Implementación de una Arquitectura de Microprocesador didáctica en VHDL

Cristian Luis Torres
María Paula Frade
Emmanuel Iarussi
Director: Martín Vazquez

Facultad de Ciencias Exactas - UNCPBA
Concurso de trabajos estudiantiles 2011
Categoría: Trabajo de Cátedra

4 de julio de 2011

Resumen

El presente trabajo fue realizado en calidad de Trabajo Final para una cátedra que presenta los fundamentos de Arquitectura de Computadoras (Arquitectura de Computadoras I) en una carrera de Informática. A fin de introducir lenguajes de descripción de hardware al contenido de dicha materia, se realizaron durante la misma diferentes implementaciones de circuitos digitales que ya eran dictados. Se propuso una implementación VHDL del microprocesador DW-B basado en el microprocesador DWARF. Además, fue deseable lograr una descripción sintetizable de la arquitectura, para que la misma pudiera ser implementada a posteriori dentro de una FPGA. Como complemento a la descripción se diseñó e implementó una herramienta traductora mediante la cual pueden desarrollarse programas para ejecutar en dicho procesador.

1. Introducción

Es ya conocida la dificultad existente en el aprendizaje de los contenidos correspondientes al área de Arquitectura de Computadoras en carreras de Informática, especialmente cuando se trata de comprender y construir arquitecturas de microprocesadores. Los mismos incluyen una gran carga de contenidos teóricos dificilmente comprensibles sin la ayuda de la experiencia práctica. En este contexto, resulta útil poseer herramientas que posibiliten dicha experimentación. Este trabajo ha sido realizado en el marco de una cátedra que recientemente introdujo la enseñanza del lenguaje VHDL(*Very High Speed Integrated Circuit Hardware Description Language*)[5] con el objetivo de atemperar dicha problemática. Este lenguaje es una heŕramienta definida por IEEE para la descripción de circuitos digitales que permite además, realizar síntesis de estas descripciones en PLD (*Programable Logic Device*) o FPGA (*Field Programmable Gate Array*).

Tal como ha sido descripto en [3], uno de los principales problemas a la hora de enseñar diseño digital es si se deben utilizar las tecnologías de FPGA para implementar los sitemas creados durante el curso. Es posible esbozar una respuesta afirmativa, si nos basamos en el valor de la experiencia de construir hardware real. La utilización de estos lenguajes permiten a los estudiantes escribir descripciones de hardware como si fuera software en vez de "cortar y conectar cables". Desde el punto de vista del profesor, resulta sencillo brindar las herramientas necesarias para que el alumno realice sus propias descripciones en laboratorios donde sólo se necesitan entornos sobre los cuales desarrollar el código. Todos estos factores ayudan a mejorar el tamaño y el desafío de los diseños sobre los que los estudiantes pueden trabajar.

El objetivo de este trabajo ha sido realizar una descripción de un procesador sencillo y didáctico, que pudiera utilizarse a modo de ejemplo para futuros trabajos de la cátedra y especialmente, para las prácticas especiales donde se aprende a utilizar este lenguaje. Para esto, se tomó como referencia el procesador DWARF [1]. Dicha elección se debe a dos motivos principales. En primer lugar, su sencillez y simplicidad en cuanto a los componentes que lo conforman, lo vuelve fácil de entender y de implementar. En segundo lugar, este procesador resulta atractivo debido a la gran cantidad de bibliografía disponible al respecto.

De todos modos, fueron realizados algunos cambios para simplificarlo, adaptándolo mejor a los objetivos propuestos. Es asi como surge el microprocesador didáctico DW-B, que posee algunas diferencias con el DWARF:

- No existe manejo de interrupciones. El mecanismo de interrupciones fue anulado únicamente con el objetivo de quitar complejidad al problema de rediseñar el procesador. No esta entre los propósitos de la Cátedra para la que fue realizado este trabajo, ahondar en estos temas.
- Las memorias son implementadas dentro de la FPGA y de forma separada (ROM y RAM). El DWARF poseía una única memoria externa a la FPGA. Además, al incorporarlas, las mismas trabajan de forma sincrónica.
- Se alteró el juego de instrucciones, agregando las operaciones lógicas "and" y "or", aritméticas "sub" y de salto "jo". Además, como fue anulado el manejo de interrupciones, debió quitarse la instrucción de retorno de interrupción "reti".

A continuación, se comenzará por una breve explicación del procesador DW-B (descripción arquitectónica, formatos de instrucción, etc.), para luego realizar el análisis detallado de todos los componentes y finalmente mostrar los resultados obtenidos con la realización de este trabajo.

2. Definición del problema

2.1. Descripción de la arquitectura DW-B

Tal como se ha dicho en secciones anteriores, a partir de la arquitectura del procesador didáctico DWARF propuesto por [1], se realizó una adaptación que lo simplificara, sin quitar la funcionalidad básica mas importante. En el siguiente esquema podemos observar, con un grado elevado de abstracción, la arquitectura interna del microprocesador. No fueron representadas en este esquema todas las conexiones por razones de complejidad en el diagrama. El decodificador de instrucción, a pesar de su aparente aislamiento en el esquema, esta conectado con todos los componentes del DW-B. Cada componente será descripto en detalle en las secciones posteriores.

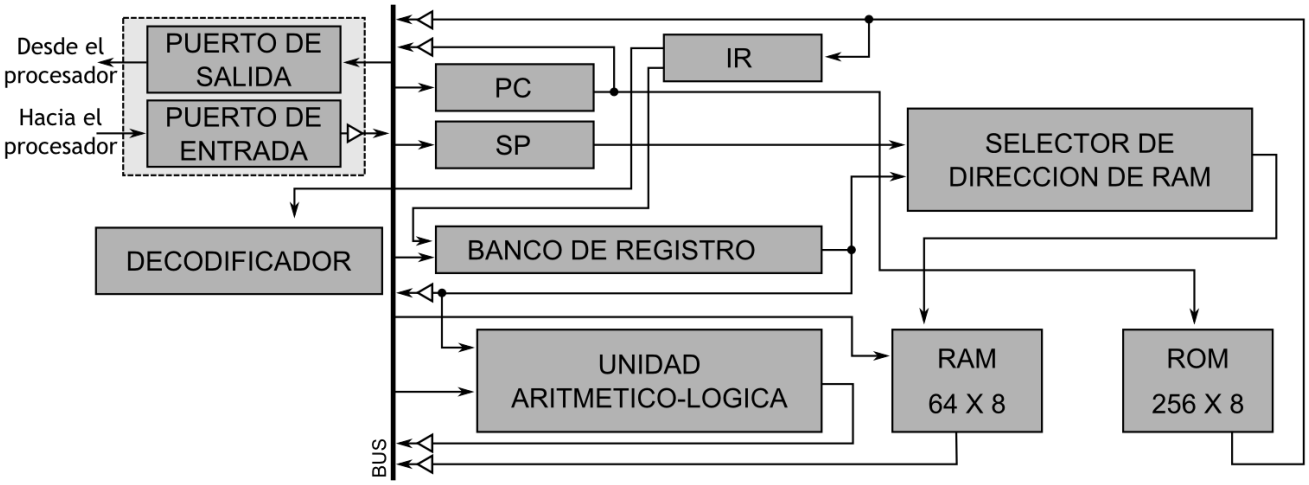

Fig. 1. Esquema de componentes del microprocesador

Según muestra el esquema, la arquitectura posee un diseño clásico, con dos memorias internas. La ROM es utilizada para almacenar el programa y la RAM se reparte en dos segmentos, sección de datos y sección de pila. Contamos además con un banco de registros con 8 unidades de un byte.

2.2. Conjunto de Instrucciones del DW-B

El DW-B es un procesador de tipo RISC (*Reduced Instruction Set Computer*), es decir que utiliza un juego de instrucciones reducido y simple que permite una ejecución más rápida y eficiente. Se dispone de un conjunto de 23 instrucciones, el cual podría extenderse a 32 en el caso de que se desee incorporar nuevas funcionalidades.

Se poseen 5 instrucciones de carga y 4 de almacenamiento, además de las instrucciones de lectura y escritura de puertos de entrada/salida. La unidad aritmético lógica provee 5 operaciones y existen 5 saltos por diferentes condiciones.

2.2.1. Formato de Datos e Instrucciones

Las instrucciones del DW-B pueden ocupar 1 o 2 bytes de espacio, según los datos necesarios para la operación que se desee realizar. Cada una contiene 5 bits iniciales que constituyen el código de operación, identificando unívocamente una instrucción, seguidos de 3 bits utilizados para retener la dirección del registro interno. Algunas instrucciones no utilizan estos últimos debido a que no operan sobre registros. Las instrucciones de 16 bits son utilizadas en caso de requerir datos o direcciones de memoria, codificando dicha

información en los 8 bits adicionales. Para el direccionamiento, dado que la memoria de datos **DATA_RAM** es de 32 bytes, se utilizarán 5 bits para direccionarla, mientras que para la memoria de pila **STACK_RAM** de 16 bytes se necesitarán 4 bits. Dado que el ancho de palabra es de 8 bits, los datos a introducir ocuparán 1 byte de espacio. A continuación se detalla una generalización de los diferentes formatos de instrucciones del DW-B:

Tabla 1. Conjunto de instrucciones

Nemónico	Descripción	Nemónico	Descripción
LOAD_REG	Carga con dir. de registro	AND_REG	Operación and
LOAD_INX	Carga con dir. indexado	OR_REG	Operación or
LOAD_INM	Carga con dir. inmediato	ADD_REG	Operación add
LOAD_DIR	Carga con dir. directo	SUB_REG	Operación sub
LOAD_IND	Carga con dir. indirecto	JC	Salto cond. por flag de carry
STORE_REG	Alm. con dir. de registro	JO	Salto cond. por flag de overflow
STORE_INX	Alm. con dir. indexado	JZ	Salto cond. por flag de cero
STORE_DIR	Alm. con dir. directo	JS	Salto cond. por flag de signo
STORE_IND	Alm. con dir. indirecto	JUMP	Salto incondicional
IN_REG	Carga de registro por puerto	JSR	Salto a subrutina
OUT_PORT	Pasaje del acumulador a puerto	RET	Retorno de subrutina
XOR_REG	Operación xor		

3. Implementación de la Arquitectura

3.1. Componentes

3.1.1. Puertos de Entrada y Salida

El microprocesador DW-B posee 2 puertos de propósito general de 8 bits, **PORT_IN** y **PORT_OUT**, los cuales son de entrada y de salida respectivamente. En caso de no haber un valor, en el puerto de salida se tendrá 'Z' (alta impedancia).

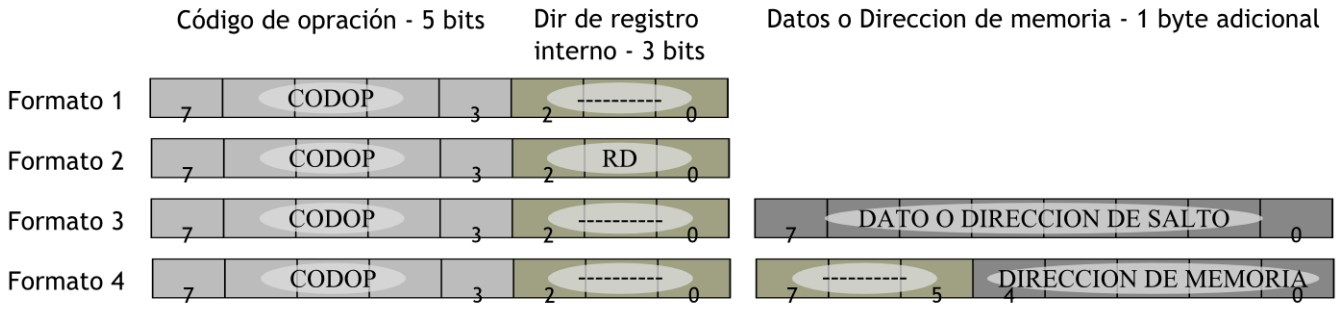

Fig. 2. Formatos de Instrucción

3.1.2. RAM de datos y pila

Consiste en una memoria de 64 bytes asincrónica respecto al clock, incorporada al microprocesador. Es necesario en este punto hacer una salvedad. La memoria original del DWARF estaba pensada para ser un

componente totalmente externo al microprocesador. Como ya se ha mencionado, la intensión guía del trabajo ha sido simplicar tanto como sea posible el diseño. Por esta razón, la memoria original fue dividida en dos unidades (ROM y RAM) de tamaños mas pequeños, para simplificar su síntesis en FPGAs de tamaño reducido, obteniendo de esta forma, un procesador dedicado.

La memoria RAM posee una señal de *escritura/no lectura* con clock bajo que permite cargar una dirección de memoria con un dato proveniente de la entrada de 8 bits conectada al bus. La señal de direccionamiento se compone de 6 bits. Es generada en el selector de RAM teniendo en cuenta que parte de la memoria se desea acceder (alta o baja) y a que localidad dentro de la misma. La lectura se realiza también asincrónicamente, para disponer de un dato sólo se debe dar el control del BUS a la salida de la RAM.

La memoria se encuentra dividida en 3 (tres) regiones:

- **DATA_RAM**: corresponde a la parte inferior de la RAM en la que se almacenan los datos cargados por el usuario y/o los que está utilizando el programa. Su tamaño es de 32 bytes, correspondiendo al rango de direcciones que va del 000000_b al 011111_b (0 a 31 en decimal).
- **STACK_RAM**: corresponde a la parte alta de la RAM en la que se almacena el PC durante las llamadas a subrutinas, o sea la pila. Su tamaño es de 16 bytes, correspondiendo al rango de direcciones que va del 110000_b al 111111_b (48 a 63 en decimal).
- **No usada**: existe un rango de memoria de 16 bytes no utilizado ubicado entre la **DATA_RAM** y la **STACK_RAM**, correspondiente a las direcciones 100000_b a 101111_b (32 a 47 en decimal). Este rango de memoria no utilizada solo se mantuvo por compatibilidad con el microprocesador DWARF, ya que este también poseía un sector de RAM inutilizado.

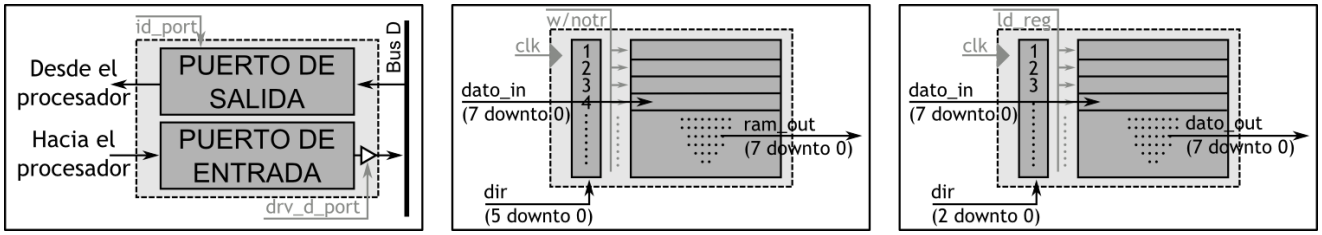

Fig. 3. Izquierda: esquema de puertos - Centro: esquema de la Memoria RAM - Derecha: esquema del Banco de Registros

3.1.3. Banco de registros

Este módulo está compuesto por 8 registros sincrónicos de 8 bits cada uno. Podemos destacar las siguientes características:

- Posee una señal de direccionamiento de de tamaño tres, conectada a los 3 bits menos significativos del registro de instrucción (RI), la cual permite referenciar los registros del 000_b al 111_b (0 a 7 en decimal).
- El dato a guardar en el banco proviene de una entrada de 8 bits conectada al BUS.
- Tiene una señal de load (**ld_reg**) que, si se encuentra alta, permite realizar la escritura en un registro del banco.

- Se destaca dentro del mismo la existencia del registro especial R0, el cual es utilizado por distintas instrucciones para el almacenamiento temporal de información. Se activa a través de la señal **r0** que permite direccionarlo ya sea para escritura o lectura según indique la señal de **ld_reg** del banco de registros.

3.1.4. Selector de dirección de Ram

Este módulo esta implementado básicamente a modo de multiplexor, el cual se encarga de generar la dirección de RAM a la que se desea acceder. Para ello necesita una señal de control **addr_sel** que permite direccionar la memoria de la siguiente manera:

1. Si es cero, se seleccionará la **DATA_RAM**.
2. Si es uno, se seleccionará la **STACK_RAM**.

Luego, las direcciones son construidas como sigue:

1. Se le concatena al valor '0' la dirección proveniente del banco de registros (desde la posición 4 a la 0), conformando una salida de 6 bits que se conecta a la entrada de direccionamiento de la RAM.
2. Se le concatena al valor "11" la dirección proveniente del puntero de pila (SP), conformando una salida de iguales características que la anterior.

3.1.5. Puntero de pila (*Stack Pointer*)

Este componente posee un registro que almacena el tope de pila que apunta a la **STACK_RAM**. Este registro tiene un tamaño de 4 bits ya que es necesario direccionar 16 bytes en la parte alta de la memoria. Podemos distinguir las siguientes señales de control:

- Señal de carga (**ld_sp**): habilita la actualización del registro.
- Señal de modificación (**inc_sp**): si esta alta indica que el SP debe incrementarse en uno, en otro caso, debe decrementarse en uno. Para realizar ambas operaciones es necesario contar con la señal **ld_sp** en alto, de lo contrario carece de efecto.

3.1.6. Contador de programa (*Program Counter*)

El contador de programa (PC) es un registro de 8 bits, debido a que se utiliza en el direccionamiento de las 256 localidades de ROM. Posee una señal de carga (**ld_pc**) que permite la actualización del registro con un valor presente en el BUS. También dispone de una señal **inc_pc** que permite incrementar en uno el contenido del registro.

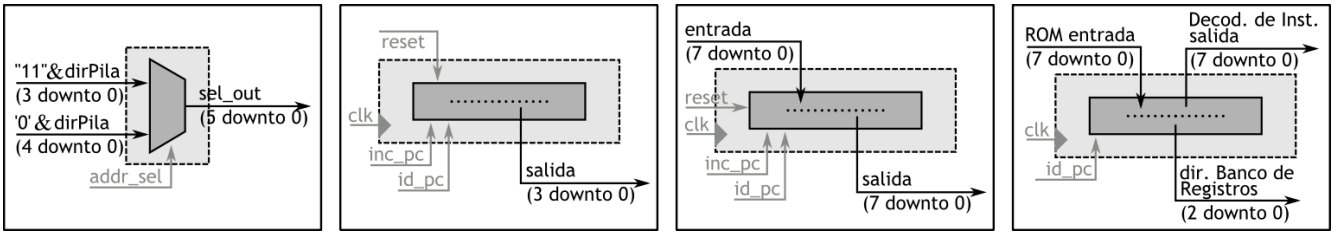

Fig. 4. Izquierda: esquema del Selector de Dirección de Ram y del Puntero de Pila - Derecha: esquema del Contador de Programa y del Registro de Instrucción

3.1.7. Registro de instrucción

El registro de instrucción (IR) es un registro de 8 bits que almacena la instrucción que está siendo ejecutada por el procesador. Posee una señal de carga **ld_ir** que permite la actualización del mismo con un valor proveniente de la salida de la ROM.

3.1.8. Memoria ROM

Consiste en una memoria de 256 bytes donde se encuentra el programa que ejecuta el microprocesador. Se aclara que la ROM debe ser compacta, o sea las direcciones ocupadas con instrucciones deben ser consecutivas e iniciar en la posición cero.

A fin de facilitar la generación del código VHDL de la ROM, el cual va a ejecutar el microprocesador DW-B, se implementó un programa traductor que realiza la codificación a partir de lenguaje ensamblador (ver sección "*Lenguaje de Ensamble*"). Este módulo posee una señal de direccionamiento 8 bits que proviene del contador de programa (PC) y una señal de salida de igual tamaño conectada al BUS y al registro de instrucción (IR).

3.1.9. Unidad aritmético-lógica

Este módulo corresponde a una unidad aritmético-lógica de 8 bits. Posee la capacidad de realizar las siguientes operaciones:

- Or (operador lógico or)
- Xor (operador lógico xor)
- Carga (load)
- Suma (add)
- Resta (sub)
- And (operador lógico and)

El sistema de representación utilizado es *complemento a 2*, permitiendo un rango de representación de -128 a 127. Sus operandos de entrada son el contenido del acumulador (**ACC**), el cual es realimentado a través del BUS, y el registro especificado por la instrucción. La salida producida son las actualizaciones, para el próximo ciclo de reloj, del acumulador y de los distintos flags dependiendo de la operación realizada.

En resumen:
ACC (Acumulador): es un registro de 8 bits.
Flags: implementados con cuatro *flip_flops*, representan lo siguiente: **carry** (c), **sign** (s), **zero**(z) y **overflow** (o).

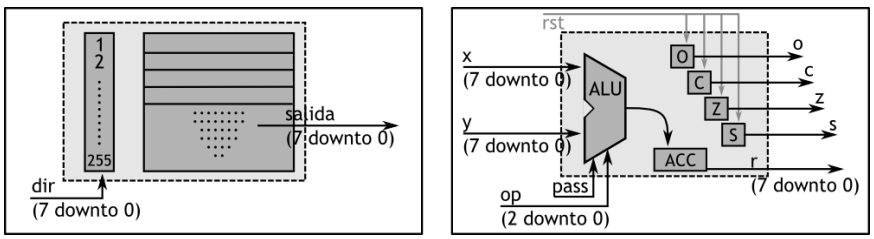

Fig. 5. Izquierda: esquema de Memoria ROM - Derecha: esquema de Unidad Aritmético-Lógica

3.1.10. Bus de datos

El bus de datos es el nexo de comunicación entre los distintos componentes que integran el procesador. Se encarga de coordinar las escrituras en el canal a través de las siguientes señales de control, manteniendo en su salida el dato correspondiente:

- **drv_d_reg**: otorga el permiso de escritura al registro.
- **drv_d_acc**: otorga el permiso de escritura al acumulador.
- **drv_d_port**: otorga el permiso de escritura al puerto de entrada
- **drv_d_rom**: otorga el permiso de escritura a la rom.
- **drv_d_data**: otorga el permiso de escritura a la ram.
- **drv_d_pc**: otorga el permiso de escritura al Program Counter.

3.1.11. Decodificador de instrucción (Unidad de Control)

La unidad de control del DW-B fue basada en la descripción en lenguaje ABEL presente en [1] y está materializada en VDHL como una máquina de estados que posee las siguientes características:

Señales de Entrada Son aquellas señales que sirven a la máquina de estados para determinar su proxima transición, dentro del grafo de transciciones.

- Clock y Reset: **clock**,**reset**.
- Instrucción actual en el Registro de Instrucción (IR).
- Flags de la ALU: **C, Z, O, S**.

Señales de Salida Son aquellas señales que utiliza la unidad de control para controlar todos los componentes del procesador, de manera que puedan ejecutarse las tareas que le son asignadas.

- **drv_d_acc, drv_d_data, drv_d_rom, drv_d_pc, drv_d_port, drv_d_reg**: conectadas al bus de datos controlan la escritura de los diferentes módulos. Sólo una de estas señales puede estar en nivel alto a la vez.
- **inc_pc, ld_pc**: conectadas al contador de programa, permiten su carga e incremento.
- **inc_sp, ld_sp**: conectadas al puntero de pila, permiten su incremento y decremento.
- **ld_ir**: conectada al registro de instrucción, habilita su carga.
- **ld_port**: conectada al puerto de salida, permite su carga.
- **ld_reg, r0**: conectadas al banco de registros, habilitan su escritura y el direccionamiento del registro especial respectivamente.
- **pass, op**: conectadas a la ALU, permiten la carga de un valor en el acumulador sin realizar operación alguna o realizando una operación en particular.
- **w_notr**: conectada a la RAM habilita su escritura.
- **addr_sel**: conectada al selector de ram, indica que parte de la memoria (data o stack) se desea direccionar.

Señales Internas Son aquellas señales que la misma unidad de control genera y utiliza en cada ciclo para determinar las acciones a realizar.

- **curr_st**: indica el estado actual de la máquina.
- **next_st**: indica el próximo estado de la máquina.

Al igual que las unidades de control de los procesadores convencionales, la del DW-B puede ser modelada con un autómata. Particularmente, el esquema de la máquina de estados de este procesador puede esquematizarse de la siguiente manera:

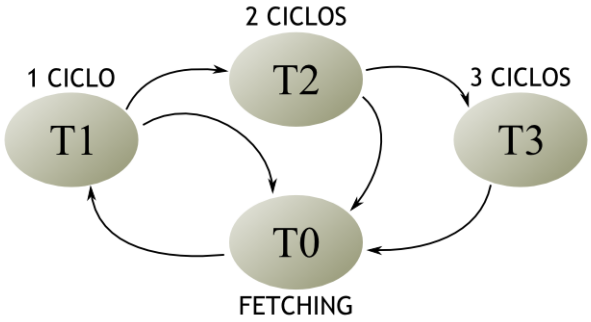

Fig. 6. Esquema de la máquina de estados de la unidad de control

Donde:

- **T0**: estado inicial de cada instrucción (donde se produce el fetching).
  - Próximo Estado: T1
- **T1**: estado donde se comienza a ejecutar la instrucción.
  - Próximo Estado: T0 si la instrucción dura un ciclo (se continua con la siguiente instrucción)
  - Próximo Estado: T1 si la instrucción dura más de un ciclo (se continua con la segunda parte de la ejecución)
- **T2**: estado de ejecución de instrucciones de 2 o màs ciclos.
  - Próximo Estado: T0 si la instrucción dura dos ciclos (se continua con la siguiente instrucción)
  - Próximo Estado: T3 si la instrucción dura 3 ciclos (se continua con la tercera parte de la ejecución)
- **T3**: estado de ejecución de instrucciones 3 ciclos.
  - Próximo Estado: T0 (se continua con la siguiente instrucción)

4. Traductor y programas en ensamblador

En conjunto con la creación de la descripción VHDL del Procesador DW-B se diseñó y desarrolló una herramienta traductora en *C++*. Dicho sistema proporciona una forma fácil y rápida de traducir los programas escritos en lenguaje ensamblador al código máquina correspondiente a cada instrucción.

4.1. Ensamblador para DW-B

Para poder programar el procesador debemos cargar la memoria de programa (ROM) con el. Como se explicitó anteriormente, cada una de ellas tiene un código máquina asociado. Por ejemplo:

Instrucción: “jump“ Código máquina:"11100000"

Por lo tanto, cuando la ROM se encuentra cargada con un programa, ésta posee un ristra de unos y ceros que conforman el volcado a memoria del programa en cuestión. Teniendo el programa escrito de esa forma, si necesitáramos corregir o simplemente entender lo que el programa realiza, nos veríamos en una tarea muy difícil de conseguir. De igual forma, si lo que queremos es programar directamente sobre el código máquina, estaríamos muy propensos a las equivocaciones y detectar un error podría llegar a ser imposible, dependiendo de la complejidad del programa. Es aquí donde surge la necesidad de crear un lenguaje que eleve el nivel de abstracción lo suficiente como para trabajar con este tipo de programas sea mas fácil para el ser humano, pero también, que lo mantenga tan abajo como se necesite para que la traducción al binario sea directa.

La solución consiste en crear un lenguaje especial (ensamblador) que hace ambas cosas. En un lenguaje de este tipo, contamos con una serie de nemónicos, nombres simbólicos de cada instrucción, y sus respectivos parámetros, datos necesarios para su ejecución en caso de que los tengan. El programa traductor lee el archivo escrito en lenguaje ensamblador y sustituye cada uno de los códigos nemotécnicos que aparecen, por su código de operación correspondiente en sistema binario. El DW-B cuenta con un lenguaje ensamblador que hemos definido para facilitar la tarea del programador y con un software que realiza la posterior traducción.

Los nemónicos que se usan para la programación del DW-B son los mismos que se definieron con anterioridad. El formato de instrucción es el siguiente:

<nemónico>[(<parámetro>)]

La aplicación diseñada traduce un programa escrito en lenguaje de ensamble a código máquina detectando y comunicando errores sintácticos. Una vez generado el código, se puede anexar a la descripción de la unidad ROM del procesador, pudiendo este ser sintetizado para permitir la ejecución del programa. Con fines didácticos, se programaron también varios algoritmos en lenguaje ensamblador, para que los alumnos pudieran experimentar con la herramienta y el procesador.

4.2. Ejemplos de ejecución

Una vez finalizado el desarrollo del procesador DW-B se ha procedido a realizar la síntesis del mismo, y utilizando para ello dos programas creados específicamente para esta arquitectura usando el conjunto de instrucciones explicitado en 1. Uno de ellos realiza el calculo el valor de la sucesion de Fiboanacci de un número x, dicho valor se obtiene del puerto de entrada y mediante el calculo recursivo correspondiente se obtiene el valor resultante.

El segundo realiza la multiplicación entre 2 numeros, A y B. Se incluyen a continuacion los codigos correspondientes en lenguaje ensamblador de ambos programas. Cabe destacar además, que ambos ejemplos constituyeron parte del material que la Cátedra adoptó para la enseñanza de diseño digital con VHDL.

4.2.1. Fibonacci

```
00 - load_inm(0)  - Se inicializa R7 con 0    18 - jump(20)
01 - store_reg(7)                             19 - ret          - Retorno al llamador
02 - load_inm(1)  - Se inicializa R6 con 1    20 - load_reg(1)  - Carga de X en acc
03 - store_reg(6)                             21 - sub(6)       - X - 1
04 - load_inm(0)  - Se inicializa R5 con 0    22 - store_reg(1) - Se carga X con X - 1
05 - store_reg(5)                             23 - jsr(10)      - FIBONACCI
06 - in(1)        - Se inicializa R1 con X    24 - load_reg(1)  - Se carga X en acc
07 - jsr(10)                                  25 - sub(6)       - X - 1
08 - load_reg(5)                              26 - store_reg(1)
09 - out                                      27 - jsr(10)      - FIBONACCI
10 - load_reg(1)  - X en acc | FIBONACCI      28 - load_inm(2)
11 - or(1)        - X or X                    29 - add(1)
12 - jz(14)       - Salto por 0               30 - store_reg(1)
13 - jump(15)                                 31 - ret          - Retorno al llamador
14 - ret          - Retorno al llamador       32 - load_reg(6)
15 - load_reg(1)  - Se carga X en acc         33 - add(5)
16 - sub(6)       - X - 1                     34 - store_reg(5)
17 - jz(32)       - Salta si 1                35 - jump(19)     - Salto al llamador
```

Del código anterior es posible abstraer el siguiente pseudocódigo:

1. Inicialización de registros con valores de futura utilidad (R[7] = 0, R[6] = 1)
2. Inicialización del registro para almacenar resultado (R[5] = 0)
3. Carga por puerto de entrada el valor X sobre el que se quiere calcular el valor de la función.
4. Comienzo del algoritmo
   a) Si el valor de X es 0, se retorna al bloque llamador
   b) Si el valor de X es 1, se suma uno al registro solución y se retorna al bloque llamador
   c) En otro caso, se llama a Fibonacci recursivamente (paso 4) para calcular $fibonacci(X-1) + fibonacci(X-2)$

4.2.2. Multiplicación de dos enteros

También se construyo un algoritmo sencillo capaz de multiplicar dos números enteros (A y B). Este ejemplo resulta mucho menos complejo que el anterior, y constituye un buen ejercicio para comenzar a utilizar la herramienta traductora, el procesador y su correspondiente lenguaje ensamblador.

```
00 - load_inm(7)  - Carga de A en acc
01 - store_reg(1) - Carga de acc en R1
02 - load_inm(4)  - Carga de B en acc
03 - store_reg(2) - Carga de acc en R2
04 - load_inm(0)  - Carga de acc con 0
05 - store_reg(3) - Inic. de resultado
06 - load_inm(1)  - Carga de acc con 1
07 - store_reg(6) - Carga de R6 con 1
08 - load_inm(0)  - Carga de acc con 0
09 - store_reg(7) - Carga de R7 con 0
10 - load_reg(2)  - Carga de B en acc
11 - add(2)
12 - jz(20)       - Si B = 0
13 - load_reg(1)  - Carga de A
14 - add(3)       - Suma A + Resultado
15 - store_reg(3) - Carga el resultado parcial
16 - load_reg(2)  - Carga de B
17 - acc sub(6)   - B = B - 1
18 - store_reg(2) - Actualización de B
19 - jump(12)     - fin del bucle
20 - load_reg(3)  - Carga de resultado en acc
```

4.3. Síntesis

En base a los dos ejemplos de la sección precedente, se realizó la síntesis en diferentes arquitecturas de FPGA para medir tiempos y espacio. El objetivo en esta etapa del trabajo no ha sido la optimización del tiempo de ciclo ni de los slices ocupados, sino simplemente mostrar que el diseño realizado es completamente síntetizable. La tabla 2 muestra algunas de la métricas resultantes de este proceso en dos FPGA Xilinx diferentes, de la familia Virtex.

Tabla 2. Métricas de sintesis en FPGA

	Virtex 4 - Speed grade 12		Virtex 5 - Speed grade 3	
	Tiempo ciclo	Slices	Tiempo ciclo	Slices
Multiplicador	5.20	564	4.17	250
Fibonacci	5.25	580	4.19	281

5. Conclusiones

En el presente trabajo se exhibe el desarrollo de una microprocesador didactico, basado en el DWARF, como parte de un proyecto final de cursada. La experiencia ha resultado muy positiva porque, por un lado, ha permitido comprender íntegramente el funcionamiento y la composición de un procesador básico, enfocandose en la funcionalidad de cada componente y en las interacciones que existen entre ellos.

Finalmente, como culminación de este trabajo, la aplicacion y el codigo fueron puestos a dispocicion de los nuevos alumnos de la materia a partir del año 2010 como material didactico. A partir de entonces se comenzo a implementar durante el transcurso de la cursada, el desarrollo de un microprocesador similar en su versión segmentada, con algunas modificaciones. Dicho proyecto es guiado y la evolucion sobre el mismo se produce de una manera incremental. Todo este trabajo se realiza en el marco de las prácticas de laboratorio, especialmente planificadas y conformadas por pequeños grupos de alumnos. Al incorporar como parte de la materia dicho proyecto, el alumno es instruido en el lenguaje VHDL, asi como tambien en el manejo de los diferentes entornos de desarrollo (en este caso, ISE Xilinx) y herramientas que le permitan simular su diseño(ModelSIM).

La experiencia de enseñanza de diseño digital utilizando VHDL ha contribuido fuertemente en el proceso de aprendizaje, generando una mayor comprensión y mejor aprendizaje, ya que junto con los contenidos propios de la arquitectura de computaduras se introducen otros conceptos como programación y simulación VHDL que colaboran en la formación integral de los alumnos.

6. Referencias

[1] *The Practical Xilinx Designer Lab Book*. Dave, Van den Bout (1997). Prentice Hall, New Jersey.

[2] *Computer Organization and Design: The Hardware/Software Interface*. D.A. Patterson y J.L. Hennessy (2009).Morgan Kaufmann.

[3] *A First Course in Digital Design Using VHDL and Programmable Logic*. Shawki Areibi (2001). 31st ASEE/IEEE Frontiers in Education Conference, Reno, NV.

[4] *Teaching Computer Architecture Using an Architecture Description Language*. Sandro Rigo, Marcio Juliato, Rodolfo Azevedo, Guido Araújo, Paulo Centoducatte. Universidad de Campinas. Brasil.

[5] *Standard VHDL Language Reference Manual* http://vhdl.org/vasg/