El uUTN8B16COP-SF

Descripción y simulación en VHDL de un procesador elemental

Ing. Gabriel Edgardo Argañaras, Marco Miretti, Lorenzo Depetris, Facundo Busano, Esteban Ceré, Luciano Brezzo

Departamento de Electrónica
UTN Facultad Regional San Francisco
San Francisco, Córdoba, República Argentina

gabriel_arganaras@arnet.com.ar, marco.miretti@gmail.com, depetrislorenzo1@gmail.com
facundobusano@gmail.com, esteban.cere@gmail.com, lucianobrezzo@outlook.com.ar

Resumen: presentación de los resultados de funcionamiento del procesador elemental uUTN8B16COP-SF descripto en VHDL. El procesador fue diseñado y simulado en el software ISIS PROTEUS 8.1 por alumnos de la asignatura Técnicas Digitales I, en el marco del trabajo final del año lectivo 2015 [1]**. La arquitectura de referencia se sintetizó luego en lenguaje VHDL y se simuló en el entorno de desarrollo ISE 9.2 de XILINX. Este informe detalla el módulo principal VHDL del procesador. Se presentan además los programas de test que se utilizaron para probar los diferentes códigos de operación.**

Palabras claves: evolución de la CPU – lenguaje VHDL – programas de test - simulación

I. INTRODUCCIÓN

El µUTN8B16COP-SF es un procesador didáctico de 8 bits con un repertorio de instrucciones de 16 códigos de operación. Su diseño forma parte de las actividades de la línea de trabajo propuesta por la asignatura Técnicas Digitales I, orientada al estudio de la arquitectura interna de los microcontroladores. Es una evolución del procesador elemental µUTN8B4COP [2] que fue diseñado por los alumnos de la asignatura Técnicas Digitales I del año 2014.

El temario de la asignatura Técnicas Digitales I de la carrera de Ingeniería Electrónica de la Universidad Tecnológica Nacional Facultad Regional San Francisco está en un proceso de actualización. En ese marco, el diseño digital tradicional se reemplazará gradualmente por el diseño utilizando lenguajes descriptores de hardware (HDL's). Esa transición requiere del aprendizaje del lenguaje, herramientas de desarrollo y también de una adaptación al método de diseño. Este desafío que se presenta es visto como una oportunidad de avance del proyecto de desarrollo del procesador: la descripción de su arquitectura interna a través del lenguaje VHDL.

El trabajo de descripción del procesador se dividió en las siguientes partes:

- Diseño de los diferentes módulos VHDL que componen la arquitectura interna básica de referencia.
- Ensamble de los módulos VHDL trabajando en estilo de descripción estructural.
- Diseño de breves rutinas de lenguaje ensamblador propias del procesador, que permitan verificar el funcionamiento de cada una de las instrucciones del set.
- Simulación de funcionamiento usando las rutinas de test. Evaluación de resultados y correcciones necesarias en las descripciones.

II. ARQUITECTURA INTERNA DEL UUTN8B16COP-SF

La arquitectura Harvard de referencia de la CPU, que se entregó a los alumnos para el desarrollo de sus trabajos, sufrió pocas modificaciones con respecto de su antecesor el µUTN8B4COP. En esta ocasión, se rediseñó el camino de datos para adaptarlo al nuevo conjunto de instrucciones, que incorpora dos modos de direccionamiento: implícito y directo. De la misma manera, la Unidad de Control (UC) amplió su bus de control para atender a las modificaciones en el camino de datos. Se agregaron más fuentes de datos a los multiplexores de los registros A y B, así como más canales al demultiplexor del registro de instrucciones (RI).

La máquina de estados interna se simplificó, quedando de cuatro estados internos, uno por cada fase del ciclo de instrucción: búsqueda, decodificación, ejecución y almacenamiento. Las diferentes versiones de su antecesor, el µUTN8B4COP no empleaban la misma cantidad de estados para todas las instrucciones. Este hecho provocaba que el tiempo de ejecución variara de un comando a otro, complicando el cálculo de rutinas de retardo. Por esta razón se decidió modificar radicalmente la máquina de estados reduciéndola a un simple contador en anillo de cuatro etapas. En cada una de ellas se van habilitando etapas del circuito asociadas a cada uno de las fases del ciclo de instrucción.

El procesador conserva el ancho del bus de direcciones que su predecesor, limitando la capacidad de memoria de programa a 256 bytes. Los códigos de instrucción ocupan una posición de memoria y tienen un ancho de palabra de 12 bits, divididos en dos campos. El campo correspondiente al código de operación es de 4 bits, y el del argumento de 8 bits. La Fig.1 muestra el diagrama de bloques de la arquitectura de referencia.

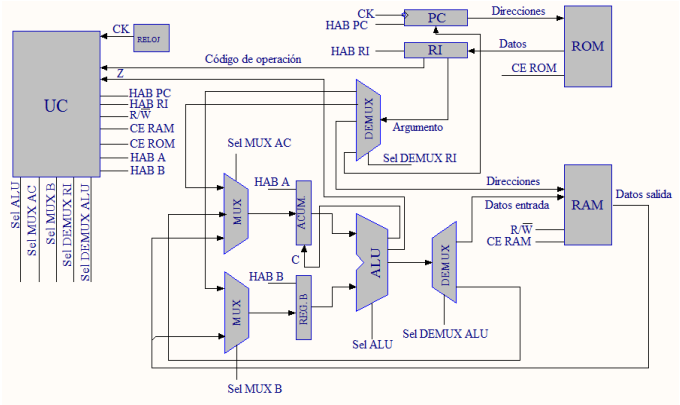

Fig. 1. Procesador µUTN8B16COP-SF, arquitectura de referencia.

La Tabla 1 muestra el set de instrucciones del procesador.

Tabla 1: repertorio de instrucciones del µUTN8B16COP-SF

Instrucciones del µUTN8B16COP			
Nenónico	***Código máquina***	***Sintaxis***	***Acción***
CargaM	00_H	CargaM,@	Carga en el registro acumulador A el contenido de la posición de memoria especificada en el argumento @.
CargaA	01_H	CargaA,#	Carga en el registro A el literal contenido en el argumento #
GuardaM	02_H	GuardaM,@	Guarda el dato actualmente almacenado en el acumulador A en la posición de memoria especificada en el argumento @.
AndA	03_H	AndA,#	Realiza la operación lógica AND entre el contenido del acumulador A y el argumento #.
OrA	04_H	OrA,#	Realiza la operación lógica OR entre el contenido del acumulador A y el argumento #.
NotA	05_H	NotA	Realiza la operación lógica NOT al contenido del acumulador A. Esta operación no tiene argumento.
OrExA	06_H	OrExA,#	Realiza la operación lógica OR EXCLUSIVA entre el contenido del acumulador A y el argumento #.
SumaA	07_H	SumaA,#	Suma aritmética del contenido actual del acumulador A más el argumento # más el bit de acarreo.
RestaA	08_H	RestaA,#	Resta al contenido actual del acumulador A el argumento #.
SumaM	09_H	SumaM,@	Suma aritmética del contenido actual del acumulador A con el contenido de la posición de memoria especificada en el argumento @ más el bit de acarreo.
RestaM	$0A_H$	RestaM,@	Resta al contenido actual del acumulador A el contenido de la posición de memoria especificada en el argumento @.
RotAI	$0B_H$	RotAI	Desplazamiento hacia la izquierda del contenido actual del acumulador A. El bit entrante proviene del bit de acarreo C. El bit saliente se guarda en el bit C. Esta operación no tiene argumento.
RotAD	$0C_H$	RotAD	Desplazamiento hacia la derecha del contenido actual del acumulador A. El bit entrante proviene del bit de acarreo C. El bit saliente se guarda en el bit C. Esta operación no tiene argumento.
BitA0	$0D_H$	BitA0,&	Pone a 0 el bit del actual contenido del registro A especificado en el argumento &. El rango permitido es 0 a 7.
BitA1	$0E_H$	BitA1,&	Pone a 1 el bit del actual contenido del registro A especificado en el argumento &. El rango permitido es 0 a 7.
Salto0	$0F_H$	Salto0,@	Salto condicional a la posición de memoria especificada en el argumento @ si el resultado de la última operación fue cero.

III. Descripción del procesador en lenguaje VHDL

La descripción del procesador se desarrolló en tres niveles jerárquicos de descripción. En el nivel más bajo de descripción se diseñaron los componentes básicos siguiendo un enfoque clásico. Por una cuestión didáctica, no se usaron bloques de librería sino que se crearon componentes a la medida del dispositivo. En este nivel se describieron registros de almacenamiento, multiplexores, demultiplexores, contadores, etc. Los estilos de descripción utilizados fueron el algorítmico y el flujo de datos.

En el nivel intermedio se crearon módulos que incorporaban componentes creados en el nivel más bajo. Se describieron entre otros la ALU, máquina de estados y la unidad de control. Los estilos de descripción utilizados fueron el algorítmico y el flujo de datos.

En el nivel superior de la descripción se interconectaron todos los módulos, de acuerdo con el diagrama de bloques de la arquitectura básica de referencia. Aquí se utilizó el estilo estructural.

En las Fig. 2 y Fig. 3 se muestra el código VHDL de la descripción del módulo principal del procesador.

En esta primera etapa del desarrollo, el procesador es un sistema cerrado sin puertos de salida o terminales de control accesibles al exterior. El bus de salidas *Dato_int* que se declara en cuerpo de la entidad es de lectura. Durante la tarea de depuración de la descripción y verificación de

funcionamiento este bus reflejó el estado de diferentes partes del procesador.

La memoria de programa y la de datos son internas. Los únicos terminales accesibles desde el exterior son las líneas de reset (RS) y el reloj (CK).

Dentro del cuerpo de la arquitectura se declaran todas las señales internas. La señal *bus_control_maquina_estados* corresponde a las funciones de salida de la máquina de estados dentro de la unidad de control, encargadas de dirigir el flujo de datos dentro del camino de datos. Luego de declaran individualmente los bits de control, buses de las memorias ROM y RAM, entradas y salidas de los registros A y B, etc.

La descripción funcional comienza con la asignación a las señales antes declaradas, de los bits correspondientes de las funciones de la máquina de estados. A continuación se comienza con la instanciación de los diferentes componentes siguiendo el diagrama de bloques de la Fig. 1:

- *conta*: contador de programa (PC)
- *reg_RI*: registro de instruicciones (RI)
- *M_ROM*: memoria de programa ROM.
- *M_RAM*: memoria de datos RAM.
- *demux_RI*: demultiplexor que distribuye el campo argumento del código de operación dentro del camino de datos.
- *mux_A*: multiplexor que alimenta al registro acumulador A. En la descripción este registro está ubicado dentro de la ALU.
- *mux_B*: multiplexor que alimenta al registro B. En la descripción está ubicado dentro de la ALU.
- *ALU*: unidad lógica – aritmética.
- *demux_ALU*: demultiplexor que distribuye el resultado de la última función realizada por la ALU a diferentes puntos del camino de datos.
- *maquina_estados*: instanciación de la máquina de estados.

```
---------------------------------------------------------------------------------------
-- UTN Facultad Regional San Francisco Técnicas Digitales I
-- Fecha de inicio:    20:49:44 20/02/2016
-- Nombre del diseño: uUTN8B16COP-SF
-- Nombre del módulo:    UCP_uUTN8B16COP - Behavioral
-- Nombre del proyecto: Microprocesador UTN8B16COP-SF
-- Descripción: CPU del procesador.
-- Revisión: 00
---------------------------------------------------------------------------------------
library IEEE;
library work;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.bloque_Contador_.all;
use work.bloque_Reg_12_bits.all;
use work.bloque_ROM.all;
use work.bloque_RAM.all;
use work.bloque_Demux_4ch_8b.all;
use work.bloque_Mux_4ch_8_bits.all;
use work.bloque_Mux_2_canales_8_bits.all;
use work.bloque_ALU_8_uUTN8B16COP.all;
use work.bloque_Demux_2_canales_8_bits.all;
use work.bloque_ Maquina_UTN8B16COP-SE.all;

entity CPU_uUTN8B16COP-SF is
    Port ( CK : in  STD_LOGIC;
           RS : in  STD_LOGIC;
           Estado : out  STD_LOGIC_VECTOR (1 downto 0);
           Dato_interno : out  STD_LOGIC_VECTOR (7 downto 0);
end CPU_uUTN8B16COP-SF;

architecture Behavioral of CPU_uUTN8B16COP-SF is
signal bus_control_maquina_estados : STD_LOGIC_VECTOR (0 to 22);
signal H_PC : STD_LOGIC;
signal Precarga :STD_LOGIC;
signal H_RI, O_RI :STD_LOGIC;
signal H_ROM :STD_LOGIC;
signal Demux_RI :STD_LOGIC_VECTOR (1 downto 0);
signal Modo_ALU :STD_LOGIC_VECTOR (3 downto 0);
signal H_A, O_A :STD_LOGIC;
signal H_B, O_B :STD_LOGIC;
signal H_F, O_F :STD_LOGIC;
signal H_RAM :STD_LOGIC;
signal RW :STD_LOGIC;
signal Demux_ALU :STD_LOGIC;
signal Mux_A :STD_LOGIC_VECTOR (1 downto 0);
signal Mux_B : STD_LOGIC;
signal Cuenta_PC : STD_LOGIC_VECTOR (7 downto 0);
signal Direccion_salto : STD_LOGIC_VECTOR (7 downto 0);
signal Dato_ROM : STD_LOGIC_VECTOR (11 downto 0);
signal Cod_op : STD_LOGIC_VECTOR (11 downto 0);
signal Arg: STD_LOGIC_VECTOR (7 downto 0);
signal Dato_B, Arg_B: STD_LOGIC_VECTOR (7 downto 0);
signal Dato_A, Arg_A: STD_LOGIC_VECTOR (7 downto 0);
signal Dir_RAM: STD_LOGIC_VECTOR (7 DOWNTO 0);
signal DE_RAM: STD_LOGIC_VECTOR (7 downto 0);
signal DS_RAM: STD_LOGIC_VECTOR (7 downto 0);
signal F_ALU_A: STD_LOGIC_VECTOR (7 downto 0);
signal F_ALU: STD_LOGIC_VECTOR (7 downto 0);
signal Z_ALU: STD_LOGIC;
signal Z_bit_ALU: STD_LOGIC;
signal C_bit: STD_LOGIC;
signal Variable_maquina: STD_LOGIC_VECTOR (4 DOWNTO 0);
signal Estados_de_maquina: STD_LOGIC_VECTOR (1 DOWNTO 0);
begin
 Variable_maquina <= (Bit_Z & Codigo_operacion(11 DOWNTO 8));
 H_PC <= bus_maquina_estados (0);
 Precarga <= bus_maquina_estados (1);
 H_ROM <= bus_maquina_estados (2);
 H_RAM <= bus_maquina_estados (3);
 R_W <= bus_maquina_estados (4);
 H_RI <= bus_maquina_estados (5);
 O_RI <= bus_maquina_estados (6);
 Demux_RI <= bus_maquina_estados (7 to 8);
 Mux_A <= bus_maquina_estados (9 to 10);
 H_A <= bus_maquina_estados (11);
 Mux_B <= bus_maquina_estados (12);
 H_B<= bus_maquina_estados (13);
 Modo_ALU <= bus_maquina_estados (14 to 17);
 Demux_ALU <= bus_maquina_estados (18);
 H_F <= bus_maquina_estados (19);
 O_F <= bus_maquina_estados (20);
 O_A <= bus_maquina_estados (21);
 O_B <= bus_maquina_estados (22);
 Argumento<= Codigo_operacion (7 downto 0);
```

Fig. 2. Descripción del módulo principal.

```
-- Inicia descripción estructural de la CPU del procesador

conta: Cont_PC port map (H_PC, RS, Precarga, CK, Cuenta_PC, Dir_salto);
reg_RI: Reg_12_bits  port map (H_RI, O_RI, CK, RS, Dato_ROM, Cod_op);
M_ROM: ROM  port map (Dato_ROM, Cuenta_PC, H_ROM);
M_RAM: RAM port map (DE_RAM, DS_RAM, Dir_RAM, H_RAM, CK,
                     RW, RS);
demux_RI: Demux_4ch_8b port map (Arg, Arg_B, Arg_A, Dir_RAM,
                                 Dir_salto, Demux_RI);
mux_A: Mux_4ch_8b port map (Arg_A, F_ALU_A, DE_RAM, Arg_A,
                            Dato_A, Mux_A);
mux_B: Mux_2ch_8b port map (Arg_B, DE_RAM, Dato_B, Mux_B);
ALU: ALU_8_uUTN8B16COP port map (Dato_A, Dato_B, F_ALU, C_bit,
                  Z_bit, H_F, O_F, H_A, O_A, H_B, O_B, CK, RS, Modo_ALU);
demux_ALU: Demux_2ch_8b port map (F_ALU, DE_RAM, F_ALU_A,
                                  Demux_ALU);
maquina_estados:  Maquina_UTN8B16COP_SE  port  map(RS,
        Variables_maquina, CK, bus_maquina_estados, Estados_maquina);

    z_bit: process (Reloj)
        begin
          if CK'event and CK='0' then
            Z_bit <= Z_ALU;
          end if;
        end process;
           Estado<=Estados_de_maquina;
           Dato_interno <= F_ALU;
end Behavioral;
```

Fig. 3. Descripción del módulo principal (continuación)

IV. PRUEBA DE FUNCIONAMIENTO DEL PROCESADOR

La descripción del procesador se verificó instrucción por instrucción utilizando programas de prueba diseñados de manera que permitieran corroborar su funcionamiento. Los programas de prueba están embebidos en la descripción del módulo de memoria ROM. Para cada prueba se debió modificar la descripción y recompilar el proyecto. La simulación del procesador se realizó utilizando la herramienta ISE Simulator dentro del entorno de desarrollo ISE de Xilinx. Las condiciones de prueba se modelaron en un archivo fuente tipo *testbench*. Las pruebas realizadas son estáticas: el software de simulación entrega una gráfica temporal donde se muestra la evolución de las señales agregadas, durante todo el tiempo definido para la prueba. Cualquier cambio de las condiciones de prueba o estados de entradas requiere que se vuelva a editar el archivo fuente *testbench*.

A continuación se describirán tres de los programas de test y se presentan los resultados de los mismos. Las escalas de tiempos de las gráficas temporales están en nanosegundos. El ciclo de instrucción ocupa cuatro ciclos de reloj, uno por cada fase. A una frecuencia de 25MHz, el procesador ejecuta una instrucción cada 160ns.

Programa de prueba 1: la Fig. 4 muestra el fragmento de la descripción del módulo de memoria ROM que contiene el programa de test 1. Este programa verifica las instrucciones *CargaA*, *CargaM*, *GuardaM* y *Salto0*. La prueba consiste en cargar un dato en una posición de memoria, recuperarlo y provocar un salto condicional hacia el principio si su contenido es 00_H. Finalmente se carga el acumulador A con 00_H. Esta acción fuerza a la instrucción Salto0 a provocar un salto incondicional similar a un comando GOTO del lenguaje BASIC.

La Fig. 5 muestra los resultados de esta prueba. La marca *B* de esa figura señala el código de operación recuperado en la fase de búsqueda del ciclo de instrucción. La marca *C* señala las direcciones apuntadas por el contador de programa. Se observa claramente que en *A*, el PC pasa de apuntar a la dirección de memoria 07_H a apuntar la dirección 00_H indicando que se produjo el salto incondicional. La marca *D* indica esa instrucción. El programa de test tuvo un tiempo de ejecución de 6,8 μs.

Programa de prueba 8: la Fig. 6 muestra el programa de test 8 embebido en la descripción del módulo de memoria ROM. En esta ocasión se verifica el funcionamiento de la instrucción ***RestaM*.** En el programa se carga el acumulador A con 02_H y realiza dos operaciones de resta del acumulador A menos el contenido de una posición de memoria. La segunda de las operaciones provoca un salto incondicional hacia el inicio del programa.

La Fig. 7 muestra a los resultados de funcionamiento del procesador luego de correr el programa de test 8. La marca *E* muestra los contenidos parciales del acumulador A.

Programa de prueba 12: la Fig. 8 muestra el programa de test 12. Se verifica el funcionamiento de la instrucción ***BitA1*.** El programa borra el acumulador A y luego va poniendo a estado lógico alto cada uno de sus bits. Finalmente entra en un lazo infinito.

La Fig. 9 muestra la parte final de los resultados de funcionamiento del procesador luego de correr el programa de test 12. En la marca *D* se observan los contenidos parciales del acumulador A. En A y E el programa queda en el bucle infinito.

```
-- viene de la descripción de la arquitectura ROM …
   --Programa test  simple guarda valores en memoria y provoca un salto
-- incondicional
with Direccion select            -- Implementación de una ROM
    dato<= X"145" when "00000000",  --00 carga A con 45H
           X"200" when "00000001",  --01  y lo guarda en 00H
           X"000" when "00000010",  --02  lee la posición 00H
           X"F00" when "00000011",  --03  Salto condicional a 00H
           X"100" when "00000100",  --04  carga A con 00H
           X"200" when "00000101",  --05 guarda A en 00H
           X"000" when "00000110",   --06  lee contenido de 00H
           X"F00" when "00000111",   --07 salto condicional a 00H y repite
                                      --  el lazo
           X"000" when others;

-- continúa la descripción …
```

Fig. 4. Módulo ROM, implementación del programa de test 1.

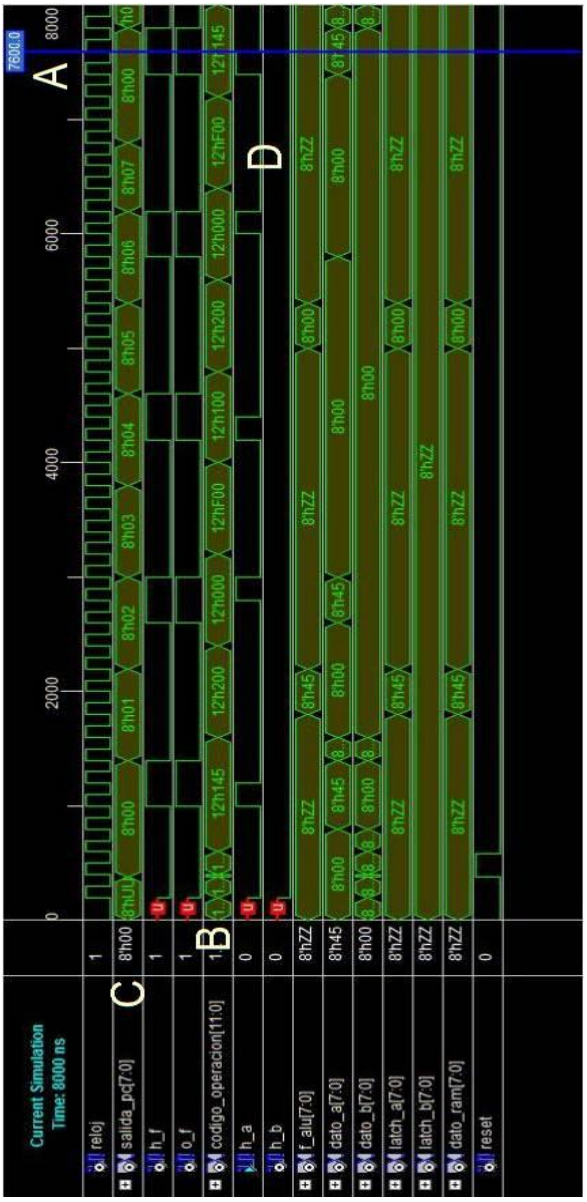

Fig. 5. Resultado del programa de prueba 1: instrucciones CargA, CargaM, GuardaM y Salto0.

```
-- viene de la descripción de la arquitectura ROM …
--octavo programa de test, realiza dos operaciones RESTA con la memoria y
--cuando el resultado es 0 salta al principio
with Direccion select                -- Implementación de una ROM
    dato<= X"101" when "00000000", -- carga A con 02
           X"200" when "00000001",-- y lo guarda en la dirección 00H
           X"102" when "00000010",-- carga A con 01
           X"A00" when "00000011",--Resta de A - contenido de 00H
           X"F00" when "00000100", --  salto condicional a 00H
           X"201" when "00000101", --  guarda el resultado 01H
           X"001" when "00000110",-- carga A con el contenido de 01H
           X"A01" when "00000111",--  Resta de A - contenido de 01H
           X"F00" when "00001000",-- salto condicional 00H
           X"202" when "00001001", --  sino guarda el resultado en 02H

-- continúa la descripción …
```

Fig. 6. Módulo ROM, implementación del programa de test 8.

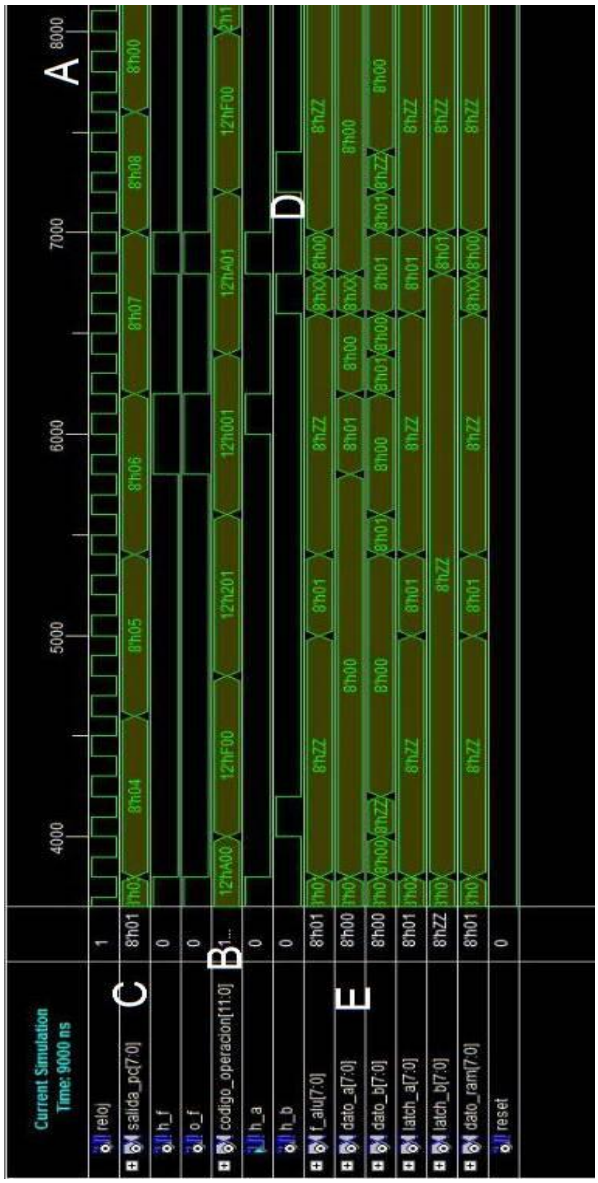

Fig. 7. Resultado del programa de prueba 8: instrucciones RestaM, y Salto0.

```
-- viene de la descripción de la arquitectura ROM …
--duodécimo programa de test, realiza varias operaciones poner a
-- a uno los bits de A y queda en un bucle infinito
with Direccion select                -- Implementación de una ROM
    dato<= X"100" when "00000000",   -- carga A con 00H
           X"E00" when "00000001", -- pone a uno el bit cero de A
           X"E01" when "00000010", -- pone a uno el bit uno de A
           X"E02" when "00000011", -- pone a uno el bit dos de A
           X"E03" when "00000100",-- pone a uno el bit tres de A
           X"E04" when "00000101",-- pone a uno el bit cuatro de A
           X"E05" when "00000110",-- pone a uno el bit cinco de A
           X"E06" when "00000111",-- pone a uno el bit seis de A
           X"E07" when "00001000", -- pone a uno el bit siete de A
           X"100" when "00001010", -- provoca un bucle infinito
           X"F0A" when "00001011", -- lazo infinito
           X"000" when others;
-- continúa la descripción …
```

Fig. 8. Módulo ROM, implementación del programa de test 12.

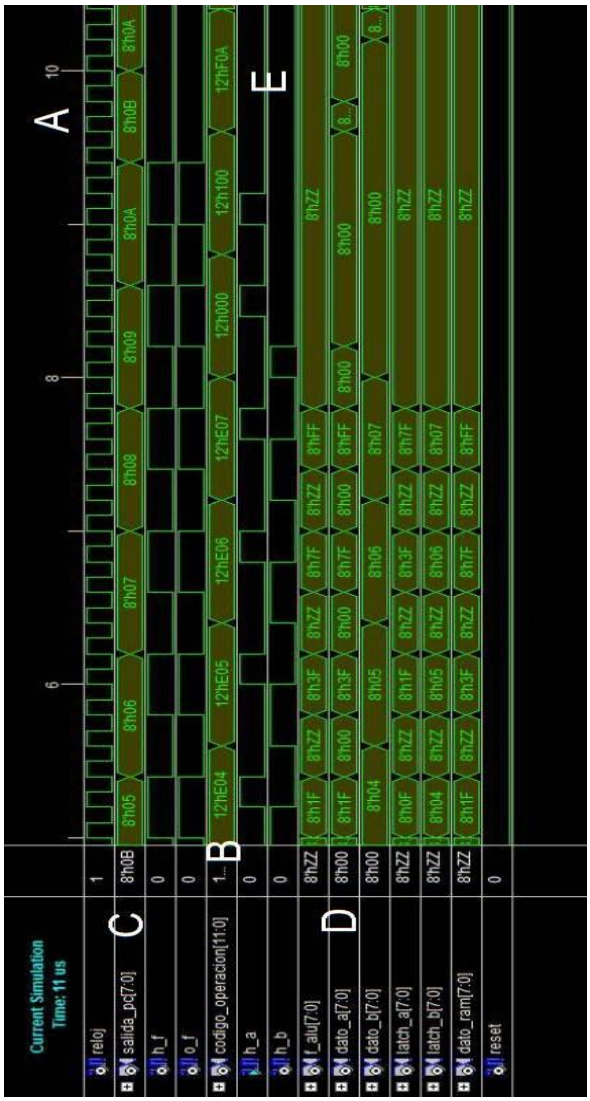

Fig. 9. Resultado del programa deprueba 12: instrucciones BitA1 y Salto0.

V. CONCLUSIONES

Se cumplió el objetivo didáctico propuesto: describir utilizando VHDL la arquitectura del UUTN8B4COP y verificar su funcionamiento por simulación. La concreción del trabajo incrementó la base de conocimientos para la actualización de los temas de la asignatura Técnicas Digitales I.

En lo que respecta a la descripción del procesador se observa la necesidad de modificar el código para las memorias ROM y RAM. El uso de comandos WITH – SELECT – WHEN para la descripción de las memorias no presentó mayores complicaciones. Se observa sin embargo que la utilización de estos mismos comandos en programas de aplicación largos se torna inviable por la longitud de la descripción. Se hace necesario profundizar el aprendizaje del lenguaje VHDL y sus recursos a fin de lograr una descripción más compacta de estos módulos.

Otro aspecto a mejorar es la salida al exterior del procesador. El diseño actual es un sistema cerrado, solo es posible observar los estados de señales internas a través del simulador. El paso siguiente en el proyecto es implementar la descripción a la placa de desarrollo Spartan 3E Starter Kit de la fábrica Xilinx. Es necesario dotar al procesador de puertos de entrada/salida que permitan ingresar o sacar datos de él.

VI. REFERENCIAS

[1] Trabajo Final Integrador, asignatura Técnicas Digitales I, carrera de Ingeniería Electrónica, Universidad Tecnológica Nacional, Facultad Regional San Francisco, San Francisco, Córdoba, Argentina – noviembre 2015.

[2] G. Argañaras, G. Cervetti, J. Ramírez, A. Yoaquino, V. Boasso, "Procesador uUTN8B4COP Diseño y simulación de un procesador didáctico" , artículo 4 FPGA y lógica programable, libro del VI Congreso de Microelectrónica Aplicada, Universidad de La Matanza, San Justo, Buenos Aires, Argentina - mayo 2015.