# Práctica 7

**Circuitos secuenciales en Dispositivos Programables**

## 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 teórico

### 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**.

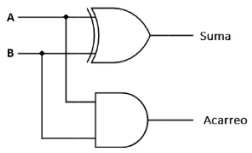


Figura 1. Medio sumador.

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.

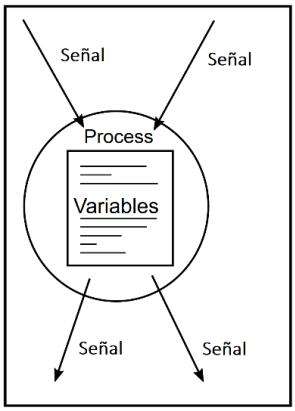


Figura 2. Uso de señales y variables en un proceso.

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;`

## Desarrollo

1. Revise el video sobre circuitos secuenciales en VHDL en: https://youtu.be/U3tXM5NTSrs
2. Cree en Vivado un nuevo proyecto llamado **Contador**, agregue un archivo de código fuente VHDL llamado **ContadorModuloM** y copie el código del Listado 1. En el código se describe un contador módulo M, donde M es configurable. El código incluye estructuras de ejecución secuenciales y concurrentes.
3. Cree el test bench **ContadorModuloM_tb** con el código del Listado 2 y simúlelo para diferentes valores de M.
4. Cree en Vivado un nuevo proyecto llamado **RegistroDesplazamiento** donde describa el registro de desplazamiento de la Fig. 3. Cree su test bench **RegistroDesplazamiento_tb** donde se pruebe el desplazamiento de dos números de 4 bits de su elección. Simule el código.

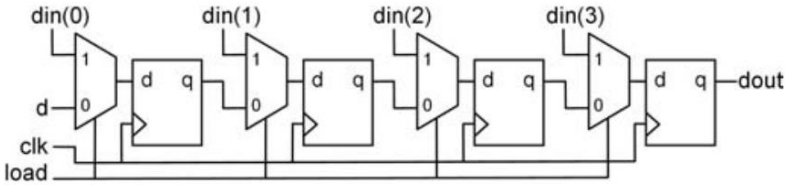


Figura 3. Registro de desplazamiento.

5. Realice un video donde describa el diagrama de tiempos de la simulación del registro de desplazamiento. Describa claramente en el diagrama los estados que toman las entradas y las salidas que producen. Indique con el cursor los tiempos en los que cambian de estado las entradas y salidas.
6. Cree en Vivado un nuevo proyecto llamado **SumadorSerie** donde describa el sumador en serie de la Fig. 4. Cree su test bench **SumadorSerie _tb** donde se pruebe la suma de dos números de: 3 bits, 4 bits y 5 bits. Los números y acarreos de entrada iniciales son a su elección. Simule el código.

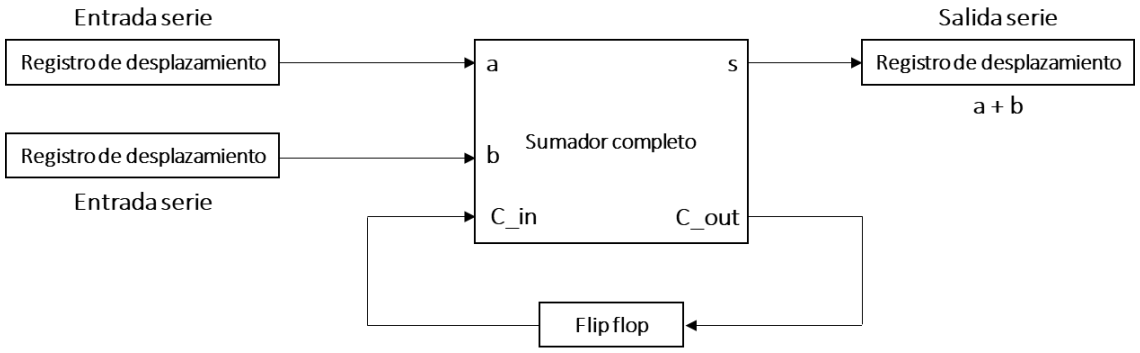


Figura 4. Sumador en serie.

7. Realice un video donde describa el diagrama de tiempos de la simulación del sumador serie. Describa claramente en el diagrama los estados que toman las entradas y las salidas que producen. Indique con el cursor los tiempos en los que cambian de estado las entradas y salidas.
8. Entregue su reporte y proyectos de Vivado (u otra herramienta de desarrollo en VHDL para FPGAs) por Moodle, los videos envíelos por medio de un enlace al drive de su cuenta universitaria.

## Conclusiones y comentarios
## Dificultades en el desarrollo
## Referencias

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- ContadorModuloM.vhd
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;
```

Listado 1. Contador Módulo M.

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- ContadorModuloM_tb.vhd


entity ContadorModuloM_tb is
end ContadorModuloM_tb;


architecture Behavioral of ContadorModuloM_tb is
    constant M : integer := 12;    -- Modulo 12 (0 - 11)
    constant N : integer := 4;
    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 -- "Instanciación por Entidad" en lugar de "Instanciación 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;
```

Listado 2. Test bench del Contador Módulo M.