Tarea 2: Descripción conductual en Verilog de un circuito digital basado en Muxes

Roberto Sánchez Cárdenas - B77059

San José, 27 de agosto de 2020
Circuitos Digitales II

Índice

1. Introducción

Este documento tiene como fin presentar la descripción conductual de un circuito digital basado en Muxes en el lenguaje de descripción de hardware iVerilog. En circuito digital fue provisto por el profesor, por lo que la tarea tenía como único fin implementar la lógica del mismo en Verilog por medio de comandos como always, if, else, etc.

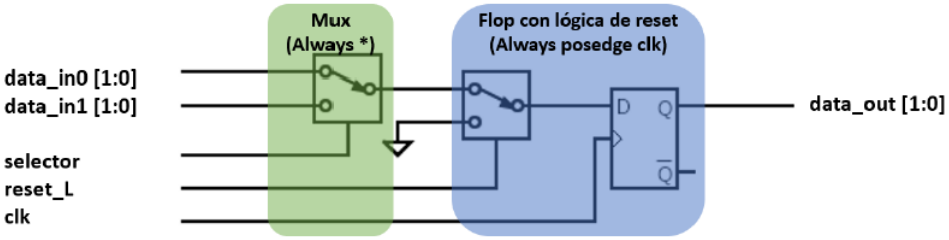

Figura 1: Circuito digital a implementar

Para compilar el archivo de Verilog y mostrar las señales generadas junto con sus resultados, basta con correr el comando **$make** en la dirección de memoria en la que se tienen los documentos de la tarea.

2. Descripción conductual

Como se mencionó previamente, una descripción conductual es describir el comportamiento del circuito que se desea implementar por medio de comandos básicos de un lenguaje de programación. La lógica utilizada para la descripción de este circuito se dividió en dos, una para el mux selector y otra para el elemento de reset y almacenamiento, esto debido a que se desea que el primer mux esté leyendo todo el tiempo y el segundo únicamente cuando llega un flanco positivo de reloj.

El primer mux toma los datos de la data_in0 o data_in1 dependiendo del valor del selector, estos datos se pasaron a una conexión reg, puesto que el dato se debe mantener mientras llega el flanco de reloj.

La segunda parte debe mantener el dato almacenado y solo se realiza un cambio al llegar un flanco de reloj, para ello se usa lógica bloqueante. Si la entrada reset_L está en 0, la salida data_out se hace 0. En el siguiente cuadro se presenta la descripción conductual del circuito mostrado en la Figura 1.

```
1  module mux(input clk,
2             input             reset_L,
3             input             selector,
4             input [1:0]       data_in0,
5             input [1:0]       data_in1,
6             output reg [1:0] data_out);
7
8     reg [1:0] salMux1; //se genera un elemento para conexi n capaz de
          almacenar datos
9
10
11    always @(*) begin
12       salMux1 = 1'b0; //se inicializa el valor de conexi n a 0
13       if (selector == 0)
14         salMux1 = data_in0; //Dependiendo del valor del selector, se
              pasa a la salida del mux
15       else
16         salMux1 = data_in1;
17    end
18
19    always @(posedge clk) begin //Se hace otro elemento que lee
          nicamente   en el flanco positivo de reloj, este elemento
         resetea o deja pasar el dato y lo guarda
20
```

```
21        if (reset_L == 0)
22          data_out <= 0; //Si reset_L esta en 0 se resetea, caso
              contrario se pasa el dato del primer mux
23        else
24          data_out <= salMux1;
25     end
26  endmodule
```

Para probar el correcto funcionamiento de la descripción realizada se utilizan las señales brindadas en el enunciado de la tarea. Para generar estas señales se diseña un probador, donde se pone el estado al que cambia cada señal en cada flanco. Se hizo de esta manera puesto que no se logró observar un patrón de comportamiento. Es códido de esta sección se muestra en el siguiente cuadro.

```
1   module probador_mux(input [0:1] data_out,
2                       output reg reset_L,
3                       output reg [0:1] data_in0,
4                       output reg [0:1] data_in1,
5                       output reg selector,
6                       output reg clk);
7     initial begin
8        $dumpfile("mux.vcd"); //Se depositan los datos en este archivo
           para leerlo en gtkwave
9        $dumpvars;
10       $display ("\t\t\tclk,\tdata_in0,\tdata_in1,\tdata_out,\treset_L
           ␣,\tselector");
11
12       $monitor($time, "\t%b\t%b\t\t%b\t\t%b\t\t%b␣\t\t%b", clk,
           data_in0, data_in1, data_out, reset_L, selector); //Datos a
           mostar
13
14       {selector, reset_L} <= 'b0; //Se inicializan estas entradas a 0
15       {data_in0, data_in1} <= 2'b00; //Se inicializan entradas en 00
16
17
18
19       @(posedge clk); //Como los pulsos no siguen una secuencia, se
           deben generar uno por uno en cada flanco
20       reset_L <= 1;
21
22       @(posedge clk);
```

```
23        data_in0 <= 2'b11;
24        data_in1 <= 2'b10;
25
26        @(posedge clk);
27        selector <= 1;
28        data_in0 <= 2'b01;
29        data_in1 <= 2'b00;
30
31        @(posedge clk);
32        data_in0 <= 2'b00;
33        data_in1 <= 2'b10;
34
35        @(posedge clk);
36        selector <= 0;
37        data_in0 <= 2'b11;
38        data_in1 <= 2'b11;
39
40        @(posedge clk);
41        selector <= 1;
42        data_in0 <= 2'b00;
43        data_in1 <= 2'b01;
44
45        @(posedge clk);
46        selector <= 0;
47        data_in0 <= 2'b10;
48        data_in1 <= 2'b00;
49
50        @(posedge clk);
51
52        @(posedge clk); //Para mostrar dos ciclo m s
53
54
55  $finish;
56
57     end // initial begin
58
59     initial clk <= 0;
60     always #2 clk <= ~clk;
61
62  endmodule // probador_mux
```

Por último, se diseña lo que se llama un banco de pruebas. Esta sección es muy sencilla puesto que el comando AUTOINST hace todo el trabajo si el modulo descriptivo y de probador están bien. En este banco se conecta la descripción con el probador. El código se presenta en el siguiente cuadro.

```
1  `timescale 1s / 1ms
2
3  `include "mux.v"
4  `include "probador_mux.v"
5
6  module BancoPruebas;
7     wire [1:0] data_in0, data_in1, data_out; //Se generan los cables
          de entrada (buses)
8     wire       clk, salMux1, reset_L, selector;
9
10
11
12    mux p_cond(/*AUTOINST*/
13               // Outputs
14               .data_out                (data_out[1:0]),
15               // Inputs
16               .clk                     (clk),
17               .reset_L                 (reset_L),
18               .selector                (selector),
19               .data_in0                (data_in0[1:0]),
20               .data_in1                (data_in1[1:0]));
21
22
23    probador_mux probador(/*AUTOINST*/
24                          // Outputs
25                          .reset_L                 (reset_L),
26                          .data_in0                (data_in0[1:0]),
27                          .data_in1                (data_in1[1:0]),
28                          .selector                (selector),
29                          .clk                     (clk),
30                          // Inputs
31                          .data_out                (data_out[1:0]));
32
33
```

```
34 endmodule
```

Para compilar todo el sistema completo basta con compilar el banco de pruebas, puesto que esta es la unión de ambos códigos. Para simplificar este proceso se diseño un makefile, al estar en la dirección del makefile solo se corre el comando **make**. El código del makefile se adjunta en el siguiente cuadro.

```
1 all: iverilog gtkwave
2
3 iverilog:
4         iverilog BancoPruebaConductual_mux.v -o prueba
5         vvp prueba
6 gtkwave:
7         gtkwave mux.vcd
```

3. Resultados

Para comprobar los resultados en el enunciado de la tarea de proporcionaron una serie de señales y lo que se esperaba obtener. En la siguiente imagen se muestra lo que se esperaba de la descripción.

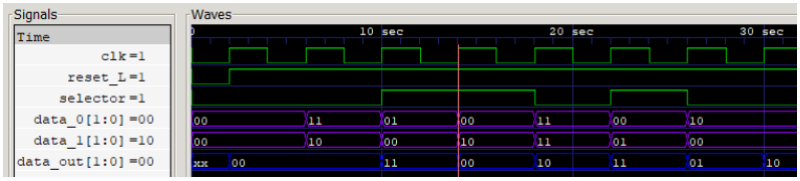

Figura 2: Respuesta esperada

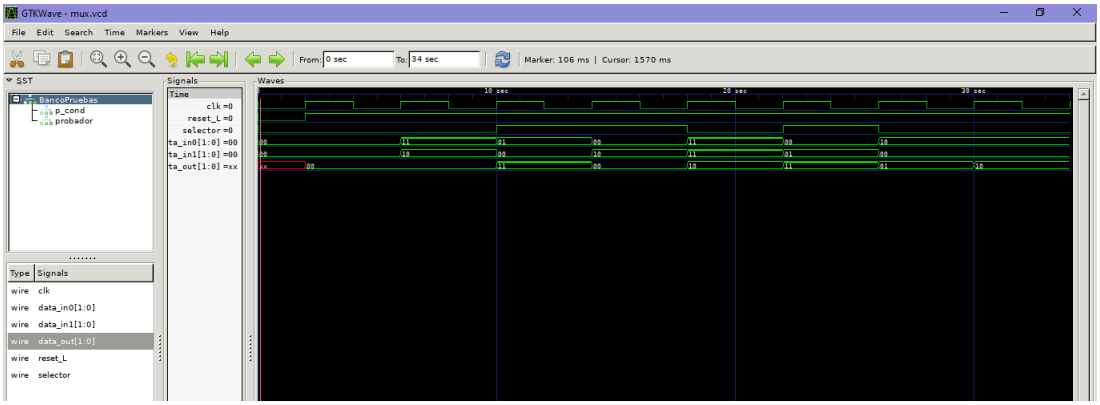

Figura 3: Respuesta obtenida al correr el makefile

Como se puede observar, la descripción obtenida concuerda con las respuestas esperadas, por lo que la descripción propuesta es correcta.

Se realizó una segunda prueba en la que el comando reset_L pasa a 0 en el 4to flanco de reloj.

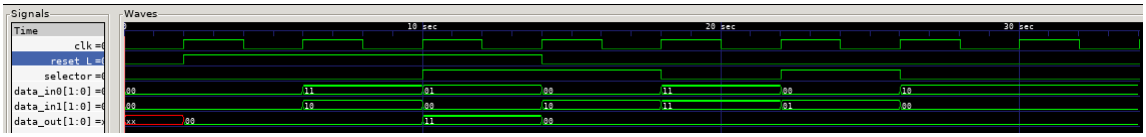

Figura 4: Cambio de señal de reset_L

4. Diagrama obtenido

A partir de la descripción realizada en el archivo mux, se obtuvo un diagrama generado por yosys, el cual se presenta en la siguiente figura.

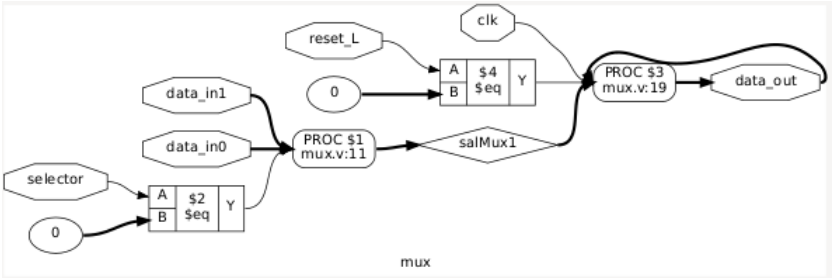

Figura 5: Diagrama del circuito obtenido con yosys

5. Resultados

Se puede observar el correcto funcionamiento de circuito puesto que, siempre y cuando el reset esté en 1, cuando el selector es leído en 1 en un flanco, cuando llega el siguiente flanco de reloj se pasa el valor de la señal correspondiente. Por ejemplo, cuando el selector está en 0 en el primer flanco, no ocurre un cambio. Cuando llega el segundo flanco positivo, la señal de data_in0, pasa a data_out.

Se observa que el comando de reset_L funciona de manera correcta puesto que al pasar a 0 en el cuarto flanco, la salida se resetea.

6. Tiempo empleado

Descripción conductual: 1 hora y 20 minutos
Probador: 50 minutos
Banco de pruebas: 8 minutos
Makefile: 4 minutos
Pruebas: 10 minutos
Redacción de documento escrito: 1 hora