Universidad Autónoma De Querétaro
Facultad de Ingeniería

Autonomous University of Querétaro

Faculty of Engineering.

Career: Automation Engineer.

Subject: Systems with reconfigurable logic.

Student: Martínez Murillo Omar Yarif.
Student: Diego Joel Zúñiga Fragoso.
Student: Daniela del Carmen Manríquez Navarro.
Student: Joselyn Gallegos Abreo.

Practice 5: Flip-Flops

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Introduction

This practice explores the design and implementation of different types of flip-flops and a clock divider using VHDL in Quartus. The flip-flops studied include D, JK, SR, and T, both in synchronous and asynchronous configurations. These flip-flops are fundamental building blocks in sequential logic circuits, each serving specific purposes such as data storage, toggling, and control.

In addition, a clock divider was implemented using a synchronous D flip-flop. The clock divider is essential for generating lower frequency signals from a higher frequency input, which is useful in timing and control applications. By assigning inputs and outputs on the FPGA, the behavior of each flip-flop and the clock divider can be observed in real-time, providing practical insights into sequential circuit design.

Methodology

Step 1: Flip Flop type D

- First, the D-type flip flop was studied in its synchronous and asynchronous form.
- Then, its code was programmed in Quartus.
- Finally, it was loaded into the FPGA and its tests were performed.

Step 2: Flip Flop type JK

- First, the JK-type flip flop was studied in its synchronous and asynchronous form.
- Then, its code was programmed in Quartus.
- Finally, it was loaded into the FPGA and its tests were performed.

Step 3: Flip Flop type SR

- First, the SR-type flip flop was studied in its synchronous and asynchronous form.
- Later, its code was programmed in synchronous and asynchronous versions in quartus.

- Finally, it was loaded into the FPGA and its tests were performed.

Step 4: Flip Flop type T

- First, the T-type flip flop was studied in its synchronous and asynchronous form.
- Then, its code was programmed in Quartus.
- Finally, it was loaded into the FPGA and its tests were performed.

Step 5: Clock divisor

Once the code for the D-type flip flop was obtained, it was only cascaded several times to divide the clock frequency to the desired one.

Results

Base algorithm Flip-flop type D

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.std_logic_arith.all;
4
5   entity Candado_D is
6       port
7       (
8           i_LD    : in std_logic;      -- Señal de reloj
9           i_D     : in std_logic;
10          o_Q     : out std_logic;
11          o_Qc    : out std_logic
12      );
13  end Candado_D;
14
15  architecture Candado of Candado_D is
16      signal Q  : std_logic;
17      signal Qc : std_logic;
18  begin
19      Q <= (i_D nand i_LD) nand Qc;
20      Qc <= (i_LD nand (not i_D)) nand Q;
21
22      o_Q     <= Q;
23      o_Qc    <= Qc;
24  end Candado;
```

Base algorithm Flip-flop type JK

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.std_logic_arith.all;
4
5   entity Candado_JK is
6       port
7       (
8           i_LD    : in std_logic;      -- Señal de reloj
9           i_J     : in std_logic;
10          i_K     : in std_logic;
11          o_Q     : out std_logic;
12          o_Qc    : out std_logic
13      );
14  end Candado_JK;
15
16  architecture Candado of Candado_JK is
17      signal Q  : std_logic;
18      signal Qc : std_logic;
19  begin
20      Q   <= (Q and i_K and i_LD) nor Qc;
21      Qc  <= (Qc and i_J and i_LD) nor Q;
22
23      o_Q     <= Q;
24      o_Qc    <= Qc;
25  end Candado;
```

Base algorithm Flip-flop type SR

```
2   use IEEE.std_logic_1164.all;
3
4   entity Candado_SR_sinc is
5       port
6       (
7           -- Entradas y salidas de candado SR con compuertas NOR
8           i_R     : in std_logic;
9           i_S     : in std_logic;
10          i_LD    : in std_logic;      -- Señal de reloj
11          o_Q     : out std_logic;
12          o_Qc    : out std_logic
13      );
14  end Candado_SR_sinc;
15
16  architecture Candado of Candado_SR_sinc is
17      signal Q      : std_logic;
18      signal Qc     : std_logic;
19  begin
20      -- Candado SR con compuertas NOR
21      Q       <= (i_R and i_LD) nor Qc;
22      Qc      <= (i_S and i_LD) nor Q;
23
24      o_Q     <= Q;
25      o_Qc    <= Qc;
26
27  end Candado;
```

```
 4  entity Candado_SR_asin is
 5      port
 6      (
 7          -- Entradas y salidas de candado SR con compuertas NOR
 8          i_R_nor      : in std_logic;
 9          i_S_nor      : in std_logic;
10          o_Q_nor      : out std_logic;
11          o_Qc_nor     : out std_logic;
12          -- Entradas y salidas de candado SR con compuertas NAND
13          i_R_nand      : in std_logic;
14          i_S_nand      : in std_logic;
15          o_Q_nand      : out std_logic;
16          o_Qc_nand     : out std_logic
17      );
18  end Candado_SR_asin;
```

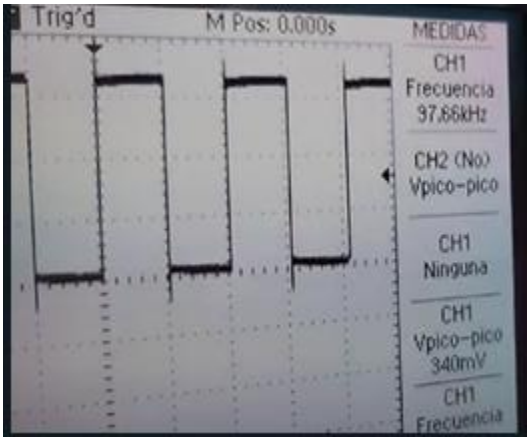
```
20  architecture Candado of Candado_SR_asin is
21      signal Q_nor       : std_logic;
22      signal Qc_nor      : std_logic;
23
24      signal Q_nand      : std_logic;
25      signal Qc_nand     : std_logic;
26  begin
27
28      -- Candado SR con compuertas NOR
29      Q_nor      <= i_R_nor nor Qc_nor;
30      Qc_nor     <= i_S_nor nor Q_nor;
31
32      o_Q_nor      <= Q_nor;
33      o_Qc_nor     <= Qc_nor;
34
35      -- Candado SR con compuertas NAND
36      Q_nand     <= i_R_nand nand Qc_nand;
37      Qc_nand    <= i_S_nand nand Q_nand;
38
39      o_Q_nand      <= Q_nand;
40      o_Qc_nand     <= Qc_nand;
41
42  end Candado;
```

Base algorithm Flip-flop type T

```
4   entity Candado_T is
5       port
6       (
7           i_LD    : in std_logic;      -- Señal de reloj
8           i_T     : in std_logic;
9           o_Q     : out std_logic;
10          o_Qc    : out std_logic
11      );
12  end Candado_T;
13
14  architecture Candado of Candado_T is
15      signal Q  : std_logic;
16      signal Qc : std_logic;
17      signal Q1  : std_logic;
18      signal Qc1 : std_logic;
19  begin
20      Q1   <= (i_T and i_LD and Q);
21      Qc1  <= (i_LD and i_T and Qc);
22
23      Q   <= (Q1 nor Qc);
24      Qc  <= (Qc1 nor Q);
25
26      o_Q     <= Q;
27      o_Qc    <= Qc;
28  end Candado;
```

Base algorithm Clock divisor.

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4
5   entity Divisor_Reloj_D is
6      port(
7      clock:in std_logic;--el reloj es de 50Mhz
8      D: in std_logic;--señal
9      o_Q: out std_logic --salida
10     );
11  end Divisor_Reloj_D;
12
13  architecture Candado_D of Divisor_Reloj_D is
14  begin
15      process(D,clock)
16      begin
17        if(clock='1')then
18        o_Q<=D;
19        end if;
20      end process;
21
22  end Candado_D;
23
```



Conclusion

The design and implementation of various flip-flops (D, JK, SR, and T) in both synchronous and asynchronous modes, along with a clock divider using a D flip-flop, provided a comprehensive understanding of sequential logic circuits. These flip-flops serve as essential components in digital systems, enabling operations such as data storage, toggling, and control.

By utilizing VHDL in Quartus and deploying the configurations onto an FPGA, we were able to observe real-time behavior. The clock divider, crucial for managing timing in digital systems, demonstrated how frequency reduction can be achieved through simple flip-flop arrangements.