) <= (B(20) and A(13));
Y25(21) <= (B(21) and A(13));
Y25(22) <= (B(22) and A(13));

-----------------------------

U112: FCA00 port map(

        A => Y24,
        Bb => Y25,
        Cin => '0',
        So => Z13,

        AC => C13

);

M(13) <= Z13(0);

----------------------------

Y26 <= (C13 & Z13(22
DOWNTO 1));

Y27(0) <= (B(0) and A(14));
Y27(1) <= (B(1) and A(14));
Y27(2) <= (B(2) and A(14));
Y27(3) <= (B(3) and A(14));
Y27(4) <= (B(4) and A(14));

Y27(5) <= (B(5) and A(14));
Y27(6) <= (B(6) and A(14));
Y27(7) <= (B(7) and A(14));
Y27(8) <= (B(8) and A(14));
Y27(9) <= (B(9) and A(14));

Y27(10) <= (B(10) and A(14));
Y27(11) <= (B(11) and A(14));
Y27(12) <= (B(12) and A(14));
Y27(13) <= (B(13) and A(14));
Y27(14) <= (B(14) and A(14));

Y27(15) <= (B(15) and A(14));
Y27(16) <= (B(16) and A(14));
Y27(17) <= (B(17) and A(14));
Y27(18) <= (B(18) and A(14));
Y27(19) <= (B(19) and A(14));

Y27(20) <= (B(20) and A(14));
Y27(21) <= (B(21) and A(14));
Y27(22) <= (B(22) and A(14));

-----------------------------

U113: FCA00 port map(

        A => Y26,
        Bb => Y27,
        Cin => '0',
        So => Z14,

        AC => C14

);
```

```
M(14) <= Z14(0);

----------------------------

Y28 <= (C14 & Z14(22
DOWNTO 1));

Y29(0) <= (B(0) and A(15));
Y29(1) <= (B(1) and A(15));
Y29(2) <= (B(2) and A(15));
Y29(3) <= (B(3) and A(15));
Y29(4) <= (B(4) and A(15));

Y29(5) <= (B(5) and A(15));
Y29(6) <= (B(6) and A(15));
Y29(7) <= (B(7) and A(15));
Y29(8) <= (B(8) and A(15));
Y29(9) <= (B(9) and A(15));

Y29(10) <= (B(10) and A(15));
Y29(11) <= (B(11) and A(15));
Y29(12) <= (B(12) and A(15));
Y29(13) <= (B(13) and A(15));
Y29(14) <= (B(14) and A(15));

Y29(15) <= (B(15) and A(15));
Y29(16) <= (B(16) and A(15));
Y29(17) <= (B(17) and A(15));
Y29(18) <= (B(18) and A(15));
Y29(19) <= (B(19) and A(15));

Y29(20) <= (B(20) and A(15));
Y29(21) <= (B(21) and A(15));
Y29(22) <= (B(22) and A(15));

-----------------------------

U114: FCA00 port map(

        A => Y28,
        Bb => Y29,
        Cin => '0',
        So => Z15,

        AC => C15

);

M(15) <= Z15(0);

----------------------------

Y30 <= (C15 & Z15(22
DOWNTO 1));

Y31(0) <= (B(0) and A(16));
Y31(1) <= (B(1) and A(16));
Y31(2) <= (B(2) and A(16));
Y31(3) <= (B(3) and A(16));
Y31(4) <= (B(4) and A(16));

Y31(5) <= (B(5) and A(16));
Y31(6) <= (B(6) and A(16));
Y31(7) <= (B(7) and A(16));
Y31(8) <= (B(8) and A(16));
Y31(9) <= (B(9) and A(16));

Y31(10) <= (B(10) and A(16));
Y31(11) <= (B(11) and A(16));
Y31(12) <= (B(12) and A(16));
Y31(13) <= (B(13) and A(16));
Y31(14) <= (B(14) and A(16));

Y31(15) <= (B(15) and A(16));
Y31(16) <= (B(16) and A(16));
Y31(17) <= (B(17) and A(16));
Y31(18) <= (B(18) and A(16));
Y31(19) <= (B(19) and A(16));

Y31(20) <= (B(20) and A(16));
Y31(21) <= (B(21) and A(16));
Y31(22) <= (B(22) and A(16));

-----------------------------

U115: FCA00 port map(

        A => Y30,
        Bb => Y31,
        Cin => '0',
        So => Z16,

        AC => C16

);

M(16) <= Z16(0);

----------------------------

Y32 <= (C16 & Z16(22
DOWNTO 1));

Y33(0) <= (B(0) and A(17));
Y33(1) <= (B(1) and A(17));
Y33(2) <= (B(2) and A(17));
Y33(3) <= (B(3) and A(17));
Y33(4) <= (B(4) and A(17));

Y33(5) <= (B(5) and A(17));
Y33(6) <= (B(6) and A(17));
Y33(7) <= (B(7) and A(17));
Y33(8) <= (B(8) and A(17));
Y33(9) <= (B(9) and A(17));

Y33(10) <= (B(10) and A(17));
Y33(11) <= (B(11) and A(17));
Y33(12) <= (B(12) and A(17));
Y33(13) <= (B(13) and A(17));
Y33(14) <= (B(14) and A(17));

Y33(15) <= (B(15) and A(17));
Y33(16) <= (B(16) and A(17));
Y33(17) <= (B(17) and A(17));
Y33(18) <= (B(18) and A(17));
Y33(19) <= (B(19) and A(17));

Y33(20) <= (B(20) and A(17));
Y33(21) <= (B(21) and A(17));
Y33(22) <= (B(22) and A(17));

-----------------------------

U116: FCA00 port map(

        A => Y32,
        Bb => Y33,
        Cin => '0',
        So => Z17,

        AC => C17

);

M(17) <= Z17(0);

----------------------------

Y34 <= (C17 & Z17(22
DOWNTO 1));

Y35(0) <= (B(0) and A(18));
Y35(1) <= (B(1) and A(18));
Y35(2) <= (B(2) and A(18));
Y35(3) <= (B(3) and A(18));
Y35(4) <= (B(4) and A(18));

Y35(5) <= (B(5) and A(18));
Y35(6) <= (B(6) and A(18));
Y35(7) <= (B(7) and A(18));
Y35(8) <= (B(8) and A(18));
Y35(9) <= (B(9) and A(18));

Y35(10) <= (B(10) and A(18));
Y35(11) <= (B(11) and A(18));
Y35(12) <= (B(12) and A(18));
Y35(13) <= (B(13) and A(18));
Y35(14) <= (B(14) and A(18));

Y35(15) <= (B(15) and A(18));
Y35(16) <= (B(16) and A(18));
Y35(17) <= (B(17) and A(18));
Y35(18) <= (B(18) and A(18));
Y35(19) <= (B(19) and A(18));

Y35(20) <= (B(20) and A(18));
Y35(21) <= (B(21) and A(18));
Y35(22) <= (B(22) and A(18));

-----------------------------

U117: FCA00 port map(

        A => Y34,
        Bb => Y35,
        Cin => '0',
        So => Z18,

        AC => C18

);

M(18) <= Z18(0);

----------------------------

Y36 <= (C18 & Z18(22
DOWNTO 1));

Y37(0) <= (B(0) and A(19));
Y37(1) <= (B(1) and A(19));
Y37(2) <= (B(2) and A(19));
Y37(3) <= (B(3) and A(19));
Y37(4) <= (B(4) and A(19));

Y37(5) <= (B(5) and A(19));
Y37(6) <= (B(6) and A(19));
Y37(7) <= (B(7) and A(19));
Y37(8) <= (B(8) and A(19));
Y37(9) <= (B(9) and A(19));

Y37(10) <= (B(10) and A(19));
Y37(11) <= (B(11) and A(19));
Y37(12) <= (B(12) and A(19));
Y37(13) <= (B(13) and A(19));
Y37(14) <= (B(14) and A(19));

Y37(15) <= (B(15) and A(19));
Y37(16) <= (B(16) and A(19));
Y37(17) <= (B(17) and A(19));
Y37(18) <= (B(18) and A(19));
Y37(19) <= (B(19) and A(19));

Y37(20) <= (B(20) and A(19));
Y37(21) <= (B(21) and A(19));
Y37(22) <= (B(22) and A(19));

-----------------------------

U118: FCA00 port map(

        A => Y36,
        Bb => Y37,
        Cin => '0',
        So => Z19,

        AC => C19

);

M(19) <= Z19(0);

----------------------------

Y38 <= (C19 & Z19(22
DOWNTO 1));

Y39(0) <= (B(0) and A(20));
Y39(1) <= (B(1) and A(20));
Y39(2) <= (B(2) and A(20));
Y39(3) <= (B(3) and A(20));
Y39(4) <= (B(4) and A(20));

Y39(5) <= (B(5) and A(20));
Y39(6) <= (B(6) and A(20));
Y39(7) <= (B(7) and A(20));
Y39(8) <= (B(8) and A(20));
Y39(9) <= (B(9) and A(20));
```

```
Y39(10) <= (B(10) and A(20));
Y39(11) <= (B(11) and A(20));
Y39(12) <= (B(12) and A(20));
Y39(13) <= (B(13) and A(20));
Y39(14) <= (B(14) and A(20));

Y39(15) <= (B(15) and A(20));
Y39(16) <= (B(16) and A(20));
Y39(17) <= (B(17) and A(20));
Y39(18) <= (B(18) and A(20));
Y39(19) <= (B(19) and A(20));

Y39(20) <= (B(20) and A(20));
Y39(21) <= (B(21) and A(20));
Y39(22) <= (B(22) and A(20));

-----------------------------

U119: FCA00 port map(

        A => Y38,
        Bb => Y39,
        Cin => '0',
        So => Z20,

        AC => C20

);


M(20) <= Z20(0);

----------------------------



Y40 <= (C20 & Z20(22
DOWNTO 1));

Y41(0) <= (B(0) and A(21));
Y41(1) <= (B(1) and A(21));
Y41(2) <= (B(2) and A(21));
Y41(3) <= (B(3) and A(21));
Y41(4) <= (B(4) and A(21));

Y41(5) <= (B(5) and A(21));
Y41(6) <= (B(6) and A(21));
Y41(7) <= (B(7) and A(21));
Y41(8) <= (B(8) and A(21));
Y41(9) <= (B(9) and A(21));

Y41(10) <= (B(10) and A(21));
Y41(11) <= (B(11) and A(21));
Y41(12) <= (B(12) and A(21));
Y41(13) <= (B(13) and A(21));
Y41(14) <= (B(14) and A(21));

Y41(15) <= (B(15) and A(21));
Y41(16) <= (B(16) and A(21));
Y41(17) <= (B(17) and A(21));
Y41(18) <= (B(18) and A(21));
Y41(19) <= (B(19) and A(21));

Y41(20) <= (B(20) and A(21));
Y41(21) <= (B(21) and A(21));
Y41(22) <= (B(22) and A(21));

-----------------------------

U120: FCA00 port map(

        A => Y40,
        Bb => Y41,
        Cin => '0',
        So => Z21,

        AC => C21

);


M(21) <= Z21(0);

----------------------------


Y42 <= (C21 & Z21(22
DOWNTO 1));

Y43(0) <= (B(0) and A(22));
Y43(1) <= (B(1) and A(22));
Y43(2) <= (B(2) and A(22));
Y43(3) <= (B(3) and A(22));
Y43(4) <= (B(4) and A(22));

Y43(5) <= (B(5) and A(22));
Y43(6) <= (B(6) and A(22));
Y43(7) <= (B(7) and A(22));
Y43(8) <= (B(8) and A(22));
Y43(9) <= (B(9) and A(22));

Y43(10) <= (B(10) and A(22));
Y43(11) <= (B(11) and A(22));
Y43(12) <= (B(12) and A(22));
Y43(13) <= (B(13) and A(22));
Y43(14) <= (B(14) and A(22));

Y43(15) <= (B(15) and A(22));
Y43(16) <= (B(16) and A(22));
Y43(17) <= (B(17) and A(22));
Y43(18) <= (B(18) and A(22));
Y43(19) <= (B(19) and A(22));

Y43(20) <= (B(20) and A(22));
Y43(21) <= (B(21) and A(22));
Y43(22) <= (B(22) and A(22));

-----------------------------

U15: FCA00 port map(

        A => Y42,
        Bb => Y43,
        Cin => '0',
        So => M(44
DOWNTO 22),

        AC => M(45)

);


end MULT0;
```

24. Divisor

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

entity AND00 is

port(
        Aa: in std_logic ;
        Ba: in std_logic ;
        Ya: out std_logic  );

end;

architecture AND0 of AND00
is
begin
Ya <= Aa AND Ba;
end AND0;

-----------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

entity XOR00 is
port(
        Ax: in std_logic ;
        Bx: in std_logic ;
        Yx: out std_logic  );

end;

architecture XOR0 of XOR00
is
begin

Yx <= Ax XOR Bx;

end XOR0;

----------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

package packageha00 is

component xor00

port(
        Ax: in std_logic ;
        Bx: in std_logic ;
        Yx: out std_logic  );

end component;

component and00

port(
        Aa: in std_logic ;
        Ba: in std_logic ;
        Ya: out std_logic  );

end component;

end packageha00;

-------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;
use packageha00.all;

entity HA00 is

port(
        A0: in std_logic ;
        B0: in std_logic ;
        S0: out std_logic ;
        C0: out std_logic  );
end;

architecture HA0 of HA00 is
begin

  U1: and00 port map(Aa =>
A0, Ba => B0, Ya => C0);
  U2: xor00 port map(Ax =>
A0, Bx => B0, Yx => S0);


end HA0;

--------------------------------------
------


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

entity or00 is

port(
        Ao: in std_logic ;
        Bo: in std_logic ;
        Yo: out std_logic  );

end;

architecture or00 of or00 is
begin
  Yo <= Ao OR Bo;
end or00;

------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

package packagefa00 is

component ha00

port(
        A0: in std_logic ;
        B0: in std_logic ;
        S0: out std_logic ;
        C0: out std_logic  );

end component;

component or00

port(
        Ao: in std_logic ;
        Bo: in std_logic ;
        Yo: out std_logic  );

end component;

end packagefa00;

------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;
use packagefa00.all;

entity fa00 is

port(
        C00: in std_logic ;
        A00: in std_logic ;
        B00: in std_logic ;
        S00: out std_logic ;
        C01: out std_logic
);

end;

architecture fa00 of fa00 is
signal Sint1, Cint1, Cint2:
std_logic;
begin

  U3: ha00 port map(  A0 =>
A00,

        B0 => B00,

        S0 => Sint1,

        C0 => Cint1);

  U4: ha00 port map(  A0 =>
C00,

        B0 => Sint1,

        S0 => S00,

        C0 => Cint2);

  U5: or00 port map(  Ao =>
Cint2,

        Bo => Cint1,
```

```
        Yo => C01);

end fa00;

--------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

package paquete_cas is

component XOR00
port(
        Ax: in std_logic ;
        Bx: in std_logic ;
        Yx: out std_logic  );
end component;

component fa00
port(
        C00: in std_logic ;
        A00: in std_logic ;
        B00: in std_logic ;
        S00: out std_logic ;
        C01: out std_logic
);

end component;

end paquete_cas;

-----------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;
use paquete_cas.all;

entity cas00 is

port(
        uno: in std_logic ;
        q_in: in std_logic ;
        p_in: in std_logic ;
        cin: in std_logic ;

        s_uno: out std_logic
;
        s_qin: out std_logic
;
        s_cas: out std_logic
;
        cout: out std_logic
);

end;

architecture cas0 of cas00 is
signal s_q, s_u, s_xo, s_cin,
sp_in, ss_cas, s_cout:
std_logic;
begin

s_u <= uno;
s_q <= q_in;
s_cin <= cin;
sp_in <= p_in;


  U6: xor00 port map(

        Ax => s_u,
        Bx => s_q,
        Yx => s_xo

  );

  U7: fa00 port map(

        C00 => s_cin,
        A00 => s_xo,
        B00 => sp_in,
        S00 => ss_cas,
        C01 => s_cout

  );

s_uno <= s_u;
s_qin <= s_q;
s_cas <= ss_cas;
cout <= s_cout;

end cas0;

---------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

package paquete_mcas00 is

component cas00

port(

        uno: in std_logic ;
        q_in: in std_logic ;
        p_in: in std_logic ;
        cin: in std_logic ;

        s_uno: out std_logic
;
        s_qin: out std_logic
;
        s_cas: out std_logic
;
        cout: out std_logic

);

end component;

end paquete_mcas00;

--------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;
use paquete_mcas00.all;

entity mod_cas00 is

port(
        M: in
std_logic_vector(22 downto 0)
;
        N: in
std_logic_vector(22 downto 0)
;
        m_uno: in std_logic
;

        S: inout
std_logic_vector(22 downto 0)
;
        R: inout
std_logic_vector(22 downto 0)
;
        m_cout: out
std_logic   );

end;

architecture mod_cas0 of
mod_cas00 is
signal RR, SS, suno, cout:
std_logic_vector(22 downto 0);
begin


  U8: cas00 port map(
        uno => m_uno,
        q_in => M(22),
        p_in => N(22),
        cin => cout (21),

        s_uno => suno(22) ,
        s_qin => RR(22),
        s_cas => SS(22),
        cout => cout(22)

  );

  U9: cas00 port map(
        uno => suno(22),
        q_in => M(21),
        p_in => N(21),
        cin => cout (20),

        s_uno => suno(21) ,
        s_qin => RR(21),
        s_cas => SS(21),
        cout => cout(21)

  );

  U10: cas00 port map(

        uno => suno(21),
        q_in => M(20),
        p_in => N(20),
        cin => cout(19),

        s_uno => suno(20) ,
        s_qin => RR(20),
        s_cas => SS(20),
        cout => cout(20)

  );


  U11: cas00 port map(

        uno => suno(20),
        q_in => M(19),
        p_in => N(19),
        cin => cout(18),

        s_uno => suno(19) ,
        s_qin => RR(19),
        s_cas => SS(19),
        cout => cout(19)

);

  U12: cas00 port map(

        uno => suno(19),
        q_in => M(18),
        p_in => N(18),
        cin => cout(17),

        s_uno => suno(18) ,
        s_qin => RR(18),
        s_cas => SS(18),
        cout => cout(18)

);


  U13: cas00 port map(

        uno => suno(18),
        q_in => M(17),
        p_in => N(17),
        cin => cout(16),

        s_uno => suno(17) ,
        s_qin => RR(17),
        s_cas => SS(17),
        cout => cout(17)

);


  U14: cas00 port map(

        uno => suno(17),
        q_in => M(16),
        p_in => N(16),
        cin => cout(15),

        s_uno => suno(16) ,
        s_qin => RR(16),
        s_cas => SS(16),
        cout => cout(16)

);
```

```
U15: cas00 port map(

        uno => suno(16),
        q_in => M(15),
        p_in => N(15),
        cin => cout(14),

        s_uno => suno(15) ,
        s_qin => RR(15),
        s_cas => SS(15),
        cout => cout(15)

);

U16: cas00 port map(

        uno => suno(15),
        q_in => M(14),
        p_in => N(14),
        cin => cout(13),

        s_uno => suno(14) ,
        s_qin => RR(14),
        s_cas => SS(14),
        cout => cout(14)

);

U17: cas00 port map(

        uno => suno(14),
        q_in => M(13),
        p_in => N(13),
        cin => cout(12),

        s_uno => suno(13) ,
        s_qin => RR(13),
        s_cas => SS(13),
        cout => cout(13)

);

U18: cas00 port map(

        uno => suno(13),
        q_in => M(12),
        p_in => N(12),
        cin => cout(11),

        s_uno => suno(12) ,
        s_qin => RR(12),
        s_cas => SS(12),
        cout => cout(12)

);

U19: cas00 port map(

        uno => suno(12),
        q_in => M(11),
        p_in => N(11),
        cin => cout(10),

        s_uno => suno(11) ,
        s_qin => RR(11),
        s_cas => SS(11),
        cout => cout(11)

);

U20: cas00 port map(

        uno => suno(11),
        q_in => M(10),
        p_in => N(10),
        cin => cout(9),

        s_uno => suno(10) ,
        s_qin => RR(10),
        s_cas => SS(10),
        cout => cout(10)

);

U21: cas00 port map(

        uno => suno(10),
        q_in => M(9),
        p_in => N(9),
        cin => cout(8),

        s_uno => suno(9) ,
        s_qin => RR(9),
        s_cas => SS(9),
        cout => cout(9)

);

U22: cas00 port map(

        uno => suno(9),
        q_in => M(8),
        p_in => N(8),
        cin => cout(7),

        s_uno => suno(8) ,
        s_qin => RR(8),
        s_cas => SS(8),
        cout => cout(8)

);

U23: cas00 port map(

        uno => suno(8),
        q_in => M(7),
        p_in => N(7),
        cin => cout(6),

        s_uno => suno(7) ,
        s_qin => RR(7),
        s_cas => SS(7),
        cout => cout(7)

);

U24: cas00 port map(

        uno => suno(7),
        q_in => M(6),
        p_in => N(6),
        cin => cout(5),

        s_uno => suno(6) ,
        s_qin => RR(6),
        s_cas => SS(6),
        cout => cout(6)

);

U25: cas00 port map(

        uno => suno(6),
        q_in => M(5),
        p_in => N(5),
        cin => cout(4),

        s_uno => suno(5) ,
        s_qin => RR(5),
        s_cas => SS(5),
        cout => cout(5)

);

U26: cas00 port map(

        uno => suno(5),
        q_in => M(4),
        p_in => N(4),
        cin => cout(3),

        s_uno => suno(4) ,
        s_qin => RR(4),
        s_cas => SS(4),
        cout => cout(4)

);

U27: cas00 port map(

        uno => suno(4),
        q_in => M(3),
        p_in => N(3),
        cin => cout(2),

        s_uno => suno(3) ,
        s_qin => RR(3),
        s_cas => SS(3),
        cout => cout(3)

);

U28: cas00 port map(

        uno => suno(3),
        q_in => M(2),
        p_in => N(2),
        cin => cout(1),

        s_uno => suno(2) ,
        s_qin => RR(2),
        s_cas => SS(2),
        cout => cout(2)

);

U29: cas00 port map(

        uno => suno(2),
        q_in => M(1),
        p_in => N(1),
        cin => cout(0),

        s_uno => suno(1) ,
        s_qin => RR(1),
        s_cas => SS(1),
        cout => cout(1)

);

U30: cas00 port map(

        uno => suno(1),
        q_in => M(0),
        p_in => N(0),
        cin => suno(0),

        s_uno => suno(0) ,
        s_qin => RR(0),
        s_cas => SS(0),
        cout => cout(0)

);

S <= SS;
R <= RR;

m_cout <= cout(3);

end mod_cas0;

-----------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;

package paquete_div00 is

component mod_cas00

port(
        M: in
std_logic_vector(22 downto 0)
;
        N: in
std_logic_vector(22 downto 0)
;
        m_uno: in std_logic
;

        S: inout
std_logic_vector(22 downto 0)
;
```

```
		R: inout
std_logic_vector(22 downto 0)
;
		m_cout: out
std_logic
);

end component;

end paquete_div00;
--------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use
ieee.std_logic_unsigned.all;
use paquete_div00.all;

entity div00 is

port(
		DIVISOR: in
std_logic_vector(22 downto 0);
		DIVIDENDO: in
std_logic_vector(22 downto 0);

--		REMINDER: OUT
std_logic_vector(22 downto 0);
		RES: out
std_logic_vector(22 downto 0)

		);

end;

architecture div0 of div00 is
signal SUM_0, SUM_1,
SUM_2, SUM_3, SUM_4,
SUM_5, SUM_6, SUM_7,
SUM_8, SUM_9, SUM_10,
SUM_11, SUM_12, SUM_13,
SUM_14, SUM_15, SUM_16,
SUM_17, SUM_18, SUM_19,
SUM_20, SUM_21, SUM_22:
std_logic_vector(22 downto 0);
signal REM_0, REM_1,
REM_2, REM_3, REM_4,
REM_5, REM_6, REM_7,
REM_8, REM_9, REM_10,
REM_11, REM_12, REM_13,
REM_14, REM_15, REM_16,
REM_17, REM_18, REM_19,
REM_20, REM_21, REM_22:
std_logic_vector(22 downto 0);
signal S_DIVIDENDO_0,
S_DIVIDENDO_1,
S_DIVIDENDO_2,
S_DIVIDENDO_3,
S_DIVIDENDO_4,
S_DIVIDENDO_5,
S_DIVIDENDO_6,
S_DIVIDENDO_7,
S_DIVIDENDO_8,
S_DIVIDENDO_9,
S_DIVIDENDO_10,
S_DIVIDENDO_11,
S_DIVIDENDO_12,
S_DIVIDENDO_13,
S_DIVIDENDO_14,
S_DIVIDENDO_15,
S_DIVIDENDO_16,
S_DIVIDENDO_17,
S_DIVIDENDO_18,
S_DIVIDENDO_19,
S_DIVIDENDO_20,
S_DIVIDENDO_21,
S_DIVIDENDO_22:
std_logic_vector(22 downto 0);
signal entrada:
std_logic_vector(22 downto 0);
begin

S_DIVIDENDO_0 <=
"0000000000000000000000"
& DIVIDENDO(22);

  U100: mod_cas00 port map(


		M => DIVISOR,
		N =>
S_DIVIDENDO_0,
		m_uno => '1',

		S => SUM_0,
		R => REM_0,
		m_cout =>
entrada(22)

  );



S_DIVIDENDO_1 <=
SUM_0(21 downto 0) &
DIVIDENDO(21);

  U101: mod_cas00 port map(


		M => REM_0,
		N =>
S_DIVIDENDO_1,
		m_uno =>
entrada(22),

		S => SUM_1,
		R => REM_1,
		m_cout =>
entrada(21)

  );


S_DIVIDENDO_2 <=
SUM_1(21 downto 0) &
DIVIDENDO(20);

  U102: mod_cas00 port map(


		M => REM_1,
		N =>
S_DIVIDENDO_2,
		m_uno =>
entrada(21),

		S => SUM_2,
		R => REM_2,
		m_cout =>
entrada(20)

  );



S_DIVIDENDO_3 <=
SUM_2(21 downto 0) &
DIVIDENDO(19);

  U103: mod_cas00 port map(


		M => REM_2,
		N =>
S_DIVIDENDO_3,
		m_uno =>
entrada(20),

		S => SUM_3,
		R => REM_3,
		m_cout =>
entrada(19)

  );


S_DIVIDENDO_4 <=
SUM_3(21 downto 0) &
DIVIDENDO(18);

  U104: mod_cas00 port map(


		M => REM_3,
		N =>
S_DIVIDENDO_4,
		m_uno =>
entrada(19),

		S => SUM_4,
		R => REM_4,
		m_cout =>
entrada(18)

  );


S_DIVIDENDO_5 <=
SUM_4(21 downto 0) &
DIVIDENDO(17);

  U105: mod_cas00 port map(


		M => REM_4,
		N =>
S_DIVIDENDO_5,
		m_uno =>
entrada(18),

		S => SUM_5,
		R => REM_5,
		m_cout =>
entrada(17)

  );


S_DIVIDENDO_6 <=
SUM_5(21 downto 0) &
DIVIDENDO(16);

  U106: mod_cas00 port map(


		M => REM_5,
		N =>
S_DIVIDENDO_6,
		m_uno =>
entrada(17),

		S => SUM_6,
		R => REM_6,
		m_cout =>
entrada(16)

  );


S_DIVIDENDO_7 <=
SUM_6(21 downto 0) &
DIVIDENDO(15);

  U107: mod_cas00 port map(


		M => REM_6,
		N =>
S_DIVIDENDO_7,
		m_uno =>
entrada(16),

		S => SUM_7,
		R => REM_7,
		m_cout =>
entrada(15)

  );


S_DIVIDENDO_8 <=
SUM_7(21 downto 0) &
DIVIDENDO(14);

  U108: mod_cas00 port map(


		M => REM_7,
		N =>
S_DIVIDENDO_8,
		m_uno =>
entrada(15),

		S => SUM_8,
		R => REM_8,
```

```
            m_cout =>
entrada(14)

  );


S_DIVIDENDO_9 <=
SUM_8(21 downto 0) &
DIVIDENDO(13);

  U109: mod_cas00 port map(


            M => REM_8,
            N =>
S_DIVIDENDO_9,
            m_uno =>
entrada(14),

            S => SUM_9,
            R => REM_9,
            m_cout =>
entrada(13)

  );


S_DIVIDENDO_10 <=
SUM_9(21 downto 0) &
DIVIDENDO(12);

  U110: mod_cas00 port map(


            M => REM_9,
            N =>
S_DIVIDENDO_10,
            m_uno =>
entrada(13),

            S => SUM_10,
            R => REM_10,
            m_cout =>
entrada(12)

  );


S_DIVIDENDO_11 <=
SUM_10(21 downto 0) &
DIVIDENDO(11);

  U111: mod_cas00 port map(


            M => REM_10,
            N =>
S_DIVIDENDO_11,
            m_uno =>
entrada(12),

            S => SUM_11,
            R => REM_11,
            m_cout =>
entrada(11)

  );


 S_DIVIDENDO_12 <=
SUM_11(21 downto 0) &
DIVIDENDO(10);

  U112: mod_cas00 port map(


            M => REM_11,
            N =>
S_DIVIDENDO_12,
            m_uno =>
entrada(11),

            S => SUM_12,
            R => REM_12,
            m_cout =>
entrada(10)

  );



 S_DIVIDENDO_13 <=
SUM_12(21 downto 0) &
DIVIDENDO(9);

  U113: mod_cas00 port map(


            M => REM_12,
            N =>
S_DIVIDENDO_13,
            m_uno =>
entrada(10),

            S => SUM_13,
            R => REM_13,
            m_cout =>
entrada(9)

  );


 S_DIVIDENDO_14 <=
SUM_13(21 downto 0) &
DIVIDENDO(8);

  U114: mod_cas00 port map(


            M => REM_13,
            N =>
S_DIVIDENDO_14,
            m_uno =>
entrada(9),

            S => SUM_14,
            R => REM_14,
            m_cout =>
entrada(8)

  );


 S_DIVIDENDO_15 <=
SUM_14(21 downto 0) &
DIVIDENDO(7);

  U115: mod_cas00 port map(


            M => REM_14,
            N =>
S_DIVIDENDO_15,
            m_uno =>
entrada(8),

            S => SUM_15,
            R => REM_15,
            m_cout =>
entrada(7)

  );


S_DIVIDENDO_16 <=
SUM_15(21 downto 0) &
DIVIDENDO(6);

  U116: mod_cas00 port map(


            M => REM_15,
            N =>
S_DIVIDENDO_16,
            m_uno =>
entrada(7),

            S => SUM_16,
            R => REM_16,
            m_cout =>
entrada(6)

  );


S_DIVIDENDO_17 <=
SUM_16(21 downto 0) &
DIVIDENDO(5);

  U117: mod_cas00 port map(


            M => REM_16,
            N =>
S_DIVIDENDO_17,
            m_uno =>
entrada(6),

            S => SUM_17,
            R => REM_17,
            m_cout =>
entrada(5)

  );


S_DIVIDENDO_18 <=
SUM_17(21 downto 0) &
DIVIDENDO(4);

  U118: mod_cas00 port map(


            M => REM_17,
            N =>
S_DIVIDENDO_18,
            m_uno =>
entrada(5),

            S => SUM_18,
            R => REM_18,
            m_cout =>
entrada(4)

  );

S_DIVIDENDO_19 <=
SUM_18(21 downto 0) &
DIVIDENDO(3);

  U119: mod_cas00 port map(


            M => REM_18,
            N =>
S_DIVIDENDO_19,
            m_uno =>
entrada(4),

            S => SUM_19,
            R => REM_19,
            m_cout =>
entrada(3)

  );


S_DIVIDENDO_20 <=
SUM_19(21 downto 0) &
DIVIDENDO(2);

  U120: mod_cas00 port map(


            M => REM_19,
            N =>
S_DIVIDENDO_20,
            m_uno =>
entrada(3),

            S => SUM_20,
            R => REM_20,
            m_cout =>
entrada(2)

  );

S_DIVIDENDO_21 <=
SUM_20(21 downto 0) &
DIVIDENDO(1);

  U121: mod_cas00 port map(


            M => REM_20,
            N =>
S_DIVIDENDO_21,
            m_uno =>
entrada(2),
```

```
                S => SUM_21,
                R => REM_21,
                m_cout =>
entrada(1)

  );

S_DIVIDENDO_22 <=
SUM_21(21 downto 0) &
DIVIDENDO(0);
```

```
  U122: mod_cas00 port map(


                M => REM_21,
                N =>
S_DIVIDENDO_22,
                m_uno =>
entrada(1),
```

```
                S => SUM_22,
                R => REM_22,
                m_cout =>
entrada(0)

  );
```

```
RES <=
"00000000000000000000001"
WHEN divisor = dividendo
ELSE
"00000000000000000000000"
WHEN dividendo < divisor
ELSE
        entrada;
```

```
--REMINDER <= SUM_3;

end div0;

------------------------------
```

25. Selector de exponente.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity MUX_EXP00 is

port(
EXP_SR: in std_logic_vector(7 downto 0);
EXP_MD: in std_logic_vector(7 downto 0);
OP: in std_logic_vector(1 downto 0);

EXP_R: out std_logic_vector(7 downto 0)

);

end;

architecture MUX_EXP0 of MUX_EXP00 is
begin

        EXP_R <= EXP_SR when (OP="01" OR OP="10") else EXP_MD;

end MUX_EXP0;
```

26. Selector del resultado de la operación.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX_EXP00 is

port(
OP_SR: in std_logic_vector(22 downto 0);
OP_M: in std_logic_vector(22 downto 0);
OP_D: in std_logic_vector(22 downto 0);
OP: in std_logic_vector(1 downto 0);

MANT_R: out std_logic_vector(22 downto 0)

);

end;

architecture MUX_EXP0 of MUX_EXP00 is
begin

        MANT_R <= OP_SR when (OP="01" OR OP="10") else OP_M WHEN OP=11
ELSE OP_D;

end MUX_EXP0;
```

27. Paquete del módulo Compue.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package paquete_compute is

---- MULTIPLEXORES PARA PERMUTAR A POR B O VICEVERSA
component MUX_X00
```

```
port(

A: in std_logic_vector(22 downto 0);
B: in std_logic_vector(22 downto 0);
F_SELMUX: in std_logic;

SAL_MUXX: out std_logic_vector(22 downto 0)

  );

end component;


---- COMPARADOR DE MANTISAS
component COMP_MANTISA00

port(


MA: in std_logic_vector(22 downto 0);
MB: in std_logic_vector(22 downto 0);

AMB: out std_logic;
BMA: out std_logic;
AEB: out std_logic

  );

end component;



---- SELECTOR DE SIGNO Y OPERACION
component SELECTOR00

port(
CODOP: in std_logic_vector(2 downto 0);

BMA: in std_logic;
AMB: in std_logic;
AEB: in std_logic;

SA: in std_logic;
SB: in std_logic;

SELMUX: out std_logic;
CIN: out std_logic;
SR: out std_logic
  );

end component;



---- SUMADOR - RESTADOR
component SUM_RES00

port(

        A: in std_logic_vector ( 22 downto 0 );
        Bb: in std_logic_vector ( 22 downto 0 );
        Cin: in std_logic ;
        SAL: out std_logic_vector ( 22 downto 0 );


        S: out std_logic;
        Z: out std_logic;
        AC: out std_logic;
        O: out std_logic
  );

end component;


---- MULTIPLICADOR
component MULT00

port(

        A: in std_logic_vector ( 22 downto 0 );
        B: in std_logic_vector ( 22 downto 0 );
        M: out std_logic_vector ( 45 downto 0 )
  );

end component;



---- DIVISOR
component DIV00

port(

        DIVISOR: in std_logic_vector(22 downto 0);
        DIVIDENDO: in std_logic_vector(22 downto 0);
```

```
--              REMINDER: OUT std_logic_vector(22 downto 0);
                RES: out std_logic_vector(22 downto 0)
  );

end component;



---- MUX SELECCIONADOR DE EXPONENTE
component MUX_EXP00

port(

EXP_SR: in std_logic_vector(7 downto 0);
EXP_MD: in std_logic_vector(7 downto 0);
OP: in std_logic_vector(1 downto 0);

EXP_R: out std_logic_vector(7 downto 0)

  );

end component;


---- MUX SELECCIONADOR DE RESULTADO ESPERADO DE LA MANTISA
component MUX_RESOP00

port(


OP_SR: in std_logic_vector(22 downto 0);
OP_M: in std_logic_vector(22 downto 0);
OP_D: in std_logic_vector(22 downto 0);
OP: in std_logic_vector(1 downto 0);

MANT_R: out std_logic_vector(22 downto 0)



  );

end component;

end paquete_compute;
```

28. Top del Módulo Compute.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use paquete_compute.all;

entity MODULO_COMPUTE00 is

port(
SALMUX_B: in std_logic_vector(22 downto 0);
SALMUX_A: in std_logic_vector(22 downto 0);

MA_NATURAL: in std_logic_vector(22 downto 0);
MB_NATURAL: in std_logic_vector(22 downto 0);

EXP_I: in std_logic_vector(7 downto 0);

CODOP: in std_logic_vector(2 downto 0);

SA: in std_logic;
SB: in std_logic;


MANTISA: out std_logic_vector (22 downto 0);
EXP_O: out std_logic_vector(7 downto 0);
SIGNO: out std_logic;


O: out std_logic;
AC: out std_logic;
Z: out std_logic;
S: out std_logic    );

end;
```

```
architecture MODULO_COMPUTE0 of MODULO_COMPUTE00 is
signal F_SM, F_AMB, F_BMA, F_AEB, F_CIN: std_logic;
signal MUXA_SUMA, MUXB_SUMB, SAL_SUM, SAL_DIV: std_logic_vector(22 downto
0);
signal SAL_MULT: std_logic_vector(45 downto 0);

begin

----- MUX_A
U19: MUX_X00 port map(

A => SALMUX_A,
B => SALMUX_B,
F_SELMUX => F_SM,
SAL_MUXX => MUXA_SUMA

  );

----- MUX_B
U20: MUX_X00 port map(

A => SALMUX_B,
B => SALMUX_A,
F_SELMUX => F_SM,
SAL_MUXX => MUXB_SUMB

  );



----- COMPARADOR DE MANTISAS
U21: COMP_MANTISA00 port map(
        MA => SALMUX_A,
        MB => SALMUX_B,

        AMB => F_AMB,
        BMA => F_BMA,
        AEB => F_AEB
  );


----- SELECTOR DE OPERACION Y SIGNO
U22: SELECTOR00 port map(
        CODOP => CODOP,

        BMA => F_BMA,
        AMB => F_AMB,
        AEB => F_AEB,
        SA => SA,
        SB => SB,

        SELMUX => F_SM,
        CIN => F_CIN,
        SR => SIGNO
  );


----- SUMADOR_RESTADOR
U23: SUM_RES00 port map(
        A => MUXA_SUMA,
        Bb => MUXB_SUMB,
        Cin => F_CIN,
        SAL => SAL_SUM,


        S => S,
        Z => Z,
        AC => AC,
        O => O
  );

----- MULTIPLICADOR
U24: MULT00 port map(
        A => MA_NATURAL,
        B => MB_NATURAL,
        M => SAL_MULT
  );

----- DIVISOR
U25: DIV00 port map(

        DIVISOR => MA_NATURAL,
        DIVIDENDO => MB_NATURAL,

        RES => SAL_DIV

  );


----- MUX OPERACION
U26: MUX_RESOP00 port map(


OP_SR => SAL_SUM,
OP_M => SAL_MULT(45 DOWNTO 23),
```

```
OP_D => SAL_DIV,
OP => CODOP(1 DOWNTO 0),

MANT_R => MANTISA


  );

----- MUX EXPONENTE
U27: MUX_EXP00 port map(
```

29. Sumatoria del sesgo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity SUM_SES00 is

port(
        EXP_IN: in std_logic_vector(7 downto 0) ;

        FSS_DESNOR: out std_logic;
        EXP_OUT: out std_logic_VECTOR(7 DOWNTO 0));

end;

architecture SUM_SES0 of SUM_SES00 is
signal XP: std_logic_vector(8 downto 0);
begin

        XP <= (EXP_IN + "001111111");

        EXP_OUT <= XP(7 DOWNTO 0);
        FSS_DESNOR <= XP(8);

end SUM_SES0;
```

```
EXP_SR => EXP_I,
EXP_MD => SAL_SUM(7 DOWNTO 0),
OP => CODOP(1 DOWNTO 0),

EXP_R => EXP_O
  );
End MODULO_COMPUTE0;
```

Modulo Set

30. Multiplexor de salida.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity MUX_RESULT00 is

port(
OPERADO: in std_logic_vector(31 downto 0);
VERIF_A: in std_logic_vector(31 downto 0);
VERIF_B: in std_logic_vector(31 downto 0);
FLAG1: in std_logic;
FLAG2: in std_logic;

RESULT: out std_logic_vector(31 downto 0)

);

end;

architecture MUX_RESULT0 of MUX_RESULT00 is
begin

        RESULT <= OPERADO when (FLAG1= '0' AND FLAG2= '0') else
                                VERIF_A WHEN (FLAG1= '0' AND FLAG2= '1')
ELSE
                                VERIF_B WHEN (FLAG1= '1' AND FLAG2= '0')
ELSE
                                VERIF_A;
```

```
end MUX_RESULT0;
```

31. Normalización Final.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity NORM_FINAL00 is

port(
ENTRADA: in std_logic_vector(31 downto 0);
CODOP: in std_logic_vector(1 downto 0);
CIN: in std_logic;
AC: in std_logic;

FLAG_ZE: out std_logic;
NORMALIZADO: out std_logic_vector(31 downto 0)

);

end;

architecture NORM_FINAL0 of NORM_FINAL00 is
signal S, S_O: std_logic;
signal EXPONENTE, EXP_O: std_logic_vector(7 downto 0);
signal MANTISA, MANT_O: std_logic_vector(22 downto 0);
signal NORM: std_logic_vector(31 downto 0);

begin

S <= ENTRADA(31);
EXPONENTE <= ENTRADA(30 downto 23);
MANTISA <= ENTRADA(22 DOWNTO 0);


        NORM <= ENTRADA when ((CODOP="01" OR CODOP="10") AND (CIN='0')
AND (AC='1')) else
                                    (ENTRADA(31 DOWNTO 1)&"0") WHEN
(MANTISA = "00000000000000000000001" AND CODOP="00") ELSE
                                    (S & (EXPONENTE - x"01") & MANTISA(21
downto 0)&"0") WHEN (MANTISA <= x"7FFFFF" AND MANTISA > x"3FFFFF") ELSE
                                    (S & (EXPONENTE - x"02") & MANTISA(20
downto 0)&"00") WHEN (MANTISA <= x"3FFFFF" AND MANTISA > x"1FFFFF") ELSE
                                    (S & (EXPONENTE - x"03") & MANTISA(19
downto 0)&"000") WHEN (MANTISA <= x"1FFFFF" AND MANTISA > x"FFFFF") ELSE
                                    (S & (EXPONENTE - x"04") & MANTISA(18
downto 0)&"0000") WHEN (MANTISA <= x"FFFFF" AND MANTISA > x"7FFFF") ELSE
                                    (S & (EXPONENTE - x"05") & MANTISA(17
downto 0)&"00000") WHEN (MANTISA <= x"7FFFF" AND MANTISA > x"3FFFF") ELSE
                                    (S & (EXPONENTE - x"06") & MANTISA(16
downto 0)&"000000") WHEN (MANTISA <= x"3FFFF" AND MANTISA > x"1FFFF") ELSE
                                    (S & (EXPONENTE - x"07") & MANTISA(15
downto 0)&"0000000") WHEN (MANTISA <= x"1FFFF" AND MANTISA > x"FFFF") ELSE
                                    (S & (EXPONENTE - x"08") & MANTISA(14
downto 0)&"00000000") WHEN (MANTISA <= x"FFFF" AND MANTISA > x"7FFF") ELSE
                                    (S & (EXPONENTE - x"09") & MANTISA(13
downto 0)&"000000000") WHEN (MANTISA <= x"7FFF" AND MANTISA > x"3FFF")
ELSE
                                    (S & (EXPONENTE - x"0A") & MANTISA(12
downto 0)&"0000000000") WHEN (MANTISA <= x"3FFF" AND MANTISA > x"1FFF")
ELSE
                                    (S & (EXPONENTE - x"0B") & MANTISA(11
downto 0)&"00000000000") WHEN (MANTISA <= x"1FFF" AND MANTISA > x"FFF")
ELSE
                                    (S & (EXPONENTE - x"0C") & MANTISA(10
downto 0)&"000000000000") WHEN (MANTISA <= x"FFF" AND MANTISA > x"7FF")
ELSE
                                    (S & (EXPONENTE - x"0D") & MANTISA(9
downto 0)&"0000000000000") WHEN (MANTISA <= x"7FF" AND MANTISA > x"3FF")
ELSE
                                    (S & (EXPONENTE - x"0E") & MANTISA(8
downto 0)&"00000000000000") WHEN (MANTISA <= x"3FF" AND MANTISA > x"1FF")
ELSE
                                    (S & (EXPONENTE - x"0F") & MANTISA(7
downto 0)&"000000000000000") WHEN (MANTISA <= x"1FF" AND MANTISA > x"FF")
ELSE
                                    (S & (EXPONENTE - x"10") & MANTISA(6
downto 0)&"0000000000000000") WHEN (MANTISA <= x"FF" AND MANTISA > x"7F")
ELSE
                                    (S & (EXPONENTE - x"11") & MANTISA(5
downto 0)&"00000000000000000") WHEN (MANTISA <= x"7F" AND MANTISA > x"3F")
ELSE
                                    (S & (EXPONENTE - x"12") & MANTISA(4
downto 0)&"000000000000000000") WHEN (MANTISA <= x"3F" AND MANTISA > x"1F")
ELSE
                                    (S & (EXPONENTE - x"13") & MANTISA(3
downto 0)&"0000000000000000000") WHEN (MANTISA <= x"1F" AND MANTISA > x"F")
ELSE
```

```
                                        (S & (EXPONENTE - x"14") & MANTISA(2
downto 0)&"00000000000000000000") WHEN (MANTISA <= x"F" AND MANTISA > x"7")
ELSE
                                        (S & (EXPONENTE - x"15") & MANTISA(1
downto 0)&"000000000000000000000") WHEN (MANTISA <= x"7" AND MANTISA >
x"3") ELSE
                                        (S & (EXPONENTE - x"16") &
MANTISA(0)&"0000000000000000000000") WHEN (MANTISA <= x"3" AND MANTISA
> x"1") ELSE
                                        (S & (EXPONENTE) & MANTISA) WHEN
(MANTISA = "00000000000000000000000") ELSE

                                        (S & (EXPONENTE - x"17")
&"00000000000000000000000");

FLAG_ZE <= '1' WHEN MANTISA = x"00000" ELSE '0';

S_O <= NORM(31);
EXP_O <= NORM(30 DOWNTO 23);
MANT_O  <= NORM(22 DOWNTO 0);

NORMALIZADO <= NORM;
end NORM_FINAL0;
```

32. Verificador.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity CHECKIT00 is

port(
IN_CHECK: in std_logic_vector(31 downto 0);

ZE: in std_logic;
FSS_DESNOR: in std_logic;
DBcero: in std_logic;
OPerror_I: std_logic;

RESULTADO: out std_logic_vector(31 downto 0);

MAS_INF: out std_logic;
MENOS_INF: out std_logic;
CERO_MAS: out std_logic;
CERO_MENOS: out std_logic;
DESNOR: out std_logic;
NAN: out std_logic;

OVERFLOW: out std_logic;
UNDERFLOW: out std_logic;
ZERO_DIV: out std_logic;
OPerror_O: out std_logic
);

end;

architecture CHECKIT0 of CHECKIT00 is
signal S, MAI, MEI, CMA, CME, DN, NN: std_logic;
signal EXPONENTE: std_logic_vector(7 downto 0);
signal MANTISA: std_logic_vector(22 downto 0);

begin

S <= IN_CHECK(31);
EXPONENTE <= IN_CHECK(30 downto 23);
MANTISA <= IN_CHECK(22 DOWNTO 0);

        MAI <= '1' WHEN (S='0' AND MANTISA=x"000000" AND
EXPONENTE=x"FF") ELSE '0';
        MEI <= '1' WHEN (S='1' AND MANTISA=x"000000" AND
EXPONENTE=x"FF") ELSE '0';

        CMA <= '1' WHEN ((ZE = '1') OR ((S = '0') AND (MANTISA=x"000000" AND
EXPONENTE = x"00"))) ELSE '0';


        CME <= '1' WHEN ((ZE = '1') OR ((S = '1') AND (MANTISA=x"000000" AND
EXPONENTE = x"00"))) ELSE '0';
        DN <= '1' WHEN (((EXPONENTE = X"00") AND (MANTISA /= x"000000") )
OR (FSS_DESNOR = '1')) ELSE '0';

        NN <= '1' WHEN ((EXPONENTE = X"FF" AND MANTISA /= X"000000") OR
OPerror_I = '1') else '0';

        OVERFLOW <= '1' WHEN ((MAI OR MEI)= '1') ELSE '0';
        UNDERFLOW <= '1' WHEN (DN = '1') ELSE '0';
        ZERO_DIV <= '1' WHEN (DBcero = '1') else '0';
        OPerror_O <= OPerror_I;

        CERO_MAS <= CMA;
        CERO_MENOS <= CME;
```

```
		DESNOR <= DN;
		MAS_INF <= MAI;
		MENOS_INF <= MEI;

		NAN <= NN;


RESULTADO <= S & x"00" & MANTISA when ((CMA OR CME) = '1') ELSE
					S & x"FF" & MANTISA when (NN= '1') ELSE
IN_CHECK;

end CHECKIT0;
```

33. Paquete del módulo Set

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_SET is

-- SUMADOR DEL SESGO
component SUM_SES00
port(
		EXP_IN: in std_logic_vector(7 downto 0) ;

		FSS_DESNOR: out std_logic;
		EXP_OUT: out std_logic_VECTOR(7 DOWNTO 0)
  );

end component;

--- MULTIPLEXOR DE RESULTADOS POSIBLES
component MUX_RESULT00

port(
OPERADO: in std_logic_vector(31 downto 0);
VERIF_A: in std_logic_vector(31 downto 0);
VERIF_B: in std_logic_vector(31 downto 0);
FLAG1: in std_logic;
FLAG2: in std_logic;

RESULT: out std_logic_vector(31 downto 0)

  );

end component;

-- NORMALIZACION FINAL DEL RESULTADO
component NORM_FINAL00

port(
ENTRADA: in std_logic_vector(31 downto 0);
CODOP: in std_logic_vector(1 downto 0);
CIN: in std_logic;
AC: in std_logic;

FLAG_ZE: out std_logic;
NORMALIZADO: out std_logic_vector(31 downto 0)

  );

end component;



--- VERIFICACION DE LOS RESULTADOS ARROJADOS CORRECTOS
component CHECKIT00

port(
IN_CHECK: in std_logic_vector(31 downto 0);

ZE: in std_logic;
FSS_DESNOR: in std_logic;
DBcero: in std_logic;
OPerror_I: std_logic;

RESULTADO: out std_logic_vector(31 downto 0);

MAS_INF: out std_logic;
MENOS_INF: out std_logic;
CERO_MAS: out std_logic;
CERO_MENOS: out std_logic;
DESNOR: out std_logic;
NAN: out std_logic;
```

```
OVERFLOW: out std_logic;
UNDERFLOW: out std_logic;
ZERO_DIV: out std_logic;
OPerror_O: out std_logic

  );

end component;
end PAQUETE_sET;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use paquete_set.all;


entity MODULO_SET00 is

port(
S: in std_logic;
EXP: in std_logic_vector(7 downto 0);
MANTISA: in std_logic_vector(22 downto 0);

VERIF_A: in std_logic_vector(31 downto 0);
VERIF_B: in std_logic_vector(31 downto 0);

CODOP: in std_logic_vector(1 downto 0);
CIN: in std_logic;
AC: in std_logic;

OPerror: in std_logic;

SAcero: in std_logic;
SBcero: in std_logic;
MAcero: in std_logic;
MBcero: in std_logic;
DAcero: in std_logic;
DBcero: in std_logic;

RESULTADO: out std_logic_vector(31 downto 0);

MAS_INF: out std_logic;
MENOS_INF: out std_logic;
CERO_MAS: out std_logic;
CERO_MENOS: out std_logic;
```

34. Top del Módulo Set

```
DESNOR: out std_logic;
NAN: out std_logic;

OVERFLOW: out std_logic;
UNDERFLOW: out std_logic;
ZERO_DIV: out std_logic;
OPerror_F: out std_logic
);

end;

architecture MODULO_SET0 of MODULO_SET00 is
signal F_DES, F1, F2, F_ZE: std_logic;
signal EXP_SUMSES: std_logic_vector(7 downto 0);
--signal MANTISA, MANT_O: std_logic_vector(22
downto 0);
signal RES_MUX, RES_NORM, COMPLETO:
std_logic_vector(31 downto 0);

begin

---- SUMADOR SESGO
U28: SUM_SES00 port map(
		EXP_IN => EXP,

		FSS_DESNOR => F_DES,
		EXP_OUT => EXP_SUMSES
);

F1 <= (MBcero XOR SAcero);
F2 <= ((MAcero OR DAcero) XOR (SBcero));


--- MULTIPLEXOR RESULTADO
COMPLETO <= S & EXP_SUMSES & MANTISA;

U29: MUX_RESULT00 port map(
		OPERADO => COMPLETO,
		VERIF_A => VERIF_A,
		VERIF_B => VERIF_B,
		FLAG1 => F1,
		FLAG2 => F2,
		RESULT => RES_MUX
);


--- NORMALIZACION
U30: NORM_FINAL00 port map(

		ENTRADA => RES_MUX,
		CODOP => CODOP,
		CIN => CIN,
		AC => AC,

		FLAG_ZE => F_ZE,
		NORMALIZADO => RES_NORM
);

--- VERIFICADOR DE RESULTADOS
U31: CHECKIT00 port map(

		IN_CHECK => RES_NORM,

		ZE => F_ZE,
		FSS_DESNOR => F_DES,
		DBcero => DBcero,
		OPerror_I => OPerror,

		RESULTADO => RESULTADO,

		MAS_INF => MAS_INF,
		MENOS_INF => MENOS_INF,
		CERO_MAS => CERO_MAS,
```

```
        CERO_MENOS => CERO_MENOS,
        DESNOR => DESNOR,
        NAN => NAN,
        OVERFLOW => OVERFLOW,
        UNDERFLOW => UNDERFLOW,
        ZERO_DIV => ZERO_DIV,
        OPerror_O => OPerror_F
);

nd MODULO_SET0;
```

Unidad de Punto Flotante (UPF)

35. Paquete de la UPF.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_UPF is

-----------------------------------------------------MODULO CHECK
component MODULO_CHECK00
port(

        OP_A: in std_logic_vector(31 downto 0);
        OP_B: in std_logic_vector(31 downto 0);
        CODOP: in std_logic_vector(4 downto 0);

        OUT_OPA: out std_logic_vector(31 downto 0);
        OUT_OPB: out std_logic_vector(31 downto 0);
        F_DESNOR_A: out std_logic;
        F_DESNOR_B: out std_logic;
        F_OPERROR: out std_logic;
        F_SAcero: out std_logic;
        F_RAcero: out std_logic;
        F_MAcero: out std_logic;
        F_DAcero: out std_logic;
        F_SBcero: out std_logic;
        F_RBcero: out std_logic;
        F_MBcero: out std_logic;
        F_DBcero: out std_logic

);
end component;

-------------------------------------------------------MODULO READY
component MODULO_READY00
port(

        OPA: in std_logic_vector(31 downto 0);
        OPB: in std_logic_vector(31 downto 0);
        COD_OP: in std_logic_vector(1 downto 0);

        MA_NATURAL: out std_logic_vector(22 downto 0);
        MB_NATURAL: out std_logic_vector(22 downto 0);
        SAL_MUXA: out std_logic_vector(22 downto 0);
        SAL_MUXB: out std_logic_vector(22 downto 0);
        EXP: out std_logic_vector(7 downto 0);
        SA: out std_logic;
        SB: out std_logic

);
end component;


----------------------------------------------------- MODULO COMPUTE
component MODULO_COMPUTE00
port(

        SALMUX_B: in std_logic_vector(22 downto 0);
        SALMUX_A: in std_logic_vector(22 downto 0);
        MA_NATURAL: in std_logic_vector(22 downto 0);
        MB_NATURAL: in std_logic_vector(22 downto 0);
        EXP_I: in std_logic_vector(7 downto 0);
        CODOP: in std_logic_vector(2 downto 0);
        SA: in std_logic;
        SB: in std_logic;
```

```
		MANTISA: out std_logic_vector (22 downto 0);
		EXP_O: out std_logic_vector(7 downto 0);
		SIGNO: out std_logic;
		C_IN: out std_logic;
		O: out std_logic;
		AC: out std_logic;
		Z: out std_logic;
		S: out std_logic

);
end component;



------------------------------------------------------ MODULO SET
component MODULO_SET00
port(

		S: in std_logic;
		EXP: in std_logic_vector(7 downto 0);
		MANTISA: in std_logic_vector(22 downto 0);
		VERIF_A: in std_logic_vector(31 downto 0);
		VERIF_B: in std_logic_vector(31 downto 0);
		CODOP: in std_logic_vector(1 downto 0);
		CIN: in std_logic;
		AC: in std_logic;
		OPerror: in std_logic;
		SAcero: in std_logic;
		SBcero: in std_logic;
		MAcero: in std_logic;
		MBcero: in std_logic;
		DAcero: in std_logic;
		DBcero: in std_logic;

		RESULTADO: out std_logic_vector(31 downto 0);
		MAS_INF: out std_logic;
		MENOS_INF: out std_logic;
		CERO_MAS: out std_logic;
		CERO_MENOS: out std_logic;
		DESNOR: out std_logic;
		NAN: out std_logic;
		OVERFLOW: out std_logic;
		UNDERFLOW: out std_logic;
		ZERO_DIV: out std_logic;
		OPerror_F: out std_logic
);
end component;

end PAQUETE_UPF;
```

36. Top de la UPF

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use PAQUETE_UPF.all;

entity UPF00 is

port(
		A: in std_logic_vector(31 downto 0) ;
		B: in std_logic_vector(31 downto 0) ;
		CODOP: in std_logic_vector(4 downto 0) ;

		A_DESNORMALIZADO: out std_logic;
		B_DESNORMALIZADO: out std_logic;

		NO, NZ, NS: out std_logic;

		MAS_INF: out std_logic;
		MENOS_INF: out std_logic;
		CERO_MAS: out std_logic;
		CERO_MENOS: out std_logic;
		DESNORMALIZADO: out std_logic;
		NAN: out std_logic;
		OVERFLOW: out std_logic;
		UNDERFLOW: out std_logic;
		ZERODIV: out std_logic;
		OPERRONEO: out std_logic;
```

```
		RESULTADO: out std_logic_vector(31 downto 0)  );
end;

architecture UPF0 of UPF00 is
signal SA, SB, FLAG_OPERROR, S_TOSET, CIN_TOSET, AC_TOSET: std_logic;
signal FSA0, FRA0, FMA0,FDA0, FSB0, FRB0, FMB0, FDB0: std_logic;
signal EXP_READYTOCOMPUTE, EXP_TOSET: std_logic_vector(7 downto 0);
signal MA_NATURAL, MB_NATURAL, SMUX_A, SMUX_B, MANTISA_TOSET:
std_logic_vector(22 downto 0);
signal A_CHECKTOREADY, B_CHECKTOREADY: std_logic_vector(31 downto 0);

begin

---------------------------------------------CHECK
  U32: MODULO_CHECK00 port map(

		OP_A => A,
		OP_B => B,
		CODOP => CODOP,

		OUT_OPA => A_CHECKTOREADY,
		OUT_OPB => B_CHECKTOREADY,
		F_DESNOR_A => A_DESNORMALIZADO,
		F_DESNOR_B => B_DESNORMALIZADO,
		F_OPERROR => FLAG_OPERROR,
		F_SAcero => FSA0,
		F_RAcero => FRA0,
		F_MAcero => FMA0,
		F_DAcero => FDA0,
		F_SBcero => FSB0,
		F_RBcero => FRB0,
		F_MBcero => FMB0,
		F_DBcero => FDB0

		);

			---------------------------------------READY
		U33: MODULO_READY00 port map(

		OPA => A_CHECKTOREADY,
		OPB => B_CHECKTOREADY,
		COD_OP => CODOP(1 downto 0),

		MA_NATURAL => MA_NATURAL,
		MB_NATURAL => MB_NATURAL,
		SAL_MUXA => SMUX_A,
		SAL_MUXB => SMUX_B,
		EXP => EXP_READYTOCOMPUTE,
		SA => SA,
		SB => SB

		);

		-------------------------------------------- COMPUTE
		U34: MODULO_COMPUTE00 port map(

		SALMUX_B => SMUX_B,
		SALMUX_A => SMUX_A,
		MA_NATURAL => MA_NATURAL,
		MB_NATURAL => MB_NATURAL,
		EXP_I => EXP_READYTOCOMPUTE,
		CODOP => CODOP(2 downto 0),
		SA => SA,
		SB => SB,

		MANTISA => MANTISA_TOSET,
		EXP_O => EXP_TOSET,
		SIGNO => S_TOSET,
		C_IN => CIN_TOSET,
		O => NO,
		AC => AC_TOSET,
		Z => NZ,
		S => NS


		);

		U35: MODULO_SET00 port map(

		S => S_TOSET,
		EXP => EXP_TOSET,
		MANTISA => MANTISA_TOSET,
		VERIF_A => A_CHECKTOREADY,
		VERIF_B => B_CHECKTOREADY,
		CODOP => CODOP(1 downto 0),
		CIN => CIN_TOSET,
		AC => AC_TOSET,
		OPerror => FLAG_OPERROR,
		SAcero => FSA0,
		SBcero => FSB0,
		MAcero => FMA0,
		MBcero => FMB0,
		DAcero => FDA0,
		DBcero => FDB0,

		RESULTADO => RESULTADO,
```

```
MAS_INF => MAS_INF,
MENOS_INF => MENOS_INF,
CERO_MAS => CERO_MAS,
CERO_MENOS => CERO_MENOS,
DESNOR => DESNORMALIZADO,
NAN => NAN,
OVERFLOW => OVERFLOW,
UNDERFLOW => UNDERFLOW,
ZERO_DIV => ZERODIV,
OPerror_F => OPERRONEO

);

end UPF0;
```

Banco de Registros

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity BANCO_REGISTROS00 is

   Port ( CLK          : in  STD_LOGIC;
          WE           : in  STD_LOGIC;
          ADDR_WR      : in  STD_LOGIC_VECTOR (7 downto 0);
          ADDR_RD1     : in  STD_LOGIC_VECTOR (7 downto
0);
          ADDR_RD2     : in  STD_LOGIC_VECTOR (7 downto
0);
          DIN          : in  STD_LOGIC_VECTOR (31 downto 0);
          DOUT1        : out STD_LOGIC_VECTOR (31 downto 0);
          DOUT2        : out STD_LOGIC_VECTOR (31 downto 0)

          );
end BANCO_REGISTROS00;


architecture BANCO_REGISTROS0 of BANCO_REGISTROS00 is
TYPE MEM_TYPE IS ARRAY (0 TO (2**8)-1 ) OF STD_LOGIC_VECTOR(DIN'RANGE);
SIGNAL MEM : MEM_TYPE;

begin
-- ESCRITURA DE MEMORIA
        PMEM : PROCESS( CLK )
        BEGIN
                IF( RISING_EDGE(CLK) )THEN
                        IF( WE = '1' )THEN

                                MEM(CONV_INTEGER(ADDR_WR)) <= DIN;

                        END IF;
                END IF;
        END PROCESS PMEM;


-- LECTURA DE MEMORIA
        DOUT1 <= MEM(CONV_INTEGER(ADDR_RD1));
        DOUT2 <= MEM(CONV_INTEGER(ADDR_RD2));

end BANCO_REGISTROS0;
```

Memoria de Datos

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity MEMORIA_DATOS00 is
        GENERIC(
                NBITS_ADDR : INTEGER := 8;
                NBITS_DATA : INTEGER := 32
        );
  Port ( CLK : in  STD_LOGIC;
       ADDR : in  STD_LOGIC_VECTOR (NBITS_ADDR-1 downto 0);
       DIN : in  STD_LOGIC_VECTOR (NBITS_DATA-1 downto 0);
       WE : in  STD_LOGIC;
       DOUT : out  STD_LOGIC_VECTOR (NBITS_DATA-1 downto 0));
end MEMORIA_DATOS00;

architecture MEMORIA_DATOS0 of MEMORIA_DATOS00 is
TYPE MEM_TYPE IS ARRAY ((2**NBITS_ADDR)-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(DIN'RANGE);
SIGNAL MEM : MEM_TYPE := (  "01000001001000000000000000000000",

        "00111111100000000000000000000000",

        "00000000000000000000000000000000",


        "00111111111100011001100110011001", --1

        "01000001110011011100111100001101", --2

        "01000000110110100010100000100100",--3

        "11001011111001010011001111000100",--4

        "01000100100101100100011001101111",--5

        "01000100000001011100111110100110",--6

        "11000000011100110101111000010011",--7

        "01000101101011111101001101010010",--8

        "11000101101001111001101011110101",--9

        "01000000000111010010100011100000",--10


        "01000000101000000000000000000000",-- #5 11

        "01000000101000000000000000000000",-- #5 12

        "01000000101000000000000000000000",-- #5 13

        "01000000101000000000000000000000",-- #5 14

        "01000000101000000000000000000000",-- #5 15

        "01000000101000000000000000000000",-- #5 16

        "01000000101000000000000000000000",-- #5 17

        "01000000101000000000000000000000",-- #5 18

        "01000000101000000000000000000000",-- #5 19

        "01000000101000000000000000000000",-- #5 20


                                                            OTHERS =>
        X"00000000");
```

```
begin
-- ESCRITURA DE MEMORIA
		PMEM : PROCESS( CLK )
		BEGIN
			IF( RISING_EDGE(CLK) )THEN
				IF( WE = '1' )THEN
					MEM(CONV_INTEGER(ADDR)) <= DIN;
				END IF;
			END IF;
		END PROCESS PMEM;

-- LECTURA DE MEMORIA
		DOUT <= MEM(CONV_INTEGER(ADDR));

end MEMORIA_DATOS0;
```

Memoria de Programa

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity MEMORIA_PROGRAMA00 is
    Port ( A : in  STD_LOGIC_VECTOR (10 downto 0);
           D : out  STD_LOGIC_VECTOR (31 downto 0));
end MEMORIA_PROGRAMA00;

architecture MEMORIA_PROGRAMA0 of MEMORIA_PROGRAMA00 is

-- CARGA Y ALMACENAMIENTO
		CONSTANT OPCODE_LW			: STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000";
		CONSTANT OPCODE_SW			: STD_LOGIC_VECTOR(4 DOWNTO 0) := "10000";

-- OPERACIONES ARITMETICAS
		CONSTANT OPCODE_SUMA : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00001";
		CONSTANT OPCODE_RESTA		: STD_LOGIC_VECTOR(4 DOWNTO 0) := "00010";
		CONSTANT OPCODE_MULT : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00011";
		CONSTANT OPCODE_DIV			: STD_LOGIC_VECTOR(4 DOWNTO 0) := "00100";

-- SALTOS
		CONSTANT OPCODE_B			: STD_LOGIC_VECTOR(4 DOWNTO 0) := "10001";
		CONSTANT OPCODE_BEQ			: STD_LOGIC_VECTOR(4 DOWNTO 0) := "10010";

-- REGISTROS
		CONSTANT R0			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000000";
		CONSTANT R1			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000001";
		CONSTANT R2			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000010";
		CONSTANT R3			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000011";


CONSTANT R4			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000100";
		CONSTANT R5			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000101";
		CONSTANT R6			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000110";
		CONSTANT R7			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000111";
		CONSTANT R8			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001000";
		CONSTANT R9			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001001";
		CONSTANT R10			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001010";
		CONSTANT R11			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001011";
		CONSTANT R12			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001100";
		CONSTANT R13			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001101";
		CONSTANT R14			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001110";
		CONSTANT R15			: STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001111";

-- SIN USO
```

```
        CONSTANT SU                                    :
STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000000";

-- INICIO DIRECCION
        CONSTANT INI                                   : STD_LOGIC_VECTOR(2
DOWNTO 0) := "000";


        TYPE ARR IS ARRAY (0 TO 2**11 - 1) OF STD_LOGIC_VECTOR(31
DOWNTO 0);
        CONSTANT ROM : ARR := (


                OPCODE_LW & R0 & SU & INI & x"00", -- CARGA EL NUMERO
DE MUESTRAS AL REGISTRO 0
                OPCODE_LW & R1 & SU & INI & x"01", -- CARGA EL NUMERO 1
POSITIVO AL REGISTRO 1
                OPCODE_LW & R2 & SU & INI & x"01", -- CARGA  EL NUMERO 1
POSITIVO AL REGISTRO 2
                OPCODE_LW & R6 & SU & INI & x"02", -- CARGA EL NUMERO 0
POSITIVO AL REGISTRO 6

                OPCODE_LW & R4 & SU & "101" & x"0D", -- SUMA, PASA    ---
CARGA EL DATO DE LA MEMORIA DE DATOS (DIR[3])
                OPCODE_LW & R3 & SU & "111" & x"03", -- SUMA,
INCREMENTA, PASA  --- CARGA EL PESO DE LA MEMORIA DE DATOS (DIN[13])
                OPCODE_MULT & R4 & R3 & R5 & "000",  --- MULTIPLICA R4 Y
R3  --- MULTIPLICA EL PESO POR LA SEÑAL
                OPCODE_SUMA & R6 & R5 & R6 & "000", -- SUMA R6  Y R5 --
SUMA ACUMULATORIA DE PONDERACION
                OPCODE_SUMA & R2 & R1 & R2 & "000", -- SUMA R2  Y R1 --
INCREMENTA UNA ITERACION AL NUMERO DE MUESTRAS TOMADAS

                OPCODE_BEQ & R0 & R2 & INI & x"04", -- SUMA R6  Y R5 --
PREGUNTA SI LA ITERACION ES IGUAL AL NUMERO DE MUESTRAS

                        -- SI NO ES IGUAL REGRESA EL CONTADOR DEL
PROGRAMA A LA DIRECCION 4

                        -- SI ES IGUAL CONTINUA CON EL PROGRAMA.

                OPCODE_SUMA & R6 & R1 & R6 & "000", -- SUMA EL SESGO A
LA SUMA ACUMULATORIA.

                OPCODE_SW & SU & R6 & INI & x"FF", -- ALMACENA EL
RESULTADO EN LA ULTIMA DIRECCION DE MEMORIA.

                OTHERS=>(OTHERS=>'0')
        );
begin
        D <= ROM( CONV_INTEGER(A) );
end MEMORIA_PROGRAMA0;
```

Contador de Programa

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;



entity CONTADOR_PROGRAMA00 is

   Port ( CLK               : in  STD_LOGIC;
      WPC                  : in  STD_LOGIC;
      DIN           : in  STD_LOGIC_VECTOR (10 downto 0);
            DOUT           : out STD_LOGIC_VECTOR (10 downto 0)

                        );
end CONTADOR_PROGRAMA00;


architecture CONTADOR_PROGRAMA0 of CONTADOR_PROGRAMA00 is

begin

        PC : PROCESS( CLK )
        BEGIN
                IF( RISING_EDGE(CLK) OR FALLING_EDGE(CLK) )THEN
                        IF( WPC = '1' )THEN
```

```
                    DOUT <= DIN;

                END IF;
            END IF;
        END PROCESS PC;



end CONTADOR_PROGRAMA0;
```

Sumador del Contador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity SUMADOR_CONTADOR00 is

    Port ( DIN_IN               : in  STD_LOGIC_VECTOR (10 downto 0);
           DIR_OUT              : out STD_LOGIC_VECTOR (10
downto 0)

                    );
end SUMADOR_CONTADOR00;


architecture SUMADOR_CONTADOR0 of SUMADOR_CONTADOR00 is
begin

DIR_OUT <= DIR_IN + "00000000001";



end SUMADOR_CONTADOR0;
```

Contador de Direcciones.

37. Sumador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;



entity SUMA_DIR00 is

    Port ( CLK          : in  STD_LOGIC;
        FS              : in  STD_LOGIC;
        ACC_IN          : in  STD_LOGIC_VECTOR (7 downto 0);
               DIR_IN   : in  STD_LOGIC_VECTOR (7 downto 0);
               DIR_OUT  : out STD_LOGIC_VECTOR (7 downto 0)

                    );
end SUMA_DIR00;


architecture SUMA_DIR0 of SUMA_DIR00 is

begin

        SD : PROCESS( CLK )
        BEGIN
            IF( RISING_EDGE(CLK) )THEN
                IF( FS = '1' )THEN

                    DIR_OUT <= ACC_IN + DIR_IN;

                END IF;
            END IF;
        END PROCESS SD;



end SUMA_DIR0;
```

38. Acumulador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;



entity ACC_DIR00 is

    Port ( CLK              : in  STD_LOGIC;
        RESET       : in  STD_LOGIC;
                FI          : in  STD_LOGIC;
                CODOP : in  STD_LOGIC_VECTOR (4 downto 0);
                DIR_OUT     : BUFFER STD_LOGIC_VECTOR (7 downto 0)

                    );
end ACC_DIR00;


architecture ACC_DIR0 of ACC_DIR00 is

begin

        AD : PROCESS( CLK, RESET )
        BEGIN

        if reset = '1' then
                DIR_OUT <= "00000000";

        ELSIF( FAllING_EDGE(CLK) )THEN
                        IF( CODOP = "00000" )THEN
                                IF( FI = '1' )THEN

                                DIR_OUT <= DIR_OUT + 1;

                                END IF;
                        END IF;
        END IF;

        END PROCESS AD;

end ACC_DIR0;
```

39. Paquete del contador de direcciones

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_CONTDIR is


component SUMA_DIR00
port(

                CLK             : in  STD_LOGIC;
        FS          : in  STD_LOGIC;
        ACC_IN      : in  STD_LOGIC_VECTOR (7 downto 0);
                DIR_IN : in  STD_LOGIC_VECTOR (7 downto 0);
                DIR_OUT         : out STD_LOGIC_VECTOR (7 downto 0)

);
end component;




component ACC_DIR00
port(

                CLK             : in  STD_LOGIC;
        RESET       : in  STD_LOGIC;
                FI              : in  STD_LOGIC;
                CODOP : in  STD_LOGIC_VECTOR (4 downto 0);

                DIR_OUT         : BUFFER STD_LOGIC_VECTOR (7 downto 0)

);
end component;
```

```
end PAQUETE_CONTDIR;
```

40. TOP del Contador de Direcciones

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use paquete_contdir.all;


entity CONTADOR_DIRECCIONES00 is

    Port ( CLK                : in  STD_LOGIC;
           RESET              : in  STD_LOGIC;


           CODOP : in  STD_LOGIC_VECTOR (4 downto 0);
      DIR          : in  STD_LOGIC_VECTOR (10 downto 0);

           FP_O               : OUT  STD_LOGIC;

           DIR_OUT            : out STD_LOGIC_VECTOR (7 downto 0)

                      );
end CONTADOR_DIRECCIONES00;


architecture CONTADOR_DIRECCIONES0 of CONTADOR_DIRECCIONES00 is

signal ACCR: std_logic_vector(7 downto 0);
begin

U1: SUMA_DIR00 port map(

           CLK => CLK,
      FS => DIR(10),
      ACC_IN => ACCR,
           DIR_IN  => DIR(7 DOWNTO 0) ,
           DIR_OUT => DIR_OUT

  );


U2: ACC_DIR00 port map(
           CLK => CLK,
      RESET  => RESET ,
           FI  => DIR(9),
           CODOP  => CODOP,

           DIR_OUT          => ACCR

  );

      FP_O <= DIR(8) when CODOP = "00000" ELSE '0';


end CONTADOR_DIRECCIONES0;
```

Unidad de Control

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;



entity CONTROL00 is

   Port ( CLK                  : in  STD_LOGIC;
              RESET  : in std_logic;
              ENABLE              : in STD_LOGIC;
       CODOP                : in  STD_LOGIC_VECTOR(4 DOWNTO 0);
              BANDERAS : out STD_LOGIC_VECTOR (13 downto 0)
                    );
end CONTROL00;


architecture CONTROL0 of CONTROL00 is

begin

          PC : PROCESS( CLK, RESET, CODOP)
          BEGIN

IF (RESET = '1') THEN
          BANDERAS <= "00000000000000";
          END IF;


-----------------------------------------------------------
          IF( CLK'EVENT OR CLK = '1' )THEN -- DURANTE EL FLANCO DE SUBIDA


           IF (RESET = '0') THEN
IF (CODOP = "00000") THEN --- LW
          BANDERAS <= "01000000000000";
END IF;

IF (CODOP = "10000") THEN -- SW
          BANDERAS <= "00001000000000";
END IF;

IF (CODOP = "00001") THEN --- ADD
          BANDERAS <= "01100100000001";
END IF;

IF (CODOP = "00010") THEN --- SUB
          BANDERAS <= "01100100000010";
END IF;

IF (CODOP = "00011") THEN --- MULT
          BANDERAS <= "01100100000011";
END IF;

IF (CODOP = "00100") THEN --- DIV
          BANDERAS <= "01100100000100";
END IF;

IF (CODOP = "10010") THEN --- BEQ
          BANDERAS <= "10000010000010";
END IF;

IF (CODOP = "10001") THEN --- B
          BANDERAS <= "10000000000000";
END IF;

IF (CODOP = "11111") THEN --- NOP
          BANDERAS <= "00000000000000";
END IF;
```

```
        END IF;

END IF;

        IF(FALLING_EDGE(CLK))THEN -- INC PC
                IF (RESET = '0') THEN
                                BANDERAS <= "00000000100000";
                END IF;
        END IF;

                END PROCESS PC;

end CONTROL0;
```

Paquete del Procesador de Punto Flotante

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package PAQUETE_PROCESADOR is

------------------------------------------------------ UNIDAD DE CONTROL
component CONTROL00
port(

                CLK             : in  STD_LOGIC;
                RESET   : in std_logic;
        CODOP           : in  STD_LOGIC_VECTOR(4 DOWNTO 0);

                BANDERAS : out STD_LOGIC_VECTOR (13 downto 0)

);
end component;



-------------------------------------------------------- UNIDAD DE PUNTO FLOTANTE
component UPF00
port(

        A: in std_logic_vector(31 downto 0) ;
        B: in std_logic_vector(31 downto 0) ;
        CODOP: in std_logic_vector(4 downto 0) ;

        A_DESNORMALIZADO: out std_logic;
        B_DESNORMALIZADO: out std_logic;
        MAS_INF: out std_logic;
        MENOS_INF: out std_logic;
        CERO_MAS: out std_logic;
        CERO_MENOS: out std_logic;
        DESNORMALIZADO: out std_logic;
        NAN: out std_logic;
        OVERFLOW: out std_logic;
        UNDERFLOW: out std_logic;
        ZERODIV: out std_logic;


OPERRONEO: out std_logic;
        RESULTADO: out std_logic_vector(31 downto 0)

);
end component;



----------------------------------------------------- CONTADOR DE PROGRAMA
component CONTADOR_PROGRAMA00
port(

                CLK             : in  STD_LOGIC;
        WPC                     : in  STD_LOGIC;
                RESET : in std_logic;
        DIN             : in  STD_LOGIC_VECTOR (10 downto 0);
                DOUT            : out STD_LOGIC_VECTOR (10 downto 0)

);
end component;



------------------------------------------------------ MEMORIA DE DATOS
```

```
component MEMORIA_DATOS00
port(

                        CLK : in  STD_LOGIC;
        ADDR : in  STD_LOGIC_VECTOR (7 downto 0);
        DIN : in  STD_LOGIC_VECTOR (31 downto 0);
        WE : in  STD_LOGIC;
        DOUT : out  STD_LOGIC_VECTOR (31 downto 0)

);
end component;



------------------------------------------------------ MEMORIA DE PROGRAMA
component MEMORIA_PROGRAMA00
port(

                        A : in  STD_LOGIC_VECTOR (10 downto 0);
        D : out  STD_LOGIC_VECTOR (31 downto 0)

);
end component;



-------------------------------------------------------- SUMADOR DEL CONTADOR
component SUMADOR_CONTADOR00
port(

                        DIR_IN              : in  STD_LOGIC_VECTOR (10 downto 0);
                        DIR_OUT                          : out STD_LOGIC_VECTOR (10
downto 0)

);
end component;



------------------------------------------------------ BANDO DE REGISTROS
component BANCO_REGISTROS00
port(

                        CLK                 : in  STD_LOGIC;
        WE                                  : in  STD_LOGIC;
        ADDR_WR     : in  STD_LOGIC_VECTOR (7 downto 0);
                                        ADDR_RD1           : in  STD_LOGIC_VECTOR (7 downto
0);
                                        ADDR_RD2           : in  STD_LOGIC_VECTOR (7 downto
0);
        DIN                         : in  STD_LOGIC_VECTOR (31 downto 0);
        DOUT1                       : out STD_LOGIC_VECTOR (31 downto 0);
                        DOUT2                : out STD_LOGIC_VECTOR (31 downto 0)

);
end component;



-------------------------------------------------------- CONTADOR DE DIRECCIONES
component CONTADOR_DIRECCIONES00
port(

                        CLK                 : in  STD_LOGIC;
                RESET           : in  STD_LOGIC;


                        CODOP : in  STD_LOGIC_VECTOR (4 downto 0);
        DIR             : in  STD_LOGIC_VECTOR (10 downto 0);

                        FP_O    : OUT  STD_LOGIC;

                        DIR_OUT            : out STD_LOGIC_VECTOR (7 downto 0)

);
end component;


end PAQUETE_PROCESADOR;
```

TOP del Procesador de Punto Flotante para el Analisis de Señales.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use paquete_procesador.all;


entity PROCESADOR00 is

   Port ( CLK               : in  STD_LOGIC;
          RESET             : in  STD_LOGIC;
          SEL               : in std_logic;


          CP                : OUT  STD_LOGIC_VECTOR (3 downto 0);
     FLAGS                  : OUT  STD_LOGIC_VECTOR (13 downto 0);
          BANDERAS_UPF: OUT STD_LOGIC_VECTOR(11 DOWNTO 0);

          MEM_OUT255        : out STD_LOGIC_VECTOR (31 downto 0)

                 );
end PROCESADOR00;


architecture PROCESADOR0 of PROCESADOR00 is
signal F_MUXCONT, F_WRITEENABLE, F_MUXOP, F_READDATA, F_WRITEDATA,
F_MUXSAL, F_COMPAND, F_MUXBRANCH, F_WPC, F_PASA, MASCERO,
MENOSCERO: STD_LOGIC;
signal OPUPF: STD_LOGIC_VECTOR(4 DOWNTO 0);

SIGNAL SAL_MUXCONT, BUS_CP, BUS_SUM, SAL_MUXBRANCH:
STD_LOGIC_VECTOR(10 DOWNTO 0);
SIGNAL SAL_MUXCONTDIR, SAL_MUXOP, BUS_CONTDIR: STD_LOGIC_VECTOR(7
DOWNTO 0);

signal B_CONTROL: STD_LOGIC_VECTOR(13 DOWNTO 0);

signal BUS_PROGRAMA, BUS_RD1, BUS_RD2, BUS_RESULTADO, BUS_DATOS,
SAL_MUXSAL: std_logic_vector(31 downto 0);

begin

------------------------------------ UNIDAD DE CONTROL
U1: CONTROL00 port map(


          CLK => CLK,
          RESET => RESET,
     CODOP  => BUS_PROGRAMA(31 DOWNTO 27),

          BANDERAS  => B_CONTROL

  );
------------------------------------ UNIDAD DE PUNTO FLOTANTE
U2: UPF00 port map(

     A => BUS_RD1,
     B => BUS_RD2,
     CODOP => B_CONTROL(4 DOWNTO 0),

     A_DESNORMALIZADO => BANDERAS_UPF(11) ,
     B_DESNORMALIZADO => BANDERAS_UPF(10),


     MAS_INF => BANDERAS_UPF(9),
     MENOS_INF => BANDERAS_UPF(8),
     CERO_MAS => MASCERO,
     CERO_MENOS => MENOSCERO,
     DESNORMALIZADO => BANDERAS_UPF(5),
     NAN => BANDERAS_UPF(4),
     OVERFLOW => BANDERAS_UPF(3),
     UNDERFLOW => BANDERAS_UPF(2),
     ZERODIV => BANDERAS_UPF(1),
     OPERRONEO => BANDERAS_UPF(0),
     RESULTADO => BUS_RESULTADO  );

------------------------------------ CONTADOR DE PROGRAMA
U3: CONTADOR_PROGRAMA00 port map(

          CLK => CLK,
     WPC  => B_CONTROL(5),
          RESET  => RESET,
     DIN         => SAL_MUXCONT,
          DOUT  => BUS_CP

  );

------------------------------------ MEMORIA DE DATOS
U4: MEMORIA_DATOS00 port map(

          CLK => CLK,
     ADDR  => SAL_MUXCONTDIR,
     DIN  => BUS_RD2,
     WE  => B_CONTROL(9),
```

```
        DOUT  => BUS_DATOS

  );


------------------------------------- MEMORIA DE PROGRAMA
U5: MEMORIA_PROGRAMA00 port map(

                            A => BUS_CP,
        D => BUS_PROGRAMA

  );

------------------------------------- SUMADOR DEL CONTADOR
U6: SUMADOR_CONTADOR00 port map(

                            DIR_IN  => BUS_CP,
                            DIR_OUT             => BUS_SUM

  );

------------------------------------- BANCO DE REGISTROS
U7: BANCO_REGISTROS00 port map(

                            CLK => CLK,
        WE  => B_CONTROL(12),
        ADDR_WR => SAL_MUXOP,
                                        ADDR_RD1  => BUS_PROGRAMA(26 DOWNTO 19),
                                        ADDR_RD2            => BUS_PROGRAMA(18 DOWNTO
11),
        DIN  => SAL_MUXSAL,
        DOUT1  => BUS_RD1,
                            DOUT2  => BUS_RD2

  );

------------------------------------- CONTADOR DE DIRECCIONES
U8: CONTADOR_DIRECCIONES00 port map(

                            CLK => CLK,
                RESET => RESET,

                            CODOP  => BUS_PROGRAMA(31 DOWNTO 27),
        DIR  => BUS_PROGRAMA(10 DOWNTO 0),

                            FP_O  => F_PASA,

                            DIR_OUT =>   BUS_CONTDIR
  );

  ---------------------------------------------------------------------- MULTIPLEXOR ANTES DEL
CONTADOR DE PROGRAMA
  SAL_MUXCONT <= BUS_SUM WHEN (B_CONTROL(13) = '0') ELSE
SAL_MUXBRANCH;

  ---------------------------------------------------------------------- MULTIPLEXOR DE
SALTOOOS
  SAL_MUXBRANCH <= BUS_PROGRAMA(10 DOWNTO 0) WHEN (F_MUXBRANCH
= '0') ELSE BUS_SUM;

  ----------------------------------------------------------------------- AND DEL SALTO
  F_MUXBRANCH <= B_CONTROL(7) AND (MASCERO OR MENOSCERO);

  BANDERAS_UPF(7) <= MASCERO;
  BANDERAS_UPF(6) <= MENOSCERO;

  ----------------------------------------------------------------------- MULTIPLEXOR DEL
CONTADOR DE DIRECCIONES
  SAL_MUXCONTDIR <= BUS_PROGRAMA(7 DOWNTO 0) WHEN (F_PASA='0') ELSE
BUS_CONTDIR;

  ---------------------------------------------------------------------- MULTIPLEXOR SELECTOR DE
OPERANDO
  SAL_MUXOP <= BUS_PROGRAMA(26 DOWNTO 19) WHEN (B_CONTROL(11) = '0')
ELSE BUS_PROGRAMA(10 DOWNTO 3);

  ----------------------------------------------------------------------  MULTIPLEXOR  SALIDA DE
RESULTADOS
  SAL_MUXSAL <= BUS_DATOS WHEN (B_CONTROL(8)='0') ELSE
BUS_RESULTADO;

  CP <= BUS_CP(3 DOWNTO 0);
  FLAGS <= B_CONTROL;



  MEM_OUT255 <= BUS_DATOS when (BUS_DATOS /=
"10111111100110011110001000001100" AND (SEL = '0') ) ELSE
                                                    X"00000000" WHEN ((BUS_RESULTADO =
"01000000100000000000000000000000")) ELSE BUS_RESULTADO;

end PROCESADOR
```

BIBLIOGRAFÍA

[1] AESOFT, «AESOFT,» Septiembre 2011. [En línea]. Available: http://www.aesoft.com.ec/. [Último acceso: Septiembre 2014].

[2] AESOFT, «AESOFT,» Septiembre 2011. [En línea]. Available: http://www.aesoft.com.ec/proyectos/mercado.pdf. [Último acceso: Septiembre 2014].

[3] A. Grediaga Olivo y J. Pérez Martínez, «Procesadores,» de *Diseño de Procesadores en VHDL*, Primera ed., Alicante, Publicaciones Universidad de Alicante, p. 225.

[4] M. S. University, «Montana State - College of Engineering,» 2011. [En línea]. Available: http://www.coe.montana.edu/.

[5] M. Sangwan y A. A. A., «Design and Implementation of Single Precision Pipelines Floating Point Co-Processor,» de *Intenational Conference on Advanced Electronic Systems (ICAES)*, New Delhi, 2013.

[6] V. H. García Ortega y et al., «Microprocesador didactico de arquitectura RISC implementado en un FPGA,» *e-Gnosis,* vol. Esp, pp. 1-8, 2009.

[7] V. H. García Ortega, *Trabajo Terminal 851,* D.F, 2006.

[8] S. G. Shiva, Computer Organization, Design, and Architecture, Quinta ed., Boca Raton, FL.: CRC Press Taylor & Francis Group, 2014.

[9] J. Martin A, UFC0465: Montaje de componentes y periféricos microinformáticos, España: IC Editorial, 2012.

[10] A. V. Deshmukh, Microcontrollers: Theory and Applications, New Delhi: Tata McGraw-Hill, 2005.

[11] M. Slater, A Guide to RISC Microprocessors, San Diego, California: Academic Press, INC..

[12] N. H. E. Weste y D. Money H., CMOS VLSI DESIGN a circuits and systems perspective, Massachusetts, USA: Pearson, 2011.

[13] D. Money H. y S. L. Harris, Digital Design and Computer Architecture, San Francisco, CA: Elsevier, 2014.

[14] B. Zeidman, «Introduction to CPLD and FPGA Design,» [En línea]. Available: www.chalknet.com.

[15] L. Semiconductor, «User's Guide "ispMACH4256ZE Breakout Board Evaluation Kit",» 2012. [En línea]. Available: www.latticesemi.com.

[16] L. Semiconductor, «User's Guide: "MachXO2 Breakout Board Evaluation Kit",» Enero 2014. [En línea]. Available: www.latticesemi.com. [Último acceso: 2015].

[17] L. Fauset, Fundamentals od Neural Networks, Architectures, algorithms, and aplications, United States of America: Prentice Hall, Englewood Cliffs, 1994.

[18] e. a. Martin T. Hagan, Neural Network Design, Boston: PWS Publishing Company, 1996.

[19] J. Cavanagh, Computer Arithmetic and Verilog HDL, CRC Press Taylor & Francis Group, 2009.