

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DESARROLLO DE UN MICROPROCESADOR ARM7

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL
ELECTRICISTA

PATRICIO ISAÍAS SOTO KORT

PROFESOR GUÍA:
VÍCTOR GRIMBLATT HINZPETER

MIEMBROS DE LA COMISIÓN:
HELMUTH THIEMER WILKENS
NICOLÁS HUMBERTO BELTRÁN MATURANA

SANTIAGO DE CHILE
MARZO DE 2008

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: PATRICIO ISAÍAS SOTO KORT
FECHA: 10 DE MARZO DE 2008
PROF. GUÍA: Sr. VÍCTOR GRIMBLATT HINZPETER

“DESARROLLO DE UN MICROPROCESADOR ARM7”

Este trabajo de título tiene como objetivo generar la capacidad de diseñar circuitos integrados digitales por medio de un dispositivo específico, un microprocesador ARM7. Se utilizan lenguajes de descripción de hardware que permiten utilizar una metodología *top-down*, comenzando por un modelo comportamental de alto nivel del circuito hasta obtener un modelo estructural.

El desarrollo contempla la utilización inicial de la herramienta ArchC, la cual permite obtener una descripción de un procesador de manera relativamente sencilla en lenguaje SystemC. Con este modelo diseñado se podrán realizar pruebas que comprueben el funcionamiento del procesador, para continuar con su descripción en lenguaje Verilog. Este último tiene la capacidad de describir circuitos a más bajo nivel y posee la ventaja de estar más maduro por la cantidad de años que ha sido utilizado por ingenieros.

El diseño del microprocesador en Verilog se realiza programando cada módulo que conforma el *datapath* junto al módulo de control que genera las señales necesarias para ejecutar una instrucción. La programación de estos módulos se realiza considerando las estructuras conocidas por la literatura relacionada al tema y diseñando secciones que no se encuentran documentadas de forma libre. A partir de esta programación se genera un modelo Verilog que combina descripciones comportamentales y estructurales.

Cada módulo debe ser probado de forma individual, asegurando su funcionalidad de forma independiente, para luego realizar una validación global del sistema. La verificación del modelo es una constante durante su diseño, así en cada etapa se realizan pruebas generales al microprocesador con programas escritos de manera que abarquen todas las instrucciones y los casos límites. Muchos errores son detectados solamente cuando el microprocesador es probado con una secuencia de instrucciones.

En conclusión se obtuvo un microprocesador funcional, con las instrucciones más importantes implementadas. Las pruebas fueron satisfactorias logrando corregir muchos errores a causa de desconocimiento y/o descuido. El objetivo principal se cumplió debido a que en el trabajo se ha debido aprender tanto acerca de sistemas digitales como de lenguajes de descripción de hardware. Estos últimos requieren mucha experiencia en su uso, ya que describen algoritmos de forma paralela y no secuencial como los típicos lenguajes de software a los cuales la mayoría está acostumbrado.

Finalmente queda a disposición un modelo que puede ser utilizado con fines académicos y con la posibilidad de completar su funcionalidad y/o ser usado como núcleo de un sistema sobre una FPGA.

a mi familia y amigos

Agradecimientos

Quiero agradecer a mi familia, la cual me acompaño siempre en este largo camino, lleno de cosas buenas y malas. Un agradecimiento especial a mi madre, Verónica, quien me enseñó a ser la persona que soy ahora, que me enseñó a ser responsable y constante. También destacar a mis hermanos, Emilio, Humberto y Francisco quienes me influenciaron a ser una buena persona y me enseñaron distintas formas de ver la vida y disfrutarla.

Agradezco a mi profesor guía Víctor Grimblatt, que fue quien me dió a conocer el tema de la electrónica desde otra especialidad, circuitos integrados digitales, la cual combina mi gusto por los circuitos y por la programación. Sin su propuesta de tema y su apoyo durante el trabajo no hubiese sido posible esta tesis.

No puedo dejar de agradecer a mis amigos, Kashu, Julio, Llopez, Tollero, Negro, Cata, Isa, Tuno, JP, Pollo, Diego, Cepech y tantos otros quienes estuvieron conmigo a lo largo de la carrera y me apoyaron en todo momento para realizar este trabajo.

Por último agradezco muchas otras personas que aunque no las nombro, fueron parte importante en mi vida, aportando consejos, ayudas, sonrisas y mucho ánimo.

Índice general

1. Introducción	1
1.1. Objetivos	1
1.2. Alcances	2
1.3. Antecedentes	2
1.3.1. Lenguajes de descripción de hardware	2
1.3.2. Microprocesador ARM7	4
1.3.3. Metodología	6
2. Especificaciones del microprocesador ARM7	8
2.1. Modelo de programador	8
2.2. Set de instrucciones	10
2.3. Pipeline, tiempos y señales	11
2.4. Resumen de especificaciones	13
3. Modelo del microprocesador en ArchC	15
3.1. Diseño	15
3.2. Validación	17
4. Modelo en Verilog del microprocesador	19
4.1. Características del modelo	19
4.2. Diseño	19
4.2.1. Registro de dirección - <i>Address Register</i>	20
4.2.2. Incrementador de dirección - <i>Address Incrementer</i>	20
4.2.3. Banco de registros - <i>Register Bank</i>	21
4.2.4. Multiplicador - <i>Multiplier</i>	22
4.2.5. Desplazador rápido - <i>Barrel Shifter</i>	22
4.2.6. Unidad Aritmético Lógica - <i>Arithmetic logic unit</i>	23
4.2.7. Registro de datos de escritura - <i>Write data register</i>	24
4.2.8. Registro de datos de escritura - <i>Read data register</i>	24
4.2.9. Pipeline para instrucciones - <i>Instruction Pipeline</i>	26
4.2.10. Control principal - <i>Main Control</i>	26
4.3. Validación	28
4.3.1. Método	28

ÍNDICE GENERAL

4.3.2. Ejemplo de ejecución de código	29
5. Conclusiones	33
Bibliografía	34
A. Descripción de señales de control	36
B. Descripción de módulos	39
B.1. Banco de registros	39
B.2. Multiplicador	39
B.3. Control de transmisión de datos por bloque	39
C. Diagrama completo de ARM7	43
D. Archivos de modelo de microprocesador	45

Índice de figuras

1.1. Diagrama del núcleo ARM7TDMI.	5
1.2. Flujo de diseño a utilizar.	7
2.1. Registros general y <i>Program Counter</i>	9
2.2. Formato del registro de estado de programa	10
2.3. Formatos del set de instrucciones.	11
2.4. Fases no traslapadas a partir del reloj.	13
3.1. Proceso para obtención de simulador en ArchC.	16
4.1. Diagrama de registro de dirección.	20
4.2. Diagrama de incrementador de dirección.	21
4.3. Diagrama del entorno del banco de registros.	21
4.4. Diagrama de desplazador (<i>barrel shifter</i>).	23
4.5. Diagrama de unidad aritmética lógica.	24
4.6. Diagrama de registro de datos de escritura.	25
4.7. Diagrama de registro de datos de lectura.	25
4.8. Diagrama de pipeline de instrucciones.	26
4.9. Diagrama de control principal.	27
4.10. Señales del microprocesador en prueba de multiplicación (1)	31
4.11. Señales del microprocesador en prueba de multiplicación (1)	32
B.1. Diagrama interior de banco de registros.	40
B.2. Estructura del multiplicador.	41
B.3. Generador de producto parcial.	41
B.4. Estructura de control de transmisión de datos por bloque.	42
C.1. Diagrama de microprocesador ARM7.	44

Índice de tablas

2.1. Vectores de excepciones	10
2.2. Set de instrucciones implementado	12
B.1. Enumeración de registros	40
B.2. Enumeración de registros	41

Capítulo 1

Introducción

En la actualidad compartimos con toda clase de circuitos integrados de forma continua en nuestra vida cotidiana. Los computadores se ha convertido en una herramienta fundamental para gran parte de los trabajos existentes, teléfonos móviles son de uso común, dispositivos para escuchar música y ver videos se hacen cada vez más populares, etc. Es por esto que en este proceso de integración de la tecnología en la sociedad ha sido necesario generar herramientas que permiten desarrollar hardware de manera fácil y sin fallas, con tal de satisfacer los requerimientos de los consumidores junto con acelerar y abaratar los procesos de diseño, prueba y fabricación de los dispositivos electrónicos.

Parte de las herramientas claves para el desarrollo de sistemas electrónicos son los lenguajes de descripción de hardware. Estos permiten por medio de texto describir el comportamiento o estructura de un circuito y ,con el software indicado, obtener la lista de componentes e interconexiones necesarias para fabricar un dispositivo.

Es así como quienes deseen desarrollar conceptos, sistemas o circuitos con funcionalidad comprobada deben conocer las distintas características tanto del sistema a desarrollar como del lenguaje de descripción, por medio del estudio y la utilización de las herramientas involucradas, siguiendo una metodología que permita diseñar y verificar el circuito logrando un producto final útil para quien lo necesite.

1.1. Objetivos

El objetivo general de este trabajo es generar la capacidad de desarrollar un circuito integrado digital por medio de un dispositivo específico como lo es un microprocesador ARM7, de manera de utilizar las herramientas que existen disponibles, conociendo la metodología utilizada en el proceso y aprender de forma práctica acerca de las ventajas y desventajas asociadas a los lenguajes de descripción de sistemas.

Los objetivos específicos son los siguientes:

- Modelar el microprocesador ARM7 en SystemC con ayuda de ArchC.
- Obtener un modelo sintetizable del microprocesador ARM7 en Verilog a partir del modelo comportamental.
- Validar los modelos obtenidos por medio de programas de prueba definidos para este propósito.
- Generar una descripción a nivel de compuertas (*netlist*) sintetizando el modelo RTL del microprocesador ARM7.

1.2. Alcances

Este trabajo contempla el desarrollo de un microprocesador ARM7 por medio de un lenguaje de descripción de sistemas y se considerarán los siguientes puntos:

- El set de instrucciones del modelo será el básico, sin incluir funciones del set THUMB característico del modelo ARM7TDMI e instrucciones de coprocesador.
- Traspaso del modelo desde SystemC a Verilog para poder realizar la síntesis del modelo.
- Proceso de síntesis completo, es decir traducción, optimización y mapeo del circuito.

El modelo será verificado y validado por medio programas de prueba y evaluaciones comparativas, las cuales se realizarán en pasos iterativos del desarrollo y el traspaso del modelo comportamental a descripción RTL.

1.3. Antecedentes

A continuación se explican diferentes elementos utilizados en este trabajo, necesario para comprender el contexto de este trabajo.

1.3.1. Lenguajes de descripción de hardware

Un lenguaje de descripción de hardware es una clase de lenguaje de computador que permite describir circuitos electrónicos. Esta descripción puede ser procesada de distintas maneras con el objetivo de generar un diseño que funcione y pueda ser fabricado. Básicamente estos lenguajes permiten:

- Diseñar
- Optimizar
- Verificar

El *diseño* de un circuito es parte de la motivación del ingeniero para utilizar un lenguaje de descripción de hardware pero no la única. Un modelo puede ser diseñado en distintos niveles de abstracción y dependerá del lenguaje, las opciones de niveles en que se pueda describir un circuito. Se habla de un modelo comportamental cuando se describen las acciones que lleva a cabo un sistema y no se intenta observar el detalle de las componentes que tendrá el circuito a nivel físico. En cambio un modelo estructural busca definir claramente los componentes que interactúan en el sistema y como están interconectados entre sí.

La *optimización* también se puede realizar a distintos niveles de abstracción aunque la diferencia se encuentra en el nivel de automatización del proceso. Al diseñar un modelo comportamental, la optimización es comúnmente realizada de forma manual debido a que este modelo plasma la idea del diseñador y es éste quien puede cambiar la configuración del diseño más fácilmente. En un modelo estructural o en general de más bajo nivel, los componentes pueden ser optimizados de manera automática con mayor facilidad con tal de disminuir su cantidad o aumentar la velocidad del circuito y a la vez no cambiar el comportamiento del sistema.

La *verificación* de un sistema es parte fundamental del diseñador, con el deseo de encontrar un circuito sin fallas se pueden realizar pruebas al diseño sin grandes costos como el fabricar el dispositivo y probarlo. Gracias a los lenguajes de descripción de hardware se puede simular el sistema, someterlo a pruebas con condiciones límites y todo esto por medio de software.

En este trabajo se utilizan principalmente dos lenguajes de descripción de hardware, SystemC y Verilog, los cuales se describen a continuación.

SystemC

SystemC es un lenguaje hecho en C++ estándar extendiendo el lenguaje con el uso de librerías de clases [4]. Se dice comúnmente que es un lenguaje de descripción de sistemas, permitiendo un diseño de alto nivel con la posibilidad de verificación del sistema de forma integrada abarcando hardware y software. Este lenguaje es apropiado para particionamiento de sistemas, evaluación y verificación en el asignamiento de bloques para la implementación de hardware o software, y para la arquitectura y medición de las interacciones entre bloques funcionales.

Con el auge de las plataformas SoC¹, SystemC se convierte en una alternativa deseable para el desarrollo de éstas, donde es necesario analizar la interacción de los componentes a nivel de sistema y disminuir el tiempo utilizado en el proceso.

Junto a esta iniciativa nace una nueva herramienta que hace uso de SystemC, llamada ArchC[11]. ArchC es un lenguaje de descripción de arquitectura diseñado por el Laboratorio

¹SoC: Sigla que significa *System on a Chip*, plataforma compuesta por variados componentes electrónicos como microprocesador, interfaz de comunicación, memorias, regulador de voltaje, etc. reunidos en un solo circuito integrado.

de Sistemas Computacionales del Instituto de Computación de la Universidad de Campinas. Este lenguaje está pensado para evaluar rápidamente áreas como: diseño de procesadores y set de instrucciones, jerarquías de memoria y otros aspectos de la investigación sobre arquitectura de computadores. Actualmente ArchC se encuentra en su versión 2.0 en fase beta y sus desarrollo algo detenido, aunque continua siendo una herramienta muy útil para analizar un microprocesador.

En primera instancia el modelo del microprocesador en este trabajo ha sido hecho con ArchC, el cual entrega una forma fácil de generar código SystemC para el dispositivo. Más adelante se explicará con más detalle la utilización de ArchC.

Verilog

Verilog es un acrónimo para *Verifying Logic* y es un HDL² usado para el diseño y documentación de sistemas electrónicos[7]. Permite el diseño de un hardware en variados niveles de abstracción y fue diseñado para tener una sintaxis similar al lenguaje de programación C de manera de ser familiar para los ingenieros y ser aceptado fácilmente.

A diferencia de SystemC, este lenguaje ya está bastante maduro y es muy utilizado por diseñadores de hardware, con la ventaja de tener todas las herramientas necesarias para realizar síntesis de circuitos. Por esta razón, en este trabajo se traspasará el modelo del procesador a una descripción en Verilog, ya que la síntesis para lenguaje SystemC es muy inmadura todavía y no asegura buenos resultados.

1.3.2. Microprocesador ARM7

El microprocesador ARM7 forma parte de la familia de *Advanced RISC Machines* (ARM) de microprocesadores de 32 bits para propósito general, los cuales ofrecen alto rendimiento a muy bajo consumo y costo[6]. Su arquitectura está diseñada con el objetivo de ser simple pero poderosa, por esta razón ha sido escogido este procesador para desarrollarlo en este trabajo. Las principales características que posee son:

- Bus de datos de 32 bits
- Set de instrucciones RISC³
- Arquitectura Von Neumann⁴
- Pipeline de 3 etapas: Fetch, Decode y Execute

²HDL: Sigla que significa *Hardware Description Language*.

³RISC: Sigla que significa *Reduced Instruction Set Computer*.

⁴La arquitectura Von Neumann corresponde a procesadores con un solo bus de datos y direcciones tanto para datos como para instrucciones

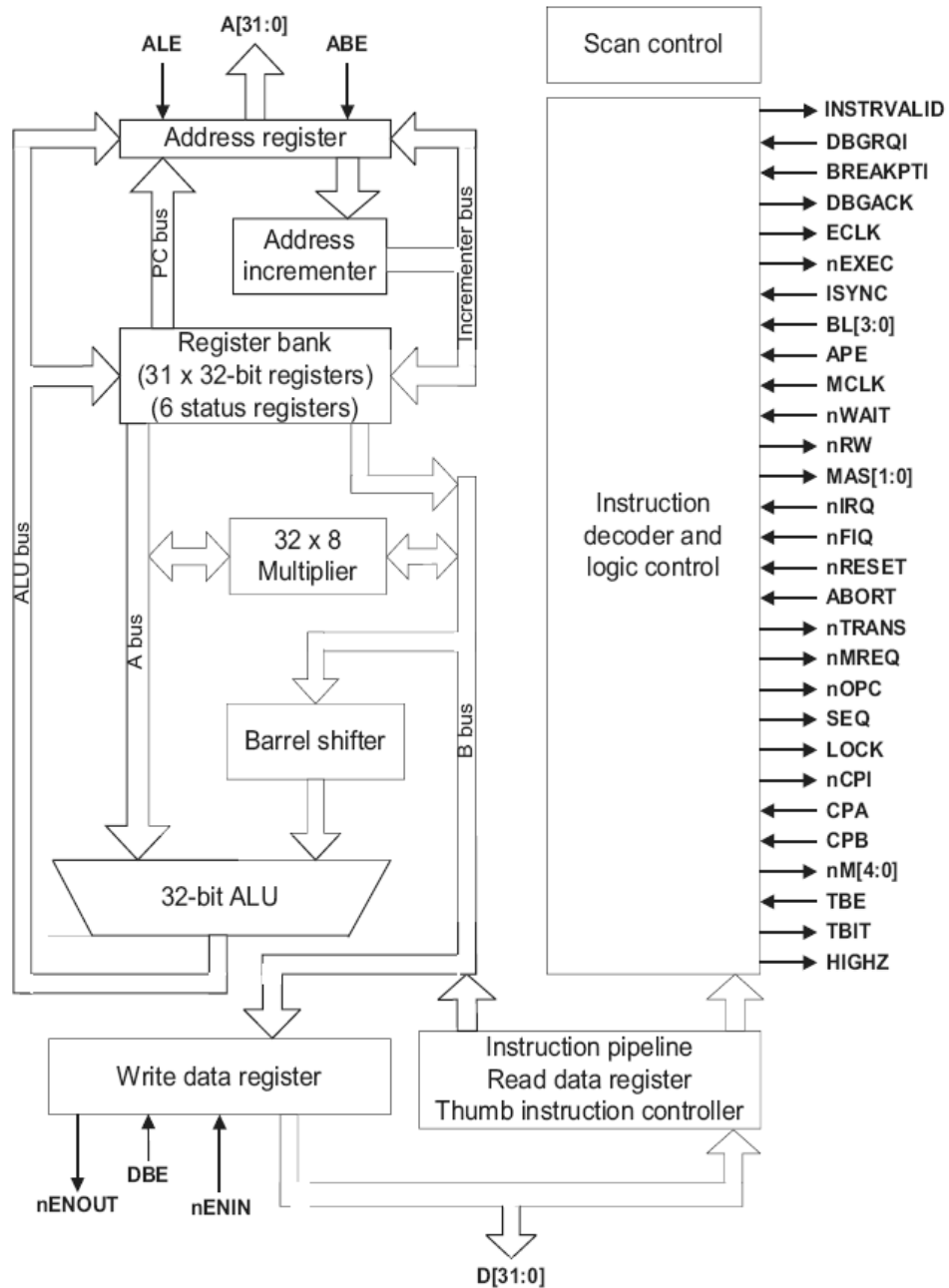


Figura 1.1: Diagrama del núcleo ARM7TDMI.

Este microprocesador posee los bloques que se pueden ver en la figura 1.1, los cuales le dan una funcionalidad básica. Otras funciones pueden agregarse al procesador por medio del manejo de coprocesador que posee el dispositivo, por ejemplo se puede agregar la capacidad de manejar números en punto flotante.

Es importante destacar que el trabajo se realiza en base al microprocesador ARM7TDMI el cual posee un set de instrucciones adicional llamado THUMB. Este set está formado por instrucciones de 16 bits y permite mayor densidad de código. Además el ARM7TDMI posee una interfaz de depuración que junto al set de instrucciones THUMB no será implementado en este trabajo, pero que es fácilmente ampliable para alguien que desee el procesador con todas sus funciones.

1.3.3. Metodología

Dentro del diseño de circuitos integrados con herramientas EDA⁵, el típico flujo se realiza a través de distintos niveles de abstracción de un circuito tanto partiendo desde el concepto hasta llegar al circuito físico (metodología *top-down*) o viceversa (metodología *bottom-up*). Actualmente el diseño de hardware va a la par con el diseño de software, debido a que en gran parte de los casos se buscan sistemas para aplicaciones específicas, y por la necesidad de disminuir el tiempo de desarrollo para llegar al mercado, se opta por utilizar metodologías traslapadas de hardware y software, exigiendo una coordinación entre ambos diseños. Al contrario de esto, en este trabajo no existe un interés por desarrollar software por lo que la metodología utilizada no contempla etapas de este tipo.

Al utilizar SystemC es posible ocupar una metodología de diseño y verificación basada en un modelo de alto nivel, obteniendo un modelo traducido desde SystemC a Verilog[13]. En este caso se utilizará el flujo de diseño de la figura 1.2, donde el modelo comportamental será descrito en ArchC/SystemC y traspasado a un diseño Verilog manualmente aprovechando el conocimiento adquirido al diseñar el modelo en ArchC.

Básicamente el flujo mostrado comprende los siguientes puntos:

1. Especificaciones del dispositivo (funciones que realiza y forma de realizarlas)
2. Descripción comportamental en ArchC
3. Verificación de la descripción comportamental de acuerdo a las especificaciones por medio de programas ejecutados en el simulador del microprocesador.
4. Traducción a Verilog de forma manual.
5. Verificación de descripción en Verilog utilizando los mismos programas de prueba anteriores y comparando los comportamientos de ambos modelos.

⁵EDA: Sigla para *Electronic Design Automation*. Refiere a las herramientas computacionales hechas para el diseño y producción de sistemas electrónicos.

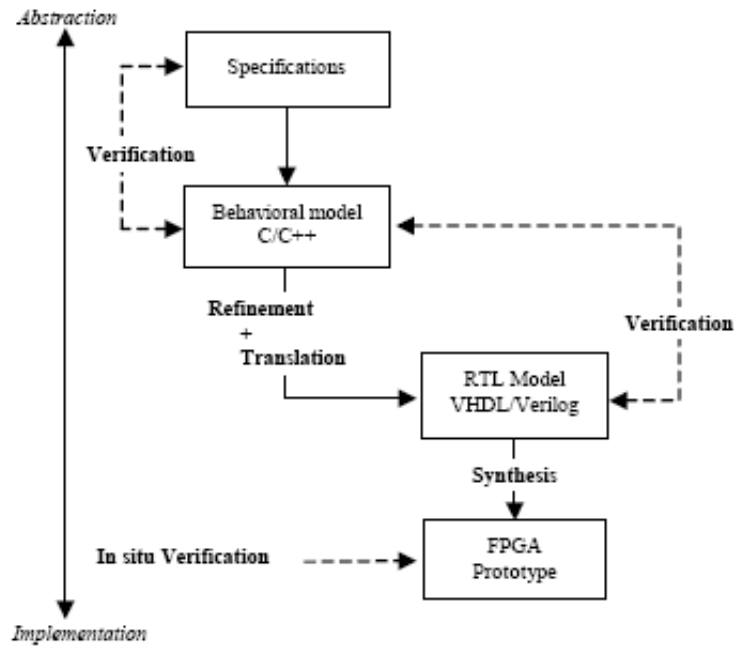


Figura 1.2: Flujo de diseño a utilizar.

6. Síntesis de Verilog para obtener lista de componentes reales.

Los pasos de verificación tienden a ser iterativos hasta encontrar un buen desempeño del circuito. Finalmente se puede traspasar el circuito a una plataforma FPGA para realizar un prueba física del diseño, pero este paso se considera un objetivo opcional en este trabajo.

Capítulo 2

Especificaciones del microprocesador ARM7

Para comprender como opera este microprocesador a continuación se describirán los detalles más importantes y cuales son las características implementadas en el modelo desarrollado.

2.1. Modelo de programador

El procesador ARM7 está diseñado para soportar dos formatos de memoria, *big endian* y *little endian*. Aunque esto permite intercambiar datos almacenados en distinto orden, no es relevante para el objetivo de este trabajo, ya que se le considera como una función adicional para ampliar la compatibilidad del procesador. Por esta razón, el procesador desarrollado soporta solamente formatos *little endian*.

ARM7 cuenta con 7 modos de operación que permiten cambiar de entorno en casos especiales:

- *User* (usr): Modo normal de operación.
- *FIQ* (fiq): Diseñado para soportar transferencias de datos o procesos de canales.
- *IRQ* (irq): Usado para propósito general en el control de interrupciones.
- *Supervisor* (svc): Modo protegido para el sistema operativo.
- *Abort mode* (abt): Entra cuando se suspende una obtención de datos o instrucciones.
- *System* (sys): Para modo de usuario privilegiado en el sistema operativo.
- *Undefined* (und): Entra cuando una instrucción no definida es ejecutada.

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

ARM-state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und


 = banked register

Figura 2.1: Registros general y *Program Counter*

Dentro del procesador se encuentra un banco de registros compuesto por 31 de uso general y 6 para el estado del procesador. En la figura 2.1 se observan los registros de ARM7. Se puede observar que en distintos modos de operación se utilizan algunos registros distintos marcados con un triángulo oscuro.

El estado actual del procesador se mantiene en el registro *CPSR* y los registros *SPSR_mode* guardan el registro de estado actual del modo de operación anterior, para luego recuperar el estado anterior copiando *SPSR* a *CPSR*. Un *Program Status Register* (PSR) contiene los flags de condición, que indican las características del resultado de una operación, flags de deshabilitación de interrupciones, flag de estado (el cual se mantiene en cero siempre porque no existe estado THUMB) y los bits del modo de operación (ver figura 2.2).

El flujo normal de un programa puede ser interrumpido por las excepciones. Las excepciones o interrupciones permiten controlar ciertos sucesos en el microprocesador realizando un cambio del modo de operación y saltando a una dirección del programa que está predefinida. En la tabla 2.1 se pueden ver las distintas excepciones, el vector predefinido para cada una y el modo de operación al que cambia el procesador.

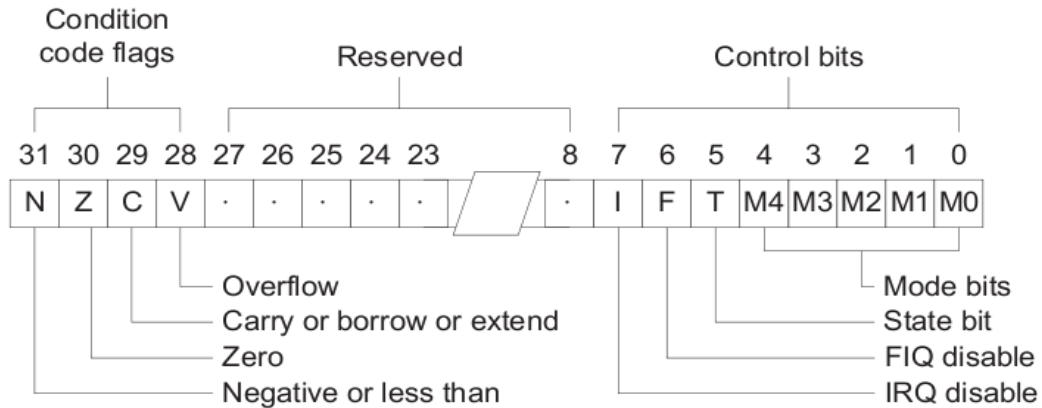


Figura 2.2: Formato del registro de estado de programa

Dirección	Excepción	Modo al que entra
0x00000000	Reset	Supervisor
0x00000004	Instrucción indefinida	Undefined
0x00000008	Interrupción de software	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reservado	Reservado
0x00000018	IRQ	IRQ
0x00000004	FIQ	FIQ

Tabla 2.1: Vectores de excepciones

El desarrollo del este procesador no planea una fase de prueba con dispositivos externos, por esta razón sólo se implementan las interrupciones por software e instrucción indefinida y reset. A pesar de esto, se ha programado la forma del sistema de excepciones con tal de implementar fácilmente las excepciones restantes en caso de ser deseadas.

2.2. Set de instrucciones

El set de instrucciones es donde más se debe prestar atención y es parte indispensable para obtener un procesador funcional. Cada instrucción esta caracterizada por una codificación o combinación de bits y para mayor comprensión se agrupan en formatos distintos. Estos formatos se pueden observar en la figura 2.3, donde se aprecia una característica común entre todos ellos, el campo de condición (los 4 bits más significativos). Este campo permite que todas las instrucciones sean condicionales, es decir que serán ejecutadas si y sólo si la

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond	0	0	I	Opcode				S	Rn	Rd	Operand 2										<i>Data Processing / PSR Transfer</i>										
Cond	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm	<i>Multiply</i>														
Cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rn	1	0	0	1	Rm	<i>Multiply Long</i>														
Cond	0	0	0	1	0	B	0	0	Rn	Rd	0	0	0	0	1	0	0	1	Rm	<i>Single Data Swap</i>											
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn	<i>Branch and Exchange</i>							
Cond	0	0	0	P	U	0	W	L	Rn	Rd	0	0	0	0	1	S	H	1	Rm	<i>Halfword Data Transfer: register offset</i>											
Cond	0	0	0	P	U	1	W	L	Rn	Rd	Offset				1	S	H	1	Offset	<i>Halfword Data Transfer: immediate offset</i>											
Cond	0	1	I	P	U	B	W	L	Rn	Rd	Offset						<i>Single Data Transfer</i>														
Cond	0	1	1													1	<i>Undefined</i>														
Cond	1	0	0	P	U	S	W	L	Rn	Register List							<i>Block Data Transfer</i>														
Cond	1	0	1	L	Offset												<i>Branch</i>														
Cond	1	1	0	P	U	N	W	L	Rn	CRd	CP#	Offset					<i>Coprocessor Data Transfer</i>														
Cond	1	1	1	0	CP	Opc	CRn	CRd	CP#	CP	0	CRm	<i>Coprocessor Data Operation</i>																		
Cond	1	1	1	0	CP	Opc	L	CRn	Rd	CP#	CP	1	CRm	<i>Coprocessor Register Transfer</i>																	
Cond	1	1	1	1	Ignored by processor												<i>Software Interrupt</i>														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figura 2.3: Formatos del set de instrucciones.

condición se cumple.

En la tabla 2.2 se presenta el set de instrucciones que se ha implementado en el microprocesador ARM7 desarrollado. Los detalles de cada instrucción se pueden ver en la página web de ARM[5] o el datasheet[1].

2.3. Pipeline, tiempos y señales

Como ya se ha dicho, el microprocesador ARM7 posee un pipeline de 3 etapas:

- Instruction Fetch (IF): Obtiene un instrucción desde la memoria.
- Instruction Decode (ID): Decodifica un instrucción preparando las señales que controlarán el *datapath*¹ en el siguiente ciclo.
- Execute (EX): Ejecuta la operación especificada utilizando el *datapath* para esto.

¹Datapath es la parte del microprocesador que reúne todas las unidades para procesamiento de datos, tales como ALU, multiplicador, banco de registros, etc.

Mnemónico	Instrucción	Acción
ADC	Suma con carry	$Rd := Rn + Op2 + Carry$
ADD	Suma	$Rd := Rn + Op2$
AND	AND	$Rd := Rn \text{ AND } Op2$
B	Salto	$R15 := \text{dirección}$
BIC	Bit Clear	$Rd := Rn \text{ AND NOT } Op2$
BL	Salto con enlace	$R14 := R15, R15 := \text{dirección}$
CMN	Comparación negativa	$CPSR \text{ flags} := Rn + Op2$
CMP	Comparación	$CPSR \text{ flags} := Rn - Op2$
EOR	Or exclusivo	$Rd := Rn \text{ XOR } Op2$
LDM	Carga de múltiples registros	Manipulación de stack (pop)
LDR	Carga de registro desde memoria	$Rd := (\text{dirección})$
MLA	Multiplicación Acumulada	$Rd := (Rm * Rs) + Rn$
MOV	Mueve registro o constante	$Rd := Op2$
MRS	Mueve estado/flags de PSR a un registro	$Rn := PSR$
MSR	Mueve registro a estado/flags de PSR	$PSR := Rm$
MUL	Multiplicación	$Rd := Rm * Rs$
MVN	Mueve registro negado	$Rd := 0xFFFFFFFF \text{ EOR } Op2$
ORR	OR	$Rd := Rn \text{ OR } Op2$
RSB	Resta inversa	$Rd := Op2 - Rn$
RSC	Resta inversa con carry	$Rd := Op2 - Rn - 1 + Carry$
SBC	Resta con carry	$Rd := Rn - Op2 - 1 + Carry$
STM	Almacenamiento múltiple	Manipulación de stack (push)
STR	Almacenamiento de registro a memoria	$j\text{dirección}j := Rd$
SUB	Resta	$Rd := Rn - Op2$
SWI	Interrupción de software	Llamada del SO
SWP	Intercambio de registro con memoria	$Rd := [Rn \ [Rn]] := Rm$
TEQ	Test de igualdad a nivel de bits	$CPSR \text{ flags} := Rn \text{ EOR } Op2$
TST	Test bits	$CPSR \text{ flags} := Rn \text{ AND } Op2$

Tabla 2.2: Set de instrucciones implementado

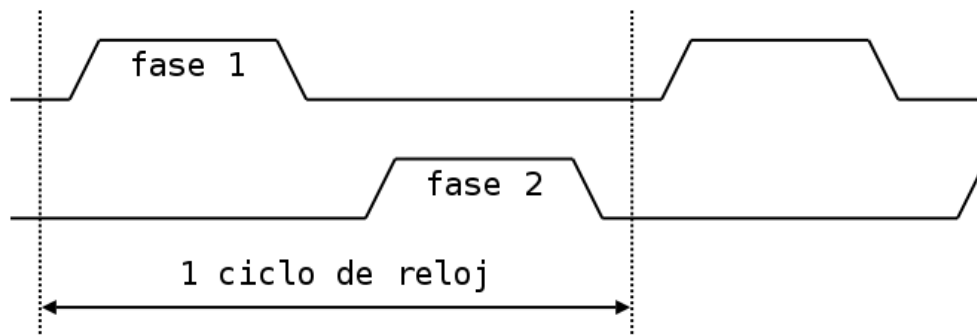


Figura 2.4: Fases no traslapadas a partir del reloj.

El pipeline permite que estas 3 etapas puedan ser realizadas simultáneamente y que la cantidad de ciclos para un instrucción sea en promedio cercana a uno. Para esto el procesador necesita que varias instrucciones puedan ser obtenidas de memoria y guardadas en una cola.

Este procesador se caracteriza por utilizar latches transparentes en vez de flip-flops, es decir los registros no se graban por medio un cantos positivos o negativos del reloj sino que a partir del reloj se obtienen dos fases que no se traslapan[2] (ver figura 2.4), éstas activan por nivel los diferentes registros del procesador. Con la utilización de dos fases no traslapadas disminuyen los problemas con carreras críticas de las señales.

Otra característica que se debe tener en cuenta es como se comporta la memoria externa. El procesador original posee la capacidad de variar el comportamiento de las señales de comunicación con la memoria para adaptarse a diferentes clases como SRAM o DRAM. En este trabajo sólo se ha implementado el soporte para memorias SRAM o ROM, lo que quiere decir que la dirección de memoria y todas las señales de control varían en la fase 1 y los datos son entregados en la fase 2.

2.4. Resumen de especificaciones

Características implementadas:

- Formato Little endian
- Banco de registro de 37 registros de 32-bits
- Multiplicador con algoritmo Booth (multiplica 8 bits por ciclo).
- Barrel Shifter
- ALU
- Pipeline de 3 etapas

- Decodificación de instrucciones y control lógico
- Sistema de excepciones con prioridad
- Excepciones Reset, Instrucción indefinida y interrupción por software.
- Soporte para memorias SRAM y ROM.
- Instrucciones listadas en la tabla 2.2.

Características descartadas:

- Soporte de 2 formatos de memoria (big y little endian)
- Instrucciones para uso de coprocesador
- Set de instrucciones THUMB
- Interrupciones Abort, FIQ y IRQ.
- Soporte para memorias DRAM.
- Módulos externos al core del microprocesador como ICEBreaker o controlador TAP.

Capítulo 3

Modelo del microprocesador en ArchC

3.1. Diseño

Como se explicó anteriormente ArchC permite modelar un microprocesador y simularlo de manera de facilitar la descripción y estudio de éste. Para esto ArchC requiere que se programen como mínimo dos archivos que describen la arquitectura general del procesador y con estos puede generar una colección de archivos en lenguaje SystemC, los cuales son compilados para obtener un simulador del procesador.

En la figura 3.1 se aprecian los archivos básicos que utiliza ArchC y los pasos que se siguen para obtener un simulador. Principalmente se tienen 3 archivos:

- *arm7tdmi.ac*: Este archivo contiene las características de la memoria interna del microprocesador, es decir el largo de palabra, registros, memoria y el tipo de endian del procesador. En este caso el procesador tiene la posibilidad de escoger little-endian o big-endian y se ha escogido la primera posibilidad desechando la capacidad de doble-endian del circuito real.
- *arm7tdmi_isa.ac*: En este archivo se declaran principalmente los formatos de las instrucciones del procesador, las instrucciones y su decodificación. También entrega la posibilidad de escribir la sintaxis del lenguaje assembly del procesador, para luego poder generar un programa ensamblador. Respecto a esto último, se dejaron descritas la mayor parte de las instrucciones con su sintaxis correspondiente, pero problemas con instrucciones de sintaxis muy complejas impidieron completar el set de instrucciones.
- *arm7tdmi_isa.cpp*: Este archivo se caracteriza por ser programado directamente en SystemC. Es donde se describe el comportamiento de cada instrucción del procesador, utilizando una programación dividida en funciones que pueden describir las acciones de en común de todas las instrucciones, luego de las que comparten un formato en particular y finalmente el comportamiento de la instrucción en particular.

Luego con estos archivos se ejecuta el programa *acsim* (ArchC Simulator Generator)[12]. Éste extrae la información de los archivos descritos anteriormente y genera todos los módulos

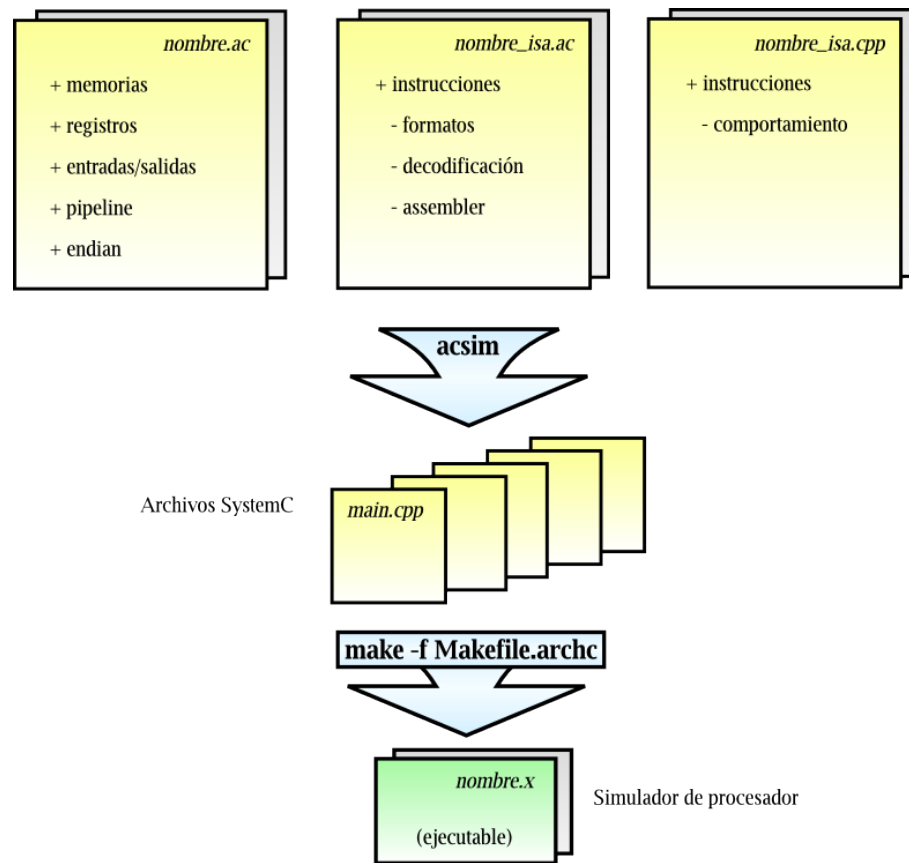


Figura 3.1: Proceso para obtención de simulador en ArchC.

SystemC y/o clases C++ necesarios para la arquitectura del simulador. Estos archivos puede compilarse y obtener el simulador del procesador.

El diseño realizado fue descrito de forma funcional, es decir las instrucciones están programadas en SystemC sin considerar los ciclos que están utilizan. Realizar un diseño de esta clase permite, en caso hipotético de estar diseñando un procesador totalmente nuevo, generar un set de instrucciones funcional, donde su codificación sea correcta y los objetivos básicos para el procesador se cumplan, además de tener una interfaz de prueba simple y sin considerar detalles técnicos que pueden ser vistos más adelante y sólo retrasan el diseño.

En este caso, el microprocesador esta diseñado, pero la creación del modelo en ArchC no deja de ser importante para comprender el set de instrucciones, los diferentes formatos que poseen y el tipo de codificación que se utiliza. La idea es preocuparse solamente de como se codifica cada instrucción y qué función realiza sin entender el cómo lo realiza.

3.2. Validación

Una vez realizado el modelo debe ser validado, es decir que cumpla con los requerimientos básicos como reconocer las instrucciones que se le entregan y ejecutarlas de forma eficaz.

Así puede diseñarse un programa y hacerlo correr en el procesador entregándolo como argumento al simulador obtenido del modelo como se explicó anteriormente. Para facilitar la tarea de depuración del modelo, en el mismo se agrega código que entrega el valor actual de las diferentes variables. Esto se realiza como en el ejemplo a continuación.

Tenemos el siguiente código parte de una función en C:

```
int x, y, z;
x = 5;
y = 2;

if( x < y )
    z = x-2;
else
    z = y+2*x;
```

Este código traducido a assembly para ARM7 es:

```
;R0=x R1=y R2=z

MOV    R0,#5
MOV    R1,#2
```

```

CMP      R0,R1
SUBMI   R2,R0,#2
ADDPL   R2,R1,R0,LSL #1

```

Luego este código se convierte a lenguaje binario por medio del ensamblador y se ejecuta con el simulador. Los resultados son los siguientes:

```

R0=0 R1=0 R2=0 R3=0 R4=0 R5=0 R6=0 R7=0 R8=0
V=0 C=0 Z=0 N=0
PC=4 Mode = 16
Processing movi instruction: Type_DataProcI-> movi
  R0=5 R1=0 R2=0 R3=0 R4=0 R5=0 R6=0 R7=0 R8=0
V=0 C=0 Z=0 N=0
PC=8 Mode = 16
Processing movi instruction: Type_DataProcI-> movi
  R0=5 R1=2 R2=0 R3=0 R4=0 R5=0 R6=0 R7=0 R8=0
V=0 C=0 Z=0 N=0
PC=12 Mode = 16
Processing cmpr instruction: Type_DataProcR-> cmpr
  R0=5 R1=2 R2=0 R3=0 R4=0 R5=0 R6=0 R7=0 R8=0
V=0 C=0 Z=0 N=0
PC=16 Mode = 16
Processing subi instruction:
  R0=5 R1=2 R2=0 R3=0 R4=0 R5=0 R6=0 R7=0 R8=0
V=0 C=0 Z=0 N=0
PC=20 Mode = 16
Processing addr instruction: Type_DataProcR-> addr
  R0=5 R1=2 R2=12 R3=0 R4=0 R5=0 R6=0 R7=0 R8=0
V=0 C=0 Z=0 N=0
PC=24 Mode = 16

```

El resultado de la simulación y que valores muestre en pantalla dependerá del código utilizado para la depuración. En este caso, se muestran los valores de los 8 primeros registros internos, de los flags de operación, el *program counter* y el modo de operación del procesador. Se observa que las instrucciones se han ido ejecutando correctamente según sus condiciones.

Capítulo 4

Modelo en Verilog del microprocesador

4.1. Características del modelo

El modelo del procesador en Verilog puede realizarse a distintos niveles de abstracción y dependerá del objetivo inmediato del modelo y el conocimiento de los módulos para saber en que nivel trabajar. Si no se tiene conocimiento acerca de la estructura interna de los bloques, requerirá realizar una descripción comportamental y luego traspasar el modelo a un nivel estructura, aunque este paso dependerá de la complejidad de la descripción y del sintetizador con que se cuente. Así un bloque no será necesario describirlo a nivel estructural si observamos que el sintetizador de código Verilog logra un resultado eficiente.

El resultado final del modelo se ha descrito en parte de forma estructural y en forma comportamental. Esto debido a que se conoce la estructura de bastantes módulos del procesador, pero existen detalles que no son analizados en los libros especializados. Es por esto que el modelo necesita de un sintetizador y optimizador eficiente para obtener buenos resultados.

Otro detalle a destacar es que existen cables desconectados, al encontrarse el modelo sin todas las funciones originales pero con la posibilidad de implementarlas fácilmente.

4.2. Diseño

Para realizar el diseño en Verilog del procesador se deben considerar distintos módulos y secciones del modelo que en su mayoría se pueden observar en la figura 1.1. A continuación se describen por separado.

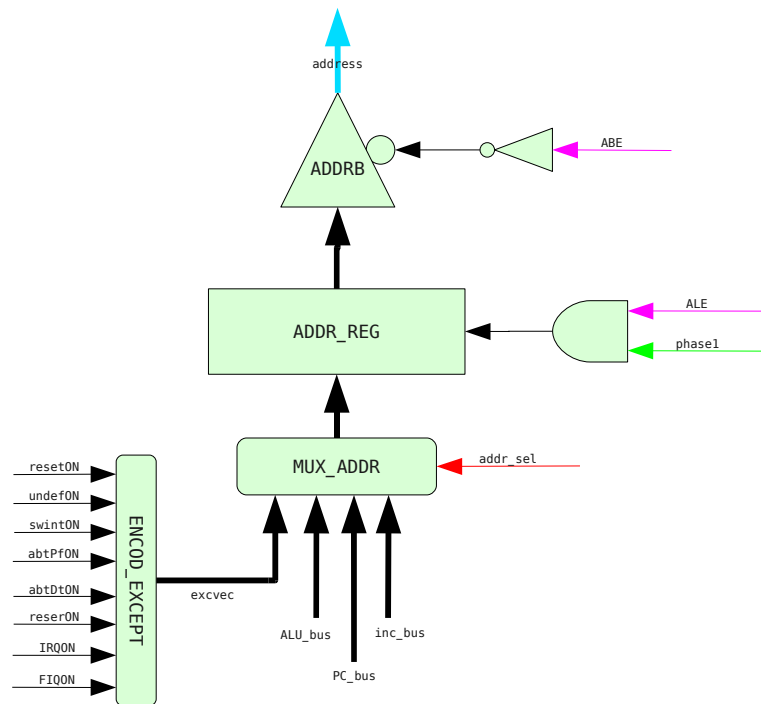


Figura 4.1: Diagrama de registro de dirección.

4.2.1. Registro de dirección - *Address Register*

Este registro mantiene la dirección que se desea leer desde la memoria. Posee 4 posibles fuentes desde donde obtener su valor: vector de excepción, resultado de la ALU, *Program Counter* y la dirección incrementada; las cuales están multiplexadas y son controladas por una señal de selección (figura 4.1).

El registro de dirección siempre graba su nuevo valor en la fase 1 a menos que *ALE* este en nivel bajo. Por otra parte, un buffer de tres estados permite que la salida pueda ser puesta en alta impedancia si *ABE* esta en nivel bajo.

El encoder para las excepciones posee de entradas las señales que indican si una excepción fue activada, así internamente y según la prioridad de estas, escoge un vector de excepción según la tabla 2.1.

4.2.2. Incrementador de dirección - *Address Incrementer*

Este modulo es muy simple y se encarga de incrementar en 4 la dirección de memoria, evitando utilizar la ALU para este objetivo. Como se observa en la figura 4.2, antes del incrementador se tiene un latch, el cual permite que no se genere un ciclo infinito, al grabarse en la fase 2, entre el registro de dirección y su incrementador. Además posee una señal de control de grabación ya que en algunas instrucciones se necesita tener constante de un ciclo

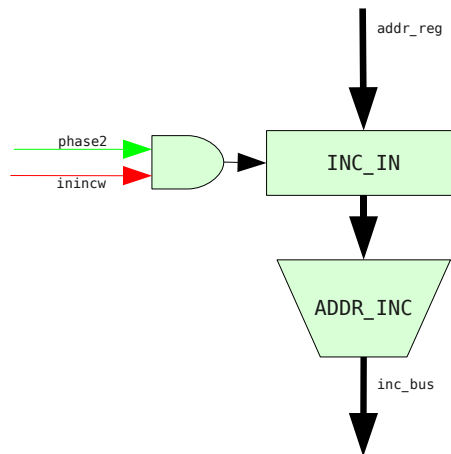


Figura 4.2: Diagrama de incrementador de dirección.

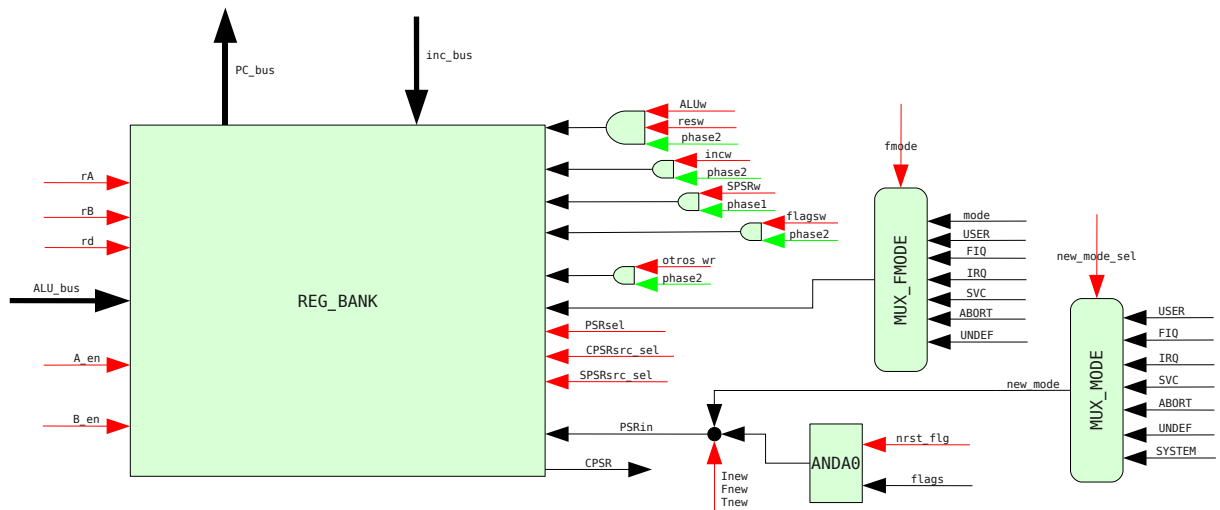


Figura 4.3: Diagrama del entorno del banco de registros.

a otro el valor del latch.

4.2.3. Banco de registros - *Register Bank*

El banco de registros esta compuesto por 31 registros generales y 6 registros para el estado del procesador. En la figura 4.3 se pueden observar las distintas señales de entrada y salida para este bloque. A continuación se describen:

- *Buses*: hay 2 buses de entrada, ALU bus y inc bus; y uno de salida que entrega el Program Counter (PC bus).
- *rA*, *rB*, *rd*: estas entradas indican el número de registro que se desea seleccionar para el bus A, el bus B y el registro que se desea escribir el resultado de la ALU.

- *A_en*, *B_en*: si su nivel es alto ponen en alta impedancia las salidas A y B hacia los buses respectivos.
- *Señales de escritura*: *ALUw* y *resw* permiten escribir el bus de la ALU en el registro seleccionado; *incw* escribe el bus del incrementador en el PC; *SPSRw* escribe en el registro SPSR del modo seleccionado por *fmode* el valor del bus ALU o el CPSR; y *flagsw*, *reservw*, *Iw*, *Fw*, *Tw* y *modew* permiten escribir distintos campos desde *PSRin*, el bus ALU o un registro SPSR a CPSR.
- *CPSRsrc_sel*, *SPSRsrc_sel*: seleccionan la fuente de escritura para CPSR y SPSR respectivamente.
- *nrst_flg*: permite dejar en cero los flags de operación cuando esta en nivel bajo.
- *Valores para PSRin*: al banco de registro se entregan nuevos valores para CPSR que permiten controlar el estado del procesador.

Para mayores detalles sobre la arquitectura interna del banco de registro, ver los anexos al final de este documento.

4.2.4. Multiplicador - *Multiplier*

El multiplicador esta implementado de manera de reducir el tiempo de cálculo. Usa el algoritmo de Booth para disminuir la cantidad de productos parciales y los suma por medio de *carry save adders*¹, dándole la capacidad de multiplicar 32x8 bits en un ciclo, por lo tanto una multiplicación completa necesitará máximo 4 ciclos más uno o dos ciclos necesarios para grabar los registros implicados. El sistema detecta cuando el multiplicador posee ceros o unos en los bits más significativos y detiene el proceso, lográndose una disminución de ciclos.

Las entradas importantes para el multiplicador son los dos valores a multiplicar, los valores para la acumulación (cuando a la multiplicación se le suma una constante) y bits que indican si los operandos se consideran con signo, si hay acumulación, si se realiza una multiplicación de 64 bits y el ciclo en que va el algoritmo.

4.2.5. Desplazador rápido - *Barrel Shifter*

El modulo de *Barrel shifter* se encarga de hacer desplazamiento a valores de 32-bits en un solo ciclo, esto permite aumentar la eficiencia del procesador ya que un desplazador de este tipo integrado ahorra mucho tiempo de ejecución y permite instrucciones con mayor funcionalidad. Posee de entradas el bus B que tiene el valor a desplazar; el carry guardado en CPSR, que se utiliza para la función RRX²; el tipo de desplazamiento, izquierda o derecha

¹En los anexos se explica con mayor detalle como se realiza la multiplicación

²*Rotate Right Extended* (RRX) es un tipo de rotación que se aplica al realizar ROR #0 (Rotate Right). Esta desplaza una posición a la derecha el valor de 32-bits, al bit 31 se asigna el carry de entrada y el bit 0 es el carry de salida.

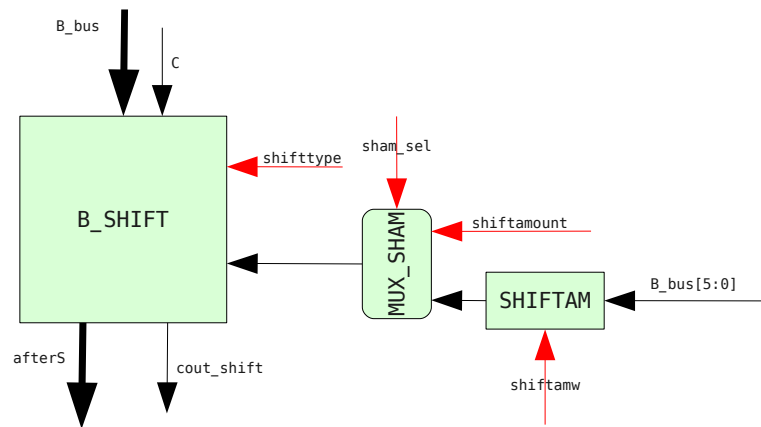


Figura 4.4: Diagrama de desplazador (*barrel shifter*).

y lógico, aritmético o rotación; y la cantidad de desplazamientos que se desean. Sus salidas son el valor desplazado y el carry de salida que corresponde al último bit que fue desplazado fuera de la fila.

Existen dos métodos para obtener la cantidad de desplazamiento deseada, es por esto que en la figura 4.4 se aprecia un multiplexor. Una manera es entregar el valor inmediato del desplazamiento y otra es a través de un registro, donde primero se debe obtener del banco de registros, luego se extraen los 6 bits menos significativos y se graban en el latch SHIFTAM de la figura.

4.2.6. Unidad Aritmético Lógica - *Arithmetic logic unit*

Este modulo es parte indispensable del microprocesador ya que realiza la mayor parte de las operaciones matemática. Los operandos del módulo, es decir el bus A y el resultado del bus B después del desplazamiento, son grabados a unos latches antes de operarse. Esto permite guardar en la fase 1 los valores y mantenerlos constantes durante la operación, aparte que impide un posible loop si se escoge un registro como operando y como destino, ya que el tramo entre registros es desde ALU a banco de registros y no de banco de registros a banco de registros.

La ALU además de realizar las operaciones de las instrucciones de procesamiento de datos, puede incrementar en 4 su salida, función necesaria para implementar las funciones de transferencia de datos en bloque. También tiene la capacidad de dejar intacta una entrada tanto desde el bus A como el segundo operando.

Si se observa la figura 4.5 se pueden percibir 3 detalles importantes rodean a la ALU. En primer lugar el multiplexor MUX_CIN permite escoger que carry utilizar entre el carry actual

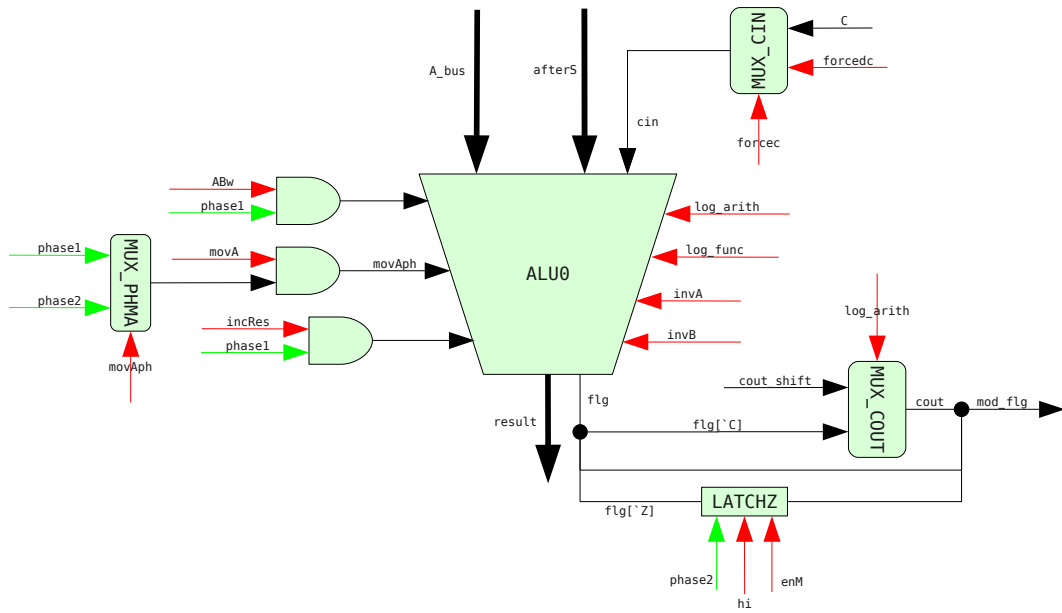


Figura 4.5: Diagrama de unidad aritmética lógica.

o un carry forzado que se utiliza para implementar la resta³. También el carry de salida se puede escoger, ya que dependerá si se ha realizado una operación aritmética o lógica; en el primer caso el carry obtenido en la ALU es importante y en segundo caso, el carry de la ALU no tiene significado por lo tanto se utiliza el carry obtenido en el *barrel shifter*. Por último, el bloque LATCHZ que se observa en la figura se utiliza para detectar si el resultado de la multiplicación de 64-bits es cero, observando si los primeros 32 bits son cero y los 32 bits restantes.

4.2.7. Registro de datos de escritura - *Write data register*

En este registro se graban los datos que se desean grabar a memoria. Como se ve en la figura 4.6, los datos provienen del bus B y con el módulo SEL_MAS_ENC selecciona según el tamaño del bloque de datos que se desea transferir, que pueden ser 32-bits (*word*), 16-bits (*half-word*) o 8-bits (*byte*). Luego los datos se graban en el latch de salida y un buffer de tres estados se encarga de poner en alta impedancia la salida en caso de no desearse.

4.2.8. Registro de datos de escritura - *Read data register*

Al contrario del registro anterior, este registro recibe los datos de entrada que se desean escribir en un registro. Los datos recibidos pueden ser considerados de 4 tipos: *unsigned byte*, *signed byte*, *unsigned halfword*, *signed halfword* y *word*. El módulo SEL_MAS_DEC de la figura 4.7 permite obtener los datos en el bus B de forma correcta, escogiendo entre byte,

³El microprocesador considera los números negativos con representación complemento-2, siendo posible implementar la resta como una suma del 1^{er} operando más el 2^{do} operando negado más 1.

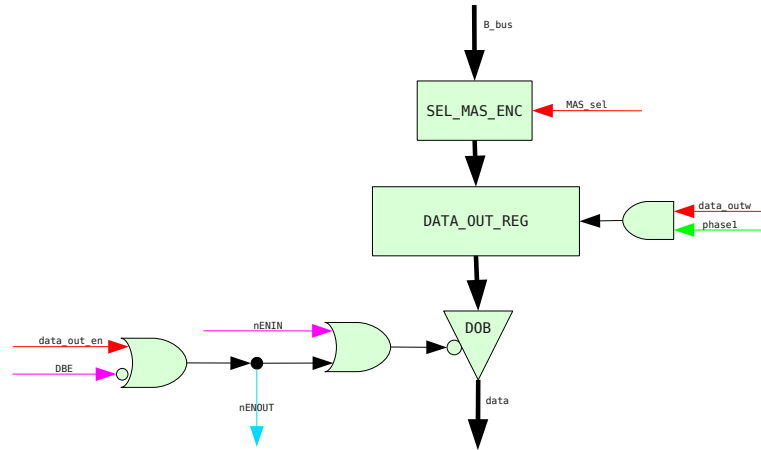


Figura 4.6: Diagrama de registro de datos de escritura.

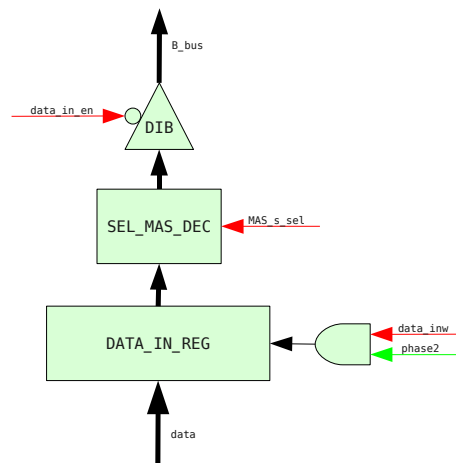


Figura 4.7: Diagrama de registro de datos de lectura.

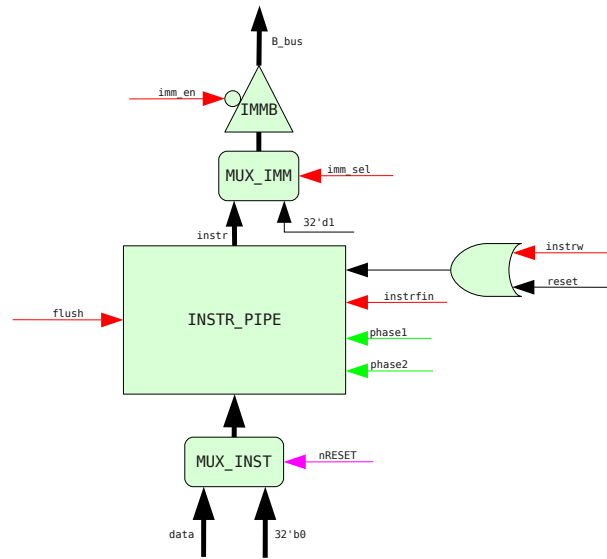


Figura 4.8: Diagrama de pipeline de instrucciones.

half-word o word y extendiendo el signo o no, dependiendo si son de tipo *unsigned* o *signed*. Finalmente es necesario un buffer de tres estados porque existen distintas fuentes para el bus B.

4.2.9. Pipeline para instrucciones - *Instruction Pipeline*

El pipeline para instrucciones se encarga de guardar las instrucciones que se obtienen de memoria, quedando encoladas. Para desechar la primera instrucción que entro a la cola se utiliza *instrfin*, indicando que ya fue ejecutada. Además se puede vaciar la cola con la señal *flush*(ver figura 4.8).

Cuando el reset del microprocesador es activado, se comienzan a grabar instrucciones que no producen efectos, permitiendo luego desactivado el reset se realicen acciones de cambio de contexto⁴.

Para escoger un el valor inmediato desde la instrucción se ocupa el multiplexor MUX_IMM, con la posibilidad de escoger el valor 1 para otros propósitos.

4.2.10. Control principal - *Main Control*

El control principal se encarga de enviar las señales de control al *datapath*, controlando la ejecución de las instrucciones. En la figura 4.9 se observa un diagrama general del control que posee varios bloques funcionales (Nota: en el diagrama faltan muchos detalles, por ejemplo

⁴Luego de un reset se respalda PC y CPSR, se cambia a modo Supervisor, se desactivan las interrupciones y se fuerza al PC a ser 0x00.

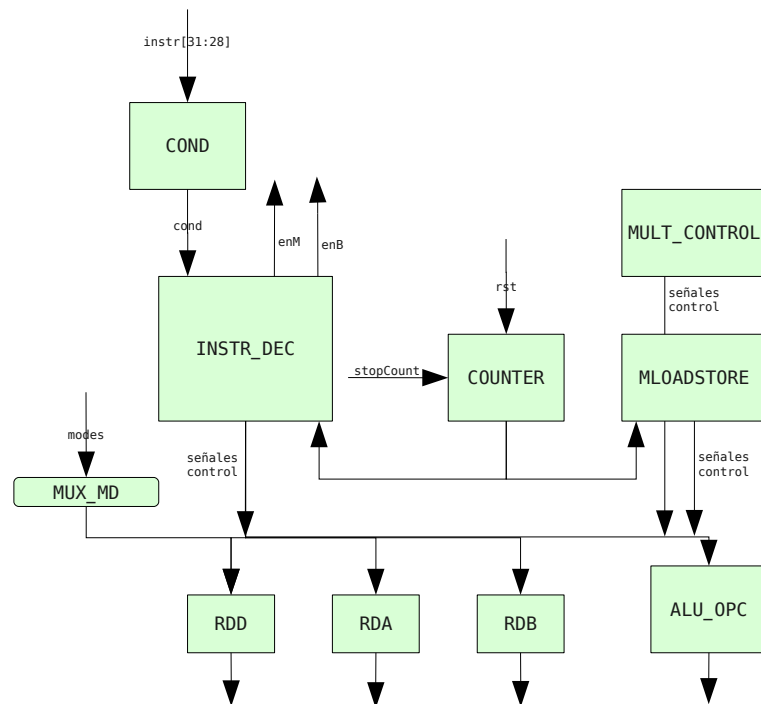


Figura 4.9: Diagrama de control principal.

que la instrucción es entrada de los bloques más importantes). Los tres considerados más importantes son el decodificador de instrucciones (INSTR_DEC), el control de multiplicación (MULT_CONTROL) y el control para instrucciones de transferencia de datos en bloque (MLOADSTORE):

Decodificador de instrucciones - *Instruction Decoder*

Este módulo se encarga de generar las señales de control de la mayoría de las instrucciones exceptuando la multiplicación y la transferencia de datos en bloque. Igualmente aunque no genere estas señales está encargado de detectar cuando una instrucción es multiplicación o de transferencia en bloque, habilitando el módulo correspondiente para cada caso.

Este bloque es el que manda la decodificación ya que además cuando la condición de la instrucción no se cumple, anula la ejecución en el datapath.

Control de multiplicación - *Multiplier Control*

Este control es habilitado por la señal *enM* y entrega las señales para el multiplicador. Se encarga de indicar al multiplicador en que ciclo va su ejecución y recibe desde el multiplicador el número de ciclos que necesitará el algoritmo.

Control para instrucciones de transferencia de datos en bloque - *Block Data Transfer Control*

Al igual que el control de multiplicación se habilita con una señal proveniente del decodificador de instrucciones. Cuando *enB* esta en nivel alto, este bloque toma control de las señales hacia el datapath. Es el único bloque que utiliza la señal *stopCounter* para detener la cuenta ya que en los ciclos de transmisión se mantiene el mismo estado copiando distintos registros.

Además se tienen los siguientes módulos:

- COND: bloque para comprobar si la condición indicada por la instrucción se cumple. Aunque no se muestra en la figura, además de tener de entrada el campo de condición de la instrucción, tiene de entrada los flags de condición.
- COUNTER: este bloque lleva la cuenta de los ciclos ejecutados. Puede ser detenido por medio de *stopCount* y es utilizado por los 3 bloques importantes antes nombrados. Es módulo es muy importante ya que indica el estado de las instrucciones y sin este el controlador no sería considerado una máquina de estado finito.
- MUX_MD: multiplexor para escoger diferentes modos de operación, utilizado para los decodificadores de registros.
- Decodificadores de registros (RDD, RDA, RDB): el banco de registros debe recibir un número de registro según sus propios índices, diferentes a los vistos por un usuario. Para esto los decodificadores necesitan el número de registro con notación normal, el modo de operación y si se desea acceder un registro PSR.
- ALU_OPC: se encarga de decodificar el *opcode* de una instrucción de procesamiento de datos generando las señales que controlan la ALU.

La decodificación de la instrucción en el control principal se realiza en la fase 2 que es cuando la instrucción cambia y los resultados de la decodificación, es decir las señales de control, se aplican en la fase 1 siguiente. El control mantiene las señales de salida en registros y estos pueden ser grabados desde los tres módulos de control importantes descritos anteriormente, dependiendo si se habilita la multiplicación, la transferencia de datos en bloque o ninguna de estas. Así en la fase 1 se graban las señales correspondientes y comienza la ejecución de la instrucción.

4.3. Validación

4.3.1. Método

El proceso de validación del microprocesador se realiza por medio de un programa cargado directo a una memoria descrita en Verilog. El modelo del circuito en primera instancia es

traducido a elementos electrónicos genéricos, aquí debe ser probado con el programa escrito para estos fines. Luego el modelo puede ser compilado y optimizado, esto quiere decir que utilizando librerías que contienen compuertas ya diseñadas y probadas se genera un circuito con elementos electrónicos reales, que dependerán de la tecnología que se ocupe. El optimizado siempre dependerá de las condiciones temporales y de área que el usuario imponga a su modelo. Luego con el circuito sintetizado con componentes reales se debe volver a verificar y validar.

Para simular la secuencia de instrucciones típicas que manejará el microprocesador y abarcar la mayor cantidad de casos, las pruebas se realizan con un programa escrito en C y compilado para ARM7. Los pasos a seguir son los siguientes:

1. Escribir el programa deseado en C.
2. Programar el código de inicialización en assembly que residirá en la dirección cero o vector de reset del procesador.
3. Compilar ambos códigos para el procesador ARM7.
4. Escribir el *linker script* que define las secciones de memoria y la posición de las funciones que se enlazarán.
5. Utilizando un enlazador generar el código binario del programa final, usando el programa en C y el programa de inicialización compilados.
6. Traspasar a ASCII hex el código binario para que tenga un formato que soporte Verilog.
7. Generar un archivo testbench en Verilog que instancie el módulo del procesador y el módulo de la memoria, junto con agregar las acciones que tomará la prueba para mostrar el funcionamiento interno del procesador.
8. Dentro del archivo Verilog de la memoria o el testbench, utilizar la tarea *\$readmemh* para cargar el programa dentro de la memoria.
9. Compilar el testbench, simular y analizar los resultados.

4.3.2. Ejemplo de ejecución de código

El ejemplo consiste en un programa que multiplica dos números.

1. El programa será código del programa en C para probar es:

```
int foo( x, y ){  
    return x*y;  
}
```

2. Programa en assembly para inicialización:

```
reset_vector:
    ldr sp,=stack_space
    bl  foo
```

3. Se compilan los programas sin linkear con los programas correspondientes generando dos archivos compilados.

4. El *linker script* es:

```
SECTIONS
{
    . = 0x00000;
    .text : { *(.text) }
    . = 0x01000;
    .data : {
        *(.data)
        stack_space = .;
    }
    .bss : { *(.bss) }
}
```

5. Luego se obtiene el código compilado en formato binario *elf*.

6. Se traspasa a hexadecimal, quedando la siguiente secuencia de instrucciones:

```
0:  e59fd000      ldr    sp, [pc, #0]    ; 0x8
4:  eb000000      bl     0xc
8:  00001000      andeq  r1, r0, r0
c:  e1a0c00d      mov    ip, sp
10: e92dd800      push  {fp, ip, lr, pc}
14: e24cb004      sub   fp, ip, #4      ; 0x4
18: e24dd008      sub   sp, sp, #8      ; 0x8
1c: e50b0010      str   r0, [fp, #-16]
20: e50b1014      str   r1, [fp, #-20]
24: e51b2010      ldr   r2, [fp, #-16]
28: e51b3014      ldr   r3, [fp, #-20]
2c: e0030392      mul   r3, r2, r3
30: e1a00003      mov   r0, r3
34: e24bd00c      sub   sp, fp, #12     ; 0xc
38: e89da800      ldm   sp, {fp, sp, pc}
```

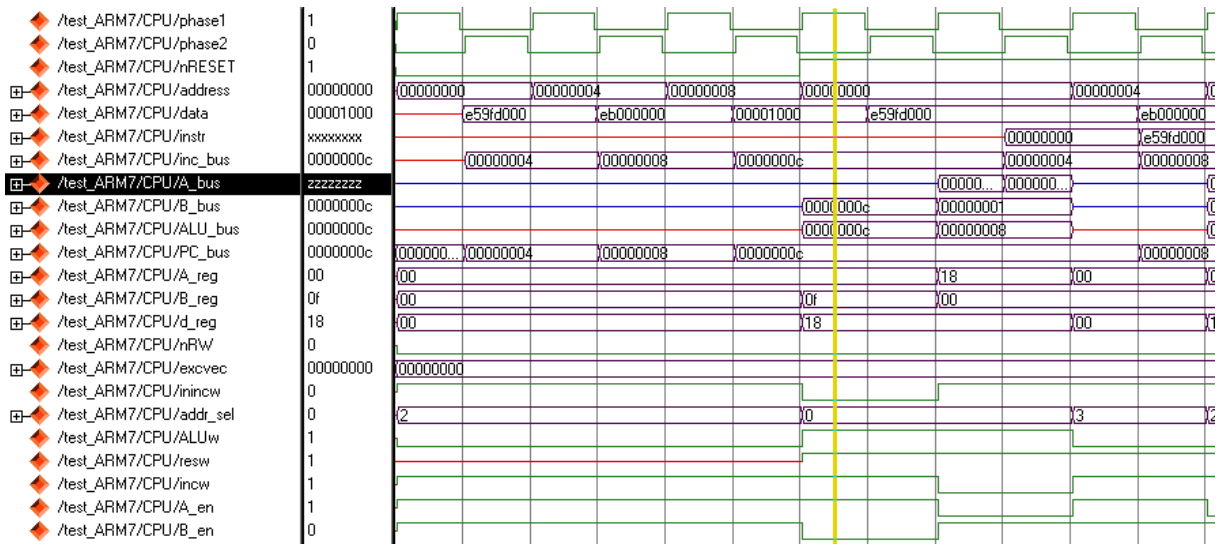


Figura 4.10: Señales del microprocesador en prueba de multiplicación (1)

7. El código para el testbench hará un reset inicial y luego captará los cambios en los registros, guardándolos en un archivo. Además se fijan todos los registro inicialmente en cero para propósitos de debugueo, exceptuando los registro 0 y 1 que se utilizan para guardar los factores de la multiplicación. Para efectos de este ejemplo se mostrará la simulación gráfica de las señales resultantes.
8. En el módulo de memoria se carga el archivo que contiene el programa.
9. En las figuras 4.10 y 4.11 se observan parte de los resultados de la simulación. Una revisión exhaustiva por los ciclos permite observar que el comportamiento y los resultados son correctos, es decir el modelo del microprocesador se comporta satisfactoriamente.

El programa de este ejemplo utiliza solamente las instrucciones LDR, BL, AND, MOV, STM, SUB, STR, MUL y LDM. En realidad se ha probado el procesador con programas que abarcan más instrucciones y además de ser cada instrucción probada por separado, y los resultados siguen siendo satisfactorios.

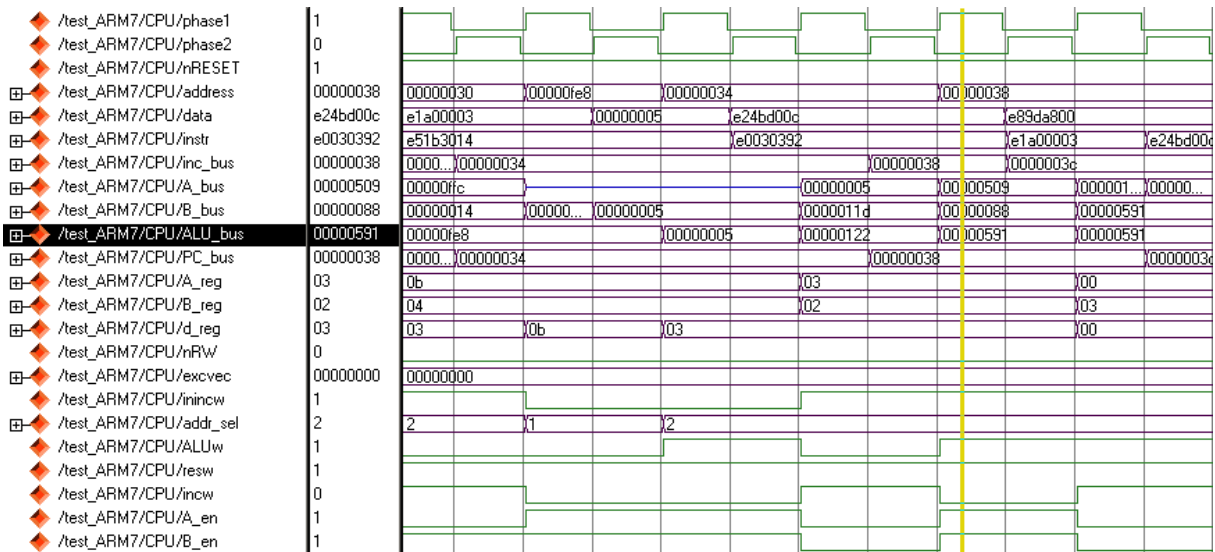


Figura 4.11: Señales del microprocesador en prueba de multiplicación (1)

Capítulo 5

Conclusiones

El desarrollo de un microprocesador ARM7 ha permitido a su realizador adquirir diversa experiencia dirigida hacia el diseño de circuitos integrados digitales. Esto quiere decir que el objetivo de generar capacidad para generar circuitos integrados ha sido cumplida. Las diversas etapas que se han superado permitieron que se los conocimientos contemplados sean de una gran amplitud.

Se llevado a cabo una metodología para obtener un procesador funcional donde la primera experiencia técnica adquirida fue acerca de los lenguajes ArchC y SystemC. Se comprobó que su utilización permite obtener modelos de mayor nivel para ser simulados y probados con mayor rapidez. Es importante destacar que la descripción comportamental de un sistema siguiendo una metodología *top-down* es muy ventajosa para disminuir errores posteriores.

Dentro del marco de lenguajes de descripción de hardware más nuevos siempre se debe tener en cuenta el nivel de desarrollo y madurez del lenguaje. Por ejemplo, ArchC siendo una herramienta muy útil para analizar microprocesadores, todavía no se encuentra en un estado estable en el cual se puedan obtener resultados sin inconvenientes. Esto conlleva tiempo gastado en descubrir detalles del programa que podrían ser utilizados de manera más provechosa.

El modelo en Verilog se implemento de manera satisfactoria, aprendiendo detalles técnicos muy importantes. El lenguaje Verilog debe ser visto desde un punto diferente que los lenguajes de software, debido a que su ejecución no es secuencial y lo que se programa es hardware. Es así como surgen muchos problemas que sólo la experiencia puede evitar.

Una ventaja clara de Verilog sobre SystemC o ArchC es el nivel de madurez del lenguaje. La cantidad de herramientas que existen y el nivel de conocimiento por parte de ingenieros es mucho mayor, entregando un lenguaje estandarizado, estable y con mucha documentación para consultar. Además se probó que no existen problemas con combinar código de manera comportamental y estructural. En la mayoría de los casos el sintetizador obtuvo los resultados deseados y muy eficientes.

Se han obtenido conocimientos bastos sobre el microprocesador ARM7, el cual posee un set de instrucciones ARM compatible con toda su familia. Se ha comprendido su funcionamiento con detalle, permitiendo en un futuro programar sobre una plataforma de esta familia con mayores herramientas.

Los trabajos futuros que podrían realizarse acerca del tema son:

- Implementar todas las excepciones del microprocesador aprovechando que el sistema en general está diseñado.
- Completar todas las funciones relacionadas con la compatibilidad con otros sistemas, como soporte *big-endian*, soporte para memorias DRAM, señales de control externo, etc.
- Diseñar soporte para set de instrucciones THUMB.
- Terminar set de instrucciones ARM implementando instrucciones para el coprocesador.
- Optimizar modelo del microprocesador, eliminando señales redundantes y mejorando estructuras de los módulos.
- Traspasar a una plataforma FPGA el procesador para probar de forma física su funcionamiento.
- Utilizar el modelo como parte un sistema completo para algún uso específico, permitiendo integrar todo en un solo chip.

Bibliografía

- [1] Atmel Corporation. *ARM7TDMITM (Thumb[®]) Datasheet*, January 1999.
- [2] Steve Furber. *ARM System-on-Chip Architecture (2nd Edition)*. Addison-Wesley Professional, 2000.
- [3] Hee Chul Kang Geun Young Jeong, Ju Sung Park. A study on multiplier architecture optimized for 32-bit processor with 3-stage pipeline. In *2004 International SoC Design Conference*, pages 546–550. COEX Seoul Korea, October 2004.
- [4] Open SystemC Initiative. Systemc homepage. <http://www.systemc.org>.
- [5] ARM Ltd. Arm information center. <http://infocenter.arm.com>.
- [6] ARM Ltd. ARM The Architecture for the Digital World. <http://www.arm.com>.
- [7] Michael McNamara. Verilog.com. <http://www.verilog.com>.
- [8] Pouyan Afshar Mohammad Noman and Olugbenga Odesina. Low-power embedded processor design. Technical report, University of Connecticut, 2004.
- [9] David A. Patterson and Hohn L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface 3rd Edition*. Morgan Kaufmann, 2005.
- [10] Lauro Rizzatti. Defining the TLM-to-RTL Design Flow. *EETimes*, January 2007.
- [11] The ArchC Team. The ArchC Architecture Description Language. <http://www.archc.org>.
- [12] The ArchC Team. *The ArchC Architecture Description Language v2.0 Reference Manual*, August 2007.
- [13] Emmanuel Vaumorin and Thierry Romanteau. From Behavioral to RTL Design Flow in SystemC. *Design and Reuse*, March 2004.
- [14] Dr. Johannes Wolkerstorfer. Computer arithmetic: Algorithms and hardware design.

Anexos A

Descripción de señales de control

Señal	Descripción
<i>Address Register</i>	
addr_sel	selecciona el bus de entrada para el registro de dirección
inincw	para escribir latch de entrada del incrementador
<i>Register Bank</i>	
A_en	habilita salida a bus A
B_en	habilita salida a bus B
regA	número del registro para bus A
fmodeA	modo de operación del registro A a leer
PSRonA	indica que se desea leer un registro PSR en A
regB	número del registro para bus B
fmodeB	modo de operación del registro B a leer
PSRonB	indica que se desea leer un registro PSR en B
regd	número del registro de destino
fmoded	modo de operación del registro de destino
PSRond	indica que se desea grabar a un registro PSR
fmode	modo de operación de registro SPSR que se desea utilizar
CPSRsrc_sel	selecciona la fuente desde donde grabar CPSR
SPSRsrc_sel	selecciona la fuente desde donde grabar SPSR
ALUw	para escribir el resultado de la ALU al registro indicado
incw	para escribir bus de incrementador a PC
SPSRw	para escribir un registro SPSR
flags	para escribir flags en registro CPSR
reservw	para escribir los bits reservados en registro CPSR
Iw	para escribir bit deshabilitador de IRQ en registro CPSR
Fw	para escribir bit deshabilitador de FIQ en registro CPSR
Tw	para escribir bit de estado en registro CPSR
modew	para escribir modo de operación en registro CPSR

ANEXOS A. DESCRIPCIÓN DE SEÑALES DE CONTROL

Señal	Descripción
<i>Bits de estado (CPSR)</i>	
new_mode_sel	selecciona un nuevo modo de operación
nrst_flg	pone en cero flags de condición
Inew	Valor de I nuevo
Fnew	Valor de F nuevo
Tnew	Valor de T nuevo
<i>Multiplier</i>	
acum	indica si se activa la acumulación
long	indica si se realiza una multiplicación de 64-bits
ini	indica si se están inicializando los operandos
cycle	indica el ciclo en que se encuentra el algoritmo
sum_en	habilita la salida suma del multiplicador
carry_en	habilita la salida carry del multiplicador
opsw	para escribir operandos en latches internos del multiplicador
sig	indica si son operandos con signo
cinw	para escribir carry in interno de multiplicador
hi	indica si se esta calculando un resultado de 64-bits
<i>Barrel Shifter</i>	
shifttype	indica el tipo de desplazamiento que se desea
shiftamount	indica la cantidad de desplazamientos
sham_sel	selecciona entre shiftamount o la cantidad dentro del registro especial
shiftamw	para escribir registro especial para cantidad de desplazamientos
<i>ALU</i>	
ABw	para escribir latches de entrada a la ALU
opcode	código que indica que operación realizar
movA	indica si se desea mover directamente el bus A
movAph	indica en que fase se desea mover el bus A
incRes	para incrementar el resultado de la ALU en 4
<i>In/Out Bus B</i>	
MAS_sel	selecciona que tipo de dato es la salida
data_outw	para escribir bus B en registro de datos de salida
data_out_en	habilita salida a bus de datos
data_inw	para escribir bus de datos a registro de datos de entrada
data_in_en	habilita entrada de datos a bus B
MAS_s_sel	selecciona el tipo de datos de entrada

ANEXOS A. DESCRIPCIÓN DE SEÑALES DE CONTROL

Señal	Descripción
<i>Instruction Pipeline</i>	
imm_en	habilita valor inmediato para bus B
imm_sel	selecciona valor inmediato desde instrucción
instrfin	indica que la instrucción actual termina su ejecución
instrw	para grabar nueva instrucción al pipeline
flush	vacía pipeline
<i>Señales de excepciones</i>	
resetClear	limpia flag de reset
swiSet	setea flag de interrupción por software
swiClear	limpia flag de interrupción por software
undefSet	setea flag de instrucción indefinida
undefClear	limpia flag de instrucción indefinida

Anexos B

Descripción de módulos

B.1. Banco de registros

En la figura B.1 se muestra la estructura interna del banco de registro diseñado. Aquí se observan las señales comentadas anteriormente, donde tanto los registros SPSR como el registro CPSR tienen un fuente escogida por medio de un multiplexor. La señal *fmode* indica tanto que registro SPSR se escribe y cual se lee. La numeración del banco de registros es la indicada en la tabla B.1, donde destaca que si se indica el registro 31 se está seleccionando un registro PSR.

B.2. Multiplicador

El algoritmo del multiplicador requiere de 3 bloques indispensables para su efectiva funcionalidad. En primer lugar se necesita un llamado *Booth recoder*, este toma 3 bits del multiplicador y los codifica para realizar una multiplicación rápida de estos 3 bits. En la tabla B.2 se encuentra la codificación que realiza. Luego el resultado se aplica al generador de producto parcial como el que se muestra en la figura B.3. Estos productos parciales se suman por medio de *carry save adders*, los cuales son sumadores que no arrastran el resto de la suma de cada bit sino que su resultado es un valor de suma y otro de carries. Éstos, como se observan en la figura B.2, se suman en la ALU del microprocesador. Así el registro multiplicador (Rs) se va desplazando 8 bits por ciclo y hasta haber calculado los 4 productos parciales de multiplicar 32x8 bits. La suma y restos parciales se guardan en registros de 64 bits que en cada ciclo van rotando 8 bits. Para conocer más detalles del algoritmo ver referencias [8], [3], [14] y [2].

B.3. Control de transmisión de datos por bloque

Para implementar la transmisión por bloque se requirió de la estructura que se observa en la figura B.4. En principio se graba al latch REG_LIST la lista de registros que se obtiene de la instrucción. Luego el seleccionador escoge el primer registro de la lista y este se utiliza en

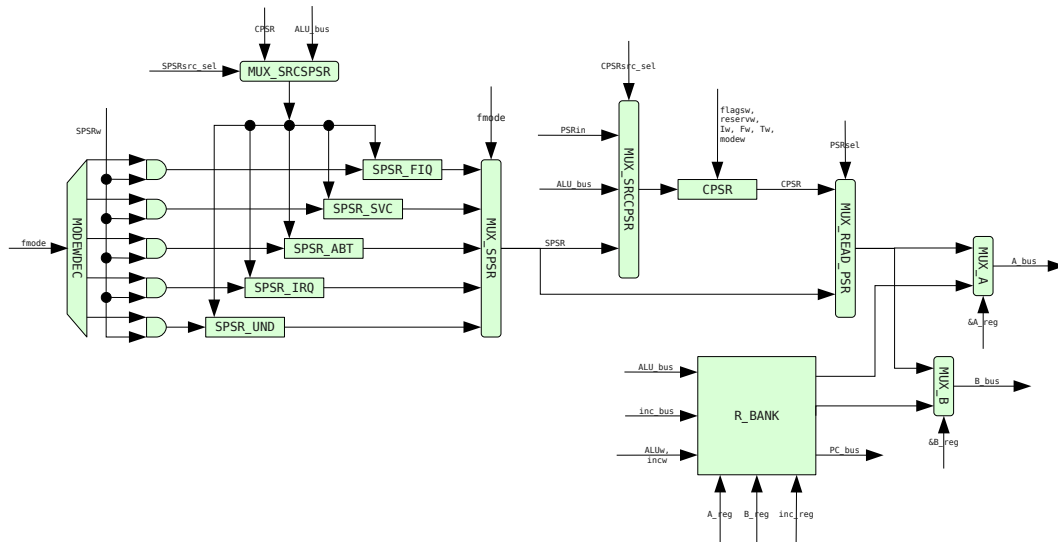


Figura B.1: Diagrama interior de banco de registros.

Índice de registro	Registro	Índice de registro	Registro
0	R0	16	R8.fiq
1	R1	17	R9.fiq
2	R2	18	R10.fiq
3	R3	19	R11.fiq
4	R4	20	R12.fiq
5	R5	21	R13.fiq
6	R6	22	R14.fiq
7	R7	23	R13.svc
8	R8	24	R14.svc
9	R9	25	R13.abt
10	R10	26	R14.abt
11	R11	27	R13 irq
12	R12	28	R14 irq
13	R13	29	R13.und
14	R14	30	R14.und
15	R15	31	PSR

Tabla B.1: Enumeración de registros

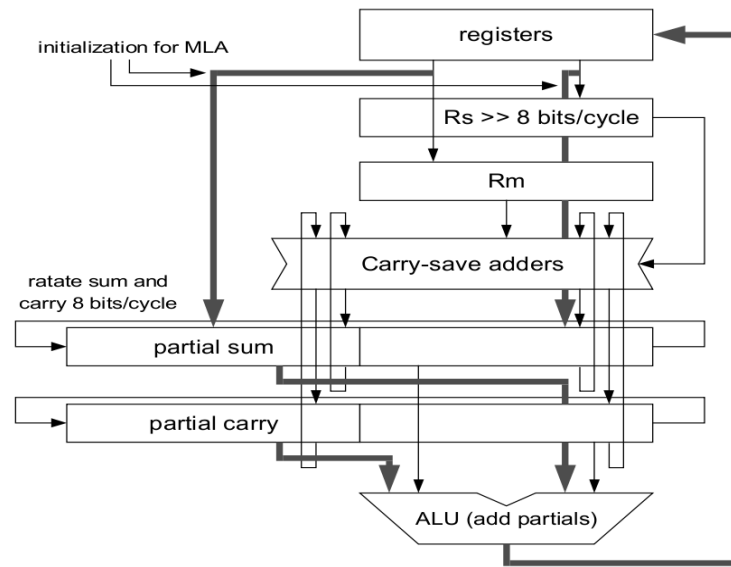


Figura B.2: Estructura del multiplicador.

b_{i+1}	b_i	b_{i-1}	Producto Parcial	neg	shift	en	
0	0		0	0 A	0	0	0
0	0		1	+1 A	0	0	1
0	1		0	+1 A	0	0	1
0	1		1	+2 A	0	1	1
1	0		0	-2 A	1	1	1
1	0		1	-1 A	1	0	1
1	1		0	-1 A	1	0	1
1	1		1	0 A	0	0	0

Tabla B.2: Enumeración de registros

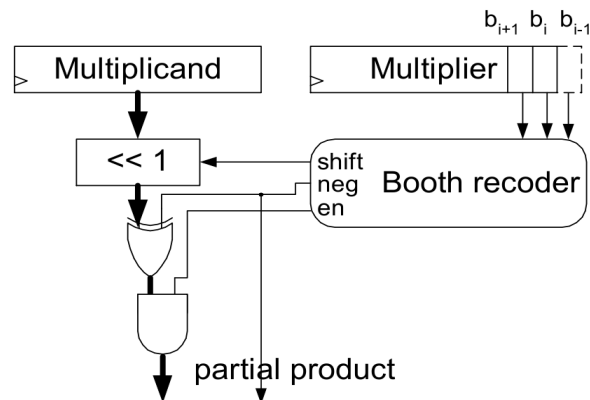


Figura B.3: Generador de producto parcial.

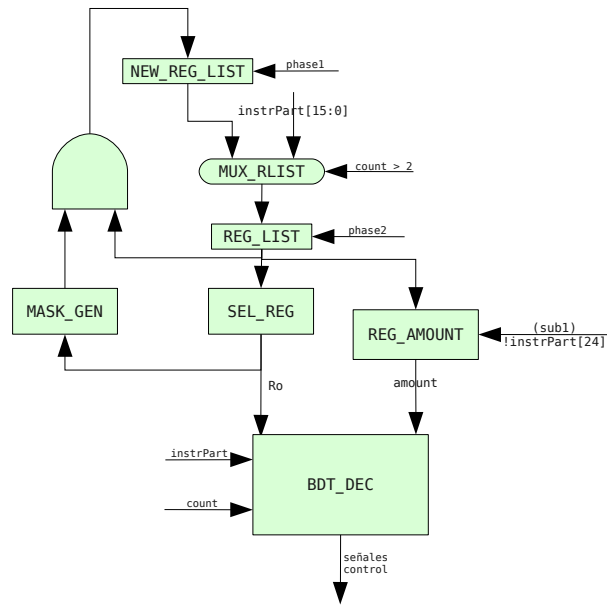


Figura B.4: Estructura de control de transmisión de datos por bloque.

el decodificador de la instrucción (BDT_DEC). Otra entrada al decodificador es la cantidad de registros por transferir que se obtiene de la lista de registros. Luego en un ciclo posterior MASK_GEN, se encarga de generar una mascara para borrar el registro ya utilizado de la lista de registros, así se renueva esta lista y continua el algoritmo hasta no tenemos más registros en ésta.

Anexos C

Diagrama completo de ARM7

En la figura C.1 se puede muestra el diagrama completo del microprocesador ARM7. Algunos detalles no mostrados anteriormente son 3 latches que indican si se activo el reset, una instrucción indefinida o una interrupción por software. Estos permiten que una vez detectadas y controladas esta excepciones, este flag sea limpiado. Además permiten formar una cola en caso de llegar excepciones simultáneamente. Y por último, existe un negador para nRESET ya que se necesita esta señal en algunos módulos.

Anexos D

Archivos de modelo de microprocesador

En el documento se adjunta un CD-ROM con los archivos de los modelos en ArchC y Verilog en sus correspondientes carpetas.

ArchC

arm7tdmi.ac	contiene la descripción de la arquitectura de memoria del procesador
arm7tdmi_isa.ac	contiene declaración de instrucciones, sus formatos y su decodificación
arm7tdmi_isa.cpp	contiene la descripción del comportamiento de cada instrucción
test	archivo hexadecimal de ejemplo

Verilog

adders.v	contiene módulos sumadores
address_inc.v	incrementador de dirección
alu_opcode.v	decodificador de opcode que genera señales para la ALU
alu.v	unidad aritmética lógica
arm7.v	modelo de microprocesador ARM7 con todos los módulos instanciados
barrel_shifter.v	desplazador rápido
buffers.v	archivo para buffers
exc_encoder.v	codificador para vector de excepción
instr_decoder.v	decodificador de instrucciones
instr_pipe.v	pipeline para instrucciones
latches.v	archivo con toda clase de latches
main_control.v	control principal
memoria.v	memoria utilizada en pruebas
mloadstore.v	control para instrucciones de transmisión de datos en bloque
mod_gates.v	archivo que contiene bloques especiales para operaciones lógicas
mult_control.v	controlador de multiplicación
multiplier.v	multiplicador
muxes.v	archivo con toda clase de multiplexores
prueba.hx	archivo de prueba hexadecimal
qsort_small.hx	archivo de prueba con algoritmo quick sort
reg_decoder.v	decodificador para numeración de registros
register_bank.v	banco de registros
sel_mas.v	seleccionadores de tipos de datos
test_arm7.v	archivo testbench para procesador
test.hx	archivo de prueba hexadecimal
testmult.hx	archivo de prueba de multiplicación