Escuela Politécnica nacional
Facultad de Ingeniería Eléctrica y Electrónica
Laboratorio de Sistemas Digitales
http://detri.epn.edu.ec
Quito, Ecuador

LABORATORIO DE SISTEMAS DIGITALES

PRÁCTICA No. 8

Fecha: 03/07/2017-07/07/2017

Tema: FLIP-FLOPS

1. Objetivos:

Familiarizar al estudiante con la utilización y funcionamiento de circuitos de dispositivos del tipo de arreglos lógicos programables que realizan operaciones aritméticas binarias y funciones lógicas.

2. Preparatorio

I. Consultar la distribución de pines y la tabla de funcionamiento de los Circuitos integrados: 7476, 74107, 74109, 74112. Esta información servirá para la elaborar el circuito para esta Práctica.

II. Hacer una breve descripción del software Active HDL, software necesario para la simulación de FPGAs. El software lo pueden encontrar en el siguiente enlace https://www.aldec.com/students/student.php?id=4.

Implementación Física y Proteus

III. Diseñar, utilizando solamente compuertas NAND, un flip – flop S – R síncrono activado con señal de reloj CLK en estado alto y que tenga PRESET Y CLEAR.

IV. Con el circuito integrado 7476, o algún equivalente, diseñar un flip – flop tipo D y tipo T.

V. Con el circuito integrado 7476, o algún equivalente, en configuración de flip – flop tipo J - K, diseñar un contador asincrónico ascendente según la siguiente tabla. Incluya el circuito de borrado manual.

Día	Módulo
Lunes	12
Martes	13
Miércoles	14
Jueves	10
Viernes	11

Simulación Active HDL

VI. Elaborar el código en VHDL que permita implementar un flip – flop S – R Asincrónico

VII. Realizar el código VHDL necesario para implementar un contador ascendente del módulo indicado en la siguiente tabla.

Día	Módulo
Lunes	187
Martes	165
Miércoles	234
Jueves	192
Viernes	223

3. Parte Practica

Implementar los circuitos correspondientes a los ítems III, IV, V del trabajo preparatorio y presentar la simulación de los literales VI y VII en Active HDL.

4. Informe

I. Hacer el análisis de los resultados obtenidos en esta práctica. Comente las modificaciones hechas a su circuito y las causas que las motivaron.

II. Investigar acerca de la utilidad de los flip – flops en circuitos prácticos

III. Explicar porque se le considera al flip-flop como la unidad básica de memoria y comente la siguiente afirmación “Los flip – flops son la base de las memorias”.

IV. Consultar sobre los circuitos detectores de flanco para Flip – Flops.”.

V. Utilizando flip – flops tipo J -K, diseñar un contador asíncrono módulo 47, con control ascendente – descendente y control de arranque y detención. Presente su diseño simulado en paquete computacional Proteus.

5. Conclusiones

6. Recomendaciones

BIBLIOGRAFÍA:

[1] TOCCI/WIDMER/MOSS. “Sistemas Digitales. Principios y Aplicaciones”. Prentice Hall. 10ma. Edición. 2007.

[2] TECHNISCHE UNIVERSITÄT CHEMNITZ, «VHDL-Online,» Technische Universität Chemnitz, [En línea]. Available: https://www.vhdl-online.de. [Último acceso: 7 6 2017].

Realizado por: Ing. Víctor Reyes. – Profesor Ocasional 2

Aprobado por: Ing. Ramiro Morejón – Jefe del Laboratorio de Sistemas Digitales

Escuela Politécnica nacional
Facultad de Ingeniería Eléctrica y Electrónica
Laboratorio de Sistemas Digitales
http://detri.epn.edu.ec
Quito, Ecuador

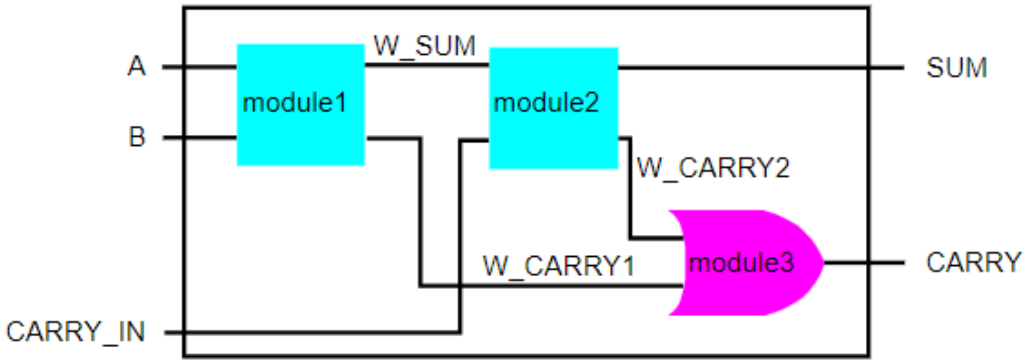
Material de Apoyo

Elementos de Apoyo

Component

VHDL permite un diseño de modelo jerárquico, lo que significa que un módulo se puede componer de varios submódulos. Las conexiones entre estos submódulos se definen dentro de la arquitectura de un módulo superior.

Por ejemplo, para la implementación de un sumador completo se pueden utilizar 2 semi-sumadores y una compuerta or como se muestra en la figura.



Para realizar un código que emplee components primero debemos declarar la estructura de cada uno de los componentes que se emplearan, cabe destacar que esta estructura debe ser la misma que la del componente en su codificación original, es decir que si existe un component HALFADDER debe existir un archivo con extensión .vhd que contenga la funcionalidad del bloque. A continuación, se muestra la sintaxis de declaración de un component, esta sección de código debe estar dentro de la arquitectura antes del comienzo (begin).

```
component HALFADDER
    port (A, B       : in bit;
          SUM, CARRY : out bit);
end component;
```

Par instanciar un component se utiliza port map, este básicamente permitirá definir que datos se van a utilizar como entradas y salidas. Por ejemplo, en el siguiente código se muestra la forma de instanciar el componente HALFADDER.

Nota: “MODULE2” es una etiqueta que va a identificar la instanciación dado que se pueden realizar varias instancias de un mismo component.

```
MODULE2 : HALFADDER
    port map (W_SUM, CARRY_IN,
              SUM, W_CARRY2);
```

Process

Consiste en un conjunto de sentencias que se ejecutan de forma secuencial, una vez que alguna de las señales de la lista de sensibilidad a cambiado su valor. La sintaxis de un Process se muestra a continuación.

```
PROCESS [lista de sensibilidad]
        [declaración de variables]
      BEGIN
        [sentencias secuenciales]
      END PROCESS;
```

Detección de Flanco de Subida y Flanco de Bajada.

Para detectar un flanco de subida se utiliza el siguiente código.

```
if(clk='1' and clk'EVENT) then
-- //Codigo que se desea que se ejecute cuando se detecte flanco de subida
end if;
```

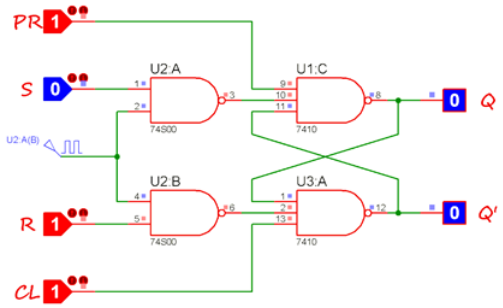
Para detectar un flaco de bajada se utiliza el siguiente código.

```
if(clk='0' and clk'EVENT) then
-- //Codigo que se desea que se ejecute cuando se detecte flanco de bajada
end if;
```

Ejemplos

Diseño, de un flip – flop S – R síncrono activado con señal de reloj CLK en estado alto con PRESET Y CLEAR, por medio de compuertas nand.

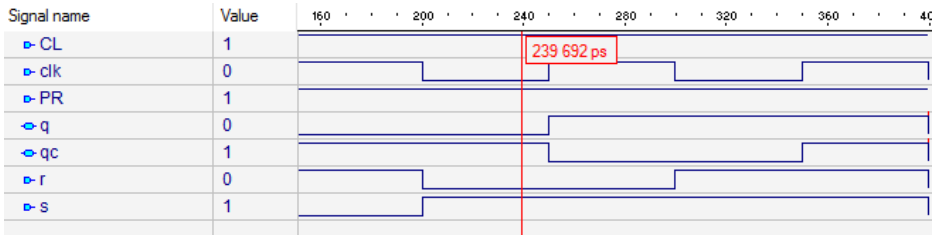
La lógica implementada es la siguiente



```
-- Declaración de Librerias Necesarias
library IEEE;
use IEEE.std_logic_1164.all;
// En entity se declara las señales que van a ser usadas como entradas y salidas
entity codigosr is
  port(
        CL : in STD_LOGIC;
        PR : in STD_LOGIC;
        clk : in STD_LOGIC;
        r : in STD_LOGIC;
        s : in STD_LOGIC;
        q : inout STD_LOGIC := '0'; -- Por medio de un valor por defecto
        qc : inout STD_LOGIC := '1' -- se establece que el Flip Flop comienza en 0
        // q y qc se declara como inout dado que van a ser usadas tanto como entradas
        //como salidas
  );
end codigosr;

architecture codigosr of codigosr is
---- Declaración de Señales ----
signal NET24 : STD_LOGIC;
signal NET28 : STD_LOGIC;
begin
// En esta sección se da la logica de un flip Flop SR en base a compuertas NAND
NET28 <= not(clk and s);
NET24 <= not(r and clk);
q <= not(qc and NET28 and PR);
qc <= not(CL and NET24 and q);
end codigosr;
```

A continuación, se muestra el resultado del código del Flip Flop SR



Flip FLOP JK

También se puede construir el código de Flip Flops básicos en base a su tabla mediante la utilización de un proceso, implementando su tabla de verdad.

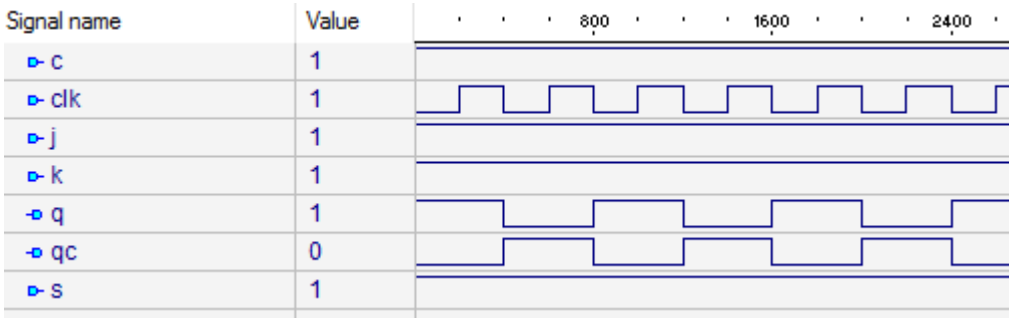
CK	J	K	Q<t+1>	$\bar{Q}$<t+1>
0	X	X	Q<t>	$\bar{Q}$<t>
⊓	0	0	Q<t>	$\bar{Q}$<t>
⊓	0	1	0	1
⊓	1	0	1	0
⊓	1	1	$\bar{Q}$<t>	Q<t>

```
-- Declaración de librerias necesarias
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- Declaración de señales de entrada y salida
entity jk is
     port(
          j : in STD_LOGIC;
          k : in STD_LOGIC;
          clk : in STD_LOGIC;
          c : in STD_LOGIC;
          s : in STD_LOGIC;
          q : out STD_LOGIC;
          qc : out STD_LOGIC
          );
end jk;

architecture jk of jk is
begin
-- El proceso se inicia cada vez que cambia clk
PROCESS(clk)
-- TMP variable temporal
-- que va a permitir cargar q y qc en cada ciclo del proceso
variable TMP: std_logic := '0';
begin
-- Si la entrada c (clear) es 0 q es 0
if (c='0') then
TMP:='0' ;
-- Si la entrada s (set) es 1 q es 1
elsif ( s='0' ) then
TMP :='1' ;
else
-- Se detecta un flanco de bajada
if(clk='0' and clk'EVENT) then
-- Se implementa la tabla segun el comportamiento deseado
if(j='0' and k='0')then
TMP:=TMP;
elsif(j='1' and k='1')then
TMP:= not TMP;
elsif(j='0' and k='1')then
TMP:='0';

else
TMP:='1';
end if;
end if;
end if;
q<=  TMP;
qc<= not TMP;
end PROCESS;
end jk;
```

A continuación, se muestra el resultado del código del Flip Flop JK, usando JK el valor de 1 lógico.



Contador MODULO 16 Asincrónico utilizando Components

```
library IEEE;
use IEEE.std_logic_1164.all;
-- Declaración de Variables de Entrada y Salida
entity contador is
  port(
       clk : in STD_LOGIC;
       q0 : inout STD_LOGIC;
       q1 : inout STD_LOGIC;
       q2 : inout STD_LOGIC;
       q3 : inout STD_LOGIC
  );
end contador;

architecture contador of contador is
---- Declaración del Componente Flip Flop JK -----
component jk
  port (
       c : in STD_LOGIC;
       clk : in STD_LOGIC;
       j : in STD_LOGIC;
       k : in STD_LOGIC;
       s : in STD_LOGIC;
       q : out STD_LOGIC;
       qc : out STD_LOGIC
  );
end component;
---- Declaración de la Constante VCC     -----
constant VCC_CONSTANT    : STD_LOGIC := '1';
---- Declaración de Señales ----
signal NET248 : STD_LOGIC :='1';
signal VCC : STD_LOGIC;
signal valor : STD_LOGIC_VECTOR(3 downto 0);
```

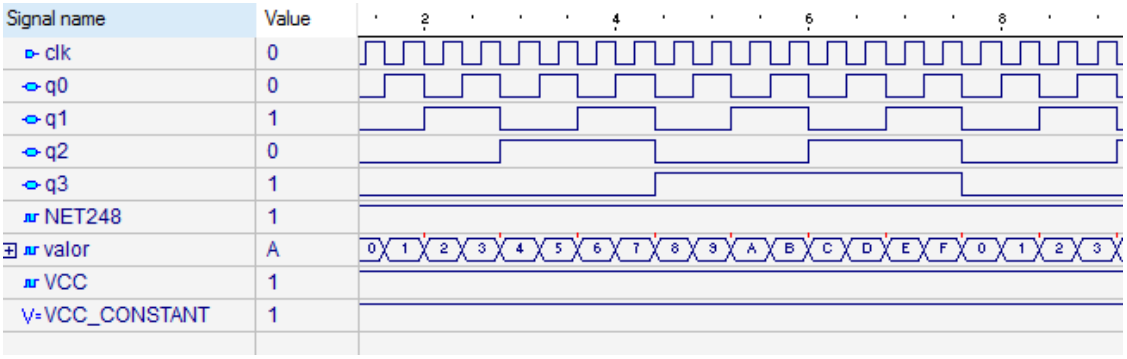
```
---- Se instancian Cuatro Veces el Flip FLop JK ----
--- Dado que un contador modulo 16 es de 4 vits  ---
VCC <= VCC_CONSTANT;
U1 : jk
  port map(
        c => NET248,
        clk => clk,
        j => VCC,
        k => VCC,
        q => q0,
        s => VCC
  );

U2 : jk
  port map(
        c => NET248,
        clk => q0,
        j => VCC,
        k => VCC,
        q => q1,
        s => VCC
  );

U3 : jk
  port map(
        c => NET248,
        clk => q1,
        j => VCC,
        k => VCC,
        q => q2,
        s => VCC
  );

U4 : jk
  port map(
        c => NET248,
        clk => q2,
        j => VCC,
        k => VCC,
        q => q3,
        s => VCC
  );
---- Calga del valor del contador en la señal valor ----
valor <= q3&q2&q1&q0;
end contador;
```

A continuación, se muestra el resultado del código del Contador Modulo 16 Asincrónico.

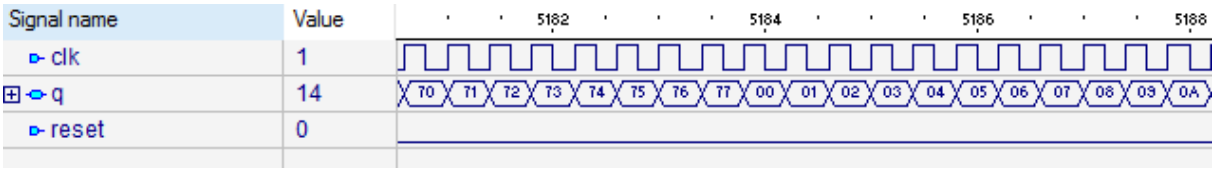


Contador MODULO 120

Para realizar un contador módulo 120 por detección de flanco y por medio de sumas en cada flanco tenemos que identificar el 120 en binario 1111000 entonces el contador va ir de 0 a 119 y 119 en binario es 1110111.

```
--Declaración de Librerias Necesarias
library IEEE;
use IEEE.STD_LOGIC_1164.all;
--Esta libreria nos permite usar sumas
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Declaración de Señales de entrada y Salida
entity contador120 is
     port(
     clk : in STD_LOGIC;
     reset : in  STD_LOGIC;
     q: inout STD_LOGIC_VECTOR(6 DOWNTO 0) :="0000000"
         );
end contador120;
architecture contador120 of contador120 is
begin
    -- Creamos un proceso contador
    -- que se lansara cada vez que cambie el valor del reset o del reloj
    contador: process ( reset, clk) begin
         -- Indicamos que si se activa la señal de reset el contador se
         --carga en 0
        if reset = '1' then
            q <= "0000000";
        -- identificamos un flanco de subida
        --  rising_edge(clk) es equivalente a (clk='1' and clk'EVENT)
        elsif rising_edge(clk) then
            -- identificamos el 119 para que del 119 regrese a 0
            if q = "1110111" then
                q <= "0000000";
            else
                -- se aumenta el valor del contador en cada ciclo de relog
                q <= q + 1;
            end if;
        end if;
    end process;
end contador120;
```

A continuación, se muestra el resultado del código del Contador Modulo 16 Asincrónico con Reset.



Para la simulación se recomienda ver el siguiente video, solo tomar en cuenta que la versión actual tiene unas pequeñas diferencias.
https://www.youtube.com/watch?v=XFDfopQFM9s

También es posible a partir de diagramas bde generar código y viceversa para lo cual se recomienda ver el siguiente video.
https://www.youtube.com/watch?v=5dpwBTZfRAM