## Ejercicios Combinacionales.

## Ejercicio 1.

Describir y simular un circuito de lógica combinacional que cumpla con lo siguiente:

```
entity MyVoter is
    Port ( piX : in STD_LOGIC;
           piY : in STD_LOGIC;
           piZ : in STD_LOGIC;
           poResult : out STD_LOGIC
         );
end MyVoter;
```

| piX piY piZ | poResult |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 0 |
| 0 1 0 | 0 |
| 0 1 1 | 1 |
| 1 0 0 | 0 |
| 1 0 1 | 1 |
| 1 1 0 | 1 |
| 1 1 1 | 1 |

## Ejercicio 2.

Describir y simular un circuito combinacional que tenga como entrada un dígito BCD natural y como salida la parte entera del cociente de su división por tres .

```
entity MyDiv3 is
    Port ( piBCD : in STD_LOGIC_VECTOR(3 downto 0);
           poResult : out STD_LOGIC_VECTOR(2 downto 0)
         );
end MyDiv3;
```

## Ejercicio 3.

Diseñe un circuito combinacional que acepte un número de tres bits y genere un número binario de salida igual al cuadrado del número de entrada.

```
entity MySquare3 is
    Port ( piBits : in STD_LOGIC_VECTOR(2 downto 0);
           poResult : out STD_LOGIC_VECTOR(5 downto 0)
         );
end MySquare3;
```

## Ejercicio 4.

Diseñe un circuito combinacional que acepte un número de tres bits y genere un número binario de salida igual al cuadrado del número de entrada.

```
entity MySquare3 is
    Port ( piBits : in STD_LOGIC_VECTOR(2 downto 0);
           poResult : out STD_LOGIC_VECTOR(5 downto 0)
         );
end MySquare3;
```

## Ejercicio 5.

Describir y simular un circuito de lógica combinacional que cumpla con lo siguiente:

```
entity MyBooleanALU is
    Port ( piData1 : in STD_LOGIC_VECTOR (3 downto 0);
           piData2 : in STD_LOGIC_VECTOR (3 downto 0);
           piOpCode : in STD_LOGIC_VECTOR (2 downto 0);
           poResult : out STD_LOGIC_VECTOR (3 downto 0);
           poZeroFlag : out STD_LOGIC;
           poOneFlag : out STD_LOGIC);
end MyBooleanALU;
```

El sistema debe realizar las siguientes operaciones bit a bit entre los datos 1  y 2

| piOpCode | Operaciones | poZeroFlag | poOneFlag |
|---|---|---|---|
| 000 | and | '1' si poResult 0x0 | '1' si poResult 0xF |
| 001 | nand | '1' si poResult 0x0 | '1' si poResult 0xF |
| 010 | or | '1' si poResult 0x0 | '1' si poResult 0xF |
| 011 | nor | '1' si poResult 0x0 | '1' si poResult 0xF |
| 100 | xor | '1' si poResult 0x0 | '1' si poResult 0xF |
| 101 | xnor | '1' si poResult 0x0 | '1' si poResult 0xF |
| 110 | 0xF | '1' si poResult 0x0 | '1' si poResult 0xF |

| | | | |
|---|---|---|---|
| 111 | 0x0 | '1' si poResult 0x0 | '1' si poResult 0xF |

## Ejercicio 6.

Describir y simular un circuito que muestre en un display de 7 segmentos el valor hexadecimal que representan cuatro pulsadores a la entrada.

```
entity MyHexDecoder is
    Port ( piData : in STD_LOGIC_VECTOR(3 downto 0);
           poHexValue : in STD_LOGIC_VECTOR(6 downto 0)
          );
end MyHexDecoder;
```

## Ejercicios Secuenciales.

## Ejercicio 7.

Describa y simule el siguiente circuito FF-D con señales de control asíncronas Rst, Pst, y Ena sincrónico.

```
entity MyFFD is
    Port ( piClk : in STD_LOGIC;
           piRst : in STD_LOGIC;
           piPst : in STD_LOGIC;
           piEna : in STD_LOGIC;
           piD : in STD_LOGIC;
           poQ : out STD_LOGIC
          );
end MyFFD;
```

## Ejercicio 8.

Describir y simular un circuito contador hexadecimal que muestre en un display de 7 segmentos el valor de la cuenta. Si piRST es ‘1’ la cuenta se resetea, si piMODE es ‘1’ cuenta en forma descendente, si piMODE es ‘0’ cuenta en forma ascendente.

```
entity MyConterHex is
    Port ( piCLK : in STD_LOGIC;
           piENABLED: in STD_LOGIC;
           piRST: in STD _LOGIC;
           piMODE: in STD_LOGIC;
           poHexValue : in STD_LOGIC_VECTOR(6 downto 0)
```

```
        );
end MyCounterHex;
```

## Ejercicio 9.

Describir y simular un contador sincrónico de módulo M, con Rst y Ena sincrónico. Un contador de módulo cuenta hasta el módulo definido (M-1) y luego vuelve al cero. Indicando con un pulso a la salida de un ciclo de reloj cada vez que se llega al módulo.

poTc: Se pone en '1' por un ciclo cada vez que se alcanza M-1.

```
entity MyCounterMod is
   Generic(BIT_WIDTH: NATURAL:=4;
            MODULE: NATURAL:=10
           );

   Port ( piClk : in STD_LOGIC;
          piRst : in STD_LOGIC;
          piEna : in STD_LOGIC;
          poCount : out STD_LOGIC_VECTOR (BIT_WIDTH-1 downto 0);
          poTc : out STD_LOGIC
        );
end MyCounterMod;
```

## Ejercicios FSM (*Finite State Machine*)

## Ejercicio 10.

Describir y simular un circuito sincrónico secuencial con una entrada X y una salida Z, que detecte la llegada de tres ceros o tres unos consecutivos, dando una salida Z = 1 coincidiendo con la aparición del tercer bit.

```
entity MyDetector is
   Port ( piClk : in STD_LOGIC;
          piRst : in STD_LOGIC;
          piEna : in STD_LOGIC;
          piX : int STD_LOGIC;
          poZ : out STD_LOGIC
        );
end MyDetector;
```

## Ejercicio 11.

Describir y simular un circuito sincrónico secuencial con dos entradas ($X_1$ , $X_2$) y dos salidas ($Z_1$ , $Z_2$). Las entradas representan un número binario natural de dos bits, N . Si el valor presente de N es mayor que el valor inmediatamente anterior, entonces, $Z_1 = 1$. Si dicho valor es menor, entonces la salida $Z_2 = 1$ . En cualquier otro caso, $Z_1 = Z_2 = 0$.

```
entity MyCmpBefore is
   Port ( piClk : in STD_LOGIC;
          piRst : in STD_LOGIC;
          piEna : in STD_LOGIC;
          piX : int STD_LOGIC_VECTOR(1 downto 0);
          poZ : out STD_LOGIC_VECTOR(1 downto 0)
        );
end MyCmpBeforer;
```

## Ejercicio 12.

Describir y simular una máquina de estado que detecte la secuencia de entrada “101”. El sistema debe contar con una entrada de configuración que determina si la secuencia a ser detectada puede ser solapada o no. Por ejemplo:

x → 000010011000101001010100 0
ctrl=0 z → 000000000000001000010000 0
ctrl=1 z → 000000000000001000010100 0

```
entity MySecDet is
        Port (   piClk : in STD_LOGIC;
                 piRst : in STD_LOGIC;
                 piCtrl : in STD_LOGIC;
                 piX : in STD_LOGIC;
                 poZ : out STD_LOGIC
                 );
end MySecDet;
```

## Ejercicio 13.

Describir y simular un sistema que represente una máquina de operar con las siguientes especificaciones. En cada clock el sistema lee la entrada piWrite, si dicha entrada está en '1', significa que en el puerto piDataComand esta el dato/comando que indica la operación a realizar, si el valor es CMD_INIT, indica que hay una nueva secuencia de cálculo, si no hay este valor no se hace nada. Si hay un CMD_INIT, en el próximo piWrite = '1', se tiene el primer operando, luego el operador, y por último el segundo operando.

| poLedsState | Estado |
|---|---|
| 0001 | Esperando CMD_INIT |

| | |
|---|---|
| 0010 | Primer Operando |
| 0100 | Operador |
| 1000 | Segundo Operando |

```
entity MyBasicCalculator is
        Port (  piClk : in STD_LOGIC;
                piRst : in STD_LOGIC;
                piWrite : in STD_LOGIC;
                piDataCommand : in STD_LOGIC_VECTOR (8-1 downto 0);
                poRead : out STD_LOGIC;
                poResult : out STD_LOGIC_VECTOR (8-1 downto 0);
                poLedsState : out STD_LOGIC_VECTOR (4-1 downto 0)
                );
end MyBasicCalculator;

constant cCMD_INIT: std_logic_vector(8-1 downto 1) := “00000001” ;
constant cCOD_OP_ADD: std_logic_vector(8-1 downto 1) := “00000010” ;
constant cCOD_OP_SUB: std_logic_vector(8-1 downto 1) := “00000100” ;
constant cCOD_OP_OR: std_logic_vector(8-1 downto 1) := “00001000” ;
```

## Ejercicio 14.

Describir y simular un sistema en donde por una línea X se recibe, bit a bit, un número binario N, empezando por el menos significativo.

El circuito permite generar una única salida Z con el valor Z = 2 x N.

```
entity My2X is
        Generic(        N: NATURAL:=4
                        )

        Port(   piClk : in STD_LOGIC;
                piRst : in STD_LOGIC;
                piEna : in STD_LOGIC;
                piX : in STD_LOGIC;
                poZ : out STD_LOGIC_VECTOR (N-1 downto 0)
                );
end MyCounterMod;
```

## Ejercicio Integrador.