



Microtecnología y
Sistemas Embebidos

Instituto Politécnico Nacional

Centro de Investigación en Computación

Lenguajes de descripción de hardware

Práctica 4 - Latches y flip-flops

PROFESOR:

M. EN C. OSVALDO ESPINOSA SOSA

POR:

ING. RICARDO ALDAIR TIRADO TORRES

CIUDAD DE MÉXICO, 25 DE MAYO DE 2024

Tabla de contenido

1. Objetivos	2
2. Latch RS	3
3. Latch D	6
4. Flip-Flop D	9
5. Latch D vs Flip-Flop D	13
6. Conclusiones	20
7. Anexos	21
7.1. Descripciones del hardware	21
7.2. Bancos de pruebas (<i>Test Benches</i>)	24

1. Objetivos

- Comprender la operación e implementación de circuitos secuenciales importantes como el flip-flop tipo D y el contador, así como sus variantes.
- Diferenciar la implementación de un *reset* asíncrono de uno síncrono en dos flip-flop tipo D utilizando el visor RTL.
- Aprender a describir un contador con múltiples terminales de control, como el *reset* asíncrono, *clock enable* (habilitador de la señal de reloj), carga de datos y sentido del conteo, además de la generación de un tope a este mismo conteo.

2. Latch RS

Actividad 1

Compilar el código del latch RS dos veces: con y sin la directiva (*“keep”*). Utilizar en ambos casos los visores *RTLViewer* y *Technology Map Viewer* estableciendo diferencias. Simular ambos casos sin retardos y con retardos (modelo lento a 85°C).

La visualización RTL, del latch RS implementado con la directiva *keep*, se muestra en la Figura 1 y sin esta directiva en la Figura 2. Los módulos se diferencian en que el circuito con *keep* conserva más compuertas lógicas que el que no usa la directiva, además de mantener intactas a las señales internas.

La visualización con el *Technology Map Viewer*, del latch RS implementado con la directiva *keep*, se muestra en la Figura 3 y sin esta directiva en la Figura 4. Los módulos se diferencian en que el circuito con *keep* implementa una celda lógica por cada señal interna declarada en el código, mientras que el circuito sin *keep* emplea una sola celda lógica para todo el circuito.

Las simulaciones sin retardos se visualizan en la Figura 5 para el latch con directiva (*keep*) y en la Figura 6 para el latch sin directiva. Ambos circuitos funcionan de la misma manera, de tal forma que

- Si $S = 0$ y $R = 0$: La salida mantiene el último estado adquirido.
- Si $S = 1$ y $R = 0$: La salida adquiere un valor alto.
- Si $S = 0$ y $R = 1$: La salida es restablecida a un nivel bajo.
- Si $S = 1$ y $R = 1$: Es un estado prohibido que no se implementó en la simulación

Nótese que los cambios en la salida ocurren únicamente cuando CLK cambia a un nivel lógico alto.

Las simulaciones con retardos (modo lento a 85°C) se visualizan en la Figura 7 para el latch con directiva *keep* y en la Figura 8 para el latch sin directiva. Se observa que, debido a que

se simuló en el peor de los escenarios, la señal de salida Q, tarda en actualizar su valor en ambas simulaciones, no obstante, para el caso del circuito con *keep*, el retardo es ligeramente mayor en comparación con el circuito sin *keep*.

En los Anexos se localiza la descripción del latch RS. Además de declarar a las entradas y salidas, se describieron a Ri, Si, Qa y Qb como señales internas y por medio de la descripción de flujo de datos de bajo nivel, se asignaron los valores de estas señales y de la salida Q. Cabe resaltar que la diferencia entre los circuitos simulados en este apartado, es la directiva *synthesis keep* en la declaración de las señales internas.

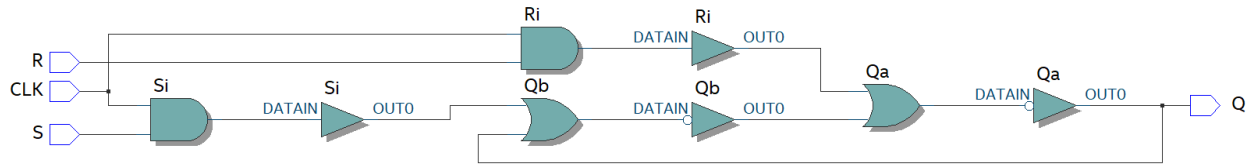


Figura 1: Diagrama RTL del Latch RS, descrito con la directiva *keep*.

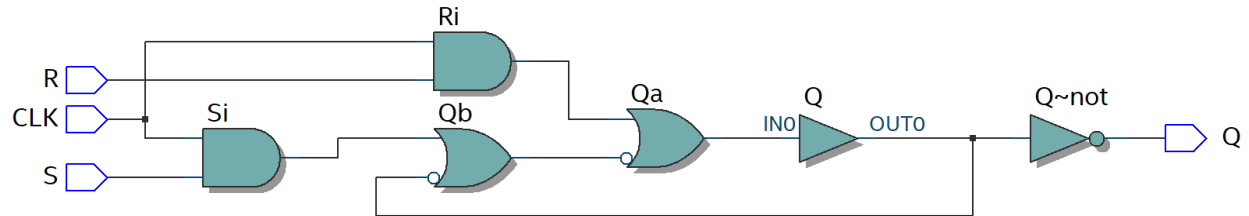


Figura 2: Diagrama RTL del Latch RS, descrito sin la directiva *keep*.

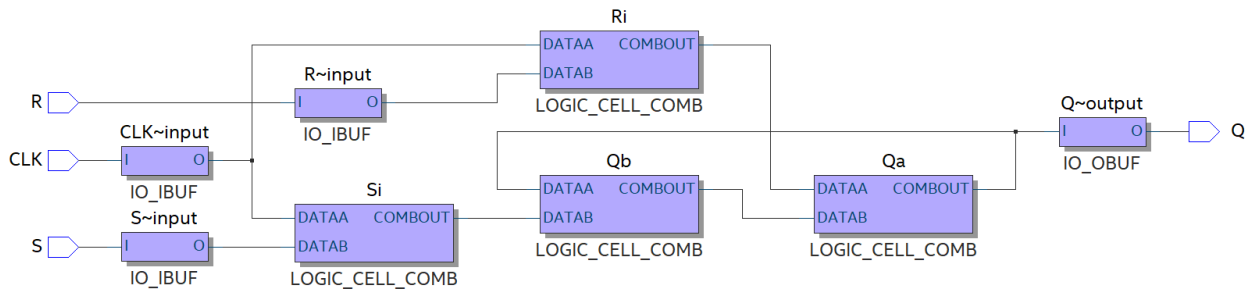


Figura 3: Latch RS (descrito con la directiva *keep*) visto desde el *Technology Map Viewer*.

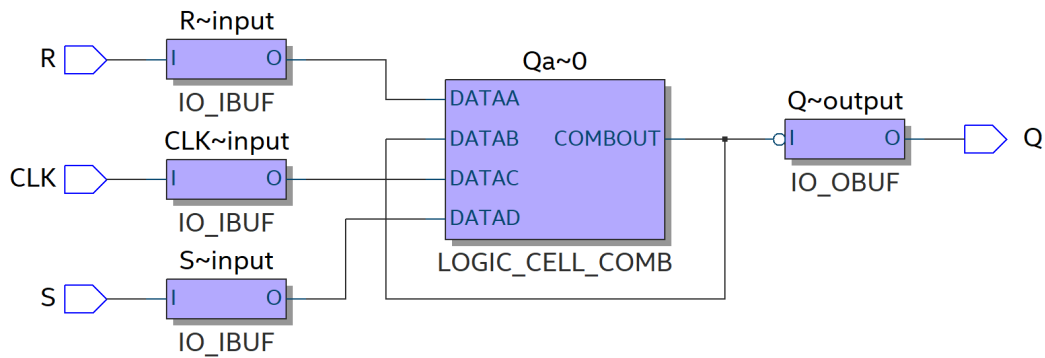


Figura 4: Latch RS (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer*.



Figura 5: Simulación sin retardos del Latch RS (descrito con la directiva *keep*) en el visor de formas de onda de ModelSim.

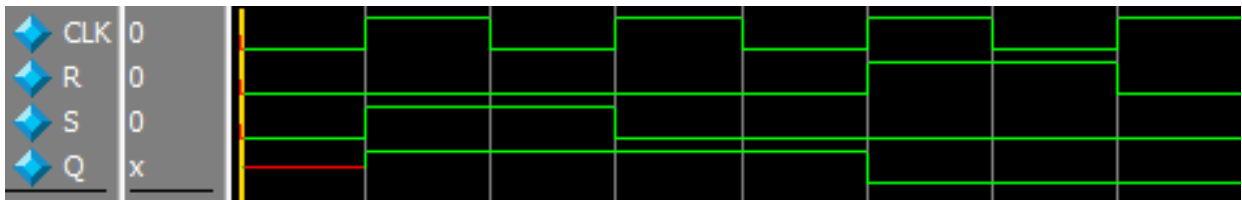


Figura 6: Simulación sin retardos del Latch RS (descrito sin la directiva *keep*) en el visor de formas de onda de ModelSim.

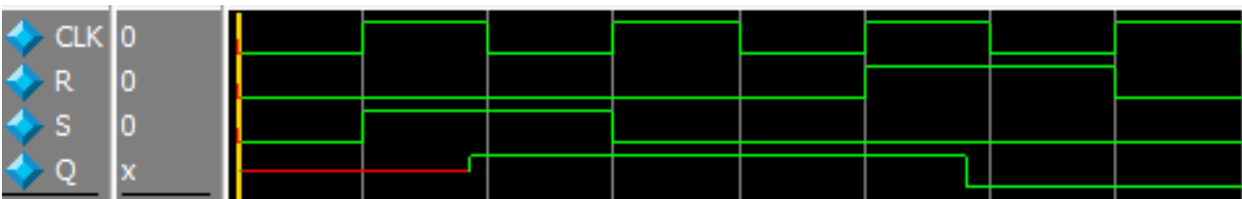


Figura 7: Simulación con retardos del Latch RS (descrito con la directiva *keep*) en el visor de formas de onda de ModelSim.

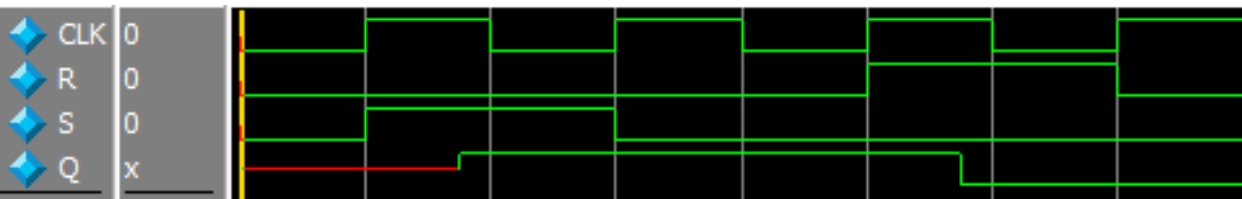
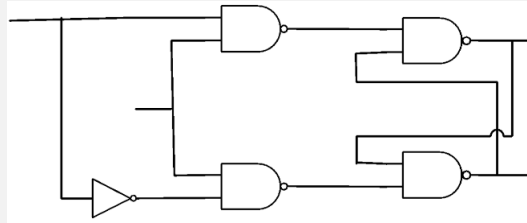


Figura 8: Simulación sin retardos del Latch RS (descrito sin la directiva *keep*) en el visor de formas de onda de ModelSim.

3. Latch D

Actividad 2

El circuito para un latch D es el siguiente:



Repetir los pasos del inciso 1.

La visualización RTL, del latch D implementado con la directiva *keep*, se muestra en la Figura 9 y sin esta directiva en la Figura 10. Los módulos se diferencian en que el circuito con *keep* conserva más compuertas lógicas que el que no usa la directiva, además de mantener intactas a las señales internas.

La visualización con el *Technology Map Viewer*, del latch D implementado con la directiva *keep*, se muestra en la Figura 11 y sin esta directiva en la Figura 12. Los módulos se diferencian en que el circuito con *keep* implementa una celda lógica por cada señal interna declarada en el código, mientras que el circuito sin *keep* emplea una sola celda lógica para todo el circuito.

Las simulaciones sin retardos se visualizan en la Figura 13 para el latch con directiva *keep* y en la Figura 14 para el latch sin directiva. Ambos circuitos funcionan de la misma manera, de tal forma que

- Si $D = 0$: La salida adquiere un valor bajo.
- Si $D = 1$: La salida adquiere un valor alto.

Nótese que los cambios en la salida ocurren únicamente cuando CLK cambia a un nivel lógico alto.

Las simulaciones con retardos (modo lento a 85°C) se visualizan en la Figura 15 para el latch con directiva *keep* y en la Figura 16 para el latch sin directiva. Se observa que, debido a que se simuló en el peor de los escenarios, la señal de salida Q, tarda en actualizar su valor en

ambas simulaciones, no obstante, para el caso del circuito con *keep*, el retardo es ligeramente mayor en comparación con el circuito sin *keep*.

En los Anexos se localiza la descripción del latch RS. Además de declarar a las entradas y salidas, se describieron a NAND1, NAND2, NAND3 y NAND4 como señales internas y por medio de la descripción de flujo de datos de bajo nivel, se asignaron los valores de estas señales y de la salida Q. Cabe resaltar que la diferencia entre los circuitos simulados en este apartado, es la directiva *synthesis keep* en la declaración de las señales internas.

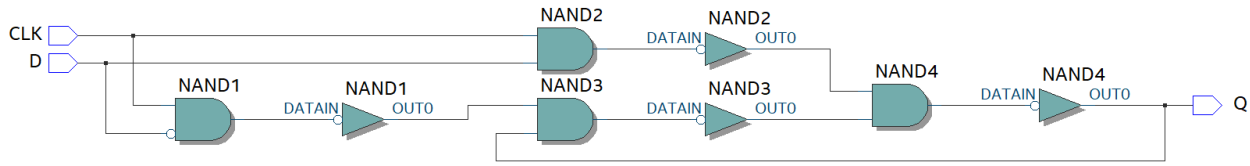


Figura 9: Diagrama RTL del Latch D, descrito con la directiva *keep*.

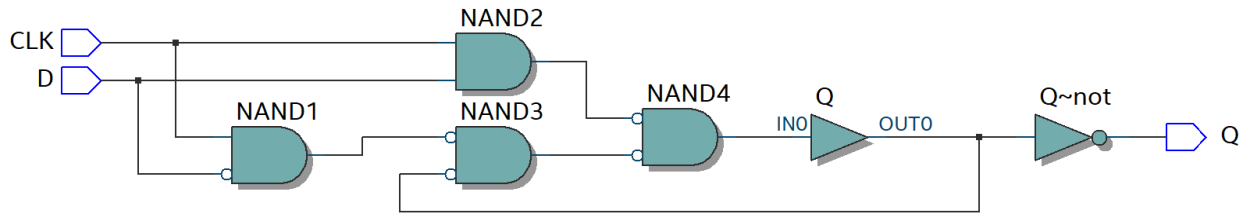


Figura 10: Diagrama RTL del Latch D, descrito sin la directiva *keep*.

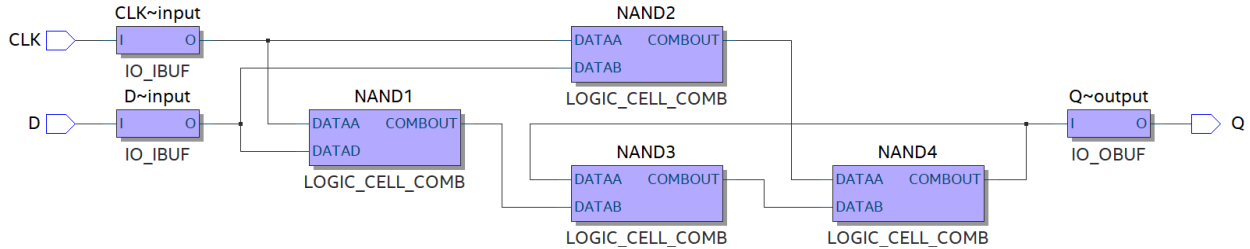


Figura 11: Latch D (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer*.

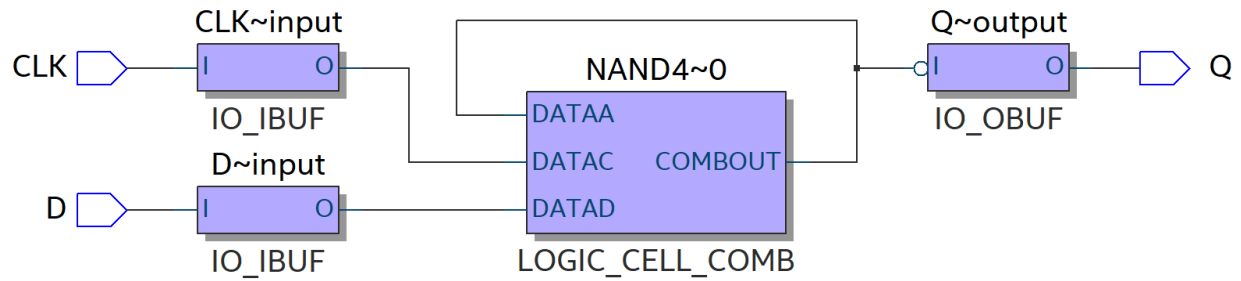


Figura 12: Latch D (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer*.



Figura 13: Simulación sin retardos del Latch D (descrito con la directiva *keep*) en el visor de formas de onda de ModelSim.



Figura 14: Simulación sin retardos del Latch D (descrito sin la directiva *keep*) en el visor de formas de onda de ModelSim.



Figura 15: Simulación con retardos del Latch D (descrito con la directiva *keep*) en el visor de formas de onda de ModelSim.

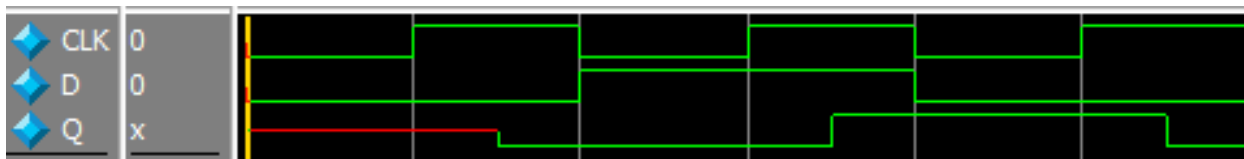
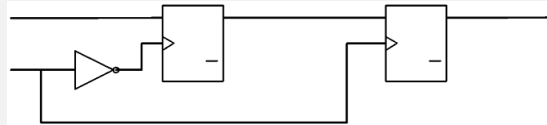


Figura 16: Simulación sin retardos del Latch D (descrito sin la directiva *keep*) en el visor de formas de onda de ModelSim.

4. Flip-Flop D

Actividad 3

Un flip-flop D puede construirse a partir de dos latch tipo D en cascada (configuración MASTER-SLAVE) de acuerdo al siguiente diagrama:



Usar dos instancias del latch D del inciso 2. Repetir los pasos del inciso 1.

La visualización RTL, del flip-flop D implementado con la directiva (*keep*), se muestra en la Figura 17 y sin esta directiva en la Figura 18. Se observa que dentro de estas instancias se tiene la descripción por flujo de datos de bajo nivel, o sea, la conexión entre compuertas lógicas (Ver Figura 19 y Figura 20). Los módulos se diferencian en que el circuito con *keep* conserva más compuertas lógicas que el que no usa la directiva, además de mantener intactas a las señales internas.

La visualización con el *Technology Map Viewer*, del flip-flop D implementado con la directiva *keep*, se muestra en la Figura 21 y sin esta directiva en la Figura 22. En la Figura 23 y Figura 24, se observa que dentro de las instancias se generan celdas lógicas. Los módulos se diferencian en que el circuito con *keep* implementa una celda lógica por cada señal interna declarada en la instancia, mientras que el circuito sin *keep* emplea una sola celda lógica para todo el circuito instanciado. Nótese que, la señal que conecta un latch D con el otro, también genera una celda lógica para el circuito con *keep*, ya que es una señal interna del módulo.

Las simulaciones sin retardos se visualizan en la Figura 25 para el flip-flop D con directiva *keep* y en la Figura 26 para el flip-flop D sin directiva. Ambos circuitos funcionan de la misma manera, de tal forma que

- **Si $D = 0$:** La salida adquiere un valor bajo.
- **Si $D = 1$:** La salida adquiere un valor alto.

Nótese que los cambios en la salida ocurren unicamente cuando CLK tiene un flanco de subida.

Las simulaciones con retardos (modo lento a 85°C) se visualizan en la Figura 27 para el flip-flop D con directiva *keep* y en la Figura 28 para el flip-flop D sin directiva. Se observa que, debido a que se simuló en el peor de los escenarios, la señal de salida Q, tarda en actualizar su valor en ambas simulaciones, no obstante, para el caso del circuito con *keep*, el retardo es ligeramente mayor en comparación con el circuito sin *keep*.

En los Anexos se localiza la descripción del flip-flop D. Tomando como instancia al latch D, declarado en la actividad 2, se realizó la descripción del módulo de forma estructural.

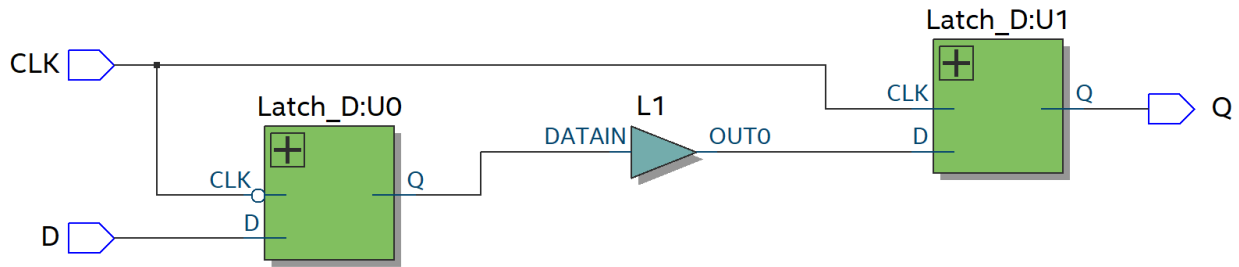


Figura 17: Diagrama RTL del flip-flop D, descrito con la directiva *keep*.

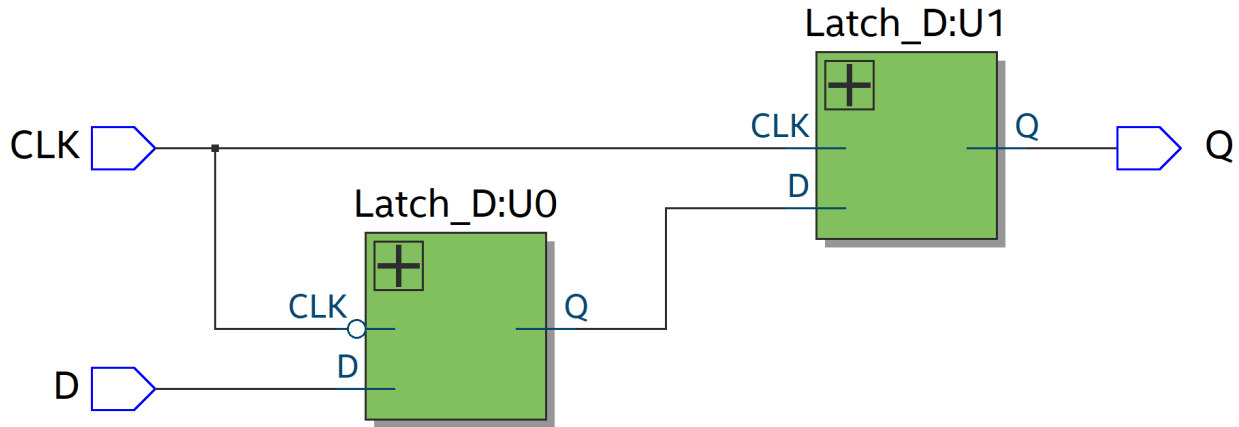


Figura 18: Diagrama RTL del flip-flop D, descrito sin la directiva *keep*.

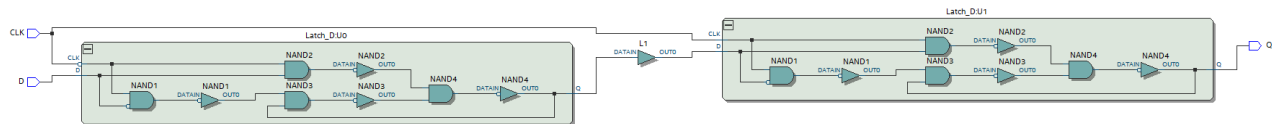


Figura 19: Diagrama RTL del flip-flop D, descrito con la directiva *keep* (acercamiento al interior de la instancia).

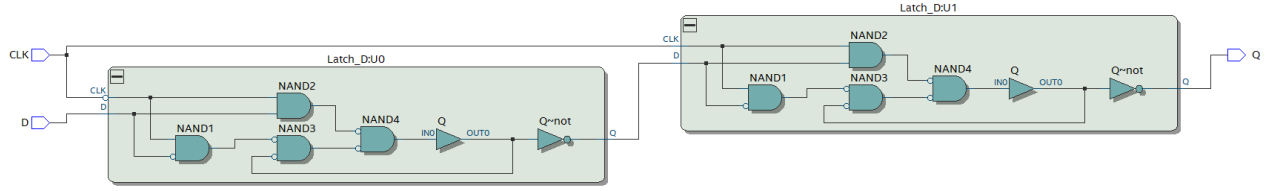


Figura 20: Diagrama RTL del flip-flop D, descrito sin la directiva *keep* (acercamiento al interior de la instancia).

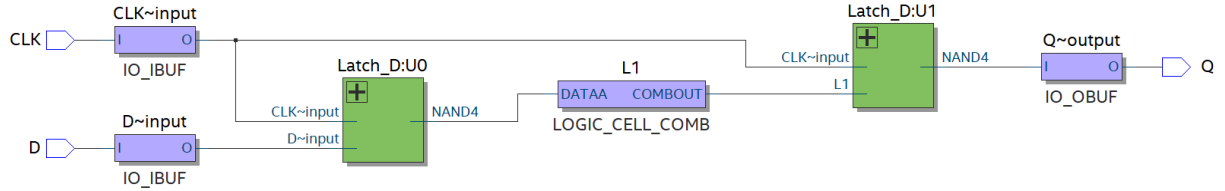


Figura 21: Flip-Flop D (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer*.

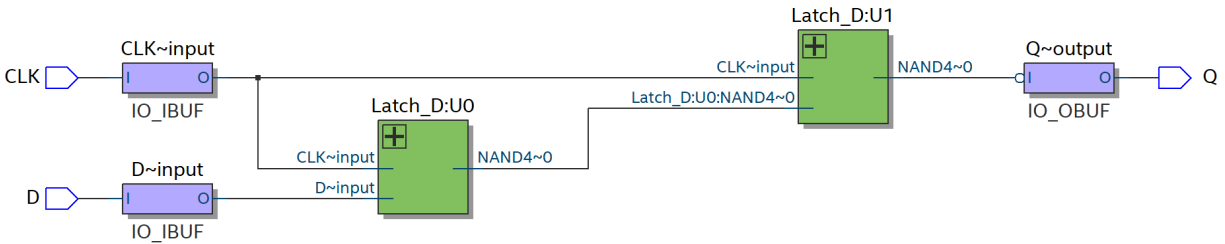


Figura 22: Flip-Flop D (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer*.

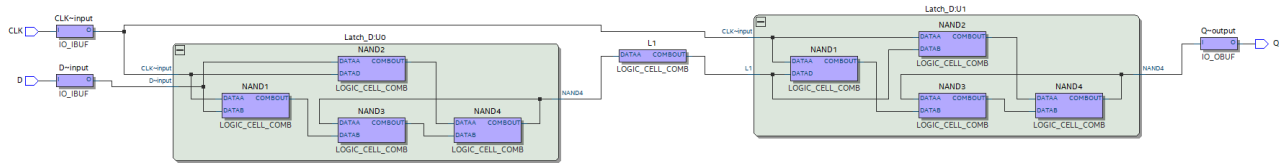


Figura 23: Flip-Flop D (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer* (acercamiento al interior de la instancia).

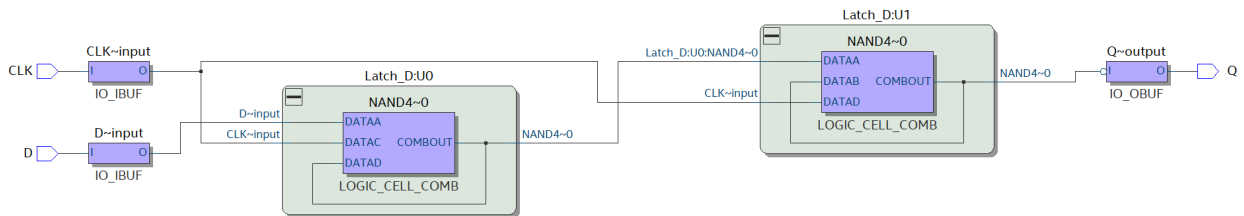


Figura 24: Flip-Flop D (descrito sin la directiva *keep*) visto desde el *Technology Map Viewer* (acercamiento al interior de la instancia).

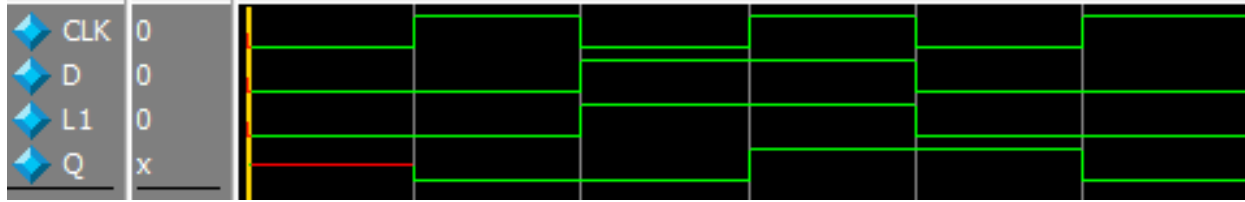


Figura 25: Simulación sin retardos del flip-flop D (descrito con la directiva *keep*) en el visor de formas de onda de ModelSim.

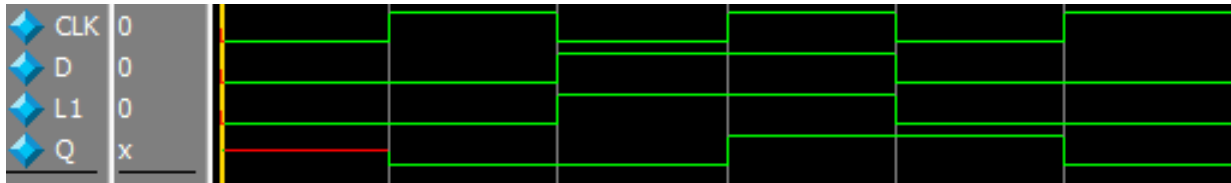


Figura 26: Simulación sin retardos del flip-flop D (descrito sin la directiva *keep*) en el visor de formas de onda de ModelSim.

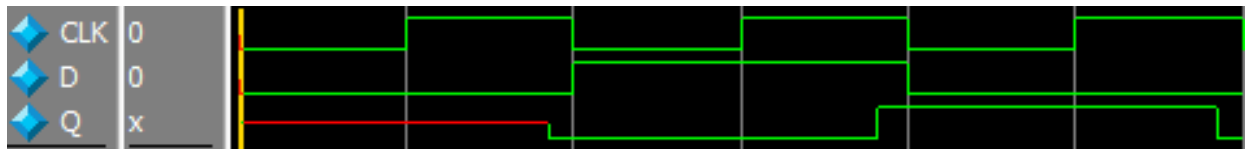


Figura 27: Simulación con retardos del flip-flop D (descrito con la directiva *keep*) en el visor de formas de onda de ModelSim.



Figura 28: Simulación sin retardos del flip-flop D (descrito sin la directiva *keep*) en el visor de formas de onda de ModelSim.

5. Latch D vs Flip-Flop D

Actividad 4

En dos proyectos separados describir por comportamiento un latch D simple y un flip-flop D simple (sólo dos entradas: D y CLK). Compilar y usar el visor que permita observar el elemento lógico donde se implementan el latch y el flip-flop. ¿En qué parte del elemento lógico se implementa el latch y en que parte se implementa el flip-flop?

La visualización RTL, del latch D descrito por comportamiento, se muestra en la Figura 29 y del flip flop D descrito por comportamiento en la Figura 30. Ahora bien, a través del *Chip Planner* se observa que el latch D se implementa en el elemento lógico de la Figura 31 y el flip-flop D en el elemento de la Figura 32 (Ver en la Figura 33 y en la Figura 34 los elementos acercados). Aunque pareciera que el latch D y el flip-flop D se efectúan, de igual forma, en un elemento lógico, se observa en la Figura 35 que el latch se implementa en la *Look Up Table* de un elemento lógico, mientras que en la Figura 36 se ve que el flip-flop se implementa de forma directa en el elemento lógico.

Las simulaciones se visualizan en la Figura 37 para el latch D y en la Figura 38 para el flip-flop D. Se visualiza un correcto funcionamiento de ambos módulos, ya que el latch varía a la salida cuando en CLK hay un cambio al nivel lógico alto, mientras que el flip-flop lo hace con un flanco de subida en CLK.

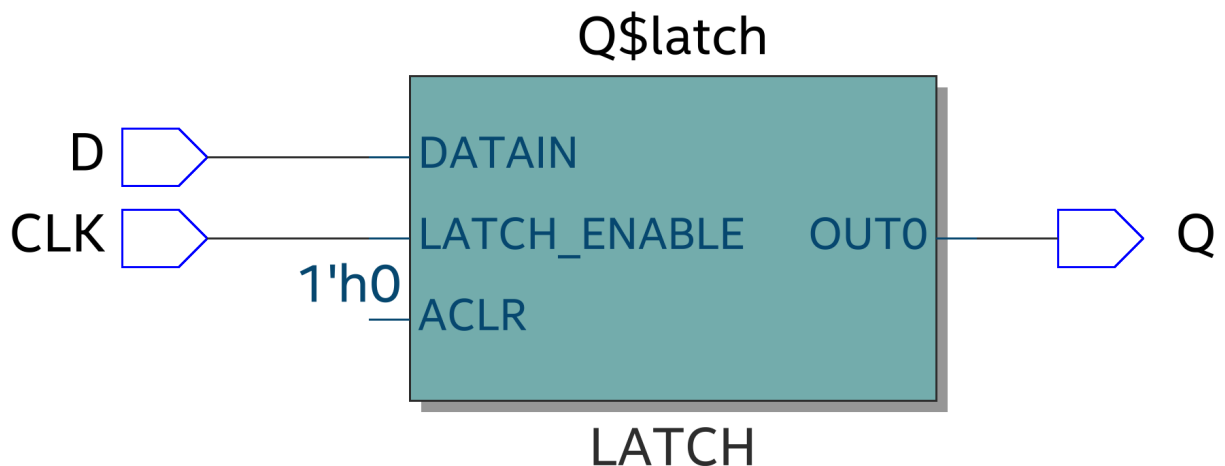


Figura 29: Diagrama RTL del latch D, descrito por comportamiento.

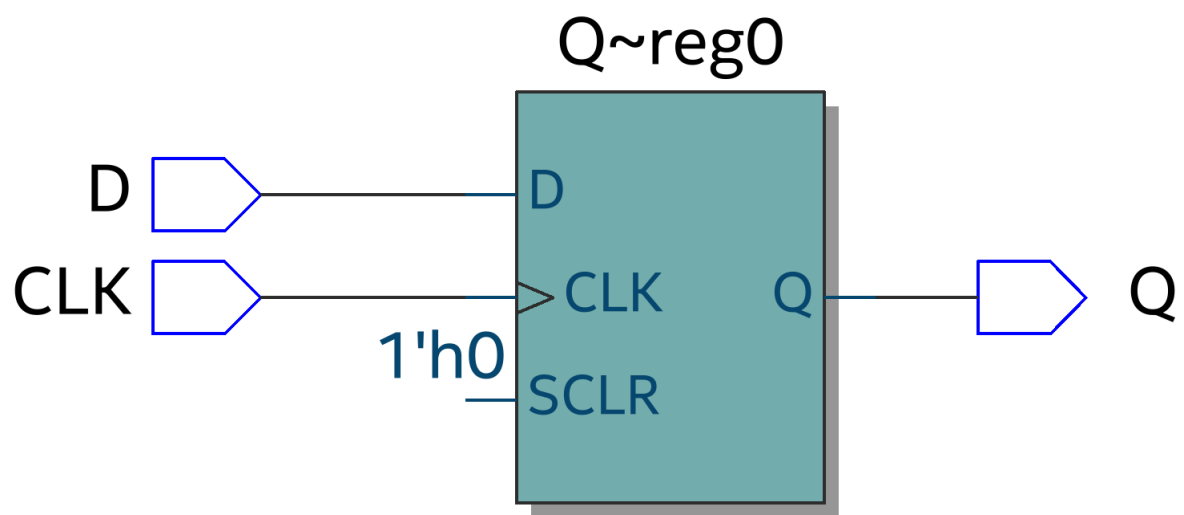


Figura 30: Diagrama RTL del flip-flop D, descrito por comportamiento.

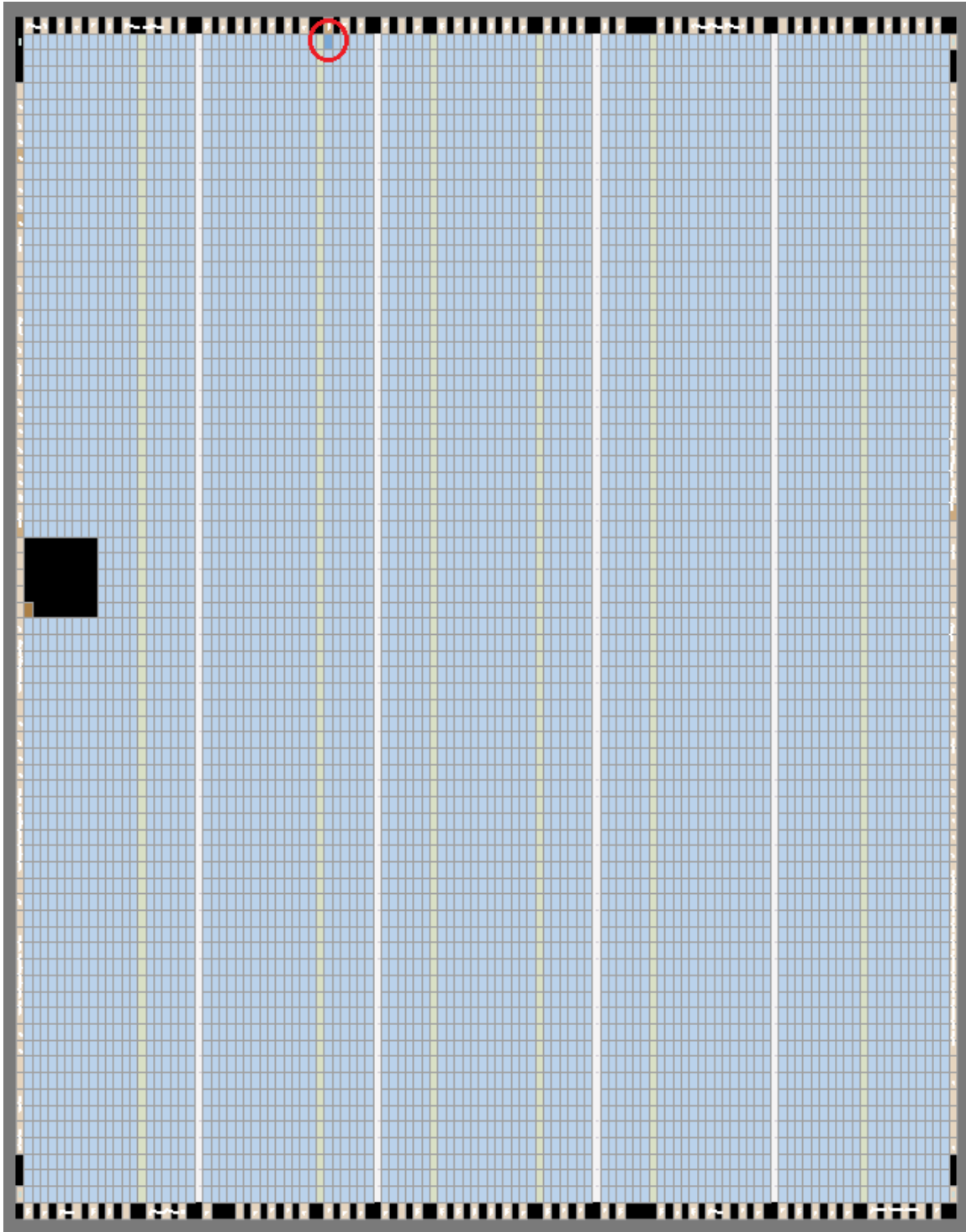


Figura 31: Latch D visto desde el *Chip Planner* (círculo rojo).

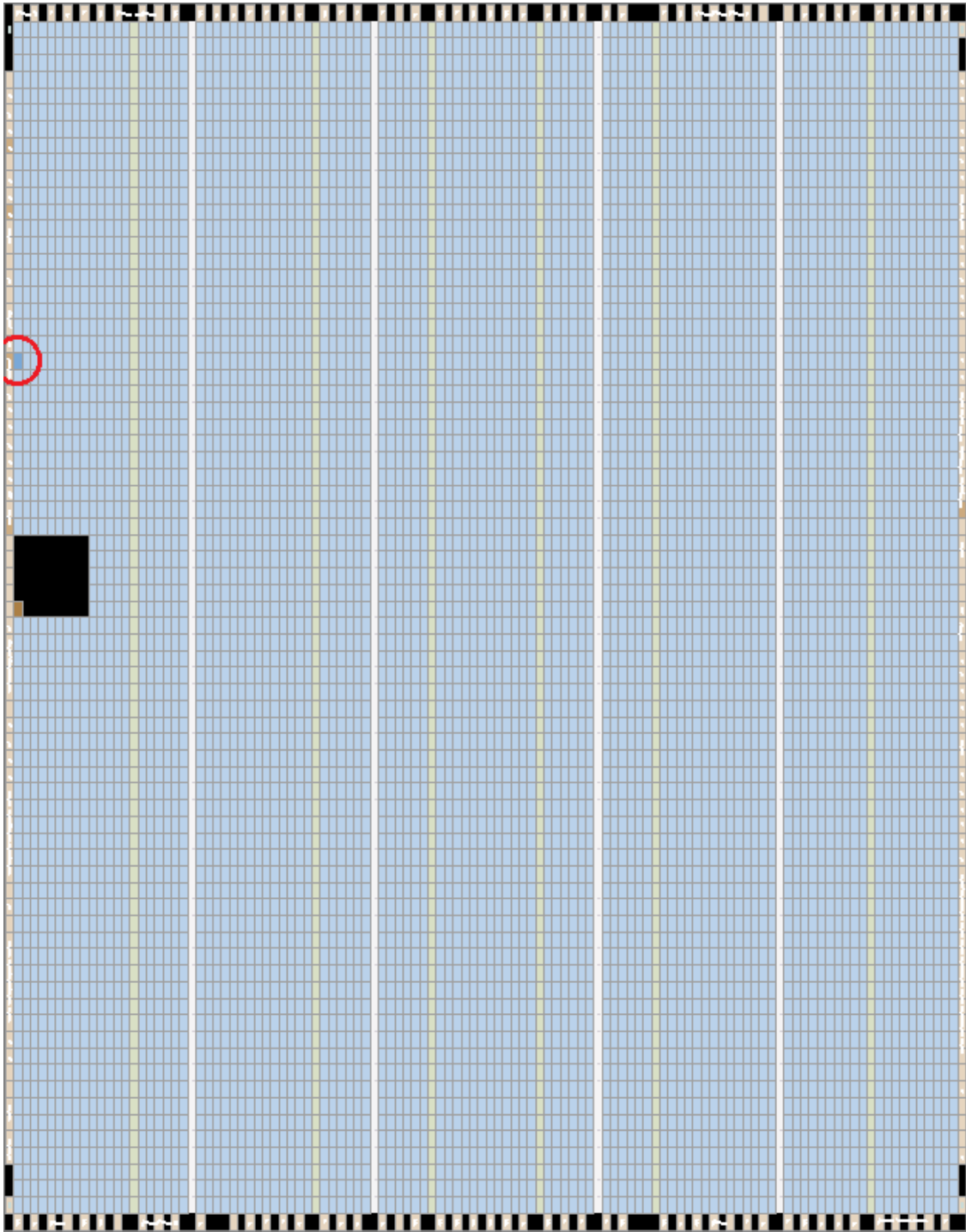


Figura 32: Flip-Flop D visto desde el *Chip Planner* (círculo rojo).

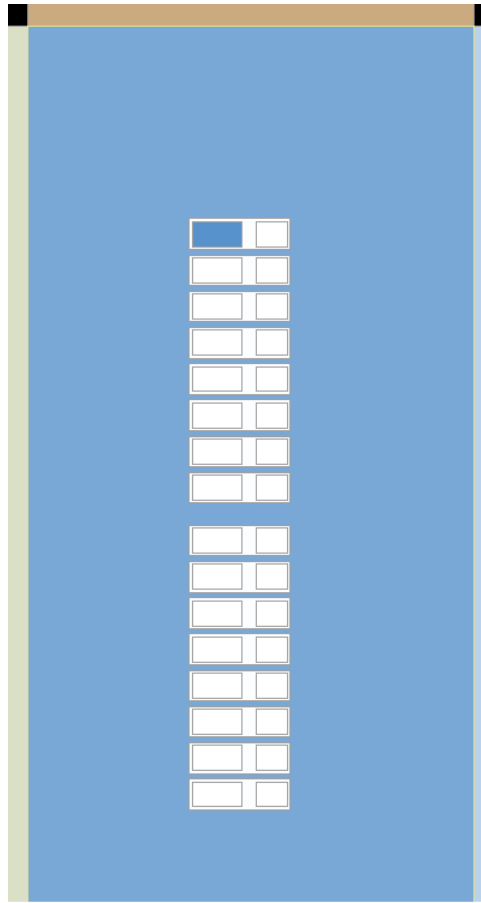


Figura 33: Latch D visto desde el *Chip Planner* (acercamiento).

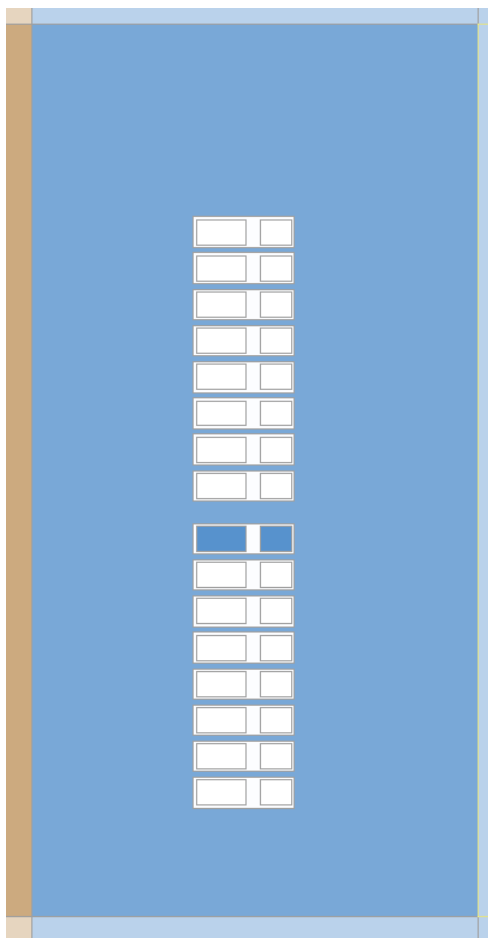


Figura 34: Flip-Flop D visto desde el *Chip Planner* (acercamiento).

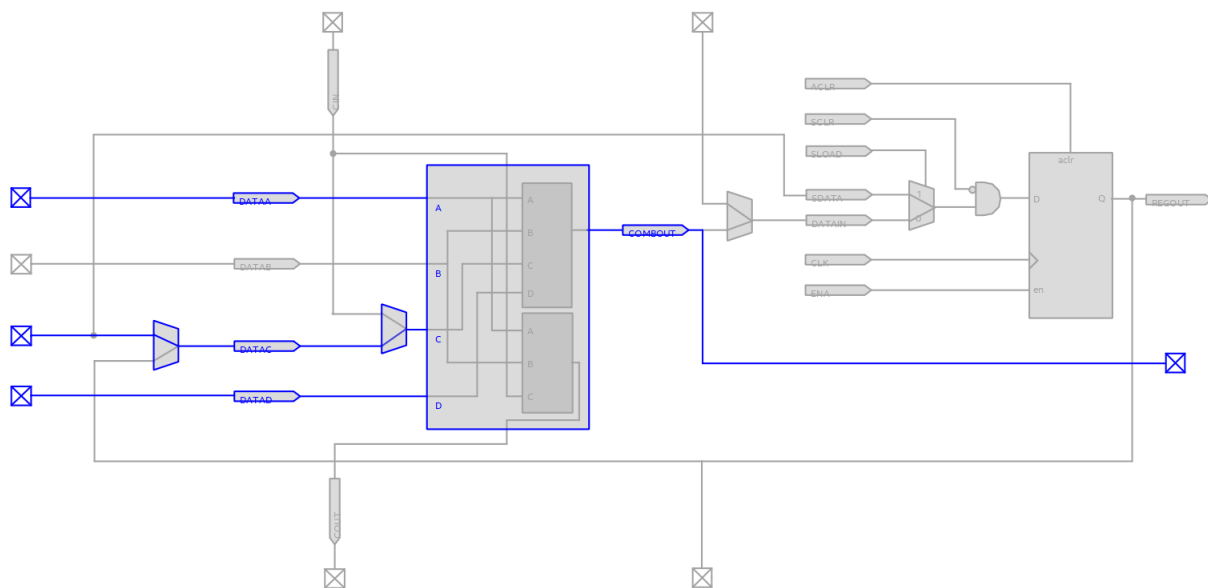


Figura 35: Latch D visto desde el *Chip Planner* (implementación en elemento lógico).

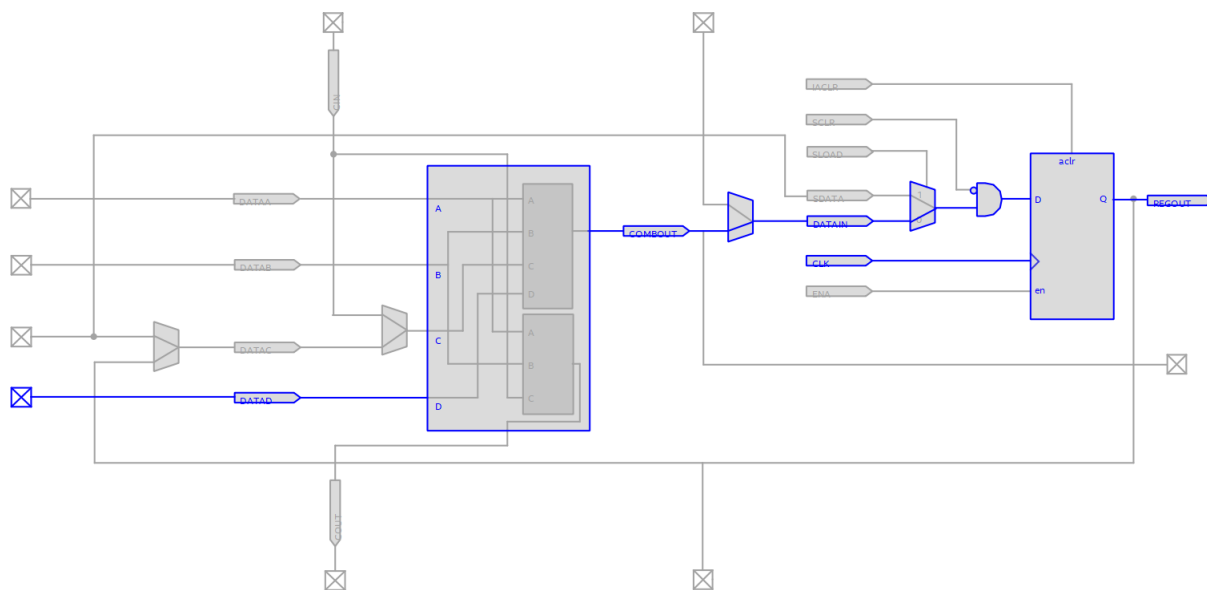


Figura 36: Flip-Flop D visto desde el *Chip Planner* (implementación en elemento lógico).

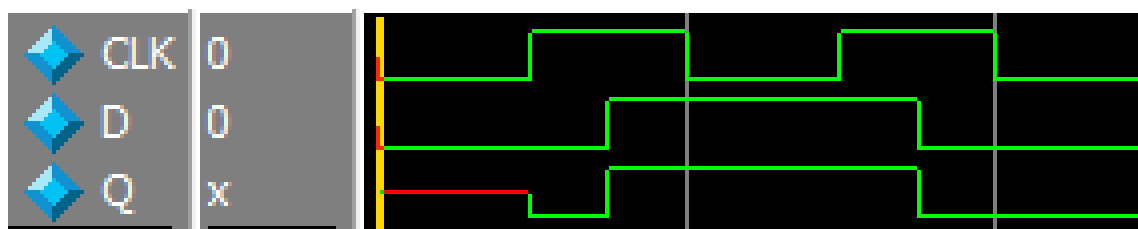


Figura 37: Simulación del latch D en el visor de formas de onda de ModelSim.

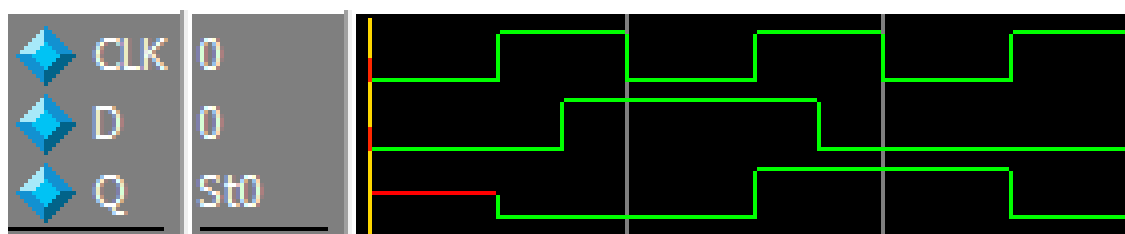


Figura 38: Simulación del flip-flop D en el visor de formas de onda de ModelSim.

6. Conclusiones

En conclusión, se implementaron los 4 circuitos en lenguaje Verilog de manera exitosa.

Para los flip flop tipo D, se implementaron de manera correcta y se diferenció el funcionamiento del *reset* asíncrono del síncrono, así como sus características físicas en el visor RTL.

Para los contadores de 8 bits, se describieron de forma adecuada. Para el contador sencillo se observó como se implementa en el visor RTL, utilizando únicamente un flip flop de 8 bits junto con un sumador en el lazo de retroalimentación. Para el contador completo, se entendió como implementar la señales de control con respecto a un orden de prioridad y se observó la complejidad de este modulo al implementar no solo un flip flop de 8 bits, sino también sumadores, multiplexores y comparadores.

Se comprobó su funcionamiento utilizando las simulaciones de forma de onda en ModelSim y se asignaron los pines correspondientes en la placa de desarrollo para programar el dispositivo y realizar las pruebas pertinentes en hardware.

En los Anexos se pueden encontrar los códigos implementados junto con sus respectivos bancos de pruebas.

7. Anexos

7.1. Descripciones del hardware

```
1 module Latch_RS(CLK, R, S, Q);
2   input  CLK, R, S;
3   output Q;
4   wire   Ri, Si, Qa, Qb /*synthesis keep*/;
5
6   assign Ri = R & CLK;
7   assign Si = S & CLK;
8   assign Qa = ~(Ri | Qb);
9   assign Qb = ~(Si | Qa);
10  assign Q  = Qa;
11
12 endmodule
```

Programa 1: Descripción en Verilog del flip flop tipo D con *reset* asíncrono.

```
1 module Latch_D(CLK, D, Q);
2   input  CLK, D;
3   output Q;
4   wire   NAND1, NAND2, NAND3, NAND4 /*synthesis keep*/;
5
6   assign NAND1 = ~(~D & CLK);
7   assign NAND2 = ~(D & CLK);
8   assign NAND3 = ~(NAND1 & NAND4);
9   assign NAND4 = ~(NAND2 & NAND3);
10  assign Q  = NAND4;
11
12 endmodule
```

Programa 2: Descripción en Verilog del flip flop tipo D con *reset* síncrono.

```
1 module Latch_D(CLK, D, Q);
2   input  CLK, D;
3   output Q;
4   wire   NAND1, NAND2, NAND3, NAND4 /*synthesis keep*/;
5
6   assign NAND1 = ~(~D & CLK);
7   assign NAND2 = ~(D & CLK);
8   assign NAND3 = ~(NAND1 & NAND4);
9   assign NAND4 = ~(NAND2 & NAND3);
10  assign Q  = NAND4;
11
```

```

12 endmodule
13
14 module FlipFlop_D(CLK, D, Q);
15     input    CLK, D;
16     output   Q;
17     wire     L1 /*synthesis keep*/;
18
19     Latch_D U0(~CLK, D, L1);
20     Latch_D U1(CLK, L1, Q);
21
22 endmodule

```

Programa 3: Descripción en Verilog del contador con *reset* asíncrono.

```

1 module L_D (
2     input    D,
3     input    CLK,
4     output   reg Q
5 );
6
7 always @ (D or CLK)
8 begin
9     if (CLK)
10    begin
11        Q <= D;
12    end
13 end
14
15 endmodule

```

Programa 4: Descripción en Verilog del contador con *reset* asíncrono, *clock enable*, carga de datos, sentido y tope de la cuenta.

```

1 module FF_D (
2     input    D,
3     input    CLK,
4     output   reg Q
5 );
6
7 always @ (posedge CLK)
8 begin
9     Q <= D;
10 end
11

```

12 `endmodule`

Programa 5: Descripción en Verilog del contador con *reset* asíncrono, *clock enable*, carga de datos, sentido y tope de la cuenta.

7.2. Bancos de pruebas (*Test Benches*)

```
1 `timescale 1 ns/ 1 ps
2 module Latch_RS_vlg_tst();
3   reg CLK;
4   reg R;
5   reg S;
6   wire Q;
7
8   Latch_RS i1 (
9     .CLK(CLK),
10    .Q(Q),
11    .R(R),
12    .S(S)
13  );
14
15  initial
16  begin
17    CLK = 0; R = 0; S = 0;
18    #10; R = 0; S = 1;
19    #20; R = 0; S = 0;
20    #20; R = 1; S = 0;
21    #20; R = 0; S = 0;
22    $display("Running testbench at CIC");
23  end
24
25  always
26  begin
27    #10; CLK = ~CLK;
28  end
29
30 endmodule
```

Programa 6: Banco de prueba para el Programa 1.

```
1 `timescale 1 ns/ 1 ps
2 module Latch_D_vlg_tst();
3   reg CLK;
4   reg D;
5   wire Q;
6
7   Latch_D i1 (
8     .CLK(CLK),
9     .D(D),
10    .Q(Q)
11  );
```

```

12
13 initial
14 begin
15     CLK = 0; D = 0;
16     #20; D = 1;
17     #20; D = 0;
18     $display("Running testbench at CIC");
19 end
20
21 always
22 begin
23     #10; CLK = ~CLK;
24 end
25
26 endmodule

```

Programa 7: Banco de prueba para el Programa 2.

```

1 `timescale 1 ns/ 1 ps
2 module FlipFlop_D_vlg_tst();
3     reg CLK;
4     reg D;
5     wire Q;
6
7     FlipFlop_D i1 (
8         .CLK(CLK),
9         .D(D),
10        .Q(Q)
11    );
12
13    initial
14    begin
15        CLK = 0; D = 0;
16        #20; D = 1;
17        #20; D = 0;
18        $display("Running testbench at CIC");
19    end
20
21    always
22    begin
23        #10; CLK = ~CLK;
24    end
25
26 endmodule

```

Programa 8: Banco de prueba para el Programa 3.

```

1  'timescale 1 ns/ 1 ps
2  module L_D_vlg_tst();
3      reg  CLK;
4      reg  D;
5      wire Q;
6
7      L_D i1 (
8          .CLK(CLK),
9          .D(D),
10         .Q(Q)
11     );
12
13     initial
14     begin
15         CLK = 0; D = 0;
16         #15; D = 1;
17         #20; D = 0;
18         $display("Running testbench at CIC");
19     end
20
21     always
22     begin
23         #10; CLK = ~CLK;
24     end
25
26 endmodule

```

Programa 9: Banco de prueba para el Programa 4.

```

1  'timescale 1 ns/ 1 ps
2  module FF_D_vlg_tst();
3      reg  CLK;
4      reg  D;
5      wire Q;
6
7      FF_D i1 (
8          .CLK(CLK),
9          .D(D),
10         .Q(Q)
11     );
12
13     initial
14     begin
15         CLK = 0; D = 0;
16         #15; D = 1;
17         #20; D = 0;

```

```
18     $display("Running testbench at CIC");
19 end
20
21 always
22 begin
23     #10; CLK = ~CLK;
24 end
25
26 endmodule
```

Programa 10: Banco de prueba para el Programa 5.