Universidad ICESI	
FACULTAD DE INGENIERÍA	
PERIODO ACADEMICO: 2024-1	
LOGICA DIGITAL FECHA ASIGNACIÓN: Semana 8	
Docente: Ing. Juan Palacios M.Sc	Taller 3.

David Dulce A00398802
Santiago Gomez Robledo A00400756

ENTREGABLE: 07/11/2024

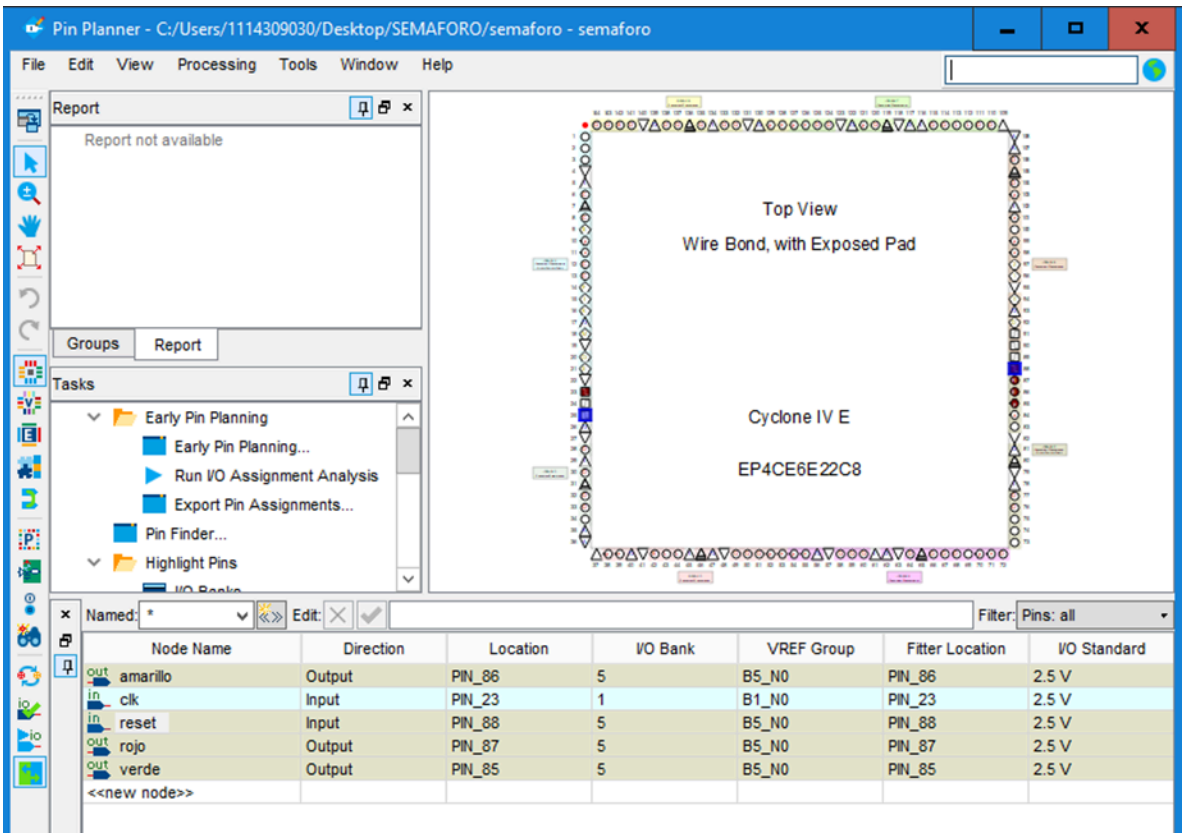
MÁXIMO GRUPOS DE 2

1. Modifique el ejercicio del semáforo de la clase pasada, tal que la secuencia sea ROJO-AMARILLO-VERDE-AMARILLO-ROJO, los tiempos y el numero de estados debe conservarse.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Creamos la entidad
entity semaforo is
  port (
        clk : in std_logic;
        reset : in std_logic;
        rojo : out std_logic;
        amarillo : out std_logic;
        verde : out std_logic
  );
end semaforo;

-- Arquitectura de la máquina de estados
architecture Behavioral of semaforo is
  type state_type is (S_ROJO, S_AMARILLO1, S_VERDE, S_AMARILLO2);
  signal state : state_type := S_ROJO;
  signal contador : integer := 0;
  constant SEGUNDO : integer := 50_000_000;
  constant TIEMPO_ROJO : integer := 6;
  constant TIEMPO_AMARILLO1 : integer := 2;
  constant TIEMPO_VERDE : integer := 6;
  constant TIEMPO_AMARILLO2 : integer := 2;
begin
  process (clk, reset)
  begin
        if reset = '1' then
        state <= S_ROJO;
        contador <= 0;
        elsif rising_edge(clk) then
```

```
        if contador < SEGUNDO * TIEMPO_ROJO - 1 and state = S_ROJO then
        contador <= contador + 1;
        elsif contador < SEGUNDO * TIEMPO_AMARILLO1 - 1 and state =
S_AMARILLO1 then
        contador <= contador + 1;
        elsif contador < SEGUNDO * TIEMPO_VERDE - 1 and state = S_VERDE
then
        contador <= contador + 1;
        elsif contador < SEGUNDO * TIEMPO_AMARILLO2 - 1 and state =
S_AMARILLO2 then
        contador <= contador + 1;
        else
        contador <= 0;
        case state is
         when S_ROJO => state <= S_AMARILLO1;
        when S_AMARILLO1 => state <= S_VERDE;
        when S_VERDE => state <= S_AMARILLO2;
        when S_AMARILLO2 => state <= S_ROJO;
        end case;
        end if;
        end if;
  end process;

  process (state)
  begin
        case state is
        when S_ROJO =>
        rojo <= '0';
        amarillo <= '1';
        verde <= '1';
        when S_AMARILLO1 =>
        rojo <= '1';
        amarillo <= '0';
        verde <= '1';
        when S_VERDE =>
        rojo <= '1';
        amarillo <= '1';
        verde <= '0';
        when S_AMARILLO2 =>
        rojo <= '1';
        amarillo <= '0';
        verde <= '1';
        end case;
  end process;
end Behavioral;
```

Pines:



Tipos y Señales

- `type state_type is (S_ROJO, S_AMARILLO1, S_VERDE, S_AMARILLO2);`: Define los estados posibles del semáforo.
- `signal state : state_type := S_ROJO;`: Señal que mantiene el estado actual del semáforo, empezando en rojo.
- `signal contador : integer := 0;`: Contador para llevar la cuenta del tiempo.
- Constantes que definen el tiempo de los diferentes estados, utilizando un reloj de 50 MHz (`SEGUNDO`).

Proceso de Máquina de Estados

El primer proceso monitorea el reloj y el reset:

- Al recibir un reset (`reset = '1'`), el estado se reinicia a `S_ROJO` y el contador se restablece a 0.
- En el flanco ascendente del reloj (`rising_edge(clk)`):

- Si el contador no ha alcanzado el tiempo correspondiente al estado actual, se incrementa.
- Si se alcanza el tiempo definido para el estado actual, el contador se reinicia y el sistema cambia al siguiente estado según la secuencia: Rojo → Amarillo1 → Verde → Amarillo2 → Rojo.

Proceso de Salidas

El segundo proceso determina el valor de las salidas según el estado actual:

- En cada estado, se establece el valor correspondiente para las luces (rojo, amarillo y verde):
  - `S_ROJO`: enciende rojo y apaga amarillo y verde.
  - `S_AMARILLO1`: enciende amarillo y apaga rojo y verde.
  - `S_VERDE`: enciende verde y apaga rojo y amarillo.
  - `S_AMARILLO2`: enciende amarillo y apaga rojo y verde.

2. Implemente el siguiente contador de 00 a 99, de igual forma explique cuál es el funcionamiento del código, el cuál será sustentado el día de la entrega. (Código corregido)

```
-- Implementación de hardware para mostrar el uso de los displays de 7 segmentos
-- en un contador de 0 a 99 corriendo en una tarjeta Cyclone IV.


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY counter99 IS
	PORT(
	CLK		: IN STD_LOGIC;
	INI	: IN STD_LOGIC;
	RESET		: IN STD_LOGIC;
	DISPLAY		: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
	TRANSISTOR	: OUT STD_LOGIC_VECTOR(1 DOWNTO 0) -- dig1 & dig2 on the EXCEL
	);
END counter99;

ARCHITECTURE AR_099 OF counter99 IS
	TYPE MAQUINA IS (UNIDADES, DECENAS);
	SIGNAL EDO_P, EDO_F : MAQUINA := UNIDADES;
```

```
        CONSTANT CONTA_RETRASO_FIN : INTEGER := 49_999_999;
        CONSTANT CONTA_SW_FIN : INTEGER := 499_999;

        SIGNAL CONTA_UNIDADES, CONTA_DECENAS : INTEGER RANGE 0 TO 9 := 0;
        SIGNAL CONTA_RETRASO : INTEGER RANGE 0 TO CONTA_RETRASO_FIN := 0;
        SIGNAL CONTA_SWITCH : INTEGER RANGE 0 TO CONTA_SW_FIN := 0;
        SIGNAL CONTADOR_PRINCIPAL : INTEGER RANGE 0 TO 99 := 0;
BEGIN
        PROCESS(CLK)
        BEGIN
        IF RISING_EDGE(CLK) THEN
        IF RESET = '1' THEN
                CONTA_UNIDADES <= 0;
                CONTA_DECENAS <= 0;
                CONTA_RETRASO <= 0;
        ELSIF INI = '1' THEN
                CONTA_RETRASO <= CONTA_RETRASO + 1;
                IF CONTA_RETRASO = CONTA_RETRASO_FIN THEN
                CONTA_RETRASO <= 0;
                CONTA_UNIDADES <= CONTA_UNIDADES + 1;
                IF CONTA_UNIDADES = 9 THEN
                        CONTA_UNIDADES <= 0;
                        CONTA_DECENAS <= CONTA_DECENAS + 1;
                        IF CONTA_DECENAS = 9 THEN
                        CONTA_DECENAS <= 0;
                        END IF;
                END IF;
                END IF;
        END IF;
        END IF;
        END PROCESS;

        PROCESS(EDO_P)
        BEGIN
        CASE EDO_P IS
        WHEN UNIDADES =>
                EDO_F <= DECENAS;
                TRANSISTOR <= "01";
        WHEN DECENAS =>
                EDO_F <= UNIDADES;
                TRANSISTOR <= "10";
        WHEN OTHERS => NULL;
        END CASE;
        END PROCESS;

        PROCESS(CLK)
```

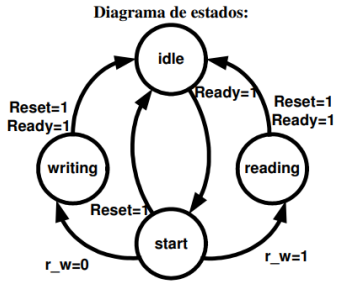
```
        BEGIN
        IF RISING_EDGE(CLK) THEN
        CONTA_SWITCH <= CONTA_SWITCH + 1;
        IF CONTA_SWITCH = CONTA_SW_FIN THEN
                CONTA_SWITCH <= 0;
                EDO_P <= EDO_F;
        END IF;
        END IF;
        END PROCESS;

        CONTADOR_PRINCIPAL <= CONTA_UNIDADES WHEN EDO_P = UNIDADES ELSE
CONTA_DECENAS;

        DISPLAY <= "0000001" WHEN CONTADOR_PRINCIPAL = 0 ELSE
                "1001111" WHEN CONTADOR_PRINCIPAL = 1 ELSE
                "0010010" WHEN CONTADOR_PRINCIPAL = 2 ELSE
                "0000110" WHEN CONTADOR_PRINCIPAL = 3 ELSE
                "1001100" WHEN CONTADOR_PRINCIPAL = 4 ELSE
                "0100100" WHEN CONTADOR_PRINCIPAL = 5 ELSE
                "0100000" WHEN CONTADOR_PRINCIPAL = 6 ELSE
                "0001111" WHEN CONTADOR_PRINCIPAL = 7 ELSE
                "0000000" WHEN CONTADOR_PRINCIPAL = 8 ELSE
                "0000100" WHEN CONTADOR_PRINCIPAL = 9 ELSE
                "0000010";
END AR_099;
```

3. Implemente un código en VHDL que permita llevar a cabo el siguiente diagrama de estados.

La máquina de estados recibe dos entradas, **“ready”** que indica cuando la memoria está preparada, **read/write (r_w)** que indica si se desea realizar una lectura o escritura y una señal de reset. La máquina de estados genera dos variables, **oe** “output enable” y we “write enable”.

Tabla de Salida

Estado	oe	we
idle	0	0
start	0	0
writing	0	1
reading	1	0



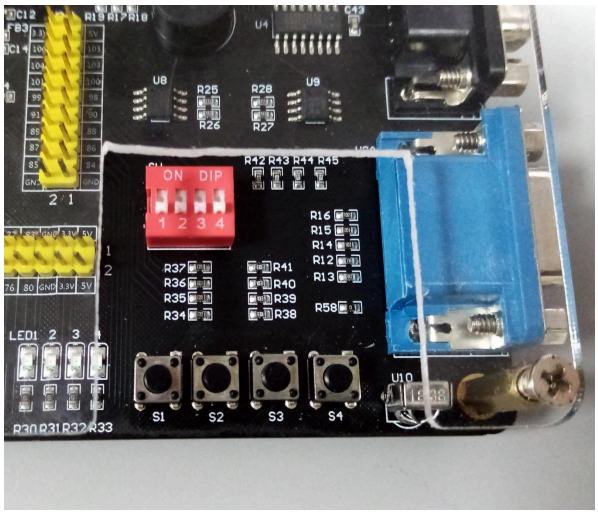

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity memory is
   Port (
      clk    : in  STD_LOGIC;  -- Pin 23  (FPGA_CLK)
      reset  : in  STD_LOGIC;  -- Pin 88  (KEY1) - Activo en bajo
      ready  : in  STD_LOGIC;  -- Pin 88  (ckey1) - Activo en bajo
      r_w    : in  STD_LOGIC;  -- Pin 89  (ckey2) - Activo en bajo
      oe     : out STD_LOGIC;  -- Pin 87  (LED1) - Activo en bajo
      we     : out STD_LOGIC   -- Pin 86  (LED2) - Activo en bajo
   );
end memory;

architecture Behavioral of memory is
   type state_type is (IDLE, START, WRITING, READING);
   signal current_state, next_state : state_type;

   -- Señales internas para manejar la lógica invertida
   signal reset_int, ready_int, r_w_int : STD_LOGIC;

begin
   -- Inversión de las señales de entrada
   reset_int <= not reset;  -- Convierte activo bajo a activo alto
   ready_int <= not ready;  -- Convierte activo bajo a activo alto
   r_w_int   <= not r_w;   -- Convierte activo bajo a activo alto

   -- Proceso de registro de estado
   SYNC_PROC: process(clk, reset_int)
   begin
      if reset_int = '1' then         -- Reset activo en alto internamente
         current_state <= IDLE;
      elsif rising_edge(clk) then
         current_state <= next_state;
      end if;
   end process;

   -- Proceso combinacional para siguiente estado y salidas
   NEXT_STATE_DECODE: process(current_state, ready_int, r_w_int)
   begin
      -- Valores por defecto de las salidas (LEDs apagados - '1' debido a la lógica invertida)
      oe <= '1';
      we <= '1';

      case current_state is
         when IDLE =>
            -- Ambos LEDs apagados en IDLE ('1' para apagado)
            oe <= '1';
            we <= '1';
            if ready_int = '1' then
               next_state <= START;
            else
               next_state <= IDLE;
            end if;
```

```
        when START =>
           if ready_int = '1' then
              if r_w_int = '0' then    -- Switch r_w en posición de escritura
                 next_state <= WRITING;
              else                     -- Switch r_w en posición de lectura
                 next_state <= READING;
              end if;
           else
              next_state <= START;
           end if;

        when WRITING =>
           we <= '0';  -- LED2 encendido (activo bajo)
           if ready_int = '1' then
              next_state <= IDLE;
           else
              next_state <= WRITING;
           end if;

        when READING =>
           oe <= '0';  -- LED1 encendido (activo bajo)
           if ready_int = '1' then
              next_state <= IDLE;
           else
              next_state <= READING;
           end if;

        when others =>
           next_state <= IDLE;
     end case;
  end process;

end Behavioral;
```

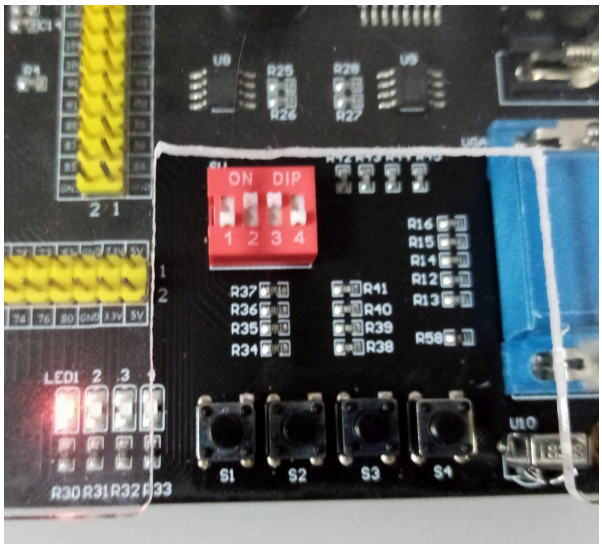
Estado de reposo:

- LED1 (oe) = APAGADO
- LED2 (we) = APAGADO



Operación de Lectura:

- Switches:
  - Switch ready (ckey1) = ON (1)
  - Switch r_w (ckey2) = ON (1) para lectura
- LEDs:
  - LED1 (oe) = ENCENDIDO
  - LED2 (we) = APAGADO

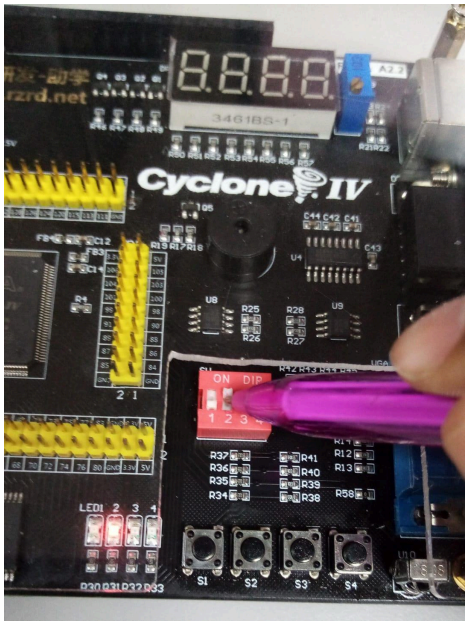


Operación de Escritura:

- Switches:
  - Switch ready (ckey1) = ON (1)
  - Switch r_w (ckey2) = OFF (0) para escritura

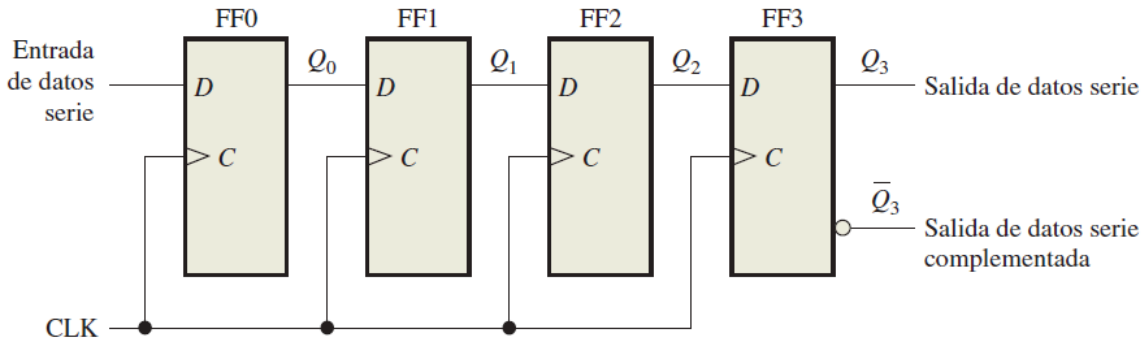
LEDs:

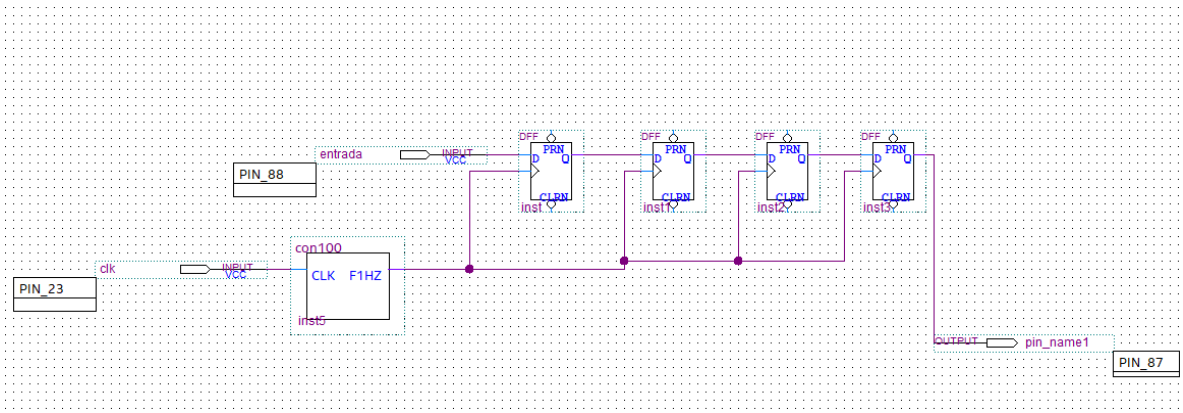
- LED1 (oe) = APAGADO
- LED2 (we) = ENCENDIDO



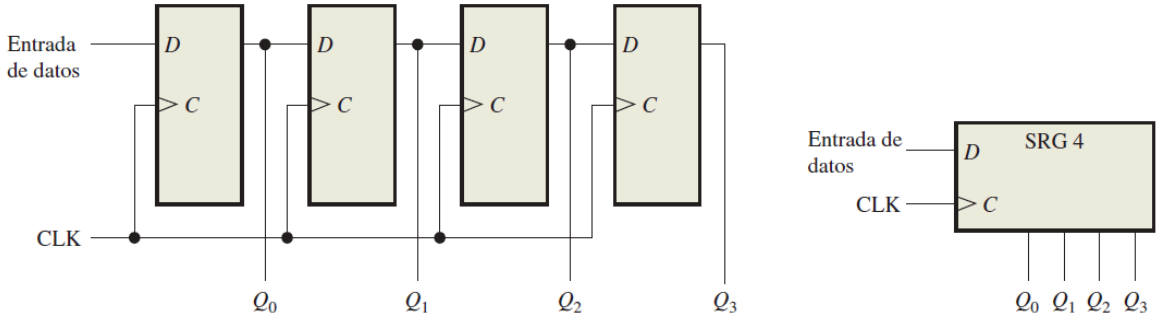
4. Mediante flip flops implemente y verifique el funcionamiento de los registros de desplazamiento Serial Input – Serial Output (SISO) y Serial Input – Parallel Output (SIPO). Nota: Los datos serán los 4 bits de los botones, la salida será los leds.

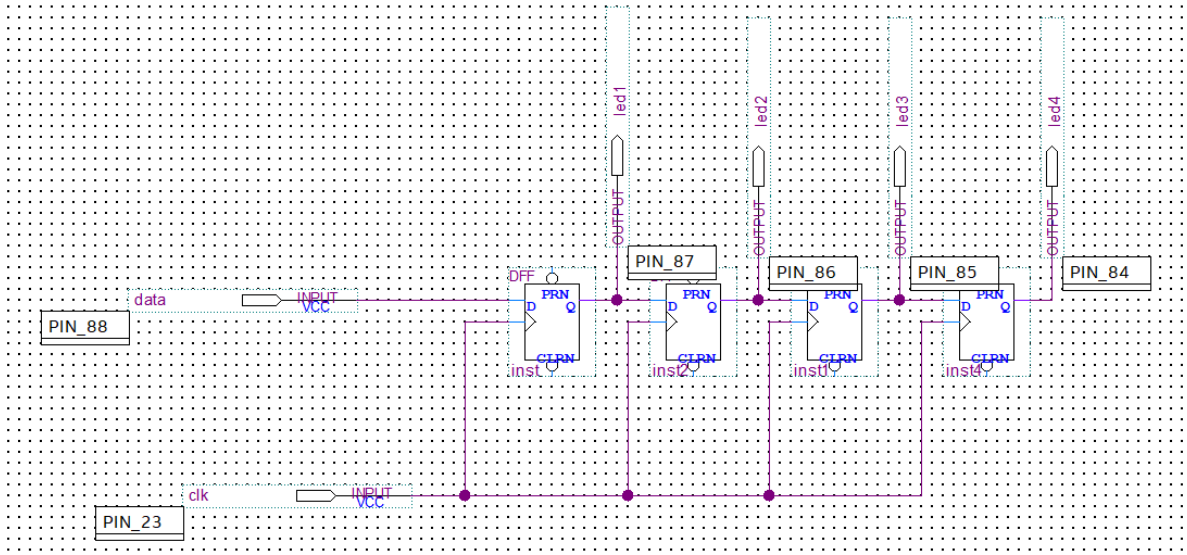
SISO





SIPO





Entregable:

En un pdf subido en intu se debe entregar los siguientes ítems.

- Código en VHDL del punto 1 con evidencia de funcionamiento (fotos)
- Código en VHDL del punto 3 con evidencia de funcionamiento (fotos).

En la clase del 7 de noviembre se debe mostrar:

- Funcionamiento en la tarjeta del punto 2.
- Funcionamiento en la tarjeta del punto 4.