Lab 03
Arquitectura de Computadores
Sección 2

Joaquín Ramírez

Mayo 18, 2020

Nota: Todos los *time scales* de los *test benches* son $\frac{1ns}{1ns}$. Para ejecutar los archivos desde terminal, ingrese *make*, con lo cual se generarán los archivos correspondientes a cada ejercicio: *ej11a, ej11b, ej12a, ej12b, ej2 y ej3.* Además, en cada *test bench* se genera un archivo *.vcd* (después de ingresar *vvp "filename"* en la terminal), el cual será ejecutado en *GTKWave.*

1.
   - TFF

     El toggle Flip Flop sirve para hacer un *toggle* al output. Esto significa, hacer un *switch*, cuando T esté activado. Como este flip flop tiene un CLK, entonces el cambio de data en q solo se puede dar en el *posedge* del clock. Se analizó el circuito, y se observa que cuando el *clear* y el *preset* son igual a 0, los valores de q y qn entran a un estado de metaestabilidad. Asimismo, cuando el $prn = 1$ y $clrn = 0$, entonces $q = 1$. Por otra parte, cuando $clrn = 1$ y $prn = 0$, q toma el valor de 0. Si ambos son igual a 1, entonces $q = \ q$, si y solo si $t = 1$. En caso contrario, no hay cambio en q. En el caso behavioral se implementó esta lógica dentro de un always que obedece al cambio de edge del clock, verificando con if-else if - else el estado de los inputs. Todo se hizo en un mismo módulo.

```
module btff(prn, t, clk, clrn, q, qn);
      input prn, t, clk, clrn;
      output reg q, qn;

always @(posedge clk)
begin
      if(clrn == 1 & prn == 0)
      begin
            q <= 1;
            qn <= 0;
      end
      else if(clrn == 0 & prn == 1)
      begin
```

```
14              q <= 0;
15              qn <= 1;
16        end
17        else if (clrn == 0 & prn == 0)
18        begin
19              q <= 1'bX;
20              qn <= 1'bX;
21        end
22        else
23        begin
24              if(t)
25              begin
26                    q <= ~q;
27                    qn <= ~qn;
28              end
29        end
30  end
31
32
33  endmodule
```

Listing 1: Behavioral TFF

El test bench genera las combinaciones distintas del clear, clock y t (prn se mantiene constante en 1) para que se pueda observar el cambio en q deseado por los *waves*.

```
1   `timescale 1ns/1ns
2   module btff_tb;
3         reg prn, t, clk, clrn;
4         inout q, qn;
5         btff g(prn, t, clk, clrn, q, qn);
6         initial begin
7               $display("time\tprn\tt\tclk\tclrn\tq\tqn")
        ;
8               prn <= 1;
9               clrn <= 0;
10              clk <= 1;
11              t <= 1;
12              #23 $finish;
13        end
14        initial begin
15              $monitor("%2d\t%b\t%b\t%b\t%b\t%b\t%b",
        $time, prn, t, clk, clrn, q, qn);
16        end
17
18        initial begin
19        #1 clrn <= 1;
20        #2 t <= ~t;
21        #2 t <=     ~t;
22        #10 t <= ~t;
23        #2 t <= ~t;
24        end
25
```

```
26        always
27        #1 clk <= ~clk;
28
29        initial begin
30              $dumpfile("btff.vcd");
31              $dumpvars;
32        end
33  endmodule
```

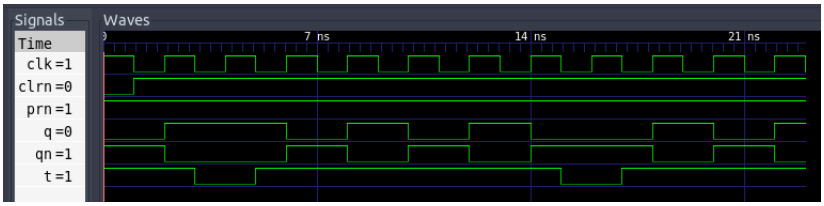
Listing 2: Behavioral TFF Test Bench

La tabla de verdad muestra mejor el cambio en q y qn conforme a la combinación de inputs.

```
1  time   prn    t      clk    clrn   q      qn
2  VCD info: dumpfile btff.vcd opened for output.
3   0     1      1      1      0      0      1
4   1     1      1      0      1      0      1
5   2     1      1      1      1      1      0
6   3     1      0      0      1      1      0
7   4     1      0      1      1      1      0
8   5     1      1      0      1      1      0
9   6     1      1      1      1      0      1
10  7     1      1      0      1      0      1
11  8     1      1      1      1      1      0
12  9     1      1      0      1      1      0
13 10     1      1      1      1      0      1
14 11     1      1      0      1      0      1
15 12     1      1      1      1      1      0
16 13     1      1      0      1      1      0
17 14     1      1      1      1      0      1
18 15     1      0      0      1      0      1
19 16     1      0      1      1      0      1
20 17     1      1      0      1      0      1
21 18     1      1      1      1      1      0
22 19     1      1      0      1      1      0
23 20     1      1      1      1      0      1
24 21     1      1      0      1      0      1
25 22     1      1      1      1      1      0
26 23     1      1      0      1      1      0
```

Listing 3: Tabla de verdad Behavioral TFF

A través de los *waves* se analiza con mayor claridad el cambio.



Por otra parte, analizando el schematic, también es posible la implementación *structural*. Para eso, se crearon submódulos.

Primero, se crearon compuertas NAND que acepten 3 y 4 inputs. Después, se implementó el *master T latch* con dichas compuertas. Asimismo, se construyó el *slave D latch* con las mismas compuerta. Finalmente, ambos latches se juntaron en un módulo grande: los inputs del T latch son los input del sistema, los input del D son los wire provenientes del T, y los output del D corresponden a los output del sistema.

```
module three_nand(a, b, c, z);
      input a, b, c;
      wire d;
      output z;
      and first_and(d, a, b);
      nand answer(z, d, c);
endmodule
```

Listing 4: Three NAND

```
module four_nand(a, b, c, d, z);
      input a, b, c, d;
      wire e, f;
      output z;
      and first_and(e, a, b);
      and second_and(f, c, d);
      nand answer(z, e, f);
endmodule
```

Listing 5: Four NAND

```
module tlatch(prn, t, nclk, clrn, q, qn, lna1, lna2);
      input prn, t, nclk, clrn;
      wire a, b;
      inout q, qn;
      output lna1, lna2;
      four_nand first_four_nand(qn, t, nclk, clrn, a);
      four_nand second_four_nand(prn, nclk, t, q, b);
      three_nand first_three_nand(a, prn, lna2, lna1);
      three_nand second_three_nand(lna1, clrn, b, lna2
   );
endmodule
```

Listing 6: Master T Latch

```
module dlatch(lna1, lna2, prn, clk, clrn, q, qn);
      input lna1, lna2, prn, clk, clrn;
      wire c, d;
      output q, qn;
      three_nand first_three_nand(lna1, clrn, clk, c);
      three_nand second_three_nand(lna2, prn, clk, d);
      three_nand third_three_nand(c, prn, qn, q);
      three_nand fourth_three_nand(q, clrn, d, qn);
endmodule
```

Listing 7: Slave D Latch

```
module stff(prn, t, clk, clrn, q, qn);
      input prn, t, clk, clrn;
      wire lna1, lna2;
      wire nclk;
      inout q, qn;
      not not_clk(nclk, clk);
      tlatch tl(prn, t, nclk, clrn, q, qn,  lna1, lna2
   );
      dlatch dl(lna1, lna2, prn, clk, clrn, q, qn);
endmodule
```

Listing 8: Structural TFF

El test bench es el mismo que en el caso behavioral.

```
`timescale 1ns/1ns
module stff_tb;
      reg prn, t, clk, clrn;
      wire q, qn;
      stff g(prn, t, clk, clrn, q, qn);
      initial begin
            $display("time\tprn\tt\tclk\tclrn\tq\tqn")
   ;
            prn <= 1;
            clrn <= 0;
            clk <= 1;
            t <= 1;
            #23 $finish;
      end
      initial begin
            $monitor("%2d\t%b\t%b\t%b\t%b\t%b\t%b",
   $time, prn, t, clk, clrn, q, qn);
      end

      initial begin
      #1 clrn <= 1;
      #2 t <= ~t;
      #2 t <=     ~t;
      #10 t <= ~t;
      #2 t <= ~t;
      end

      always
      #1 clk <= ~clk;

      initial begin
            $dumpfile("stff.vcd");
            $dumpvars;
      end
endmodule
```

Listing 9: Structural TFF Test Bench

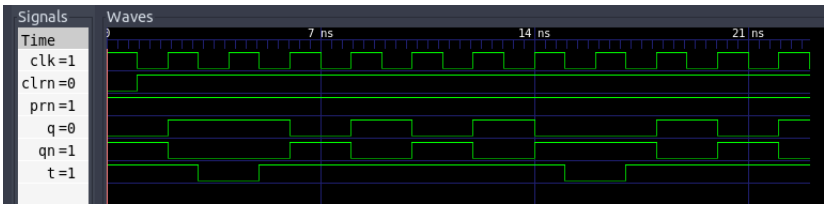
La tabla de verdad comprueba lo esperado.

```
time   prn    t      clk    clrn   q      qn
```

```
VCD info: dumpfile stff.vcd opened for output.
 0     1      1      1      0      0      1
 1     1      1      0      1      0      1
 2     1      1      1      1      1      0
 3     1      0      0      1      1      0
 4     1      0      1      1      1      0
 5     1      1      0      1      1      0
 6     1      1      1      1      0      1
 7     1      1      0      1      0      1
 8     1      1      1      1      1      0
 9     1      1      0      1      1      0
10     1      1      1      1      0      1
11     1      1      0      1      0      1
12     1      1      1      1      1      0
13     1      1      0      1      1      0
14     1      1      1      1      0      1
15     1      0      0      1      0      1
16     1      0      1      1      0      1
17     1      1      0      1      0      1
18     1      1      1      1      1      0
19     1      1      0      1      1      0
20     1      1      1      1      0      1
21     1      1      0      1      0      1
22     1      1      1      1      1      0
23     1      1      0      1      1      0
```

Listing 10: Tabla de verdad Structural TFF

Los *waves* indican el mismo comportamiento de señales.



- JKFF

  El jk Flip Flop es interesante de analizar, pues el cambio de q depende no solo de un input, aparte de clk, clrn t prn, sino de dos inputs: j y k. El comportamiento de acuerdo a las combinaciones entre clrn prn son las mismas que en el TFF. Al igual que en todo flip flop, el cambio de data en q solo se puede dar en el *posedge* del clock. Se observa que cuando el *clear* y el *preset* son igual a 0, los valores de q y qn entran a un estado de metaestabilidad. Asimismo, cuando el $prn = 1$ y $clrn = 0$, entonces $q = 1$. Por otra parte, cuando $clrn = 1$ y $prn = 0$, q toma el valor de 0. Si $clrn$ y prn son igual a 1, entonces recién se analiza los inputs j y k. Si $j = 0$ y $k = 1$, entonces $q = 0$. Por otra parte, si $j = 1$ y $k = 0$, q toma el valor de 1. Finalmente, solo si tanto j

como k son igual a 1, el valor de q será igual a la inversión del mismo ($\sim q$). Caber resaltar que si $j = k = 0$, q se mantiene igual a q_{prev}. En el caso behavioral, se implementó un módulo que verifica los casos expuestos en el *posedge* del clk. Todo se realizó en un mismo módulo.

```
module bjkff(prn, j, k, clk, clrn, q, qn);
      input prn, j, k, clk, clrn;
      output reg q, qn;

always @(posedge clk)
begin
      if(clrn == 1 & prn == 0)
      begin
            q <= 1;
            qn <= 0;
      end
      else if(clrn == 0 & prn == 1)
      begin
            q <= 0;
            qn <= 1;
      end
      else if (clrn == 0 & prn == 0)
      begin
            q <= 1'bX;
            qn <= 1'bX;
      end
      else
      begin
            if(j == 0 & k == 1)
            begin
                  q <= 0;
                  qn <= 1;
            end
            else if(j == 1 & k == 0)
            begin
                  q <= 1;
                  qn <= 0;
            end
            else if(j == 1 & k == 1)
            begin
                  q <= ~q;
                  qn <= ~qn;
            end

      end
end


endmodule
```

Listing 11: Behavioral JKFF

El test bench genera las combinaciones distintas del clear, clock

y t (prn se mantiene constante en 1) para que se pueda observar el cambio en q deseado por los *waves*.

```
‘timescale 1ns/1ns
module bjkff_tb;
      reg prn, j, k, clk, clrn;
      inout q, qn;
      bjkff g(prn, j, k, clk, clrn, q, qn);
      initial begin
            $display("time\tprn\tj\tk\tclk\tclrn\tq\
   tqn");
            prn <= 1;
            clrn <= 0;
            clk <= 1;
            j <= 0;
            k <= 0;
            #17 $finish;
      end
      initial begin
            $monitor("%2d\t%b\t%b\t%b\t%b\t%b\t%b\t%b"
   , $time, prn, j, k, clk, clrn, q, qn);
      end

      initial begin
      #1 clrn <= 1;
      #2 j <= ~j;
      #2 j <=     ~j;
      #2 j <= ~j;
      #8 j <= ~j;
      end
      initial begin
      #1 k <= ~k;
      #2 k <= ~k;
      #4 k <= ~k;
      #8 k <= ~k;
      end

      always
      #1 clk <= ~clk;

      initial begin
            $dumpfile("bjkff.vcd");
            $dumpvars;
      end
endmodule
```

Listing 12: Behavioral JKFF Test Bench

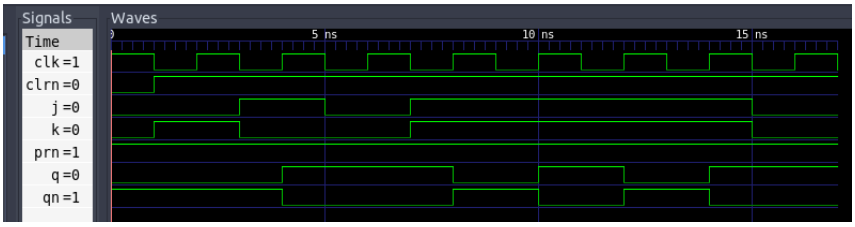
La tabla de verdad comprueba lo esperado.

```
time   prn    j      k      clk    clrn   q      qn
VCD info: dumpfile bjkff.vcd opened for output.
 0     1      0      0      1      0      0      1
 1     1      0      1      0      1      0      1
 2     1      0      1      1      1      0      1
```

```
 3     1     1     0     0     1     0     1
 4     1     1     0     1     1     1     0
 5     1     0     0     0     1     1     0
 6     1     0     0     1     1     1     0
 7     1     1     1     0     1     1     0
 8     1     1     1     1     1     0     1
 9     1     1     1     0     1     0     1
10     1     1     1     1     1     1     0
11     1     1     1     0     1     1     0
12     1     1     1     1     1     0     1
13     1     1     1     0     1     0     1
14     1     1     1     1     1     1     0
15     1     0     0     0     1     1     0
16     1     0     0     1     1     1     0
17     1     0     0     0     1     1     0
```

Listing 13: Tabla de verdad Behavioral JKFF

A través de los *waves* se analiza con mayor claridad el cambio.



Para el *structural*, se usaron las mismas compuertas NAND de cuatro y tres inputs que en el TFF. Se implementó el *master JK latch* con dichas compuertas. Además, se usó el mismo *Slave D Latch* del ejercicio anterior. Por último, ambos latches se juntaron en un módulo grande: los inputs del JK latch son los input del sistema, los input del D son los wire provenientes del T, y los output del D corresponden a los output del sistema.

```
module jklatch(prn, j, k, nclk, clrn, q, qn, lna1,
    lna2);
       input prn, j, k, nclk, clrn;
       wire a, b;
       inout q, qn;
       output lna1, lna2;
       four_nand first_four_nand(qn, j, nclk, clrn, a);
       four_nand second_four_nand(prn, nclk, k, q, b);
       three_nand first_three_nand(a, prn, lna2, lna1);
       three_nand second_three_nand(lna1, clrn, b, lna2
    );
endmodule
```

Listing 14: Master JK Latch

```
module sjkff(prn, j, k, clk, clrn, q, qn);
       input prn, j, k, clk, clrn;
```

```
      wire lna1, lna2;
      wire nclk;
      inout q, qn;
      not not_clk(nclk, clk);
      jklatch jkl(prn, j, k, nclk, clrn, q, qn, lna1,
    lna2);
      dlatch dl(lna1, lna2, prn, clk, clrn, q, qn);
endmodule
```

Listing 15: Structural JKFF

El test bench es el mismo que en el caso behavioral.

```
`timescale 1ns/1ns
module sjkff_tb;
      reg prn, j, k, clk, clrn;
      inout q, qn;
      sjkff g(prn, j, k, clk, clrn, q, qn);
      initial begin
            $display("time\tprn\tj\tk\tclk\tclrn\tq\
    tqn");
            prn <= 1;
            clrn <= 0;
            clk <= 1;
            j <= 0;
            k <= 0;
            #17 $finish;
      end
      initial begin
            $monitor("%2d\t%b\t%b\t%b\t%b\t%b\t%b\t%b"
    , $time, prn, j, k, clk, clrn, q, qn);
      end

      initial begin
      #1 clrn <= 1;
      #2 j <= ~j;
      #2 j <=    ~j;
      #2 j <= ~j;
      #8 j <= ~j;
      end
      initial begin
      #1 k <= ~k;
      #2 k <= ~k;
      #4 k <= ~k;
      #8 k <= ~k;
      end

      always
      #1 clk <= ~clk;

      initial begin
            $dumpfile("sjkff.vcd");
            $dumpvars;
      end
```

```
endmodule
```

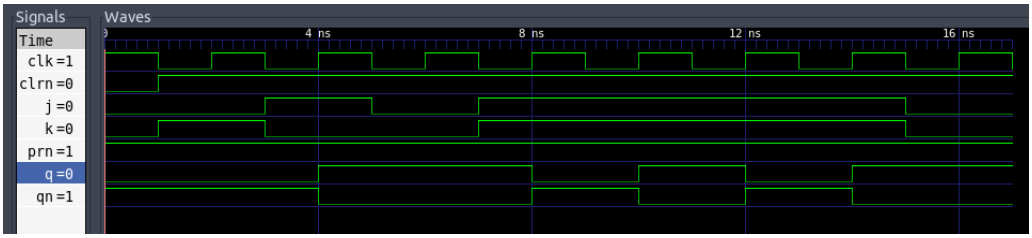
Listing 16: Structural JKFF Test Bench

La tabla de verdad comprueba lo esperado.

```
time   prn    j      k      clk    clrn   q      qn
VCD info: dumpfile sjkff.vcd opened for output.
 0     1      0      0      1      0      0      1
 1     1      0      1      0      1      0      1
 2     1      0      1      1      1      0      1
 3     1      1      0      0      1      0      1
 4     1      1      0      1      1      1      0
 5     1      0      0      0      1      1      0
 6     1      0      0      1      1      1      0
 7     1      1      1      0      1      1      0
 8     1      1      1      1      1      0      1
 9     1      1      1      0      1      0      1
10     1      1      1      1      1      1      0
11     1      1      1      0      1      1      0
12     1      1      1      1      1      0      1
13     1      1      1      0      1      0      1
14     1      1      1      1      1      1      0
15     1      0      0      0      1      1      0
16     1      0      0      1      1      1      0
17     1      0      0      0      1      1      0
```

Listing 17: Tabla de verdad Structural JKFF

A través de los *waves* se analiza con mayor claridad el cambio.



Ambas implementaciones, behavioral y structural tienen las mismas tablas de verdad y comportamiento de *waves*.

2. Schematic misterioso

   El funcionamiento de este esquemático es bastante peculiar. Para su implementación se tuvieron que crear tres submódulos: dff, buffer y mux2x1. El mux2x1 es el encargado de pasar determinada data por la "línea de acción". La señal que delimita qué data pasará, siempre será el load. Siempre que $load = 1$, el *wire* de dicho mux tomará el valor de $d[i]$. Este *wire* entra como input D al dff, o D Flip Flop. Este submódulo se encarga de pasar data a otro *wire*, y el valor de éste último será asignado a una posición específica del output paralelo $q[i]$. El buffer es el encargado de hacer la copia de esta última tarea.

```
module mux2x1(a, b, s, c);
	input a, b, s;
	output reg c;
	always @(*)begin
		case (s)
			0 : c = a;
			1 : c = b;
			default: c = c;
		endcase
	end
endmodule
```

Listing 18: MUX 2x1

```
module dff(d, clk, clrn, q);
	input d, clk, clrn;
	output reg q;
	always @(posedge clk)
	begin
		if(clrn==1)
		begin
			q = 0;
		end
		else
		begin
			q = d;
		end
	end
endmodule
```

Listing 19: D Flip Flop

```
module buffer(a, b);
	input a;
	output reg b;
	always @(*)
	begin
		b = a;
	end
endmodule
```

Listing 20: Buffer

La relación existente entre di y $q[2:0]$, es la siguiente: dado el *posedge clk* y el $clrn = 0$, el valor de $q[i]$ será el mismo valor que d, siempre y cuando, el load del tiempo previo haya sido 0.

Por otra parte, la relación entre $d[2:0]$ y do es la siguiente: dado el *posedge clk* y el $clrn = 0$, el valor de do siempre será el mismo valor que $d[0]$, siempre y cuando el load del tiempo previo haya sido 1. Si el load hubiera sido 0, entonces do depende de las compuertas anteriores. Caber resaltar que si el load se mantiene constante en 1 antes de hasta el 3er MUX, y si justo en ese momento es igual a 0, entonces el valor de do será el valor de d[1].

En *mystery.v* se "llaman" a todos los submódulos para construir el schematic.

```
module mystery(di, load, clk, clrn, d, q, do);
      input di, load, clk, clrn;
      input [2:0] d;
      output [2:0] q;
      wire m1, m2, m3;
      wire f1, f2;
      output do;

      mux2x1 first_mux(di, d[2], load, m1);
      dff first_dff(m1, clk, clrn, f1);
      buffer first_buffer(f1, q[2]);

      mux2x1 second_mux(f1, d[1], load, m2);
      dff second_dff(m2, clk, clrn, f2);
      buffer second_buffer(f2, q[1]);

      mux2x1 third_mux(f2, d[0], load, m3);
      dff third_dff(m3, clk, clrn, do);
      buffer third_buffer(do, q[0]);

endmodule
```

Listing 21: Mystery Schematic

En el test bench se realizan diferentes combinaciones entre los inputs, para analizar de cerca el comportamiento de las señales del output, tanto la serial como la paralela.

```
`timescale 1ns/1ns
module mystery_tb;
      reg di, load, clk, clrn;
      reg [2:0] d;
      wire [2:0] q;
      wire do;
      mystery g(di, load, clk, clrn, d, q, do);
      initial begin
            $display("time\tdi\tload\tclk\tclrn\td\tq\tdo");
            clrn <= 0;
            clk <= 0;
            di <= 0;
            load <= 0;
            d <= 3'b010;

            #40 $finish;
      end
      initial begin
            $monitor("%2d\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b", $time, di, load, clk, clrn, d, q, do);
      end

```

```
22
23
24        always #1 clk = ~clk;
25        always #6 d = d -1;
26        always #6 di = ~di;
27        always #20 load = ~load;
28
29        initial begin
30              $dumpfile("mystery.vcd");
31              $dumpvars;
32        end
33  endmodule
```

Listing 22: Mystery Schematic Test Bench

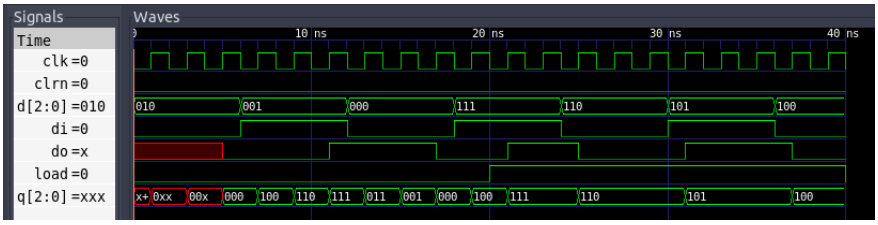
La tabla de verdad durante los 40 tiempos ayuda a observar el cambio en las señales. Como se puede ver, al inicio los outputs toman valores indeterminados, pues aún "no ha llegado" ninguna señal "conocida/determinada" al submódulo que los asignan. Asimismo, solo se puede dar la asignación durante el *posedge clk*.

```
1  time   di     load   clk    clrn   d      q      do
2  VCD info: dumpfile mystery.vcd opened for output.
3   0     0      0      0      0      010    xxx    x
4   1     0      0      1      0      010    0xx    x
5   2     0      0      0      0      010    0xx    x
6   3     0      0      1      0      010    00x    x
7   4     0      0      0      0      010    00x    x
8   5     0      0      1      0      010    000    0
9   6     1      0      0      0      001    000    0
10  7     1      0      1      0      001    100    0
11  8     1      0      0      0      001    100    0
12  9     1      0      1      0      001    110    0
13 10     1      0      0      0      001    110    0
14 11     1      0      1      0      001    111    1
15 12     0      0      0      0      000    111    1
16 13     0      0      1      0      000    011    1
17 14     0      0      0      0      000    011    1
18 15     0      0      1      0      000    001    1
19 16     0      0      0      0      000    001    1
20 17     0      0      1      0      000    000    0
21 18     1      0      0      0      111    000    0
22 19     1      0      1      0      111    100    0
23 20     1      1      0      0      111    100    0
24 21     1      1      1      0      111    111    1
25 22     1      1      0      0      111    111    1
26 23     1      1      1      0      111    111    1
27 24     0      1      0      0      110    111    1
28 25     0      1      1      0      110    110    0
29 26     0      1      0      0      110    110    0
30 27     0      1      1      0      110    110    0
31 28     0      1      0      0      110    110    0
32 29     0      1      1      0      110    110    0
33 30     1      1      0      0      101    110    0
```

```
31      1       1       1       0       101     101     1
32      1       1       0       0       101     101     1
33      1       1       1       0       101     101     1
34      1       1       0       0       101     101     1
35      1       1       1       0       101     101     1
36      0       1       0       0       100     101     1
37      0       1       1       0       100     100     0
38      0       1       0       0       100     100     0
39      0       1       1       0       100     100     0
40      0       0       0       0       100     100     0
```

Listing 23: Tabla de verdad Mystery Schematic

Los *waves* generados ayudan a comprender mejor el funcionamiento.



3. FIFO 4 Para conseguir el comportamiento mostrado en el schematic, se tuvieron que implementar una serie de submódulos. Los D Flip Flops "R*i*" se encargan de copiar cierta data a un wire, siempre que se de el *posedge clk*. Este flip flop ha sido adaptado a 8 bits, para que funcione como se plantean los *waves*. Además, se implementó el SR Latch, para que se generen las señales de los clock. Éste fue usando compuertas NOR y añadiendo un *clr*, el cual setea el output q a 0.

```
module dfffifo (d, clk, q);
      input [7:0] d;
      input  clk;
      output reg [7:0] q;
      always @(posedge clk)
      begin
            q = d;
      end
endmodule
```

Listing 24: D Flip Flop Adaptado

```
module sr(s, r, clr, q, qn);
      input s, r, clr;
      output reg q, qn;
      always @(*)
      begin
            if(clr == 1)
            begin
                  q = 0;
                  qn = 1;
```

```
            end
            else
            begin
                  q = r ~| qn;
                  qn = q ~| s;
            end
      end
endmodule
```

Listing 25: SR Latch

Estos submódulos fueron "llamados" en la implementación del módulo "top", en diferentes ocasiones, como se muestra en el schematic. Además de estos submódulos, fue necesario hacer uso de las *compuertas built-in* de Verilog AND, OR y buffer.

```
module fifo (din, write, read, clr, dout, full, empty);
      input write, read, clr;
      input [7:0] din;
      output [7:0] dout;
      output full, empty;

      wire f0q, f0qn, f1q, f1qn, f2q, f2qn, f3q, f3qn, f4q
    , f4qn;
      wire [7:0] d12, d23, d34;
      wire [0:0]clk1, clk2, clk3, clk4;
      wire clrf0;


      or orf0(clrf0, clr, clk1);
      sr f0(~clr, write, clrf0, f0q, f0qn);

      and clock1(clk1, f0q, f1qn);
      sr f1(clk1, clk2, clr, f1q, f1qn);
      dfffifo r1(din, clk1, d12);
      and clock2(clk2, f1q, f2qn);
      dfffifo r2(d12, clk2, d23);
      buf fu(full, f1q);
      and clock3(clk3, f2q, f3qn);
      dfffifo r3(d23, clk3, d34);
      sr f2(clk2, clk3, clr, f2q, f2qn);
      and clock4(clk4, f3q, ~read, f4qn);
      dfffifo r4(d34, clk4, dout);
      sr f3(clk3, clk4, clr, f3q, f3qn);
      sr f4(clk4, read, clr, f4q, f4qn);


      buf em(empty, f4qn);
endmodule
```

Listing 26: FIFO 4

Para lograr el comportamiento de señales de los *waves* planteados, fue necesario variar cada cierto tiempo cada input, generando diferentes

combinaciones, durante un periodo total de 190 ns.

```
`timescale 1ns/1ns
module fifo_tb;

      reg [7:0]din;
      reg write, read, clr;
      wire [7:0] dout;
      wire full, empty;

      fifo g(din, write, read, clr, dout, full, empty);
      initial begin
            $display("time\tclr\tdin\t\twrite\tread\tfull\
   tempty\tdout");
            clr <= 1;
            write <= 0;
            read <= 0;

            #190 $finish;
      end

initial begin
            $monitor("%2d\t%b\t%b\t%b\t%b\t%b\t%b\t%b",
   $time, clr, din, write, read,full, empty, dout);
      end

      initial begin
            #20 clr <= 0;
      end
      initial begin
            #20 din <= 8'he1;
            #20 din <= din +1;
            #20 din <= din +1;
            #20 din <= din +1;
      end

      initial begin
            #25 write <= ~write;
            #10 write <= ~write;
            #10 write <= ~write;
            #10 write <= ~write;
            #10 write <= ~write;
            #10 write <= ~write;
            #10 write <= ~write;
            #10 write <= ~write;
      end

      initial begin
            #105 read <= ~read;
            #10 read <= ~read;
            #10 read <= ~read;
            #10 read <= ~read;
            #10 read <= ~read;
```

```
51              #10 read <= ~read;
52              #10 read <= ~read;
53      end
54
55
56
57
58      initial begin
59              $dumpfile("fifo.vcd");
60              $dumpvars;
61      end
62  endmodule
```

Listing 27: FIFO 4 Test Bench

La tabla de verdad muestra el cambio del FIFO 4 a través del tiempo. Como se observa, el output *dout* no tiene un valor determinado, y eso está bien, pues aún no llega ninguna señal determinada al módulo previo que lo genera. También se ve que en ciertos momentos el FIFO 4 está lleno, pero en otros no, y en algunos incluso está totalmente vació.

```
1  time   clr    din          write read  full   empty dout
2  VCD info: dumpfile fifo.vcd opened for output.
3   0     1      xxxxxxxx     0     0     0      1     xxxxxxxx
4  20     0      11100001     0     0     0      1     xxxxxxxx
5  25     0      11100001     1     0     0      1     xxxxxxxx
6  35     0      11100001     0     0     0      0     11100001
7  40     0      11100010     0     0     0      0     11100001
8  45     0      11100010     1     0     0      0     11100001
9  55     0      11100010     0     0     0      0     11100001
10 60     0      11100011     0     0     0      0     11100001
11 65     0      11100011     1     0     0      0     11100001
12 75     0      11100011     0     0     0      0     11100001
13 80     0      11100100     0     0     0      0     11100001
14 85     0      11100100     1     0     0      0     11100001
15 95     0      11100100     0     0     1      0     11100001
16 105    0      11100100     0     1     1      1     11100001
17 115    0      11100100     0     0     0      0     11100010
18 125    0      11100100     0     1     0      1     11100010
19 135    0      11100100     0     0     0      0     11100011
20 145    0      11100100     0     1     0      1     11100011
21 155    0      11100100     0     0     0      0     11100100
22 165    0      11100100     0     1     0      1     11100100
23 175    0      11100100     0     0     0      1     11100100
```

Listing 28: Tabla de verdad FIFO 4

Los *waves* generados facilitan mucho el entendimiento del cambio en las señales.

Como se observa, los cambios en dout, full, empty son casi 100% precisos. El cambio de los inputs de igual manera. A la hora de analizar los cambios de las señales internas del módulo *fifo.v*, se puede ver que quizás no es tan exacto. Por un lado, ningún *clk* cambia a 1, siempre se mantiene en 0. Sin embargo, no habría razón para que se de esto. Ya que $d12$, $d23$ y $d34$ cambian en sus valores, al igual que *dout*, esto significa que su respectivo *clk* a tenido que pasar de 0 a 1, pues solo se transfiere la data en el *posedge*. Considero que los *waves* no son exactos quizás porque estos cambios se dan muy rápido, y no los detecta mi versión de GTKWave.

De igual manera, los *waves* obtenidos son como los presentados en la tarea.