DISEÑO VHDL DE UN PROCESADOR DE OCHO BITS E IMPLEMENTACIÓN EN UN CPLD

Alexander Pareja Giraldo, Mario Enrique Vera Lizcano
Grupo de Biolectrónica y Nanoelectrónica, EIEE, Universidad del Valle
A.A. 25360 Cali, Colombia
alexpgq@hotmail.com, mario@univalle.edu.co

ABSTRACT

This article presents the design, modeling and implementation of an 8 bits dedicate processor, which uses Von Neumann architecture. The design is realized using the characteristics of a RISC processor. In order to reduce the system complexity a design methodology based on register transfer design concept is used, and the processor is modeled using VHDL, which allow to achieve a high grade of abstraction and hierarchy. The VHDL coding presents a low-medium abstraction level and the description is oriented for synthesis in order to optimize the hardware resources of the CPLD. The hardware architecture is synthesized by MAX+PLUS II on the CPLD EPF10K20RC240-4 circuit. In order to validate this design a memory and an I/O device are interfaced to the processor, and a test program is load, which allow to execute an algorithm, and the results are saved in the I/O device.

RESUMEN

Este articulo presenta el diseño, modelamiento e implementación de un procesador dedicado de ocho bits con arquitectura tipo Von Neumann. El diseño se realiza utilizando una filosofía orientada a procesadores tipo RISC. Para contrarrestar la complejidad del sistema se utiliza una metodología de diseño basada en RTD (Register Transfer Design) y se modela el procesador con VHDL, los cuales permiten manejar un alto grado de abstracción y jerarquía en el diseño. Los códigos planteados están en un nivel de abstracción medio-bajo y son descripciones orientados para síntesis en pro de optimizar la utilización de los recursos hardware del CPLD. La arquitectura hardware es sintetizada con MAX+PLUS II sobre el dispositivo de Altera EPF10K20RC240-4. Para validar el diseño se realizó la interconexión del procesador con una memoria y un puerto I/O, cargando un programa de prueba para ejecutar un algoritmo y capturando el resultado final en el puerto.

DISEÑO VHDL DE UN PROCESADOR DE OCHO BITS E IMPLEMENTACIÓN EN UN CPLD

Alexander Pareja Giraldo, Mario Enrique Vera Lizcano
Grupo de Biolectrónica y Nanoelectrónica, EIEE, Universidad del Valle
A.A. 25360 Cali, Colombia
alexpgq@hotmail.com, mario@univalle.edu.co

ABSTRACT

This article presents the design, modeling and implementation of an 8 bits dedicate processor, which uses Von Neumann architecture. The design is realized using the characteristics of a RISC processor. In order to reduce the system complexity a design methodology based on register transfer design concept is used, and the processor is modeled using VHDL, which allow to achieve a high grade of abstraction and hierarchy. The VHDL coding presents a low-medium abstraction level and the description is oriented for synthesis in order to optimize the hardware resources of the CPLD. The hardware architecture is synthesized by MAX+PLUS II on the CPLD EPF10K20RC240-4 circuit. In order to validate this design a memory and an I/O device are interfaced to the processor, and a test program is load, which allow to execute an algorithm, and the results are saved in the I/O device.

1. INTRODUCCION

Los procesadores programables realizan operaciones complejas por medio de una secuencia de operaciones elementales denominada conjunto de instrucciones [*IS: Instruction Set*] y codificadas usualmente por medio de comandos que se agrupan en programas para implementar cualquier algoritmo. El proceso de diseño de procesadores programables representa un alto grado de complejidad para los diseñadores de sistema, complejidad que se centra en los compromisos de diseño debidos a la interacción hardware–software requerida para diseñar un procesador óptimo.

Con el fin de contrarrestar la complejidad del diseño de procesadores, este se aborda utilizando los conceptos de jerarquía y abstracción; la jerarquía esta caracterizada por dos conceptos: arquitectura y organización. La arquitectura establece un nivel jerárquico de interfaz en el cual el diseñador (hardware) y el programador (software) perciben el mismo sistema, y la organización establece un nivel jerárquico centrado en la estructura hardware del sistema.

Con referencia al conjunto de instrucciones (IS), la arquitectura la define y la organización la implementa; estos dos conceptos están íntimamente relacionados estableciendo la arquitectura del conjunto de instrucciones [*ISA: Instruction Set Architecture*], que se refiere a los atributos que tienen un impacto directo en la ejecución lógica de un programa (visibles al programador), y la organización del conjunto de instrucciones [*ISO: Instruction Set Organization*] que se refiere a los atributos funcionales y sus interconexiones (transparentes al programador), los cuales materializan especificaciones arquitectónicas. [4][6]

Para contrarrestar la complejidad del diseño hardware del procesador, se utiliza como metodología el diseño con transferencia entre registros (RTD), orientada hacia la implementación de algoritmos en hardware. El modelo RTD utilizado para el diseño de un sistema se muestra en la figura 1 y está constituido por dos unidades: la unidad de procesamiento (*Datapath*) y la unidad de control (*Controlpath*). [1]

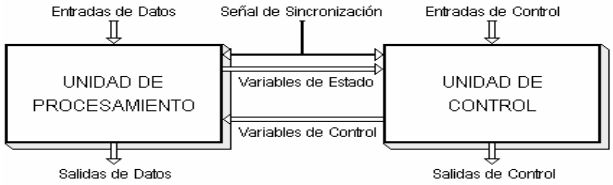

Figura 1. Diagrama de bloques del modelo RTD.

El *Datapath* esta formado por un conjunto de registros que almacenan los datos procesados (datos iniciales, datos intermedios y resultados finales), un conjunto de elementos de cálculo que desarrollan el procesamiento actual, y unos recursos de interconexión entre los registros y los elementos de cálculo. El *Controlpath* genera las señales que controlan la transferencia de información entre los diferentes componentes del *Datapath*, así como el siguiente estado de control a ejecutar [1][2][4]. La figura 2 muestra la estructura RTD general de un procesador.

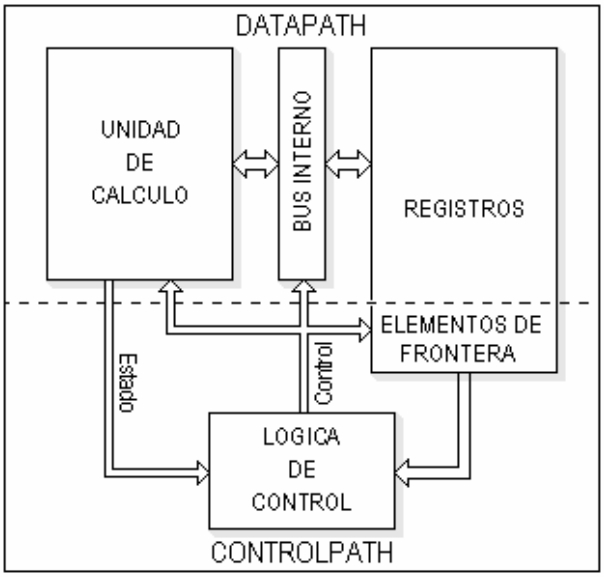

Figura 2. Estructura RTD general de un procesador.

2. DEFINICION DE LA ISA

Para definir las características de la ISA se plantean varias filosofías adoptadas por los procesadores RISC [3][5], el principal criterio de diseño a tener en cuenta es la uniformidad y la regularidad. También se deben considerar compromisos de diseño que permitan optimizar las prestaciones del procesador y faciliten su concepción. Como la ISA representa la confluencia conceptual entre el diseñador y el programador, se establecen compromisos de diseño que permitan la mejor iteración entre éstos.

2.1 Conjunto de Instrucciones

Entre las consideraciones de diseño adoptadas se encuentra la utilización de *opcodes* expansibles [7], que permiten una mayor versatilidad en la definición de un conjunto de instrucciones básicas y adicionar nuevas instrucciones cuando sea necesario.

Con este criterio se define un *opcode* con la mínima cantidad de bits del IR, que permita definir un conjunto de instrucciones y deje libres los bits necesarios para direccionamiento y ejecución. Con los cuatro bits más significativos del IR se codificaron 14 instrucciones, tres de ellas representan instrucciones que necesitan menor cantidad de campos para su interpretación y se dejan dos códigos libres para futuras expansiones. La tabla 1 resume el conjunto de instrucciones especificadas para el procesador.

Tabla 1. Conjunto de Instrucciones

OP	M	Instrucción	Mnemónico
0000		Resta con inmediato	SUBI
0001		Suma con inmediato	ADDI
0010		Or con inmediato	ORI
0011		And con Inmediato	ANDI
0100		Xor con inmediato	XORI
0101		Carga Inmediata	LDI
0110		Carga Directa por Registro	LDR
0111		Libre	Libre
1000	M	Carga	LD
1001	M	Almacenamiento	ST
1010	M	Llamada a procedimiento	CALL
1011	M	Salto Incondicional	JMP
1100		Libre	Libre
1101	M	Saltos Condicionales	Expansible
1110	M	Saltos Anticondicionales	Expansible
1111		Procesamiento-Misceláneas	Expansible

2.2 Formato de Instrucciones

Con el objeto de establecer una codificación compacta, estructurada y fácilmente decodificable se define un formato de instrucciones con longitud fija, por lo tanto es necesario definir diferentes tipos de formatos.

Para contrarrestar la complejidad hardware introducida por la multiplicidad de formatos se plantean campos relativamente uniformes y similares que se diferencian básicamente por el campo de *opcode*, el cual permite interpretar los campos complementarios.

Combinando los *opcodes* expansibles y los campos del formato se proponen cuatro tipos de formatos de acuerdo a la cantidad de bits del *opcode*: FA(4), FB(6), FC(8), FD(12). La figura 3 muestra los diferentes tipos de formato propuestos.

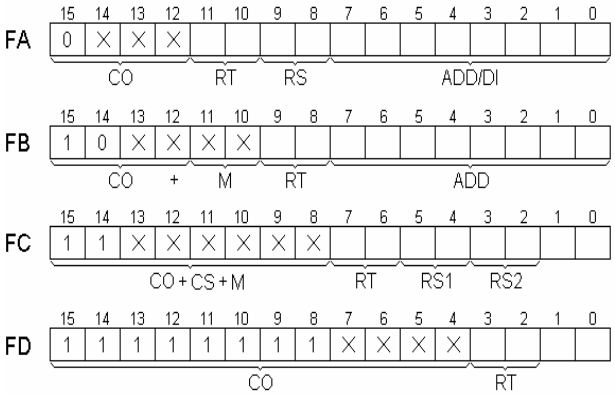

Figura 3. Tipos de formato de instrucción

2.3 Modos de Direccionamiento

Para suministrar versatilidad y flexibilidad en la programación se definen siete modos de direccionamiento suficientes para implementar las construcciones básicas de los lenguajes de programación. Estos modos de direccionamiento permiten determinar la dirección efectiva de los operandos y/o instrucciones; como solo unas pocas instrucciones utilizan las potencialidades del direccionamiento se íntegra el campo de modo en el *opcode*. Los modos de direccionamiento disponibles son:

- Implícito
- Inmediato
- Directo por registro
- Desplazamiento por registro Base
- Desplazamiento relativo al IP
- Indexado por registro
- Absoluto paginado

3. DEFINICION DE LA ISO

El procesador ejecuta las instrucciones siguiendo un ciclo básico de operación, por medio de una secuencia de microoperaciones, estas son:

- Fase de búsqueda de instrucción (IFP: Instruction Fetch Phase), en esta fase se obtiene la instrucción de memoria y se almacena en un registro de control.
- Fase de decodificación de instrucción (IDP: Instruction Decoded Phase), en esta fase se interpreta la instrucción, cuando se requiera de operandos en memoria o nuevas instrucciones se calcula la dirección efectiva para obtenerlos.
- Fase de ejecución de instrucción (IEP: Instruction Execute Phase), en esta fase se realiza el procesamiento de datos para ejecutar las instrucciones.

La ISA en conjunto con el ciclo básico de operación establecen los requerimientos funcionales mínimos para la implementación adecuada de la misma, por lo tanto se pueden plantear todos los requerimientos funcionales del procesador basados en estos dos conceptos. La descripción del procesador objeto de este articulo está basada en el diseño RTD, por lo cual se define la ISO como un conjunto de componentes de cada uno de los bloques básicos del diseño RTD.

3.1 Datapath

Los requerimientos funcionales para el *datapath* se pueden agrupar en: almacenamiento, direccionamiento, procesamiento e interfaz con memoria. De estos bloques solo son visibles al programador cuatro registros de propósito general agrupados en una unidad llamada GPR. La figura 4 muestra la estructura interna del *datapath*.

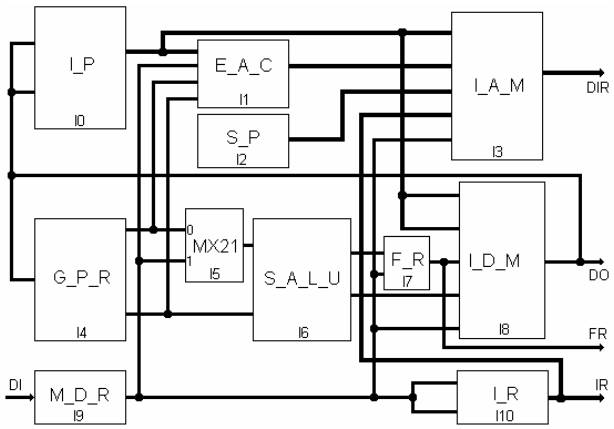

Figura 4. Estructura del *Datapath*.

3.1.1 Almacenamiento

Los requisitos funcionales básicos para el almacenamiento se establecen por la IEP y el conjunto de instrucciones, que plantean la necesidad de utilizar un conjunto de registros para almacenamiento temporal (GPR). La IFP sugiere la utilización de un registro para almacenamiento intermedio (MDR) para facilitar la interacción con memoria. La IDP establece la necesidad de un registro para almacenar las instrucciones (IR) y que estén disponibles para su decodificación en los instantes adecuados. Las instrucciones de control de programa requieren la utilización de un registro de banderas (FR) para el procesamiento de los saltos condicionales y anticondicionales.

3.1.2 Direccionamiento

Los requisitos funcionales para el direccionamiento de memoria los determinan la IFP y el formato de instrucción con el campo de dirección de la siguiente instrucción implícito que establecen la necesidad de un puntero de instrucciones (IP) para el secuenciamiento del programa.

Las instrucciones de control como el procesamiento de subprogramas e interrupciones requieren la utilización de una pila de memoria para salvaguardar temporalmente el estado del procesador y consecuentemente un registro puntero de pila (SP) para controlar este proceso. Los diferentes modos de direccionamiento planteados requieren de un bloque hardware encargado de calcular la dirección efectiva (EAC) de los operandos y/o instrucciones. El puntero de instrucciones (IP) propuesto es de 16 bits y el puntero de pila (SP)es de 8 bits

Para solucionar el compromiso establecido entre la longitud del formato y los modos de direccionamiento se diseño una unidad para el cálculo de las direcciones efectivas. La tabla 2 muestra la funcionalidad de este elemento.

Tabla 2. Modos de Direccionamiento.

Modo	S	Dirección Efectiva
Absoluto paginado	00	[IPH*100H] + MDR
Registro base	01	[RD*100H] + MDR
Relátivo al IP	10	[IP] + MDR
Indexado por Registro	11	[RD*100H] + [RS]

3.1.3 Procesamiento

Para suministrar potencialidad funcional al procesador se requiere de un bloque de hardware denominado: unidad aritmético–lógica–desplazamiento (SALU) y un selector de datos (MX21) para instrucciones con direccionamiento inmediato.

Para implementar la totalidad de instrucciones planteadas se diseño la SALU con las operaciones planteadas en la tabla 3.

Tabla 3. Operaciones de la SALU.

NA	OD	S(2)	S(1)	S(0)	NEMONICO
0	0	0	0	0	ROL B
0	0	0	0	1	SHL B
0	0	0	1	0	ROR B
0	0	0	1	1	SHR B
0	0	1	0	0	ROLB
0	0	1	0	1	SHL B
0	0	1	1	0	TRF B
0	0	1	1	1	SAR B
0	1	0	0	0	INC A
0	1	0	0	1	SUB A,B
0	1	0	1	0	SUM A,B
0	1	0	1	1	DEC A
0	1	1	0	0	OR A,B
0	1	1	0	1	AND A,B
0	1	1	1	0	XOR A,B
0	1	1	1	1	NOT A
1	X	X	X	X	----

Estas operaciones se seleccionaron de forma que se simplificará el hardware requerido; con cuatro bits hubiese sido suficiente para expresar las operaciones necesarias, sin embargo se hizo deseable incluir la operación de negación en este conjunto y la forma más eficiente fue utilizar un bit adicional para complementar el dato de entrada A, que en combinación con la operación de incremento permitió formar la operación deseada

3.1.4 Interfaz con Memoria

Los requisitos funcionales para realizar la interfaz con memoria están determinados por los múltiples elementos que necesitan interactuar con la memoria (IP, EAC, SP, MDR, FR, SALU); cada uno de estos bloques accede ya sea al bus de direcciones o al bus de datos y por lo tanto se necesita un mecanismo para controlarlo. el mecanismo planteado se concentra en la multiplexación de los buses de direcciones y de datos, no solo por ser la solución directa que esta más accesible, sino por la limitación del compilador seleccionado en el procesamiento de buses compartidos y por al ausencia en el dispositivo seleccionado de buses triestado internos. Para la interfase se requiere entonces de una interfase con las direcciones de memoria (IAM) y de una interfase con los datos de memoria (IDM).

3.2 Controlpath

Los requisitos funcionales para la unidad de control del procesador se establecen básicamente por las fases para la ejecución de una instrucción: unidad para búsqueda de instrucciones (IFETCH), unidad para decodificación de instrucciones (IDECODE) y unidad para la ejecución de instrucciones (IEXECUTE); adicionalmente se establece la necesidad de un bloque hardware para el reconocimiento de interrupciones (IAC). La unidad de control planteada unifica estos bloques en una máquina de estados y realiza las transiciones adecuadas en cada caso, la figura 5 muestra el diagrama de estados.

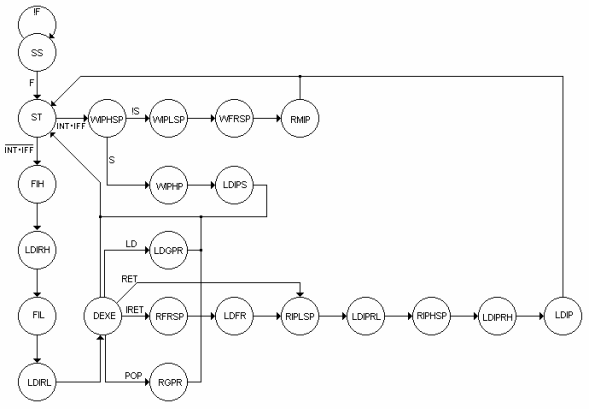

Figura 5. Diagrama de Estados para el *Controlpath*

La unidad de control se diseña pensando en el ciclo básico de operación, la máquina de estados inicia en un estado de espera y sale de éste cuando se le hace una

solicitud de fetch para obtener la instrucción de memoria y almacenarla en el IR, este procedimiento dura cinco ciclos de reloj. En el siguiente flanco se transfiere el control al estado de decodificación y ejecución (epicentro de la máquina), donde se utilizan el registro de instrucción y el registro de banderas provenientes del *datapath* para tomar decisiones de transición y generar las señales necesarias para ejecutar la instrucción indicada.

4. SINTESIS DEL PROCESADOR

El modelamiento VHDL del procesador se realiza pensando en descripciones optimizadas para síntesis, para cada unidad modelada se realizan diversas descripciones y se selecciona la que presente mejores desempeños en cuanto a área y velocidad. Por lo tanto las descripciones se realizan pensando en el hardware intentado (nivel de abstracción medio–bajo) y utilizando generalmente tipos de datos orientados a hardware (Std_logic).

El escribir código VHDL para síntesis economiza recursos hardware, para la tecnología de implementación seleccionada (CPLD) esto implica mejoras en la velocidad por la utilización de menos recursos de ruteo para la interconexión de las celdas lógicas.

Las descripciones se particionan de acuerdo a la metodología de diseño RTD y se estructuran progresivamente hasta conformar el modelamiento completo del procesador.

4.1 Datapath

Los lineamientos hechos establecen la necesidad de realizar descripciones con diferentes grados de detalle, por lo tanto se utilizan todos los estilos de modelamiento suministrados por el VHDL. Para facilitar la concepción de las descripciones para el *datapath* éstas se realizan subdividiéndolas por funcionalidades, tal como se expreso en la definición de la ISO; sin embargo la estructuración del *datapath* se realiza instanciando todos los componentes modelados directamente en el diseño de mayor jerarquía (*datapath*).

Las descripciones realizadas para los elementos de almacenamiento utilizan modelamientos secuenciales puros (con un solo proceso) controlados por la función RISIND_EDGE que infiere Flip–Flops D. Para los elementos combinacionales se utiliza bien sea un híbrido entre el modelamiento funcional y el flujo de datos o múltiples procesos; para los elementos de interfaz se utiliza la multiplexación que se infiere por medio de sentencias case.

El *datapath* requiere de 393 LES (34%) para su implementación con un fan–in promedio del 84%; el retardo crítico para este diseño es 85.5ns y la frecuencia máxima de operación es 12.87Mhz. como cada LE tiene un total de doce puertas equivalentes utilizables se requieren un total de 4716 puertas equivalentes de dos entradas para este circuito.

4.2 Controlpath

La descripción realizada para el control utiliza modelamiento secuencial con dos procesos: un proceso encargado de generar la lógica secuencial y otro proceso encargado de generar la lógica combinacional. La lógica secuencial garantiza un estado de inicialización y genera las transiciones de estado, la lógica combinacional condiciona las transiciones de estado y genera las señales de control necesarias para la ejecución de las instrucciones. La máquina de estados es de tipo Mealy porque en algunas instrucciones el registro de instrucción transfiere información a las señales de control.

El *Controlpath* requiere de 438 LES (38%) para su implementación con un fan–in promedio del 88%; el retardo crítico para este diseño es 85.1ns y la frecuencia máxima de operación es 18.62Mhz. Como cada LE tiene un total de doce puertas equivalentes utilizables se requieren un total de **5256** puertas equivalentes de dos entradas para este circuito.

4.3 Estructurando el procesador

Para estructurar el procesador basta con realizar un modelamiento estructural que instancia las dos unidades descritas previamente en esta sección. El procesador requiere de 825 LES (71%) para su implementación con un fan–in promedio del 86%; el retardo crítico para este diseño es 155.6ns y la frecuencia máxima de operación es 4.65Mhz. Como cada LE tiene un total de doce puertas equivalentes utilizables se requieren un total de 9900 puertas equivalentes de dos entradas para este circuito.

Como se puede observar de los resultados obtenidos las celdas lógicas se están utilizando eficientemente porque en promedio se supera el 86% de cargabilidad de los elementos lógicos.

5. VALIDACIÓN DEL PROCESADOR

Para verificar el correcto funcionamiento del procesador se plantea una aplicación que utiliza algunas de las instrucciones del procesador para realizar una serie de cálculos, para ello se conecta el procesador con una memoria donde se almacena el programa a ejecutar y un puerto de salida donde se captura el resultado final. Inicialmente se realizan operaciones de transferencia inmediata y direccionada, posteriormente se realizan cálculos con datos inmediatos y entre registros y se transfiere el contenido de uno de los registro a memoria,

finalmente se captura en el puerto de salida el resultado para la validación del programa. La figura 6 muestra el esquemático para la validación del procesador.

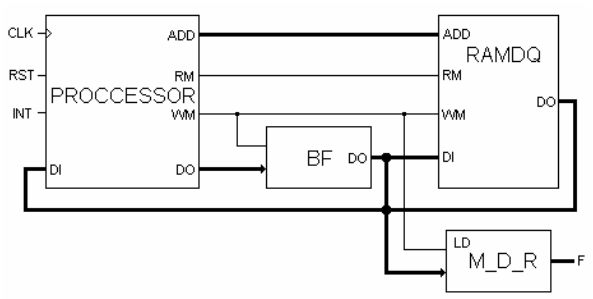

Figura 6. Esquemático para el programa de prueba.

La memoria utilizada es la megafunción suministrada por altera (lpm ramdq) con una pequeña modificación (control de lectura no registrado y salida triestado). El puerto de salida corresponde al modelo del registro M_D_R diseñado para el procesador; además se añadió un buffer triestado para controlar el acceso al bus compartido por el procesador y la memoria. La figura 7 muestra el programa utilizado para validar el funcionamiento del procesador.

```
DEPTH = 256;
WIDTH = 8;
ADDRESS_RADIX = HEX;
DATA_RADIX    = HEX;

CONTENT
  BEGIN
    [00..FF] : 00;
    000    : 50 01; --LDI  RA,01
    002    : 81 18; --LD   RB,[18H]
    004    : 58 03; --LDI  RC,03
    006    : FA E4; --ADD  RD,RC,RB
    008    : 0C 05; --SUBI RD,05
    00A    : F7 00; --INC  RA
    00C    : FB 40; --DEC  RB
    00E    : F0 40; --ROL  RB
    010    : FD A4; --AND  RC,RC,RB
    012    : 18 18; --ADDI RC,18
    014    : F8 00; --NEG  RA
    016    : 90 19; --ST   [IPH*100H+DIR],RA
    018    : 02 00; --DATO
END;
```

Figura 7. Programa de prueba.

El procesador inicialmente carga un dato inmediato (01H) en el registro A del GPR, carga un dato en el registro B almacenado en la última línea del programa de prueba (18H) y carga otro dato inmediato (03H) en el registro C del GPR. Teniendo esta información en los registros se procede a realizar algunas operaciones aritméticas y lógicas entre ellos.

Posteriormente se suma el contenido de RB (02H) con el contenido de RC(03H) y se almacena el resultado en RD (05H), luego restamos al contenido de RD un dato inmediato (05H) y el resultado se almacena en RD (00H), con estas dos operaciones se validan operaciones aritméticas entre registros y con datos inmediatos.

Luego se procede a incrementar el valor de RA (02H), luego se decrementa RB (01H) y se rota a la izquierda (02H). La siguiente operación valida las operaciones lógicas entre registros, realizando una and entre RB (02H) y RC (03H) y la almacena en RC (02H).

Finalmente se suma RC (02H) con un dato inmediato (18H) y se niega el contenido de RA (FEH) para almacenarlo en la posición de memoria 19H. La figura 8 muestra la simulación donde se puede observar la evolución correcta del programa, su correcta ejecución y el almacenamiento del contenido de RA en memoria en el ciclo 74 del reloj.

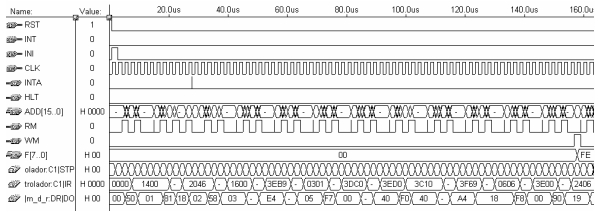

Figura 8. Simulación para el programa de prueba.

La mayoría de las instrucciones ejecutadas necesitan un mínimo de 5 ciclos de reloj y como el programa tiene 13 instrucciones se requieren aproximadamente 78 ciclos de reloj para ejecutar el programa, se debe tener en cuenta que no todas las instrucciones tienen la misma duración.

6. CONCLUSIONES Y TRABAJO FUTURO

La principal conclusión de este trabajo es resaltar la viabilidad de diseñar sistemas complejos utilizando una metodología de diseño RTD para la especificación, VHDL para la descripción y los PLDs para la implementación de sistemas complejos.

Con la metodología utilizada para el diseño del procesador, aunque no se trabaja directamente sobre los elementos lógicos (LEs), se le suministra al compilador un código VHDL en un nivel de abstracción medio-bajo orientado para síntesis que permite alcanzar mejores resultados en área y velocidad.

El procesador desarrollado permite resolver diversidad de problemas de forma secuencial suministrando versatilidad en la integración de sistemas digitales complejos, economizando recursos hardware, esfuerzo de ingeniería y tiempo de desarrollo.

Como el sistema procesador se especifica y diseña completamente considerando opciones de diseño

particulares, se cuenta con un sistema procesador con cero soporte software y por lo tanto resulta interesante plantear como trabajo futuro el desarrollo de un lenguaje ensamblador para este procesador que permita una programación rápida y eficiente.

La solución propuesta plantea todo un conjunto de consideraciones de diseño basadas en la filosofía de diseño de procesadores tipo RISC, para implementar un procesador en una arquitectura tipo CISC, por lo tanto el procesador diseñado podría ser tipo "CRISC". Para mejorar el desempeño del procesador se puede plantear como trabajo futuro el diseño de un procesador con arquitectura tipo RISC y utilizar algunas estrategias de diseño como la segmentación.

7. REFERENCIAS

[1] DAVIO Marc y otros, "Digital Systems with Algorithm Implementation", John Wiley & Sons, Belfast 1983.

[2] GAJSKI D. Daniel, "Principios de Diseño Digital", Prentice Hall, Madrid, 1997.

[3] HENNESSY L. John y PATTERSON A. David, "Arquitectura de Computadores: Un Enfoque Cuantitativo", MacGraw Hill, Madrid, 1993.

[4] MANO M. Morris y KIME R, Charles, "Fundamentos de Diseño Lógico y Computadoras", Prentice Hall , México D.F, 1998.

[5] PATTERSON A. David y HENNESSY L. John, "Computer Organization & Design: The Hardware/Software Interface", Morgan Kaufmann Publishers 2ª ed, San Francisco, 1998.

[6] STALLINGS William, "Organización y Arquitectura de Computadores: Diseño para optimizar prestaciones", Prentice Hall 5ª ed, Madrid 1999.

[7] TANENBAUM S. Andrew, "Organización de Computadoras: Un Enfoque Estructurado", Prentice Hall 4ª edición, México D.F, 2000.