UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

TÉCNICA DIGITALES I

Curso R3001

GUIA TPS VHDL

v3- 2021

Profesor: Ing. Atencio, Jerónimo

Índice

Condiciones de entrega:

- Los archivos generado deben tener como nombre el mismo que la entidad. Por ejemplo, para el ejercicio 1.1 el nombre deberá ser $myAnd2.vhd$ y para el testbench el formato del nombre es el siguiente $myAnd2\_tb.vhd$
- Todos los archivos (también los testbench) .vhd deberán ser subidos al repositorio dentro de una carpeta con el nombre **guiaXX** donde XX corresponde al numero de sección. Por ejemplo para la sección lógica combinacional la carpeta se llamaría **guia01**
- Arme un proyecto por cada sección con el nombre **guiaXX** donde XX corresponde al numero de sección. Por ejemplo para la sección lógica combinacional el proyecto se llamaría **guia01**
- Los ejercicios entregables estan indicados con la leyenda **(Obligatorio)**
- Usar el numero de parte XCA12TCPG238-3
- Todos los $process$ deben ser activos flanco ascendente, salvo indicación contraria.
- Todos los reset deben ser activos nivel alto (Resetea cuando la entrada vale '1'), salvo indicación contraria.

1. Circuitos combinacionales

1. Implemente una compuerta AND de dos entradas.

```
entity myAnd2 is
Port ( a, b : in std_logic;
       y : out std_logic);
end myAnd2;
```

2. Implemente una compuerta AND de cuatro entradas.

```
entity myAnd4 is
Port ( a : in std_logic_vector (3 downto 0);
       y : out std_logic);
end myAnd4;
```

3. Implemente la siguientes funciones lógicas utilizando los operadores lógicos
   - **S** = A $\oplus$ B $\oplus$ CI
   - **CO** = (A.B) + (B.CI) + (A.CI)

```
entity fa is
Port ( a, b, ci : in std_logic;
       s, co : out std_logic);
end fa;
```

4. Implemente un multiplexor de 4 vías de datos.

```
entity myMux4 is
Port ( a : in std_logic_vector (3 downto 0);
       c : in std_logic_vector (1 downto 0);
       y : out std_logic);
end myMux4;
```

5. Implemente la siguientes funciones lógicas utilizando when-else.

A	B	CI	CO	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- **S** = A $\oplus$ B $\oplus$ CI
- **CO** = (A.B) + (B.CI) + (A.CI)

```
entity fullAdder is
Port ( a,b,ci : in std_logic;
       s,co : out std_logic);
end fullAdder;
```

6. Implemente un decodificador de 3a8

```
entity deco3a8 is
Port ( w : in std_logic_vector (2 downto 0);
       y : out std_logic_vector (7 downto 0));
end deco3a8;
```

7. Implemente un decodificador de 3a8 con enable

```
entity deco3a8eEna is
Port ( w : in std_logic_vector (2 downto 0);
       e : in std_logic;
       y : out std_logic_vector (7 downto 0));
end deco3a8Ena;
```

8. Implemente un codificador de 8a3 con prioridad

```
entity coder8a3 is
Port ( w : in std_logic_vector (7 downto 0);
       y : out std_logic_vector (2 downto 0));
end coder8a3;
```

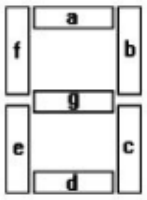
9. **(Obligatorio)**Implemente un codificador de 8a3 con prioridad, enable y salida gs

```
entity coder8a3eg is
Port ( w : in std_logic_vector (7 downto 0);
       e : in std_logic;
       gs : out std_logic
       y : out std_logic_vector (2 downto 0));
end coder8a3eg;
```

10. **(Obligatorio)**Implemente un decodificador BCD a 7 segmentos. Use *with - select*.

```
entity myDeco7Seg is
Port ( entrada : in std_logic_vector (3 downto 0);
       a, b, c, d, e, f, g: out std_logic);
end myDeco7Seg;
```

Entrada	a	b	c	d	e	f	g
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1
1010	1	0	0	1	1	1	1
1011	1	0	0	1	1	1	1
1100	1	0	0	1	1	1	1
1101	1	0	0	1	1	1	1
1110	1	0	0	1	1	1	1
1111	1	0	0	1	1	1	1



11. **(Obligatorio)** Implemente un generador de paridad par de N bits. Use *xor y for - generate*.

```
entity paridadGen is
Generic (N: integer := 4);
Port ( a   : in  std_logic_vector (N - 1 downto 0);
       sel : in  std_logic;
       p   : out std_logic);
end paridadGen;
```

El bit de paridad se calcula de la siguiente forma:

- **Paridad par:** Se cuenta la cantidad de unos, si el total es par el bit de paridad vale 0, en caso contrario vale 1.
- **Paridad impar:** Se cuenta la cantidad de unos, si el total es impar el bit de paridad vale 0, en caso contrario vale 1

Por ejemplo para N = 4: p <= a(3) xor a(2) xor a(1) xor a(0) xor sel;

Dato	Paridad par paridadSel <= '0'	Paridad impar paridadSel <= '1'
0000	0	1
0001	1	0
0101	0	1
1110	1	0

El bit de paridad se utiliza para detectar posibles errores en la transmisión de datos. Este se agrega al dato a transmitir y al ser recibido se puede verificar calculando la paridad del dato y comparándolo con el bit de paridad recibido.

12. **(Obligatorio)**Implemente un conversor gray a binario de N bits. Use *xor y for - generate*

```
entity grayBinario is
Generic (N: integer := 4);
Port ( gray     : in  std_logic_vector (N-1 downto 0);
       binario  : out std_logic_vector (N-1 downto 0));
end grayBinario;
```

Utilice las siguientes expresiones para realizar la conversión:

- $binario(i) = gray(i),\ para\ i = N - 1$
- $binario(i) = binario(i+1)\ xor\ gray(i),\ para\ (N-2) \leq i \leq 0$

La siguiente tabla muestra la conversión para 4 bits:

Binario	Gray		Binario	Gray
0000	0000		1000	1100
0001	0001		1001	1101
0010	0011		1010	1111
0011	0010		1011	1110
0100	0110		1100	1010
0101	0111		1101	1011
0110	0101		1110	1001
0111	0100		1111	1000

13. **(Obligatorio)**Implemente un conversor binario a gray de N bits utilizando for - generate.

```
entity binarioGray is
Generic (N: integer := 4);
Port ( binario : in std_logic_vector (N-1 downto 0);
       gray : out std_logic_vector (N-1 downto 0));
end binarioGray;
```

Utilice las siguientes expresiones para realizar la conversión:

- $gray(i) = binario(i),\ para\ i = N - 1$

- $gray(i) = binario(i+1)\ xor\ binario(i),\ para\ (N-2) \leq i \leq 0$

14. **(Obligatorio)**Utilizando como componente los dos VHDL creados en el punto anterior cree un conversor **binario - gray** cuando **sel = 1** y **gray - binario** cuando **sel = 0** seleccionable por una entrada de control.

```
entity conversorBinarioGray is
Generic (N: integer := 4);
Port ( entrada: in std_logic_vector (N-1 downto 0);
       salida : out std_logic_vector (N-1 downto 0);
       sel : in std_logic);
end conversiorBinarioGray;
```

2. Circuitos aritméticos

1. **(Obligatorio)**Implemente un circuito que permite dividir una magnitud de 8 bits por 1, 2, 4, 8 según se seleccione. (Use when - else)

```
entity myDivMag8 is
Port ( entrada : in std_logic_vector (7 downto 0);
       div : in std_logic_vector (1 downto 0);
       salida : out std_logic_vector (7 downto 0));
end myDivMag8;
```

2. **(Obligatorio)**Implemente un circuito que permite dividir un número signado en convenio CA2 de 8 bits por 1, 2, 4, 8 según se seleccione. (Use when - else)

```
entity myDiv8 is
Port ( entrada : in std_logic_vector (7 downto 0);
       div : in std_logic_vector (1 downto 0);
       salida : out std_logic_vector (7 downto 0));
end myDiv8;
```

3. Implemente un circuito que permite realice la suma/resta de dos números signados en convenio CA2 de N bits.

```
entity sumadorRestador is
Generic (N: integer := 4);
Port ( a : in std_logic_vector (N-1 downto 0);
       b : in std_logic_vector (N-1 downto 0);
       s_r : in std_logic;
       ov : out std_logic;
       resultado: out std_logic_vector (N-1 downto 0));
end sumadorRestador;
```

   - Si s_r vale 0, entonces resultado <= b + a
   - Si s_r vale 1, entonces resultado <= b - a

4. Implemente un comparador de magnitudes con salida mayor, menor e igual. Las salidas son activas nivel alto.

```
entity comparadorMag is
Generic (N: integer := 4);
Port ( a : in std_logic_vector (N-1 downto 0);
       b : in std_logic_vector (N-1 downto 0);
       aMayorQueB : out std_logic;
       aMenorQueB : out std_logic;
       aIgualB    : out std_logic);
end comparadorMag;
```

5. Implemente un comparador de números signados en CA2 con salida mayor, menor e igual. Las salidas son activas nivel alto.

```
entity comparadorCa2 is
Generic (N: integer := 4);
Port ( a : in std_logic_vector (N-1 downto 0);
       b : in std_logic_vector (N-1 downto 0);
       aMayorQueB : out std_logic;
       aMenorQueB : out std_logic;
       aIgualB    : out std_logic);
end comparador;
```

6. **(Obligatorio)**Implemente un comparador que permita seleccionar con una entrada magCa2 si el circuito compara magnitudes (entrada en '0') o números signados en CA2 (entrada en '1'). Las salidas son activas nivel alto.

```
entity comparador is
Generic (N: integer := 4);
Port ( a : in std_logic_vector (N-1 downto 0);
       b : in std_logic_vector (N-1 downto 0);
       magCa2 : in std_logic;
       aMayorQueB : out std_logic;
       aMenorQueB : out std_logic;
       aIgualB    : out std_logic);
end comparador;
```

7. **(Obligatorio)**Implemente un circuito sumador/restador de dos números signados en CA2 de N bits con resultado de N bits. Si la operación produce overflow sature el resultado.

```
entity sumadorRestadorSat is
Generic (N: integer := 4);
Port ( a : in std_logic_vector (N-1 downto 0);
       b : in std_logic_vector (N-1 downto 0);
       s_r : in std_logic;
       ov : out std_logic;
       resultado: out std_logic_vector (N-1 downto 0));
end sumadorRestadorSat;
```

   - Si s_r vale 0, entonces resultado <= b + a
   - Si s_r vale 1, entonces resultado <= b - a

8. **(Obligatorio)**Implemente un circuito que reciba un número signado en CA2 de 16 bits y sature el mismo a 8 bits.

```
entity sat16a8 is
Port ( a : in std_logic_vector (15 downto 0);
       b : out std_logic_vector (7 downto 0));
end sat16a8;
```

9. **(Obligatorio)**Implemente un circuito que realice operaciones aritméticas y lógicas

```
entity miniAlu is
Generic (N: integer := 4);
Port ( a : in std_logic_vector (N-1 downto 0);
       b : in std_logic_vector (N-1 downto 0);
       op : in std_logic_vector (1 downto 0);
       zero : out std_logic;
       ov : out std_logic;
       resultado: out std_logic_vector (N-1 downto 0));
end miniAlu;
```

Donde:

- **a** y **b**: son los operandos
- **r**: es el resultado de las operaciones.
- **ov**: salida que indica si hubo overflow en la operación.
- **cero**: Indica si la salida es cero
- **op**: es la operación a realizar
  - **00**: b + a signado (setea la señal de overflow)
  - **01**: b - a signado (setea la señal de overflow)
  - **10**: a and b bit a bit (coloca overflow en cero)
  - **11**: a or b bit a bit (coloca overflow en cero)

3. Integradores

1. Demuestre las siguientes igualdades:

   a) $x + yz = (x + y)(x + z)$

   b) $(x + y)(x + \bar{y}) = x$

   c) $xy + x\bar{y} = x$

2. Dibuje el circuito más simple como suma de productos y como producto de sumas para las siguientes funciones lógicas.

   a) $f(c, b, a) = \sum(m_0; m_1; m_4; m_7)$

   b) $f(c, b, a) = \sum(m_0; m_2; m_5; m_7)$

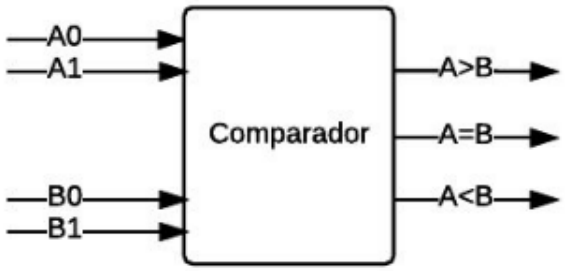
   c) $f(c, b, a) = \prod(m_1; m_2; m_5; m_7)$

   d) $f(d, c, b, a) = \sum(m_0; m_1; m_2; m_3; m_7; m_{12}; m_{13}; m_{15})$

   e) $f(d, c, b, a) = \prod(m_0; m_1; m_5; m_7; m_8; m_{10}; m_{14}; m_{15})$

3. Repita el punto anterior usando sólo compuertas NAND y NOR

4. Implemente un **comparador** de magnitudes de 2 bits. Siendo A1 y B1 los bits más significativos de las palabras A y B.



   a) Implemente la tabla de la verdad

   b) Indique las expresiones lógicas mínimas

   c) Dibuje el circuito a nivel de compuertas.

5. Implemente un circuito que realice la multiplicación de dos números de dos bits signados en CA2.

   a) Definir la cantidad de bits del resultado.

   b) Realizar tabla de verdad.

   c) Hallar las funciones lógicas mínimas

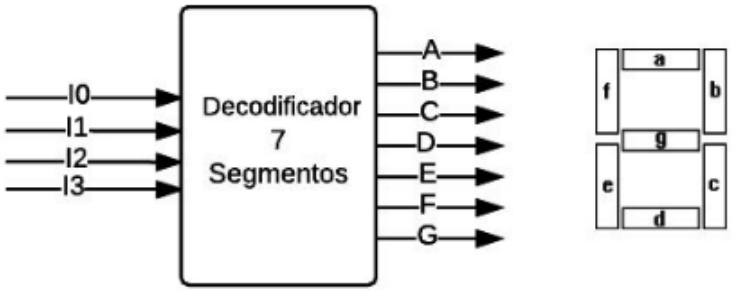
   d) Dibuje el circuito a nivel de compuertas.

6. Implemente un **comparador** de números signados en CA2 de N bits utilizando como base un comparador de magnitud.

   a) Dibuje el circuito a nivel de bloques.

b) Justifique la respuesta.

7. Implemente un decodificador de BCD a 7 segmentos. El display es cátodo común, es decir, cada segmento se prende con un '1' lógico.



a) Implemente la tabla de la verdad

b) Indique las expresiones lógicas mínimas

c) Dibuje el circuito a nivel de compuertas.

d) Implemente el circuito utilizando un codificado 4 a 8

e) Implemente utilizando un multiplexor con 4 entradas de control.

f) Implemente utilizando un multiplexor con 3 entradas de control. Escriba la tabla de la verdad reducida en una variable.

8. Implemente un circuito que calcule el valor absoluto un número signado en CA2 de 4 bits.

a) Determine la cantidad de salidas del circuito.

b) Implemente la tabla de la verdad.

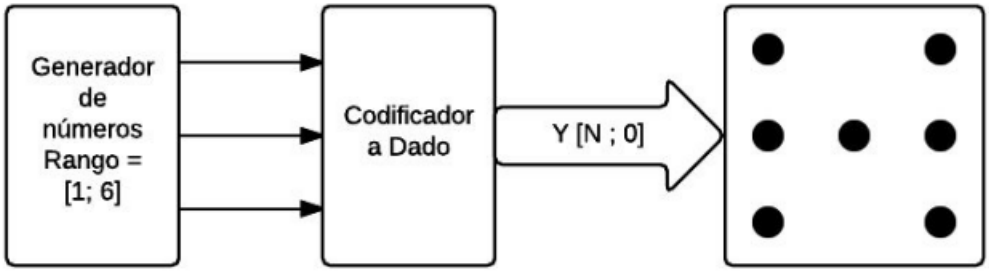
c) Indique las expresiones lógicas mínimas.

d) Dibuje el circuito a nivel de compuertas.

e) Implemente utilizando un multiplexor con 4 entradas de control.

f) Implemente utilizando un multiplexor con 3 entradas de control. Escriba la tabla de la verdad reducida en una variable.

9. Se dispone de un generador de números pseudoaleatorios de tres bits cuyo rango es [1; 6]. Realice una función lógica que convierta estos números en la representación de las caras de un dado. Cada punto del dado se prende con un '1'.



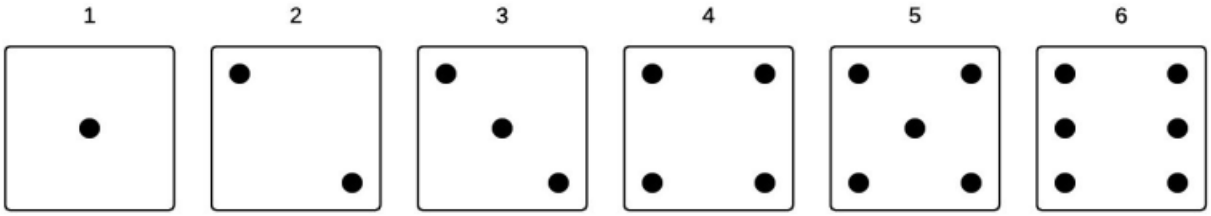
a) Determine la cantidad de salidas del codificador a dado.

b) Obtenga las funciones lógicas del codificador a dado.

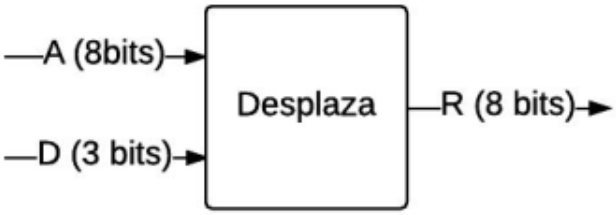
c) Dibuje el circuito resultante.

d) Dibuje el diagrama temporal para la secuencia de entrada 1; 2; 3; 4; 5; 6.

e) ¿Qué LEDs del dado se encenderán si como entrada se coloca alguna de estas combinaciones B"000" ó B"111"?



10. Implemente un circuito que realice **desplazamiento a la izquierda** de un número 8 bits sin signo (**magnitud**) y complete con ceros la parte menos significativa (Least Significant Bit: LSB).



Donde:

- **A**: es el número a desplazar.
- **D**: es la cantidad de bits a desplazar.
- **R**: es el número desplazado.

Ejemplos:

- A = "0001_0101" ; D = "000" => R = "0001_0101"
- A = "1001_0101" ; D = "001" => R = "0010_1010"
- A = "0000_0001" ; D = "111" => R = "1000_0000"

a) Escriba la tabla de la verdad

b) Implemente el circuito utilizando multiplexores

11. Repita el ejercicio anterior pero que el **desplazamiento sea a la derecha** y complete con ceros la parte más significativa (Most Significant Bit: MSB).

12. Implemente un circuito que realice un desplazamiento de un número de 8 bits sin signo (**magnitud**) **hasta que el MSB sea cero**. Agregue unos en la parte menos significativa. Ejemplos:

    - A = “0001_0101” => R = “0001_0101”
    - A = “1001_0101” => R = “0010_1011”
    - A = “1111_1111” => R = “1111_1111”
    - A = “1111_1110” => R = “0111_1111”

    a) Escriba la tabla de la verdad

    b) Implemente el circuito utilizando multiplexores

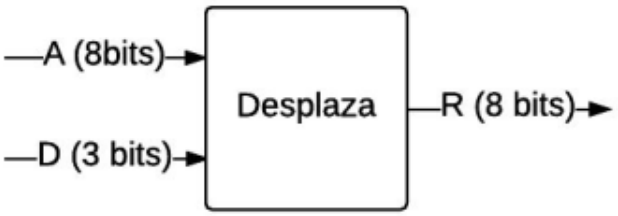
13. **(Obligatorio)**Implemente un circuito que realice un desplazamiento de un número de 8 bits sin signo (**magnitud**) **hasta que el MSB sea uno**. Agregue ceros en la parte menos significativa. Ejemplos:

    - A = “0001_0101” => R = “1010_1000”
    - A = “1001_0101” => R = “1001_0101”
    - A = “0000_0000” => R = “0000_0000
    - A = “0000_0001” => R = “1000_0000”

    a) Escriba la tabla de la verdad

    b) Implemente el circuito utilizando multiplexores

14. Implemente un circuito que realice desplazamiento de un número de 8 bits signado en **CA2**. Debe complete con cero la parte menos significativa y la más significativa con el bit de signo (MSB).



Donde:

- **A**: es el número a desplazar.
- **D**: es la cantidad de bits a desplazar.
- **R**: es el número desplazado.

Ejemplos:

- A = “0001_0101” ; D = “000” =>R = “0001_0101”
- A = “1001_0101” ; D = “001” =>R = “0010_1010”
- A = “1000_0001” ; D = “111” =>R = “0100_0000”

a) Escriba la tabla de la verdad

b) Implemente el circuito utilizando multiplexores

15. Implemente un circuito que reciba un número signado en CA2 de 16 bits y sature el mismo a 8 bits.

    a) Escriba la tabla de la verdad

    b) Implemente el circuito a nivel de compuertas.

16. Implemente un circuito a nivel de compuertas que realice el decremento de 2 a un número signado en CA2 de N bits. El circuito deberá indicar si hubo overflow al realizar la operación. **No está permitido el uso de sumadores, ni restadores**. Optimice la cantidad de compuertas usadas.

17. Implemente un circuito a nivel de compuertas que realice el incremento de 2 a un número signado en CA2 de N bits. El circuito deberá indicar si hubo overflow al realizar la operación. **No está permitido el uso de sumadores, ni restadores**. Optimice la cantidad de compuertas usadas.

18. **(Obligatorio)**Implemente un circuito que cuente la cantidad de unos en las entradas. El circuito dispone de 5 entradas

    a) Determine la cantidad de salidas del circuito.

    b) Implemente la tabla de la verdad del circuito

    c) Halle el circuito a nivel de compuertas a través del método de síntesis utilizando mapas de Karnaugh y justifique su funcionamiento.

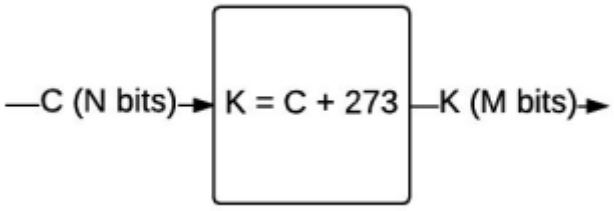
    d) Implemente utilizando únicamente sumadores completos (full adder)

    e) Implemente el código en VHDL del circuito.

```
entity bitCnt is
    port ( entrada : in  std_logic_vector (4 downto 0);
    salida : out std_logic_vector (2 downto 0));
end bitCnt;
```

    f) Implemente un testbench que demuestre el funcionamiento del circuito

19. Implemente un circuito que realice la conversión de un valor de temperatura en grados Celsius cuyo rango está entre -273 ^{o}C a 1000 ^{o}C a grados Kelvin. Los valores de temperatura de entrada y salida del circuito son números enteros los cuales deben estar codificados en CA2. Determine y justifique la cantidad de bits tanto para la entrada y salida de su circuito, así como también el rango resultante.



a) Determine la cantidad de salidas y entradas del circuito.

b) Implemente la tabla de la verdad del circuito

c) Halle el circuito a nivel de compuertas.

d) Implemente utilizando únicamente sumadores completos (full adder)

20. **(Obligatorio)**Implemente un circuito que permita calcular el promedio de 4 números binarios signados en CA2 de 4 bits cada uno.

a) Determine la cantidad de salidas del circuito para calcular correctamente el promedio. Trunque la parte decimal del numero obtenido.

b) Implemente la tabla de la verdad del circuito

c) Implemente utilizando sumadores completos (full adder)

d) Implemente el código en VHDL del circuito. Complete correctamente el generic

```
entity promedio is
    Generic (N: integer := 4);
    port ( entradaA : in std_logic_vector (4 downto 0);
           entradaB : in std_logic_vector (4 downto 0);
           entradaC : in std_logic_vector (4 downto 0);
           entradaD : in std_logic_vector (4 downto 0);
            salida : out std_logic_vector (N downto 0));
end promedio;
```

e) Implemente un testbench que demuestre el funcionamiento del circuito

21. **(Obligatorio)**Implemente un circuito combinacional que realice la multiplicación del número ingresado por la constante $3/2$ (tres medios). La entrada es una magnitud con una cantidad de bits genérica.

a) Implemente el circuito a nivel de bloques.

b) Implemente el código en VHDL del circuito. Complete correctamente el generic

```
entity multFrac is
    Generic (N: integer := 4);
    port ( entrada : in std_logic_vector (4 downto 0);
           salida : out std_logic_vector(N downto 0));
end multFrac;
```

c) Implemente un testbench que demuestre el funcionamiento del circuito

4. Circuitos secuencial

1. Implemente un latch D.

```
entity myLatchD is
    Port ( ena : in std_logic;
           d   : in std_logic;
           q : out std_logic);
     end myLatchD;
```

2. Implemente un latch D, con clear.

```
entity myLatchD_Clr is
    Port ( ena : in std_logic;
           clr : in std_logic;
           d : in std_logic;
           q : out std_logic);
     end myLatchD_Clr;
```

3. Implemente un flip-flop D con reset sincrónico.

```
entity myFFDR is
Port ( clk : in std_logic;
       rst : in std_logic;
       d : in std_logic;
       q : out std_logic);
end myFFDR;
```

4. Implemente un flip-flop D con reset sincrónico y enable.

```
entity myFFDREna is
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       d : in std_logic;
       q : out std_logic);
end myFFDREna;
```

5. **(Obligatorio)**Implemente un flip-flop D con reset sincrónico, enable y clear

```
entity myFFDREnaClr is
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       clr : in std_logic;
       d : in std_logic;
       q : out std_logic);
end myFFDREnaClr;
```

6. **(Obligatorio)**Implemente N flip-flop D con reset sincrónico y enable. Use *generic*.

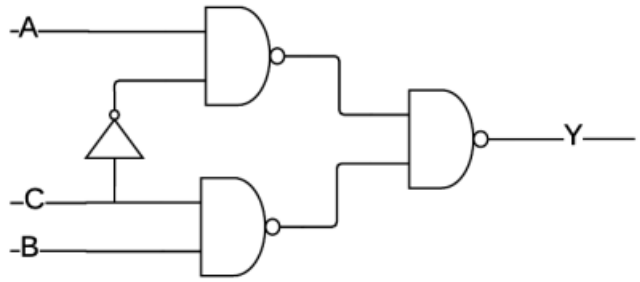
```
entity myFFDREN is
Generic (N: integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       d : in std_logic_vector (N-1 downto 0);
       q : out std_logic_vector (N-1 downto 0));
end myFFDREN;
```

7. Implemente en VHDL flip-flop T con reset sincronico.

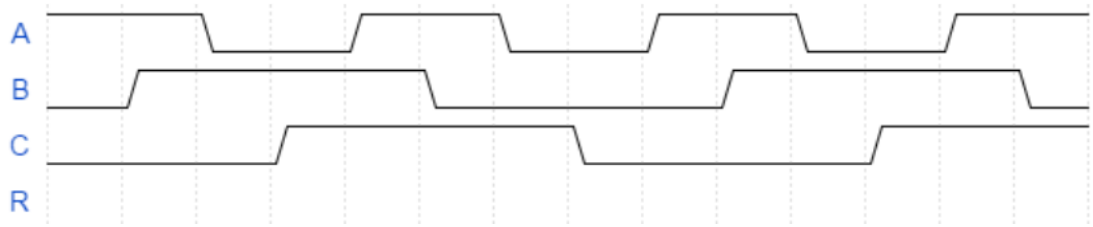
```
entity FFT is
Port ( clk : in std_logic;
       rst : in std_logic;
       t : in std_logic;
       q : out std_logic);
end FFT;
```

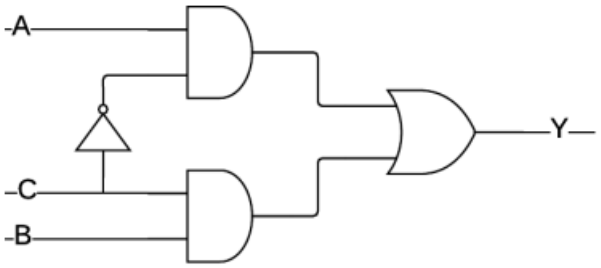
5. Timming

1. Dibuje la señal de salida de los siguientes circuitos considerando que los tiempos de propagación de cada compuerta son de 1 ns, indicarlos claramente en el diagrama. Los circuitos se encuentran estabilizados en el tiempo cero.

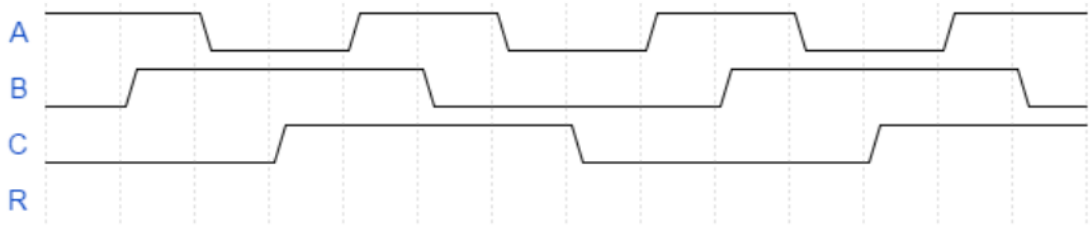


a)

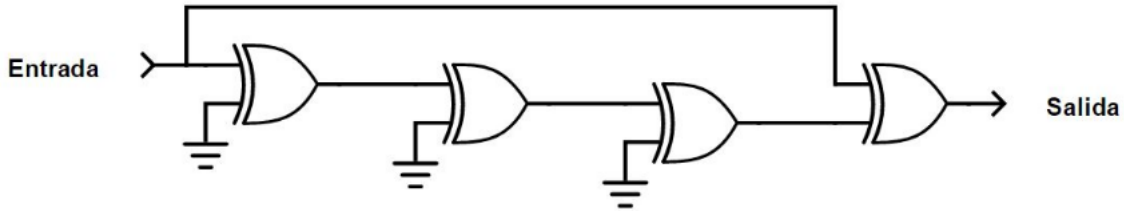




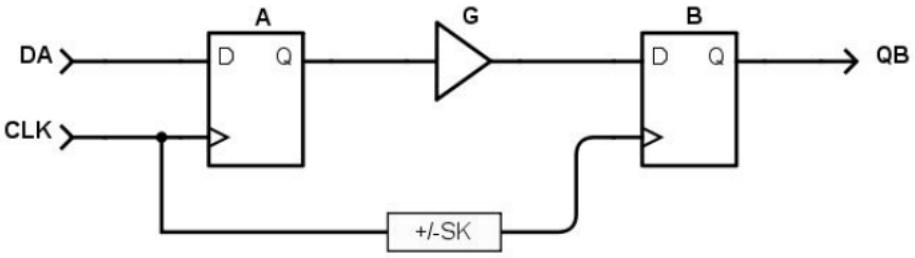
b)



2. Determine la señal de salida del siguiente circuito si el tiempo de propagación de cada compuerta es de 1 ns y se le inyecta por su entrada una señal cuadrada con una frecuencia de 1 MHz. Sea consistente con los ordenes de magnitud de los tiempos involucrados, no se pide un gráfico a escala.



3. Dado el siguiente circuito, parámetros de flip-flops y lógica combinacional



FFA	Min[ns]	Max[ns]
t_{cq}	2	3
t_s	0.2	4
t_h	0.1	0.5

FFB	Min[ns]	Max[ns]
t_{cq}	2	3
t_s	0.2	4
t_h	0.1	0.5

	Min[ns]	Max[ns]
t_g	1	2

Halle:

a) F_{MAX} para $t_{SK} = 0$

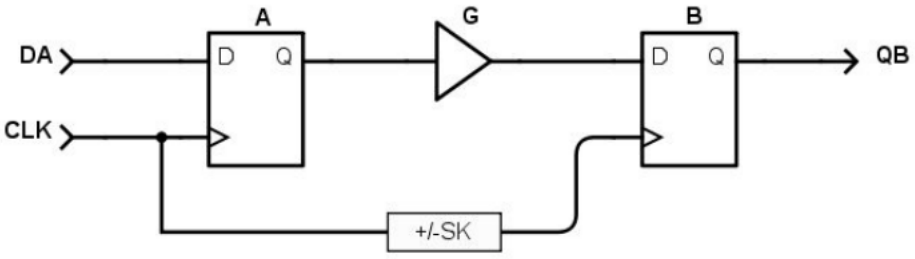
b) t_{SU_SLACK} y t_{H_SLACK} para $t_{SK} = 0$ y $F = 100MHz$

c) El skew máximo tolerable por el circuito para $F = 100MHz$

d) t_{SU_SLACK} y t_{H_SLACK} para $t_{SK} = 2ns$ y $F = 100MHz$

Justifique las respuestas utilizando diagramas temporales.

4. Dado el siguiente circuito, parámetros de flip-flops y lógica combinacional.



FFA	Min[ns]	Max[ns]
t_{cq}	1	2
t_s	0.2	4
t_h	0.1	0.5

FFB	Min[ns]	Max[ns]
t_{cq}	1	4
t_s	0.2	4
t_h	0.1	0.5

	Min[ns]	Max[ns]
t_g	1	2

Halle:

a) F_{MAX} para $t_{SK} = 0$

b) t_{SU_SLACK} y t_{H_SLACK} para $t_{SK} = 0$ y $F = 100MHz$

c) El skew máximo tolerable por el circuito para $F = 100MHz$

d) t_{SU_SLACK} y t_{H_SLACK} para $t_{SK} = 2ns$ y $F = 100MHz$

Justifique las respuestas utilizando diagramas temporales.

6. Registros - Contadores

1. **(Obligatorio)**Implemente un registro de desplazamiento de N bits con
   - Reset sincrónico
   - Enable
   - Entrada serie (si), Salida serie (so)
   - Entrada paralelo (pi) y entrada de carga paralelo (pl)
   - Salida paralelo (po)

```
entity myShiftReg is
Generic (N: integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       si  : in std_logic;
       so  : out std_logic;
       po  : out std_logic_vector (N - 1 downto 0);
       pi  : in std_logic_vector (N - 1 downto 0);
       pl  : in std_logic);
end myShiftReg;
```

2. **(Obligatorio)**Implemente un contador en Johnson de N bits, con reset sincrónico y enable

```
entity myCntJohnson is
Generic (N: integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       q   : out std_logic_vector (N - 1 downto 0));
end myCntJohnson;
```

3. **(Obligatorio)**Implemente un contador en anillo de N bits, con reset sincrónico y enable

```
entity myCntRing is
Generic (N: integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       q   : out std_logic_vector (N - 1 downto 0));
end myCntRing;
```

4. Implemente un contador binario de N bits, con reset sincrónico y enable

```
entity myCntBinarioSimple is
Generic (N: integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       q   : out std_logic_vector (N - 1 downto 0));
end myCntBinarioSimple;
```

5. **(Obligatorio)**Implemente un contador en binario de N bits con:

   - Reset sincrónico
   - Enable
   - Entrada paralelo (d) y entrada de carga paralelo (dl)
   - Salida paralelo (q)

```
entity myCntBinarioPl is
Generic (N: integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       dl  : in std_logic;
       d   : in std_logic_vector (N - 1 downto 0);
       q   : out std_logic_vector (N - 1 downto 0));
end myCntBinarioPl;
```

6. Implemente un contador binario bidireccional de N bits.

   - dir = 0: Ascendente
   - dir = 1: Descendente

```
entity myCntBiDir is
Generic (N: integer := 4);
Port ( clk: in std_logic;
       rst: in std_logic;
       ena: in std_logic;
       dir: in std_logic;
       q : out std_logic_vector (N-1 downto 0));
end myCntBiDir;
```

7. **(Obligatorio)**Implemente un código VHDL que cada M/2 y M *(use generic)* pulsos de clock ponga la salida en 1 durante 1 ciclo de clock.

```
entity myCnt is
Generic (M : integer := 100);
Port ( clk       : in std_logic;
       rst       : in std_logic;
       ena       : in std_logic;
       salidaM_2 : out std_logic;
       salidaM   : out std_logic);
end myCnt;
```

Coloque la siguiente linea de código en donde declara las signals para que se calcule automáticamente la cantidad de bits del contador en base a la cuenta a realizar:

constant N : integer := integer(ceil(log2(real (M))));
constant M_2 : unsigned (N-1 downto 0) := to_unsigned(M-1, N) / 2 ;

Deberá agregar el siguiente *use*:

use ieee.math_real.all

8. **(Obligatorio)**Implemente un código VHDL que cada p pulsos de clock ponga la salida en 1 durante 1 ciclo de clock.

```
entity myCnt2 is
Generic (N : integer := 4);
Port ( clk : in std_logic;
       rst : in std_logic;
       ena : in std_logic;
       p   : in std_logic_vector (N - 1 downto 0);
       salida : out std_logic);
end myCnt2;
```

9. **(Obligatorio)**Implemente en VHDL un Fibonacci LFSR de N bits (xor N entradas). Se recomienda usar *xor_reduce* para ello deberá agregar *use ieee.std_logic_misc.all*

```
entity lfsr_F is
    Generic (N : integer := 4);
    Port ( clk    : in  std_logic;
           rst    : in  std_logic;
           inits  : in  std_logic_vector(N - 1 downto 0);
           taps   : in  std_logic_vector(N - 1 downto 0);
           salida : out  std_logic_vector(N - 1 downto 0));
end lfsr_F;
```

- inits: Es el valor inicial que toma el LFRS (reset)
- taps: Cada elemento en uno del vector indica si forma ese bit parte de la realimentación.

10. **(Obligatorio)**Implemente en VHDL Galois LFSR de N bits (N xor de 2 entradas).

```
entity lfsr_G is
    Generic (N : integer := 4);
    Port ( clk    : in  std_logic;
           rst    : in  std_logic;
           inits  : in  std_logic_vector(N - 1 downto 0);
           taps   : in  std_logic_vector(N - 1 downto 0);
           salida : out  std_logic_vector(N - 1 downto 0));
end lfsr_G;
```

- inits: Es el valor inicial que toma el LFRS (reset)
- taps: Cada elemento en uno del vector indica si forma ese bit parte de la realimentación.

11. **(Obligatorio)**Implemente un detector de flanco ascendente y descendente. Cuando se detecta el flanco correspondiente en la señal de entrada deberá poner la salida en uno por un ciclo de clock. Recuerde que rising_edge y falling_edge deben usarse solo sobre una señal de clk

```
entity myEdgeDetector is
Port ( clk : in std_logic;
       rst : in std_logic;
       d : in std_logic;
       ascendente : out std_logic;
       descendente : out std_logic);
end myEdgeDetector;
```

12. **(Obligatorio)**Implemente un circuito que cuente la cantidad de flancos ascendentes de la señal de entrada d

```
entity myEdgeCnt is
Generic (N: integer := 4);
Port ( clk: in std_logic;
       rst: in std_logic;
       d: in std_logic;
       q : out std_logic_vector (N-1 downto 0));
end myEdgeCnt;
```

13. Implemente un circuito que haga parpadear una salida con una cadencia de 1 segundo, suponiendo que el clock del sistema es de 100MHz. Se recomienda utilizar como componente uno de los contadores implementados anteriormente.

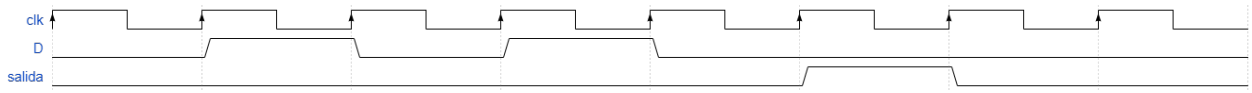
```
entity parpadeo1S is
Port ( clk : in  std_logic;
       rst : in  std_logic;
       salida : out  std_logic);
end parpadeo1S;
```

14. **(Obligatorio)**Implemente un componente en VHDL que disponga de dos entradas dataUp y dataDown

    - Cuando dataUp vale 1 se incrementa el registro en uno en cada flanco de clock.
    - Cuando dataDown vale 1 se decrementa el registro en uno en cada flanco de clock.

```
entity myRegInc is
    Generic (N : integer := 8);
    Port ( clk     : in  std_logic;
           rst     : in  std_logic;
           dataUp  : in  std_logic;
           dataDown: in  std_logic;
           regOut  : out std_logic_vector(N - 1 downto 0));
end myRegInc;
```

15. Implemente un circuito que coloque en uno su salida luego de detectar la secuencia "1010"que ingresa de forma serial por la entrada d. Cada bit de la secuencia dura un ciclo de clock. (Utilice un registro de desplazamiento)




```
entity detectorSecuencia is
    Port ( clk     : in  std_logic;
           rst     : in  std_logic;
           d       : in  std_logic;
           salida  : out  std_logic);
end detectorSecuencia;
```

16. Implemente un circuito que cada vez que la entrada gen reciba un pulso en uno por un ciclo de clock, genera la secuencia "1010"de forma serial a razón de un bit por ciclo de clock.

```
entity generadorSecuencia is
    Port ( clk     : in  std_logic;
           rst     : in  std_logic;
           gen     : in  std_logic;
           salida  : out  std_logic);
end generadorSecuencia;
```

7. Máquinas de estado

FFA	Min[ns]	Max[ns]
t_{cq}	2	3
t_s	0.2	4
t_h	0.1	0.5

FFB	Min[ns]	Max[ns]
t_{cq}	2	3
t_s	0.2	4
t_h	0.1	0.5

1. Explique la diferencia entre una máquina de estados **Mealy** y **Moore**

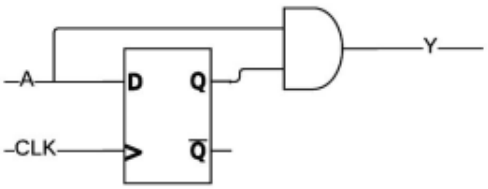
   a) Dibuje el diagrama de bloques de cada una de ellas.

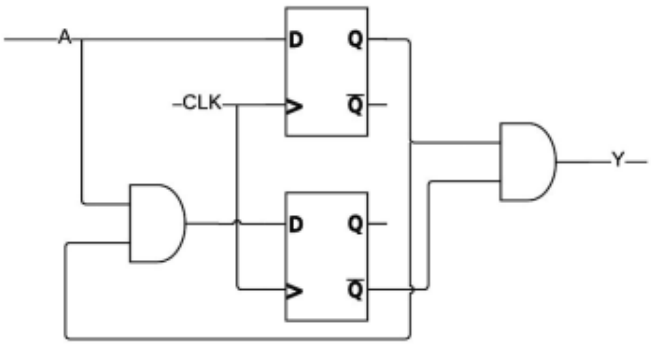
   b) Dibuje un diagrama de estados **Mealy** de un detector de la secuencia binaria “10”.

   c) Dibuje un diagrama de estados **Moore** de un detector de la secuencia binaria “10”.

   Para los diagramas de estado indique claramente cada estado, la entrada, salida y condición de reset.

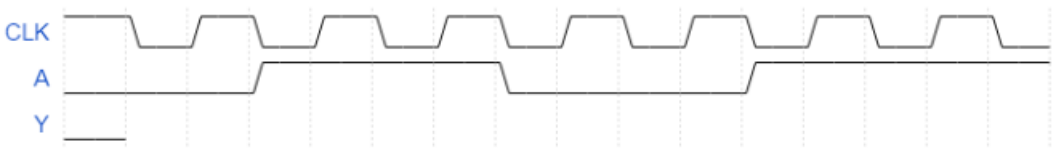
2. A partir de los siguientes circuitos:





   a) Indique a qué tipo de máquina de estados (Mealy o Moore) corresponde cada circuitos. Justifique su respuesta.

   b) Realice el diagrama temporal con las entradas dadas a continuación. Suponga que la salida Q de los FFD está en cero en el momento inicial.



3. Detector de flancos con salida Mealy.

   Implemente una máquina de estados **Mealy** que coloque su única salida en ’1’ durante un ciclo de clock cuando detecte un flanco (ascendente o descendente) en la entrada del sistema.

   a) Realice el diagrama de estados.

   b) Confeccione la tabla de transición de estados con asignación de estados.

   c) Halle las expresiones mínimas de las entradas preparatorias de los FFD y de la salida.

d) Implemente el circuito resultante.

e) Marque el camino crítico en el circuito.

f) Realice un gráfico temporal demostrando el funcionamiento.

g) Incluya un reset sincrónico.

h) Obtenga la máxima frecuencia de clock del sistema.

i) Implemente en VHDL

4. Detector de flancos con salida Moore.

   Implemente una máquina de estados **Moore** que coloque su única salida en '1' durante un ciclo de clock cuando detecte un flanco (ascendente o descendente) en la entrada del sistema.

   a) Realice el diagrama de estados.

   b) Confeccione la tabla de transición de estados con asignación de estados.

   c) Halle las expresiones mínimas de las entradas preparatorias de los FFD y de la salida.

   d) Implemente el circuito resultante.

   e) Marque el camino crítico en el circuito.

   f) Realice un gráfico temporal demostrando el funcionamiento.

   g) Incluya reset sincrónico.

   h) Obtenga la máxima frecuencia de clock del sistema.

   i) Implemente en VHDL

5. **(Obligatorio)**Implemente un circuito que genere la secuencia "10110" (el primer bit de la secuencia es el LSB) cuando la señal de entrada **start** se encuentra en el valor '1'. Si la señal de entrada **start** vale '0' el circuito pone '0' en la salida.

   a) Realice el diagrama de estados.

   b) Confeccione la tabla de transición de estados con asignación de estados.

   c) Halle las expresiones mínimas de las entradas preparatorias de los FFD y de la salida.

   d) Implemente el circuito resultante.

   e) Marque el camino crítico en el circuito.

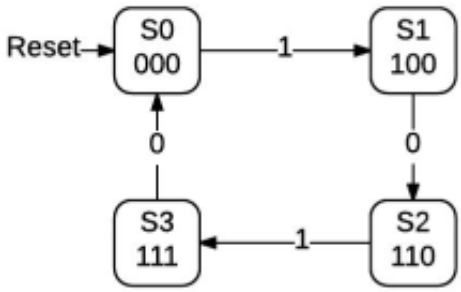
   f) Obtenga la máxima frecuencia de clock del sistema.

   g) Determine el máximo skew tolerable para una F = 100 MHz.

   h) Implemente en VHDL la maquina de estados

```
entity mySec is
Port (clk     : in std_logic;
      rst     : in std_logic;
      start   : in std_logic;
      salida  : out std_logic);
end mySec;
```

6. **(Obligatorio)**Implemente una máquina de estados **Moore** que responda al siguiente diagrama de estados.



a) Confeccione la tabla de transición de estados con asignación de estados.

b) Halle las expresiones mínimas de las entradas preparatorias de los FFD y de la salida.

c) Implemente el circuito resultante.

d) Incluya reset sincrónico.

e) Marque el camino crítico en el circuito.

f) Realice un diagrama temporal que muestre todos los estados.

g) Obtenga la máxima frecuencia de clock.

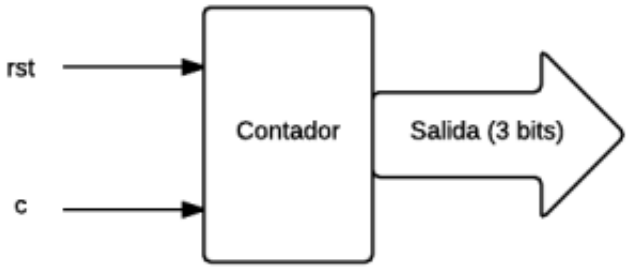
h) Calcule el slack de setup para una F = 100MHz

i) Implemente en VHDL la maquina de estados

```
entity myCntMoore is
Port (clk     : in std_logic;
      rst     : in std_logic;
      ena       : in std_logic;
      salida  : out std_logic_vector (2 downto 0));
end myCntMoore;
```

7. **(Obligatorio)**Contador Binario-Gray.

   Implemente una máquina de estado **Moore** que cuando la señal de entrada de control llamada **C** esté en '0' cuente binario y cuando esté en '1' cuente en Gray.



a) Realice diagrama de estados.

b) Confeccione la tabla de transición de estados con asignación de estados.

c) Expresiones mínimas de las entradas preparatorias de los FFD y de la salida.

d) Implemente el circuito resultante.

e) Incluya reset asincrónico.

f) Obtenga la máxima frecuencia de clock.

g) Calcule el slack de setup para una F = 100MHz

h) Implemente en VHDL la maquina de estados

```
entity myCntBinGray is
Port (clk     : in std_logic;
      rst     : in std_logic;
      c       : in std_logic;
      salida  : out std_logic_vector (2 downto 0));
end myCntBinGray;
```

8. Implemente contador de 2 bits en código gray. (Implemente maquina de estados)

```
entity myCntGray2Bits is
Port (clk       : in std_logic;
      rst       : in std_logic;
      ena       : in std_logic;
      salida    : out std_logic_vector (1 downto 0));
end myCntGray2Bits;
```

8. Testbench

1. **(Obligatorio)**Verifique el funcionamiento de forma automática del LFSR_G de 8 bits realizando el testbench automático

   Cada linea del archivo contendrá los siguientes campos:

   - Reset : 1 carácter (0 ó 1)
   - Separador : 1 carácter (una coma)
   - inits: 8 carácteres (0 ó 1)
   - Separador : 1 carácter (una coma)
   - taps: 8 carácteres (0 ó 1)
   - Separador : 1 carácter (una coma)
   - salida: 8 carácteres (0 ó 1)
   - Avance de linea (\r\n o \n)

   Ejemplo de archivo:

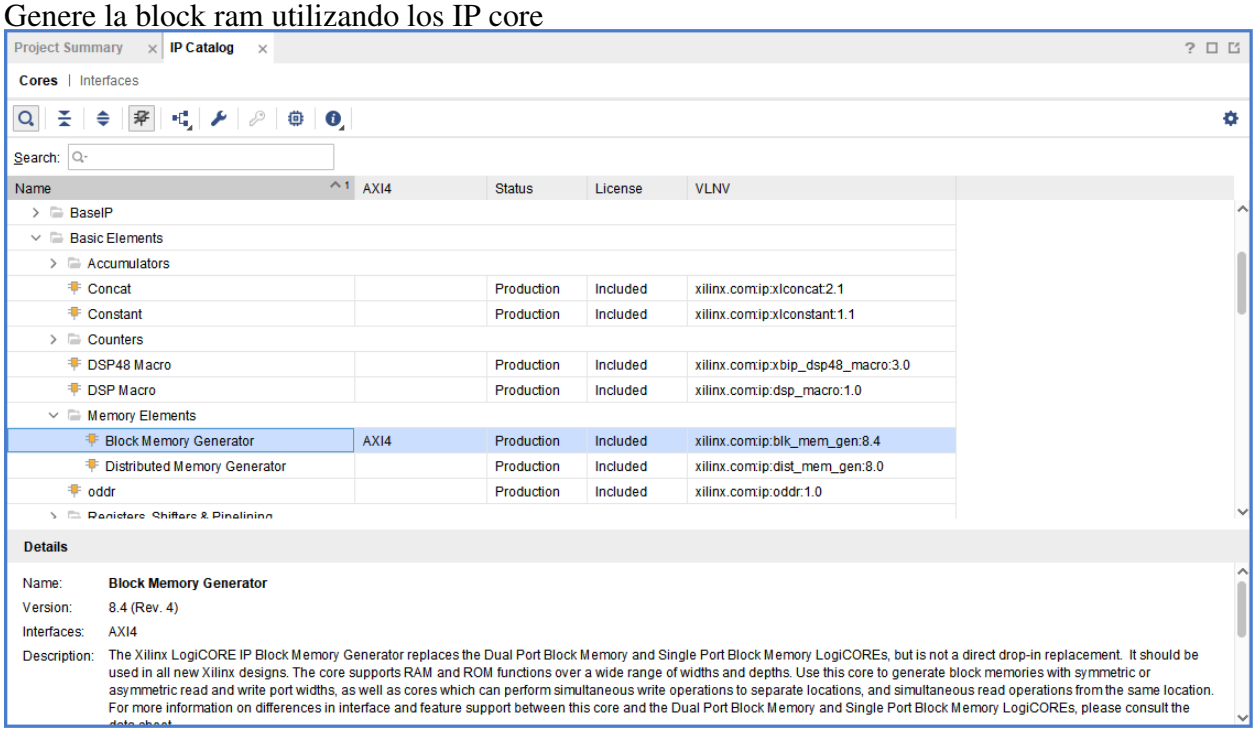
   0,11111111,11111111,00000000
   1,11111111,11111111,11111111
   1,11111111,11111111,10000000
   1,11111111,11111111,01000000
   1,11111111,11111111,00100000
   1,11111111,11111111,00010000
   1,11111111,11111111,00001000
   1,11111111,11111111,00000100
   1,11111111,11111111,00000010
   1,11111111,11111111,00000001

   El testbench deberá leer cada linea del archivo excitando las entradas y verificando si las salida del circuito a testear coincide con las del archivo de testbench. Al aparecer una diferencia entre las salidas del circuito a testear y el archivo deberá generar un report indicando error.

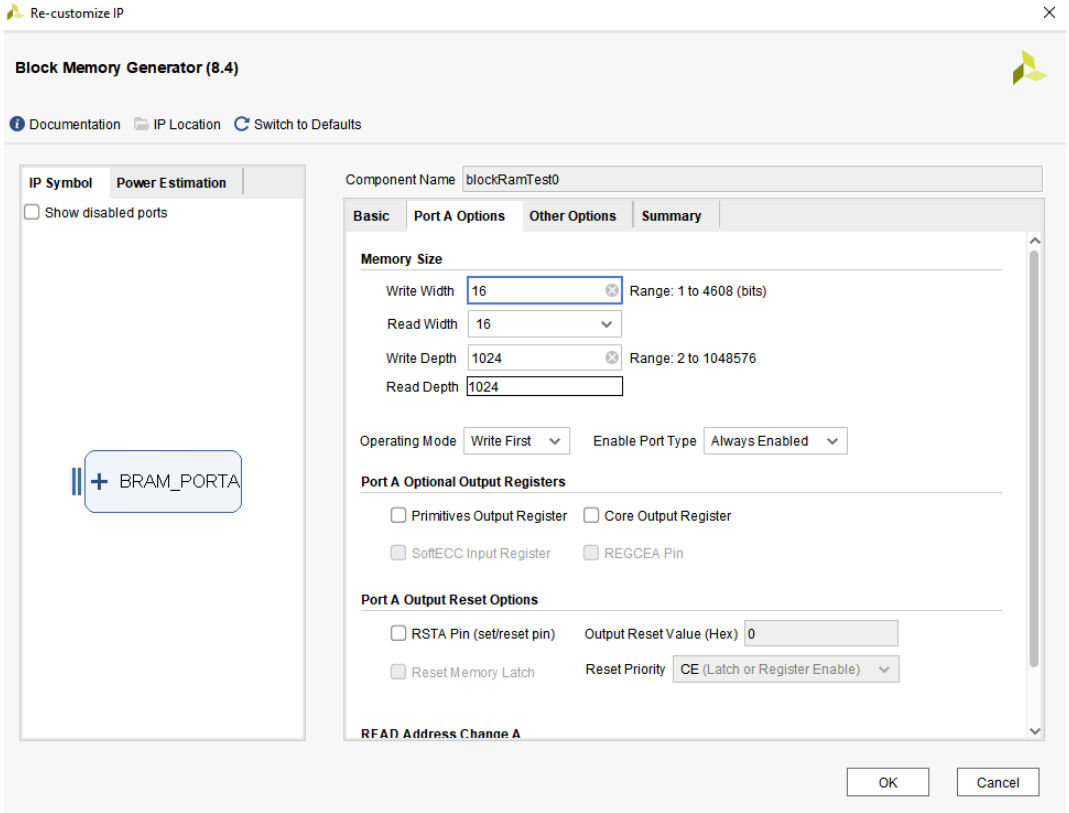
2. **(Obligatorio)**Verifique el funcionamiento de forma automática del LFSR_F de 8 bits realizando el testbench automático

9. Memoria - Block Ram

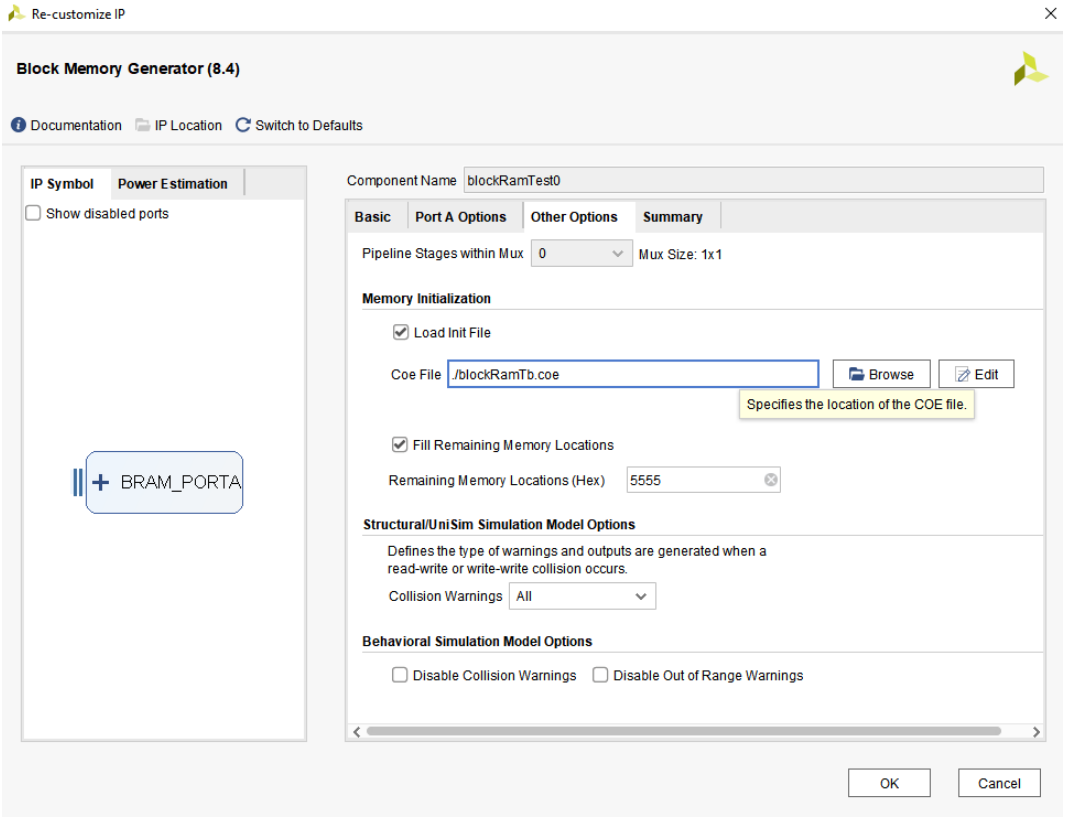
1. Realice el testbench de una blockram de 1024 posiciones y 16 bits de ancho utilizando IP core. Genere la block ram utilizando los IP core



La siguiente imagen muestra como configurar el Port A



La siguiente imagen muestra como configurar la inicialización de la memoria



A continuación se observa la entidad de la block ram a instanciar en el testbench

```
entity blockRamTest0
  port (
    clka : in std_logic;
    wea : in std_logic_vector(0 DOWNTO 0);
    addra : in std_logic_vector(9 DOWNTO 0);
    dina : in std_logic_vector(15 DOWNTO 0);
    douta : out std_logic_vector(15 DOWNTO 0)
  );
end blockRamTest0;
```

El testbench deberá realizar las siguientes operaciones sobre la block ram.

- Leer el contenido de toda la memoria y colocarlo en un archivo con el nombre blockRamRd00.txt
- Escribir en todas las posiciones de memoria de la block ram números consecutivos iniciando por el 0xAA00. Ejemplo En la posición de memoria 0x000 se almacenara 0xAA00 en la posición de memoria 0x001 se almacenara 0xAA01.
- Leer el contenido de toda la memoria y colocarlo en un archivo con el nombre blockRamRd01.txt

Cada linea del archivo generado por el testbench contendrá los campos:

- dirección
- Separador : una coma
- dato leído
- Avance de linea (\r\n o \n)