Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Universidad Autónoma de Baja California
Facultad de Ciencias Químicas e Ingeniería

CIRCUITOS DIGITALES AVANZADOS

Practica 7
Dispositivos Programables

Docente: Lara Camacho Evangelina

Alumnos:
Gómez Cárdenas Emmanuel Alberto **1261509**
León Romero Pablo Constantino **1253171**

Contenido

OBJETIVO
Diseñar circuitos secuenciales en dispositivos programables FPGA.

EQUIPO
Computadora con el IDE Xilinx Vivado u otro software para desarrollo de código para FPGAs.

FUNDAMENTO TEORICO

Flujo de datos
Cuando se desarrollan programas en VHDL u otro lenguaje de descripción de hardware, hay que recordar que la ejecución no va a ser igual a la de un programa de computadora. El código en VHDL **describe hardware**, no una serie de instrucciones de un procesador. Un circuito electrónico puede contener elementos que ejecutan acciones al mismo tiempo, en paralelo. Por ejemplo, en el medio sumador de la Fig. 1, las entradas A y B se aplican a dos compuertas lógicas, XOR y AND, y cada una obtiene una salida. Estas dos acciones son ejecutadas en paralelo y es a lo que se le llama **concurrencia**.

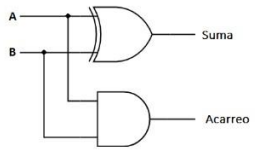


Figura 1. Medio sumador.

El lenguaje VHDL es concurrente, por lo tanto, no ejecuta las instrucciones en el orden en que están escritas. Habiendo dos o más instrucciones, no las ejecuta una tras la otra, sino que éstas pueden ejecutarse a la vez.

La instrucción básica en la ejecución concurrente es la asignación de señales por medio del símbolo <=, tal como se vio en la práctica anterior con el medio sumador:

```
suma    <= a xor b;
acarreo <= a and b;
```

Existen además otras estructuras de alto nivel que facilitan la asignación de señales con base al cumplimiento de ciertas condiciones, estas son **WHEN - ELSE** y **WITH - SELECT - WHEN**.

Estructura WHEN - ELSE

Sentencia de selección múltiple. Su sintaxis es:

```
<señal> <= <asignación_1> WHEN <condición_1> ELSE
           <asignación_2> WHEN <condición_2> ELSE
                          ...
           <asignación_n> WHEN <condición_n> ELSE
           <asignación_m>;
```

Un ejemplo de uso:

```
z <= "00" WHEN a = b ELSE
     "10" WHEN a > b ELSE
     "11";
```

Siempre tiene que haber una asignación de valor, por lo tanto, cuando una condición no va a generar cambios en la señal, se usa la palabra reservada **UNAFFECTED**.

```
z <= a WHEN b = '1' ELSE UNAFFECTED;
```

Estructura WITH - SELECT - WHEN

Es similar a la estructura switch de los lenguajes de programación para computadoras.

Realiza la asignación en base al estado de una señal. Su sintaxis es:

```
WITH <señal_1> SELECT
     <señal_2> <= <asignación_1> WHEN <estado_señal1>,
                  <asignación_2> WHEN <estado_señal2>,
                   ...
                  <asignación_n> WHEN OTHERS;
```

Un ejemplo de uso:

```
WITH x SELECT
            z <= a WHEN "00",
                 b WHEN "01",
                 c WHEN "10",
                 d WHEN OTHERS;
```

En la estructura **WITH - SELECT - WHEN**, todos los estados posibles de la señal de control

(señal_1 en la descripción) deben ser incluidos en el conjunto de opciones y no deben repetirse. A diferencia, en la estructura **WHEN_ELSE** no es forzoso incluir

todos los posibles estados, incluso puede operar con expresiones con diferentes argumentos, como se ve en el ejemplo:

```
z <= '0' WHEN reset = '0' ELSE
      d  WHEN clk = '1';
```

Estructura PROCESS

Esta estructura es concurrente respecto a que todos los PROCESS y otras sentencias concurrentes en el código se ejecutan a la vez. Sin embargo, las instrucciones que están adentro del PROCESS se ejecutan **secuencialmente**, una detrás de otra, siguiendo el orden en el que están escritas en el código. Esta estructura permite describir un circuito en base a su **comportamiento**. Su sintaxis es:

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

La **lista de sensibilidad** indica todas las señales que provocan que se ejecute el proceso, esto cuando por lo menos una de ellas cambia de valor. Por ejemplo, un proceso declarado como `process(clk,rst)` se ejecuta cada vez que las señales `clk` o `rst` cambian de estado. Todas las señales referenciadas en un proceso deberían estar en la lista de sensibilidad.

Ejemplo de uso donde se describe un flip flop D:

```
entity DFlipFlop is
port (d,clk:   in std_logic;
      pre,clr: in std_logic;
      q:       out std_logic);
end DFlipFlop;

architecture behave of DFlipFlop is
begin
   process(clk,pre,clr)
   begin
     if (clr='0') then           -- clr asíncrono tiene mayor precedencia.
      q <= '0';
     elsif(pre='0')then          -- pre asíncrono tiene la siguiente precedencia.
      q <= '1';
     elsif rising_edge(clk) then -- operación síncrona, ocurre si clr=pre='1'
      q <= d;                    -- y hay un flanco ascendente en el clk.
      end if;
end process;
end;
```

En VHDL, variables y señales son elementos distintos. Las señales se tienen que declarar entre la sentencia ARCHITECTURE y su correspondiente BEGIN. Las variables se declaran entre PROCESS y su BEGIN. Otra diferencia es que las **señales actualizan su valor solo hasta que termina la presente ejecución del proceso**, mientras que las variables toman su nuevo valor inmediatamente. Dentro de un proceso se pueden usar tanto señales como variables.

Las variables se usan generalmente para retener los resultados inmediatos en la implementación de un algoritmo, de la siguiente manera:

1. Se asigna el estado de una señal a una variable, la señal es un dato de entrada al algoritmo. Se usa el operador **:=** en la asignación a la variable.
2. Se ejecuta el algoritmo (proceso).
3. El resultado que quedó en la variable se copia a una señal, esta señal es entonces un dato de salida del algoritmo.

No se puede acceder a los valores de las variables fuera de su proceso. Una señal si puede ser accedida en uno o más procesos, pero **no más de un proceso puede modificar su estado**. En la Fig. 2 se muestra el uso que generalmente se da a señales y variables dentro de los procesos.

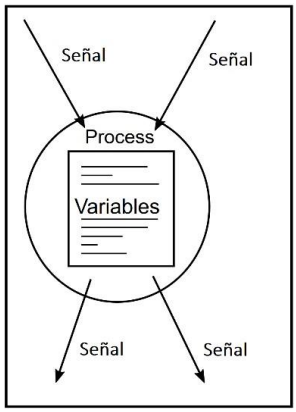


Figura 2. Uso de señales y variables en un proceso.

Ejemplo:

```
ENTITY SenalesYVariables IS
PORT (c: IN std_logic);
END ENTITY;

ARCHITECTURE behave OF SenalesYVariables IS
  SIGNAL a,b: std_logic;
BEGIN
```

```
  PROCESS(c)
    VARIABLE z: std_logic;    BEGIN      a <= c
and b; -- "a" actualiza su valor solo hasta que
termina                   -- la ejecución actual
del proceso.
    z := a or c;  -- "z" se actualiza inmediatamente.
Recuerde que "a"                      -- no ha cambiado
en este momento.
  END PROCESS;
END ARCHITECTURE;
```

Sentencias de selección secuenciales

Estructura IF - THEN - ELSE

Permiten seleccionar el código a ejecutar en base a una o más condiciones. Es una estructura secuencial. Su sintaxis es:

```
IF <condición_1> THEN
 [sentencias]
ELSIF <condición_2> THEN
 [sentencias]
ELSE
 [sentencias]
END IF;
```

Un ejemplo de uso:

```
if (s = "00") then
   z <= a;
elsif (s = "11") then
   z <= b;
else
   z <= c;
end if;
```

Puede haber más de un `elsif`. Las cláusulas `elsif` y `else` son opcionales, pero si la estructura if-then-else no está completamente especificada (no tiene else), esto implica que se va a implementar con un elemento de memoria (no un multiplexor, por ejemplo) o incluso la implementación podría no ser la adecuada al diseño (por ejemplo, el compilador podría colocar una señal directamente a tierra/vcc).

Estructura WHEN - CASE

Permite evaluar una expresión para seleccionar el código a ejecutarse. Se tienen que tomar en cuenta todos los casos, para esto se puede colocar como última opción la sentencia **WHEN OTHERS**. Su sintaxis es:

```
CASE <expresión> IS
  WHEN <valor_1> => [sentencias]
  WHEN <valor_2> => [sentencias]
  WHEN <rango_de_valores> => [sentencias]
  WHEN OTHERS => [sentencias]
END CASE;
```

Un ejemplo de uso:

```
Case s is
  when “00” =>
    z <= a;
  when “11” =>
    z <= b;
  when others =>
    z <= c;
end case;
```

Estructura LOOP

Se usa para crear bucles. En VHDL, existe el bucle **FOR** y **WHILE**. Su sintaxis es:

```
[etiqueta:] [WHILE <condición> | FOR <condición>] LOOP
[sentencias]
[exit;]
[next;]
END LOOP [etiqueta];
```

Ejemplo de uso de **FOR LOOP**:

```
inicio: for k in N-1 downto 0 loop
   Q(k) <= '0';
end loop inicio;
```

La variable `k` es implícita en el `FOR LOOP`, no necesita declararse.

Ejemplo de uso de **WHILE LOOP**:

```
inicio: while (k > 0) loop
   Q(k) <= '0'
   k := k - 1;
end loop inicio;
```

La variable `k` debe declararse como una variable de proceso, entre PROCESS y BEGIN. En este ejemplo, su formato podría ser `variable k: integer := N-1;`

Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Desarrollo

Parte 1. Contador

Codigo ContadorModuloM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ContadorModuloM is
    generic (
            M : integer := 5; -- Conteo de 0 a M-1.
            N : integer := 3  -- Numero de bits requeridos para el conteo
    );
    port (
            clk, rst : in std_logic;
            tc       : out std_logic; -- Ciclo de conteo terminado.
            conteo   : out std_logic_vector(N-1 downto 0)
    );
end ContadorModuloM;
architecture Behavioral of ContadorModuloM is
    signal cnt, cnt_siguiente : unsigned(N-1 downto 0);
begin
    process(clk, rst)
    begin
        if rst = '1' then
            cnt <= (others=>'0');
        elsif rising_edge(clk) then
            cnt <= cnt_siguiente;
        else
            cnt <= cnt;
        end if;
    end process;

    -- Si el conteo actual es M-1, el siguiente debe ser 0. De otra manera, incrementar
    -- el conteo actual.
    cnt_siguiente <= (others=>'0') when cnt=(M-1) else (cnt+1);

    -- Activar Tc cuando se llega al conteo maximo.
    tc <= '1' when cnt = (M-1) else '0';

    conteo <= std_logic_vector(cnt); -- Conteo actual en el puerto de salida.
end Behavioral;
```

Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Codigo ContadorModuloM_tb

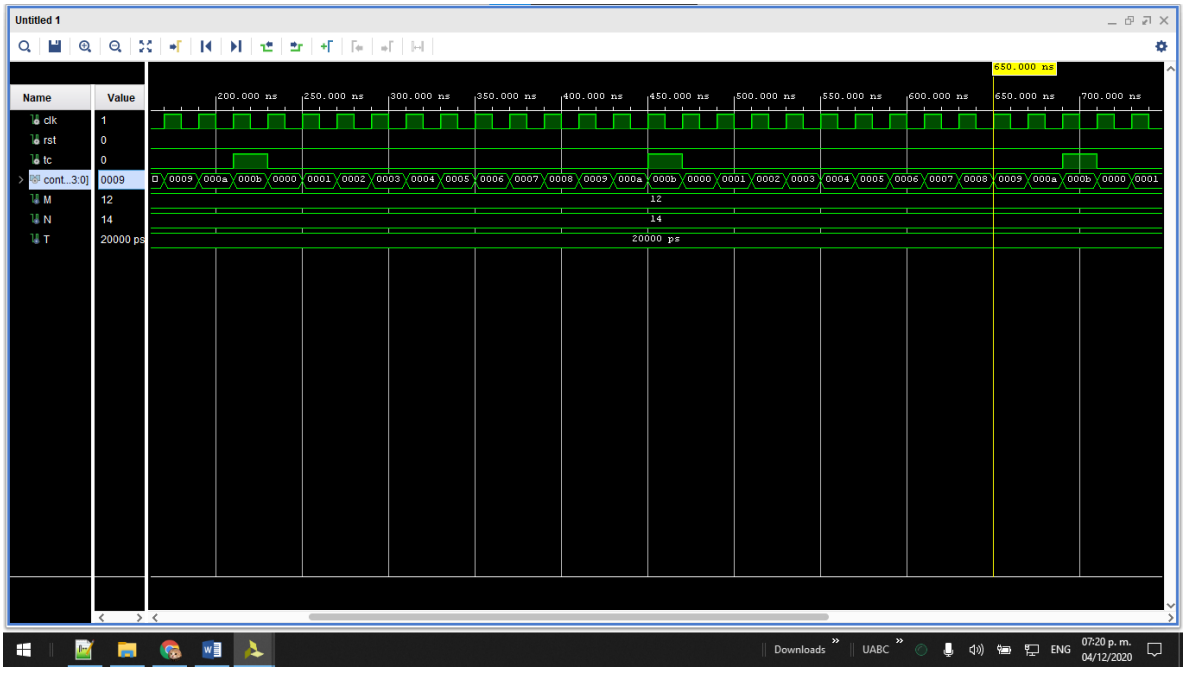
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ContadorModuloM_tb is
end ContadorModuloM_tb;

architecture Behavioral of ContadorModuloM_tb is
    constant M : integer := 12;     --Modulo 12(0 -11)
    constant N : integer := 14;
    constant T: time     := 20 ns;  --Periodo

    signal clk, rst : std_logic;
    signal tc : std_logic;
    signal conteo : std_logic_vector(N-1 downto 0);
begin
    uut : entity work.ContadorModuloM -- "Instancion por Entidad" en lugar de "Instacion por Componente",
        generic map (                 -- la palabra "work" se refiere a la biblioteca o directorio de trabajo actual,
            M => M,                   -- usando "entity work,ContadorModuloM" no es necesario agregar el componente
            N => N                    -- (component ContadorModuloM), a diferencia de como se hizo en "medio_sumador_tb.vhd".
            )
       port map (
            clk    => clk,
            rst    => rst,
            tc     => tc,
            conteo => conteo
            );
process
begin
            clk <= '0';
            wait for T/2;
            clk <= '1';
            wait for T/2; -- Ciclo de trabajo de 50%.
        end process;
        -- rst activo solo en el primer ciclo de reloj.
        rst <= '1', '0' after T/2;
end Behavioral;
```

Simulación ContadorModuloM



Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Parte 2. Registro de desplazamiento

Código RegistroDesplazamiento

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity RegistroDesplazamiento is
5      generic (
6              N : integer := 4
7      );
8      port(
9              rst, clk, load, d : in std_logic;
10             dout      : out std_logic_vector(N-1 downto 0);
11             din       : in std_logic_vector(N-1 downto 0)
12     );
13 end RegistroDesplazamiento;
14
15 architecture Behavioral of RegistroDesplazamiento is
16     signal registro : std_logic_vector(N-1 downto 0);
17 begin
18     process(clk, rst)
19     begin
20         if rst = '1' then
21             registro <= (others=>'0');
22         elsif rising_edge(clk) then
23             if load = '1' then
24                 registro <= din;
25             else
26                 registro(N-2 downto 0) <= registro(N-1 downto 1);
27                 registro(N-1) <= d;
28             end if;
29         end if;
30         dout<=registro;
31     end process;
32
33 end Behavioral;
34
```

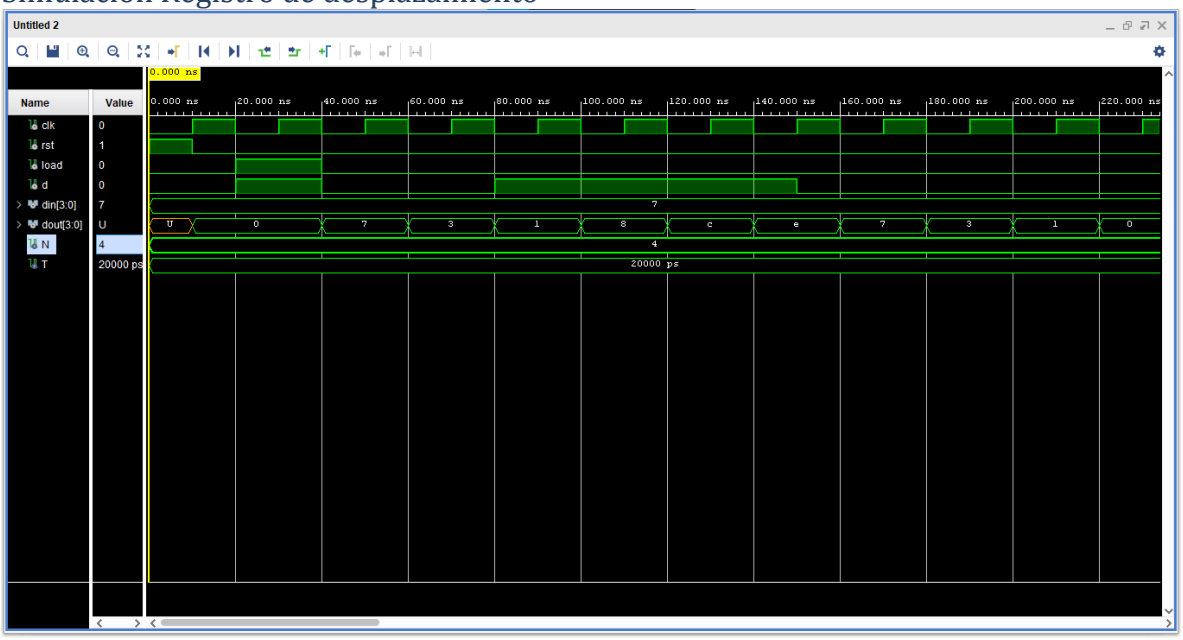
Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Código RegistroDesplazamiento_tb

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity RegistroDesplazamiento_tb is
4  end RegistroDesplazamiento_tb;
5  architecture Behavioral of RegistroDesplazamiento_tb is
6      constant N: integer := 4;
7      constant T: time := 20ns;
8
9      signal clk, rst, load, d: std_logic ;
10     signal din: std_logic_vector(N-1 downto 0);
11     signal dout: std_logic_vector(N-1 downto 0);
12
13 begin
14     uut: entity work.RegistroDesplazamiento
15     generic map(
16         N=>N
17     )
18     port map(
19         clk => clk,
20         rst => rst,
21         load => load,
22         din => din,
23         dout => dout,
24         d => d
25     );
26     process
27     begin
28         clk <= '0';
29         wait for T/2;
30         clk <= '1';
31         wait for T/2;
32     end process;
33     din <= "0111";
34     rst <= '1','0' after T/2;
35     load <= '0','1' after T, '0' after 2*T;
36     d<='0','1' after T, '0' after 2*T,'1' after 4*T,'0' after 150ns;
37 end Behavioral;
```

Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Simulación Registro de desplazamiento



Parte 3. Sumador en serie

Codigo SumadorSerie

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   -- Uncomment the following library declaration if using
6   -- arithmetic functions with Signed or Unsigned values
7   --use IEEE.NUMERIC_STD.ALL;
8
9   -- Uncomment the following library declaration if instantiating
10  -- any Xilinx leaf cells in this code.
11  --library UNISIM;
12  --use UNISIM.VComponents.all;
13
14  -- SumadorSerie.vhd
15  entity SumadorSerie is
16      port (
17          a, b, cin       : in std_logic;
18          s, cout         : out std_logic
19      );
20  end SumadorSerie;
21
22
23  architecture Behavioral of SumadorSerie is
24  begin
25      s <= (a xor b) xor (cin);
26      cout <= (a and b) or (cin and (a xor b));
27  end Behavioral;
```

Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Codigo SumadorSerie_tb

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4   entity SumadorSerie_tb is
5   end SumadorSerie_tb;
6   architecture Behavioral of SumadorSerie_tb is
7       constant N : integer := 5;
8       constant T : time    := 100ns;
9       signal clk, rst, load, a, b, c, cin, cout   : std_logic;
10      signal numA, numB, numC                     : std_logic_vector(N-1 downto 0);
11  begin
12      FlipFlopD : entity work.FlipFlopD
13          port map(
14              clk => clk,
15              rst => rst,
16              d   => cout,
17              q   => cin );
18      RegistroA : entity work.RegistroDesplazamiento
19          generic map( N => N )
20          port map(
21              clk     => clk,
22              rst     => rst,
23              load    => load,
24              din     => numA,
25              dout    => a,
26              d       => '1'        );
27      RegistroB : entity work.RegistroDesplazamiento
28          generic map(N => N )
29          port map(
30              clk     => clk,
31              rst     => rst,
32              load    => load,
33              din     => numB,
34              dout    => b,
35              d       => '1' );
36      RegistroC : entity work.RegistroDesplazamiento
37          generic map( N => N )
```

```
37          generic map( N => N )
38          port map(
39              clk     => clk,
40              rst     => rst,
41              load    => load,
42              din     => numC,
43              d       => c,
44              numOut  => numC );
45       SumadorSerie : entity work.SumadorSerie
46          port map(
47              a       => a,
48              b       => b,
49              s       => c,
50              cin     => cin,
51              cout    => cout);
52      numA    <= "01000" after T/2;
53      numB    <= "00011" after T/2;
54      rst     <= '1' after T/2, '0' after T;
55      load    <= '1' after T/2, '0' after 2*T;
56      process
57      begin
58          clk <= '0';
59          wait for T/2;
60          clk <= '1';
61          wait for T/2;
62      end process;
63  end Behavioral;
```

Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Registro de desplazamiento

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  entity RegistroDesplazamiento is
5      generic(
6          N    : integer := 4
7      );
8      port (
9          rst, clk, load, d   : in std_logic;
10         din                 : in std_logic_vector(N-1 downto 0);
11         dout                : out std_logic;
12         numOut              : out std_logic_vector(N-1 downto 0)
13     );
14 end RegistroDesplazamiento;
15
16
17 architecture Behavioral of RegistroDesplazamiento is
18     signal registro : std_logic_vector(N-1 downto 0);
19 begin
20     process(clk, rst)
21     begin
22         if rst = '1' then
23             registro <= (others=>'0');
24         elsif rising_edge(clk) then
25             if load = '1' then
26                 registro <= din;
27             else
28                 registro(N-1 downto 1) <= registro(N-2 downto 0);
29                 registro(0) <= d;
30             end if;
31         end if;
32         numOut <= registro;
33         dout <= registro(N-1);
34     end process;
35 end Behavioral;
```

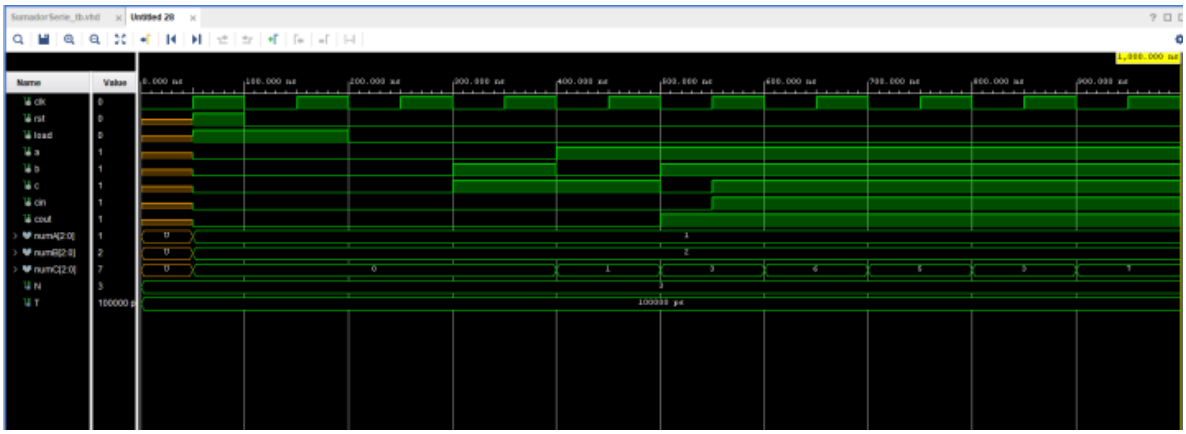
Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

Flip flop tipo D

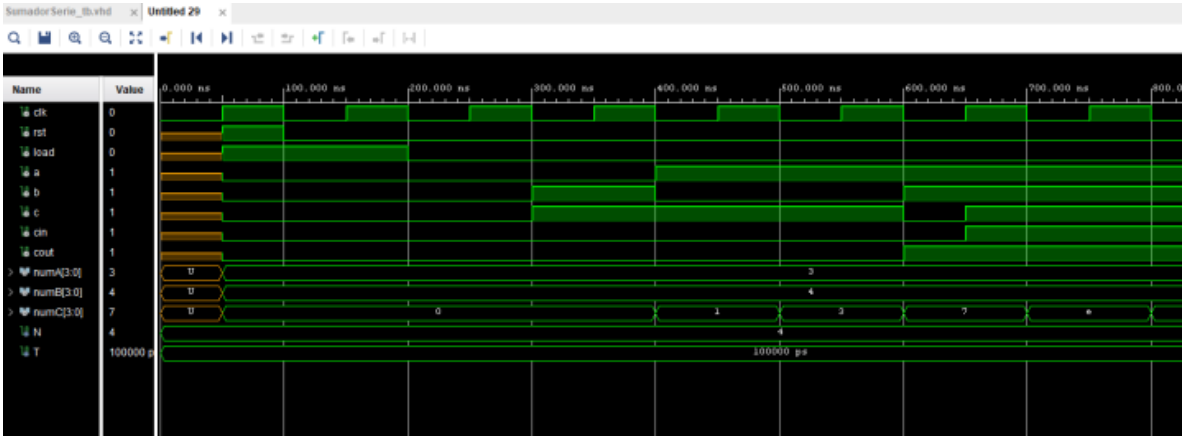
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  -- Uncomment the following library declaration if using
6  -- arithmetic functions with Signed or Unsigned values
7  --use IEEE.NUMERIC_STD.ALL;
8
9  -- Uncomment the following library declaration if instantiating
10 -- any Xilinx leaf cells in this code.
11 --library UNISIM;
12 --use UNISIM.VComponents.all;
13
14 -- FlipFlopD.vhd
15 entity FlipFlopD is
16     port (
17         d, clk, rst : in std_logic;
18         q           : out std_logic
19     );
20 end FlipFlopD;
21
22 architecture Behavioral of FlipFlopD is
23 begin
24     process(clk, rst)
25     begin
26         if rst = '1' then
27             q <= '0';
28         elsif rising_edge(clk) then
29             q <= d;
30         end if;
31     end process;
32 end Behavioral;
```

Gómez Cárdenas Emmanuel Alberto
León Romero Pablo Constantino

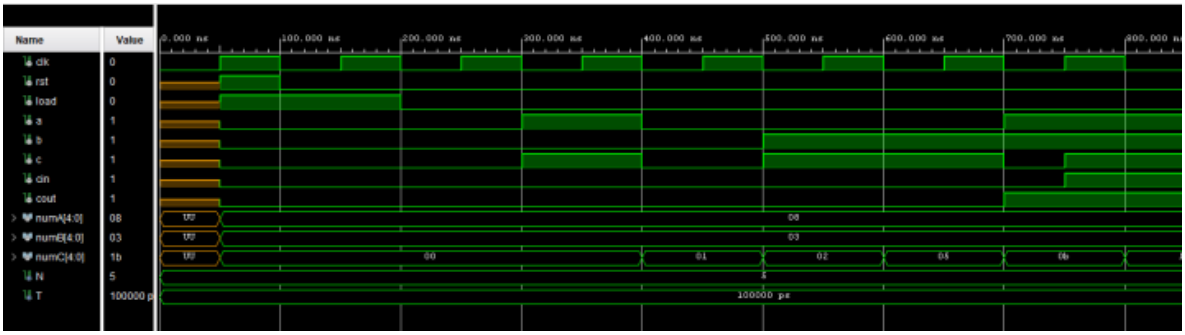
Simulación SumadorSerie 3 bits



Simulación SumadorSerie 4 bits



Simulación SumadorSerie 5 bits



CONCLUSIONES

Gómez Cárdenas Emmanuel Alberto:

En esta práctica nos pudimos familiarizar un poco más con el manejo del programa Vivado, ya que sigue siendo un poco complejo. Gracias al programa pudimos diseñar y simular circuitos con un nivel de complejidad aún más alto que los anteriores como lo es el registro de desplazamiento o el sumador.

Pablo Constantino León romero:

Esta práctica fue desafiante dado que no hemos tenido demasiada teoría en programación vhdl. Aun así, creo que se pudo lograr un buen resultado y se aprendieron bastantes cosas. Se entendí mejor el manejo de las diversas partes del código como, la entity, arquitectura, proceso, etc. Además, se vieron nuevas formas de implementar circuitos ya conocidos por de antes.

VIDEO DE PRACTICA

https://drive.google.com/drive/folders/1iFYMaCSydAgiQBsk1RVU_BWEk5s5KEQV?usp=sharing