Laboratorio 2: Introducción a los lenguajes de descripción de hardware

David Cordero, Jonathan Guzmán, Darío Rodríguez
dcorderoch@ieee.org jonathana1196@gmail.com darior1227@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—En este laboratorio se investigó sobre tecnologías de diseño digital, tanto en hardware, como lo son las FPGA, como en software, como los lenguajes de descripción de hardware SystemVerilog y VHDL, y los modelos de diseño de comportamiento y de estructura. Se diseñó en SystemVerilog en modelo de comportamiento un decodificador para display de 7 segmentos, en VHDL un Sumador completo de 4 bits en modelo de estructura, y en Ω (camiar) un contador parametrizable de N bits con reset asincrónico, que además fueron programados y probados en la tarjeta de desarrollo Altera Cyclone V DE1-SoC.**

***Palabras clave*—Contador, decodificador, FPGA, hardware, netlist, software, sumador, SystemVerilog, VHDL.**

I. Introducción

El modelado de comportamiento en los lenguajes de descripción de hardware describe lo que debe hacer un módulo de hardware en términos lógicos de sus entradas y salidas, sin entrar en detalles de los componentes que lo formen, a diferencia del modelo estructural, que describe los componentes que forman el módulo. Para diseños complejos, es posible que el modelo estructural tenga mejores resultados que el de comportamiento, pues los dispositivos de lógica programable como las FPGA tienen una cantidad de compuertas limitada, además de que existen *netlists* (descripciones de hardware teóricas) que no pueden ser sintentizadas, por lo que es posible que el software que se usa para la síntesis lógica no pueda calcular e implementarel hardware descrito en el dispositivo. Ejemplo de código en SystemVerilog:

```
// fulladder

module fulladder(input a, input b, input cin,
                 output s, output cout);

logic p;
logic g;
assign p = a ^ b;
assign g = a & b;

assign s = p ^ cin;
assign cout = g | (p & cin);

endmodule
```

como se puede observar en el código *fulladder*, se describe el resultado lógico deseado, y se delega la responabilidad de definir el cómo se combinarán las señales al software de síntesis.

```
// and_

module and_(input a, input b,
            output c);

assign c = a & b;

endmodule

// not_
module and_(input a,
            output c);

assign c = ~a;

endmodule

// or_
module or_(input a, input b,
           output c);

assign c = a | b;

// xor_
module xor_(input a, input b,
            output c);

logic w1, w2, w3, w4;

not_ N1(a, w1);
not_ N2(b, w2);

and_ A1(b, w1, w3);
and_ A2(a, w2, w4);

or_ O1 (w3, w4, c);

endmodule
```

de la misma manera se puede observar en el código *xor_* que se describe los componentes a usar para dar el resultado lógico deseado del módulo

La síntesis lógica es un proceso que transforma un lenguaje de descripción de hardware (HDL por sus siglas en inglés) en una representación diferente del hardware descrito, esta puede ser texto, o diagramas de circuitos. Este proceso es típicamente realizado por software, llamado sintetizador, que puede realizar optimizaciones para reducir la cantidad de componentes requeridos para cumplir el comportamiento descrito en el HDL.

La lógica programable es generalmente realizada en arreglos

genéricos de circuitería que pueden ser programados para realizar funciones lógicas específicas, esta generalmente se usa para prototipado de módulos de hardware pequeños, aunque dada su flexibilidad y las mejoras en manufactura, recientemente se usan para crear circuitos que realizan de manera muy eficiente tipos específicos de calculos, por lo que a veces se les clasifica como aceleradores. Las FPGA *(Field Programmable Gate Array)* son una versión moderna de dispositivos que permiten hacer lógica programable, estas usan bloques de lógica configurables para implementar lógica combinacional, y además pueden implementar lógica secuencial.

II. Sistema Desarrollado

II-A. Decodificador a Display de 7 segmentos

El Manual de usuario de la DE1-SoC especifica las conexiones a los display de 7 segmentos como descrito en la figura II-A [**?**]

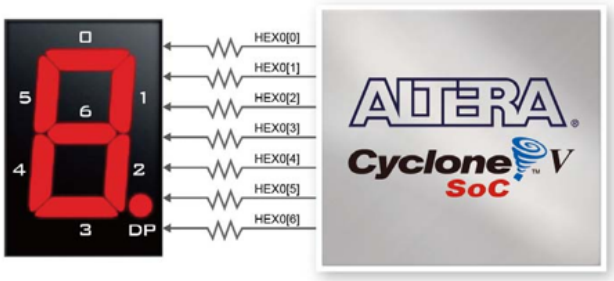

Figura 1. Conexiones entre el display de 7 segmentos HEX0 y la FPGA Cyclone V SoC

tabla de verdad para los números en hexadecimal

mint	a	b	c	d	0	1	2	3	4	5	6
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	1	1	1	0	1	1	1
11	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	0	1	0	0	1	1	1	0
13	1	1	0	1	0	1	1	1	1	0	1
14	1	1	1	0	1	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	1	1	1

Cuadro I
TABLA DE VERDAD PARA MOSTRAR NÚMEROS EN HEXADECIMAL EN EL DISPLAY DE 7 SEGMENTOS

de esto se puede implementar el decodificador en System-Verilog de la siguiente manera:

```
// deco

// asumiendo OUT[6] como HEX00
// y OUT[0] como HEX06

module deco
    (input logic [3:0] IN,
     output logic [6:0] OUT);

always begin
    case(IN)
        4'b0000: OUT = 7'b1111110;
        4'b0001: OUT = 7'b0110000;
        4'b0010: OUT = 7'b1101101;
        4'b0011: OUT = 7'b1111001;
        4'b0100: OUT = 7'b0110011;
        4'b0101: OUT = 7'b1011011;
        4'b0110: OUT = 7'b1011111;
        4'b0111: OUT = 7'b1110000;
        4'b1000: OUT = 7'b1111111;
        4'b1001: OUT = 7'b1110011;
        4'b1010: OUT = 7'b1110111;
        4'b1011: OUT = 7'b0011111;
        4'b1100: OUT = 7'b1001110;
        4'b1101: OUT = 7'b0111101;
        4'b1110: OUT = 7'b1001111;
        4'b1111: OUT = 7'b1000111;
    endcase
end

endmodule
```

II-B. Sumador de 4 bits

Para desarrolla el sumador de 4 bits completo se debía partir de un sumador de un bit, este debía desarrollarse en VHDL, el código del mismo es:

```
// fulladder

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity fulladder is
    port(a, b, c_in: in STD_LOGIC;
         sum, c_out: out STD_LOGIC);
end entity fulladder;

architecture synth of fulladder is

begin

    sum <= a XOR b XOR c_in;

    c_out <= (a AND b) OR (c_in and (a OR b));

end architecture synth;
```

Partiento de este código se puede desarrollar uno de 4 bits, cuyo código es:

```
// fulladder 4 bits

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity fullAdder4Bits is

port (A: in STD_LOGIC_VECTOR (3 downto 0);
      B: in STD_LOGIC_VECTOR (3 downto 0);
      Y: out STD_LOGIC_VECTOR (3 downto 0);
      c_out: out STD_LOGIC);
```

```
end entity fullAdder4Bits;

architecture synth of fullAdder4Bits is

signal carry: STD_LOGIC_VECTOR(3 downto 0);

begin
carry(0) <= '0';
    bit0: entity WORK.fulladder port map
    (a => A(0), b => B(0), sum => Y(0),
    c_in => carry(0), c_out => carry(1));

    bit1: entity WORK.fulladder port map
    (a => A(1), b => B(1), sum => Y(1),
    c_in => carry(1), c_out => carry(2));

    bit2: entity WORK.fulladder port map
    (a => A(2), b => B(2), sum => Y(2),
    c_in => carry(2), c_out => carry(3));

    bit3: entity WORK.fulladder port map
    (a => A(3), b => B(3), sum => Y(3),
    c_in => carry(3), c_out => c_out);

end architecture synth;
```

Al compilar dicho código podemos verificar que lo que interpreta es lo esperado

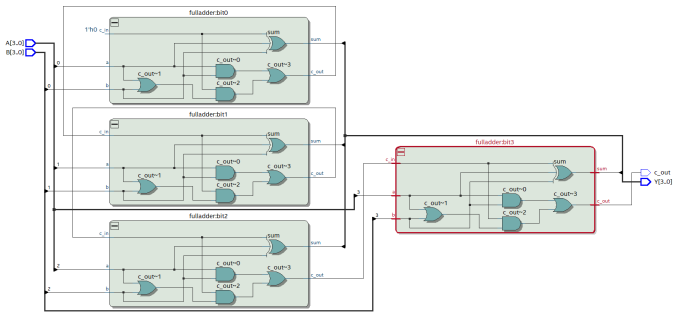

Figura 2. Sumador de 4 bits completo

Una vez que hemos verificado que la descripción de hardware del sumador es la correcta podemos conectar este con el decodificador para poder mostrar los resultados en el display, este código es el siguiente:

```
// fulladder 4 bits con display

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity fullAdder4Bits_w_Display is

port
(A, B : in  STD_LOGIC_VECTOR(3 downto 0);
Y1, Y2 : out STD_LOGIC_VECTOR(6 downto 0));

end entity fullAdder4Bits_w_Display;

architecture synth of
    fullAdder4Bits_w_Display is

component disp_deco
port(x, y, z, w : in  STD_LOGIC;
     a, b, c, d, e, f, g : out STD_LOGIC);
end component;

signal c_out : STD_LOGIC;
signal sum : STD_LOGIC_VECTOR(3 downto 0);

begin

fullAdder4Bit : entity work.fullAdder4Bits port map
(a => A, b => B, Y => sum, c_out => c_out);

display1 : entity work.decoder port map(x => sum(3),
y => sum(2), z => sum(1), w => sum(0), a => Y1(6),
b => Y1(5), c => Y1(4), d => Y1(3), e => Y1(2),
f => Y1(1), g => Y1(0));

display2 : entity work.decoder port map
(x => '0', y => '0', z => '0', w => c_out, a => Y2(6),
b => Y2(5), c => Y2(4), d => Y2(3), e => Y2(2),
f => Y2(1), g => Y2(0));

end architecture synth;
```

Al compilar dicho código podemos verificar que lo que interpreta es lo esperado

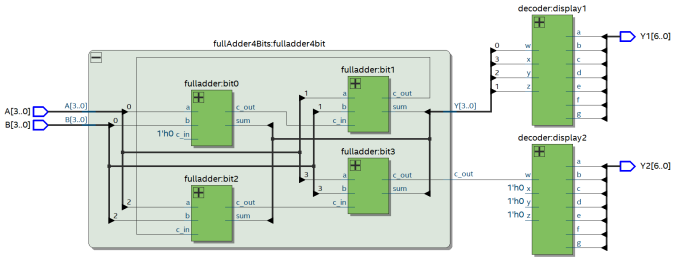

Figura 3. Sumador de 4 bits completo conectado al display

II-C. Counter

Para el contador, se necesitaba plantearlo de manera que sea asíncrono y que reciba como parámetro el número de bits, de esta manera tiene N bits, de manera que se utilizó el siguiente código:

```
module counter \#(parameter N = 8)
(input logic clk,
input logic reset,
output logic [N-1:0] q);

always_ff @(posedge clk, posedge reset)
if (reset) q <= 0;
else q <= q + 1;

endmodule
```

De esta manera, se puede expresar una tabla de verdad para ejemplificar la salida del contador si se supone un N de 3 bits

mint	a	b	c	OUTPUT
0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7

Cuadro II
TABLA DE VERDAD PARA MOSTRAR NÚMEROS DEL CONTADOR

III. RESULTADOS

III-A. Decodificador a Display de 7 Segmentos

Al simular el módulo de decodificador implementado en SystemVerilog se obtiene el resultado que se puede ver en la figura III-A

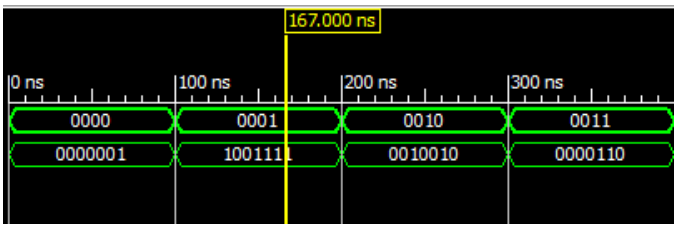

Figura 4. Waveform de simulación de decodificador (lógica negativa)

por razones de tiempo, no fue posible implementar el decodificador en la FPGA a tiempo para este informe.

III-B. Sumador completo de 4 bits

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;


entity fullAdder4Bits_tb is
end fullAdder4Bits_tb;

architecture testbench of fullAdder4Bits_tb is
signal a, b, y : STD_LOGIC_VECTOR(3 downto 0);
signal c_out   : STD_LOGIC;

begin
    UUT : entity work.fullAdder4Bits port map
    (a => a, b => b, y => y, c_out => c_out);

    a <=  "0000" after 0.05 ns,
          "0000" after 0.15 ns,
          "1111" after 0.25 ns,
          "1010" after 0.35 ns,
          "1111" after 0.45 ns;
    b <=  "0000" after 0.05 ns,
          "1111" after 0.15 ns,
          "0000" after 0.25 ns,
          "0101" after 0.35 ns,
          "1111" after 0.45 ns;

    assert a = "0000" and b = "0000" and
    y = "0000" and c_out = '0'
    report 'There is an incorrect value on
    the output led' severity error;

    assert a = "1111" and b = "1111" and
    y = "1111" and c_out = '0'
    report 'There is an incorrect value on
    the output led' severity error;

    assert a = "0000" and b = "1111" and
    y = "1111" and c_out = '0'
    report 'There is an incorrect value on
    the output led' severity error;

    assert a = "1111" and b = "0000" and
    y = "1111" and c_out = '0'
    report 'There is an incorrect value on
    the output led' severity error;

    assert a = "1010" and b = "0101" and
    y = "0100" and c_out = '1'
    report 'There is an incorrect value on
    the output led' severity error;

    assert a = "1111" and b = "1111" and
    y = "1110" and c_out = '1'
    report 'There is an incorrect value on
    the output led' severity error;

end testbench;
```

Se obtuvo el siguiente resultado, el cual no era el esperado como se puede observar a continuación, esta situación se da por no realizar un correcta interfaz entre los distintos componentes del sumador y no se logró encontrar el error a tiempo.

Figura 5. Waveform de simulación de sumador completo de 4 bits

III-C. Contador

Al simular el contador, tuvimos problemas debido a que la salida no se veía como en teoría debería de estar, debido a que el contador tiene una salida del número en binario, por lo que no se pudo obtener el wave correcto. Sin embargo, a la hora de implementarlo en la FPGA todo funcionó a la perfección.

IV. ANÁLISIS DE RESULTADOS

IV-A. Decodificador a Display de 7 Segmentos

Fue posible implementar el decodificador de números binarios de 4 bits a 7 bits para controlar el display de 7 segmentos en SystemVerilog con modelo de comportamiento, y probarlo en el simulador, la poca agilidad con la herramienta Quartus, causó considerable atraso, por lo que debemos dedicar tiempo a aprender el proceso de diseño a implementación en esta para evitar estos retrasos.

IV-B. Sumador completo de 4 bits

Fue posible implementar la descripción del hardware necesario para desarrollar un sumador completo de 4 bits, más no fue posible obtener los resultados esperados al momento de realizar el test bench del mismo, por lo que se debe aprender a realizar interfaces de componentes para futuros proyectos.

IV-C. Contador

El contador, dependiendo de los bits que lo conformen, puede dar la salida de varios numeros en binario, por lo que se utilizó los leds de la FPGA como la salida de cada uno de estos,el número de bits refleja la cantidad de números posibles, por lo que si tiene 3 bits, son 8 posiciones, 6 bits son 64 y así sucesivamente.

V. CONCLUSIONES

La FPGA es una muy útil herramienta de diseño de módulos de hardware para circuitos simples, y para poder probar su viabilidad, y correcto funcionamiento, cabe destacar que el grupo de trabajo debe aprender el proceso desde diseñar en Quartus con SystemVerilog y/o VHDL hasta implementarlo en la FPGA en el menor tiempo posible (idealmente minutos para diseños simples), para poder concentrarse en el diseño y no perder tiempo en el uso de las herramientas de desarrollo. En ste caso el libro ha sido subutilizado, por lo que se debe dedicar tiempo a leerlo más detalladamente, en el caso de los *testbench* de auto-chequeo, el sólamente hay un ejemplo, y este no explica cómo generar otros, por lo que se debe hacer más experimentación con estos. Otro de los puntos a destacar es que se puede utilizar las iteraciones en la descripción de hardware para el desarrollo de algunos sistemas electricos, como el contador,que se describio por medio de un parámetros de bits.

REFERENCIAS

[1] Harris, S. Harris, D. *Digital Design and Computer Architecture: ARM Edition*. Morgan Kaufmann, 2015.