Los archivos generado deben respetar el siguiente formato de nombre guiaDeClase02_ejercicioNumero.vhd
Si el archivo es un testbench el formato del nombre es el siguiente guiaDeClase02_ejercicioNumero_tb.vhd
Por ejemplo el archivo del ejercicio 1 debe llevar el nombre `guiaDeClase02_01.vhd` Si el ejercicio tuviera items a y b por ejemplo el nombre deberá ser `guiaDeClase02_01_A.vhd` para el punto A.
Todos los archivos deberán ser subidos al repositorio dentro de una carpeta con el nombre guiaDeClase02 junto con sus simulaciones (solo suba los archivos .vhd)
Ver guia holaFFD.pdf Todos los ejercicios llevan reset sincrónico salvo que se indique lo contrario.

1. Implemente un flip-flop D. (Compare el comportamiento con el latch)

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

2. Implemente un FFD. con entrada de clear (Reset asincrónico)

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

3. Implemente un FFD con entrada de enable.

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

4. Implemente N FFD. (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;
```

5. 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.

```
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;
```

6. Implemente un registro de desplazamiento de N bits con reset sincrónico. (Entrada serie y salida serie)

```
entity myShiftReg is
    Generic (N: integer := 4);
    Port ( clk: in  std_logic;
           rst: in  std_logic;
           d: in  std_logic;
           q : out  std_logic);
end myShiftReg;
```

7. Implemente un registro de desplazamiento de N bits con reset síncrono y enable. (Entrada serie y salida serie)

```
entity myShiftReg is
    Generic (N: integer := 4);
    Port ( clk: in  std_logic;
           rst: in  std_logic;
           ena: in  std_logic;
           d: in  std_logic;
           q : out  std_logic);
end myShiftReg;
```

8. Implemente un registro de desplazamiento de N bits con:
   - Reset sincrónico
   - Enable
   - Carga y salida paralela
   - Bidireccional (Tiene dos entradas y dos salidas serie) Para dir = 0 cuenta ascendente

```
entity myShiftReg is
    Generic (N: integer := 4);
    Port ( clk: in  std_logic;
```

```
            rst: in  std_logic;
            ena: in  std_logic;
            dir:  in  std_logic;
            dr, dl :  in  std_logic; -- Entrada serie izquierda y derecha
            d: in  std_logic_vector (N-1 downto 0); -- entrada paralela
            q : out  std_logic_vector (N-1 downto 0));
end myShiftReg;
```

9. Implemente un contador en anillo de N bits.

```
entity myRingCnt 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 myRingCnt;
```

10. Implemente un contador en Johnson de N bits.

```
entity myJohnsonCnt 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 myJohnsonCnt;
```

11. Implemente un contador binario ascendente de N bits con enable.

```
entity myCnt 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 myCnt;
```

12. Implemente un contador binario bidireccional de N bits.
    - dir = 0: Ascendente
    - dir = 1: Descendente

```
entity myCnt 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 myCnt;
```

13. Implemente un contador Johnson de N bits

```
entity myJohnsonCnt 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 myJohnsonCnt;
```

14. (Integrador) 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;
```