

# Diseño Digital

2024-2

Prof. Alonso Sanchez Huapaya

# Preguntas

# Sobre la clase pasada...

- ¿Cuáles son las 3 reglas del diseño síncrono? ¿Por qué existen?
- ¿Cuál es la estructura básica de todo circuito secuencial?
- ¿Cómo implementamos un registro en VHDL?

# Temario

- Tipo de datos dentro de numeric\_std
- Contadores
- Divisores de frecuencia
- Registro de desplazamiento serial-serial
- Registro de desplazamiento serial-paralelo
- Registro de desplazamiento paralelo-serial
- Generador de PWM

# Tipo de datos dentro de numeric\_std

- Antes de definir a los contadores, se definen los objetos de tipo “signed” o “unsigned”.

Importar: “numeric\_std” (*use ieee.numeric\_std.all;*)

Los “port” (in/out) siempre serán std\_logic ó std\_logic\_vector

Sólo declararemos las “signal” como “signed” ó “unsigned”.

Hay que convertir signed/unsigned a std\_logic para conectar...

# Tipo de datos dentro de numeric\_std

- **Ejemplo 1:** Entidad con 1 entrada de 10 bits *ejm1\_in* y 1 salida de 10 bits *ejm1\_out*.

La entidad realiza lo siguiente:  $ejm1\_out = ejm1\_in + ejm1\_in$ .

Tanto entrada como salida son “std\_logic\_vector” y representan enteros CON signo.

**IMPORTANTE:** El significado/interpretación de la cadena de bits lo da el desarrollador!

# Tipo de datos dentro de numeric\_std

- **Ejemplo 2:** Entidad con 2 entradas de 10 bits cada una  $a$ ,  $b$  y 1 salida de 10 bits  $ejm2\_out$ .

La entidad realiza lo siguiente:  $ejm2\_out = a - b$ .

Tanto entradas como salida son “std\_logic\_vector” y representan enteros CON signo.

**IMPORTANTE:** El significado/interpretación de la cadena de bits lo da el desarrollador!

# Contadores síncronos

- Lógica de salida: “cable” de r\_reg a la salida.
- Lógica de estado siguiente: incremento o decremento.
- En VHDL...
  - Diagrama RTL
  - Diagrama de tiempos

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity binary_counter4_pulse is
    port(
        clk, reset: in std_logic;

        q: out std_logic_vector(3 downto 0)
    );
end binary_counter4_pulse;

architecture two_seg_arch of binary_counter4_pulse is
    signal r_reg: unsigned(3 downto 0);
    signal r_next: unsigned(3 downto 0);
begin
    -- register
    process(clk, reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic (incrementor)
    r_next <= r_reg + 1;
    -- output logic
    q <= std_logic_vector(r_reg);

end two_seg_arch;
```




# Divisores de frecuencia

- Lógica de salida: MSB de r\_reg
- Lógica de estado siguiente: incremento o decremento.
- En VHDL...
  - Diagrama RTL
  - Diagrama de tiempos

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity binary_counter4_pulse is
    port(
        clk, reset: in std_logic;
        max_pulse: out std_logic;
        q: out std_logic_vector(3 downto 0)
    );
end binary_counter4_pulse;

architecture two_seg_arch of binary_counter4_pulse is
    signal r_reg: std_logic_vector(3 downto 0);
    signal r_next: std_logic_vector(3 downto 0);
begin
    -- register
    process(clk)
    begin
        if (reset = '1') then
            r_reg <= (others < '0');
        elsif (clk'event and clk = '1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic (incrementor)
    r_next <= r_reg + 1;
    -- output logic
    q <= std_logic_vector(r_reg);
    max_pulse <= '1' when r_reg="1111" else
        '0';
end two_seg_arch;
```



# Registro de desplazamiento serial-serial

- Lógica de salida: LSB de r\_reg (si es SR) // MSB de r\_reg (si es SL).
- Lógica de estado siguiente: operación SR ó SL de 1 bit, rellena con la entrada serial.
- En VHDL...
  - Diagrama RTL
  - Diagrama de tiempos

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_right_register is
    port(
        clk, reset: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end shift_right_register;

architecture two_seg_arch of shift_right_register is
    signal r_reg: std_logic_vector(3 downto 0);
    signal r_next: std_logic_vector(3 downto 0);
begin
    -- register
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;
    -- next-state logic (shift right 1 bit)
    r_next <= d & r_reg(3 downto 1);
    -- output
    q <= r_reg(0);
end two_seg_arch;
```

# Registro de desplazamiento serial-paralelo


- Lógica de salida: “cable” de r\_reg a la salida.
- Lógica de estado siguiente: operación SR ó SL de 1 bit, rellena con la entrada serial.
- En VHDL...
  - Diagrama RTL
  - Diagrama de tiempos

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_right_register is
    port(
        clk, reset: in std_logic;
        d: in std_logic;
        q: out std_logic_vector(31 downto 0)
    );
end shift_right_register;

architecture two_seg_arch of shift_right_register is
    signal r_reg: std_logic_vector(31 downto 0);
    signal r_next: std_logic_vector(31 downto 0);
begin
    -- register
    process(clk)
    begin
        if (reset = '1') then
            r_reg <= (others <= '0');
        elsif (clk'event and clk = '1') then
            r_reg <= r_next;
        end if;
    end process;

    -- next-state logic (shift right 1 bit)
    r_next <= d & r_reg(31 downto 1);

    -- output
    q <= r_reg(31 downto 0);
end two_seg_arch;
```



# Registro de desplazamiento paralelo-serial


- Lógica de salida: LSB de r\_reg (si es SR) // MSB de r\_reg (si es SL).
- Lógica de estado siguiente: 1 MUX:
  - Selector: entrada de 1 bit adicional a la entidad ("load").
  - Entrada 1: entrada paralela de la entidad.
  - Entrada 2: r\_reg desplazado (SR ó SL)
- En VHDL...
- Diagrama RTL
- Diagrama de tiempos

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_right_register is
    port(
        clk, reset: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end shift_right_register;

architecture two_seg_arch of shift_right_register is
    signal r_reg: std_logic_vector(3 downto 0);
    signal r_next: std_logic_vector(3 downto 0);
begin
    -- register
    process(clk)
    begin
        if (reset = '1') then
            r_reg <= (others => '0');
        elsif (clk'event and clk = '1') then
            r_reg <= r_next;
        end if;
    end process;

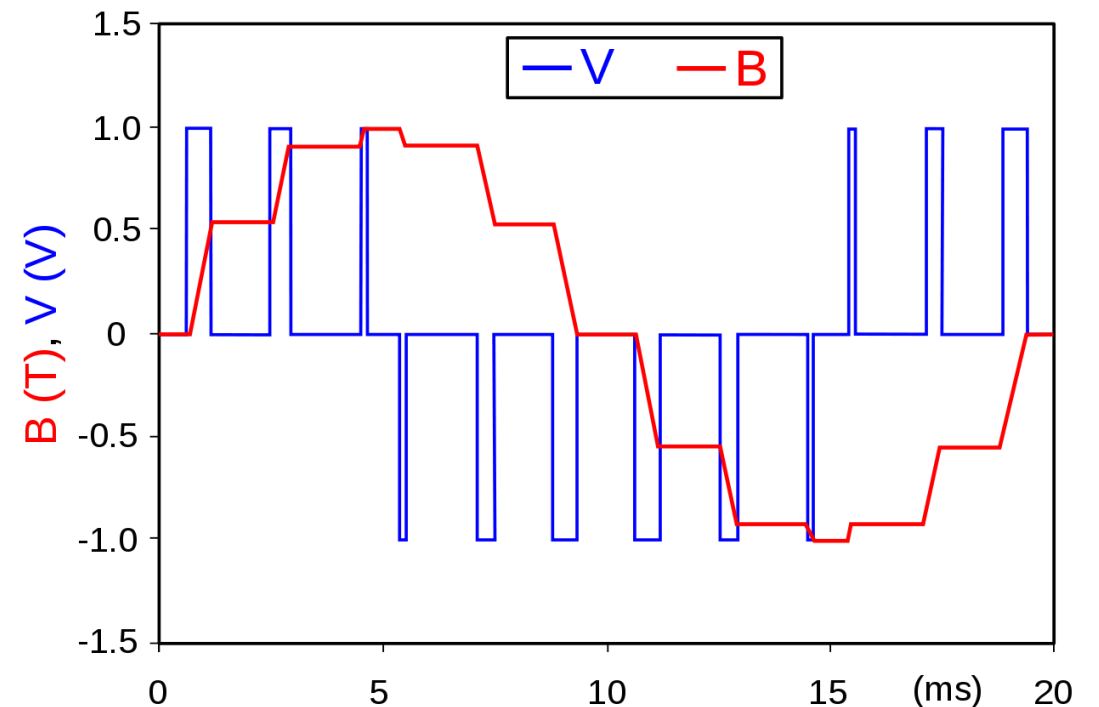
    -- next-state logic (shift right 1 bit)
    r_next <= d & r_reg(3 downto 1);

    -- output
    q <= r_reg(0);
end two_seg_arch;
```



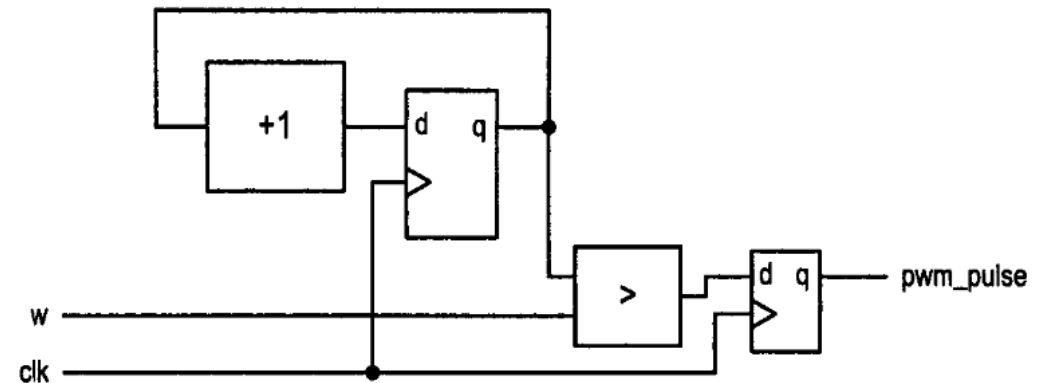
# Generador PWM

- ¿Qué es “PWM”?
  - Pulse-width modulation (modulación por ancho de pulso)
  - Genera una señal cuadrada con ancho de pulso (tiempo “en alto”) variable.
  - Ejemplo de aplicación: control de velocidad en un motor (curva azul: voltaje generado con PWM. Curva roja: corriente en inductor del motor)



# Generador PWM

- RTL básico:
  - Registro contador de “n” bits
  - Comparador, recibe cuenta del contador y una entrada de “n” bits (w en la figura)
  - Entrada “w” controla el duty-cycle de la PWM
  - Buffer de 1 bit a la salida del comparador



**Figure 9.9** Block diagram of a PWM circuit.

# Generador PWM

- VHDL y diagrama de tiempos

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity pwm is
    port(
        clk, reset: in std_logic;
        w: in std_logic_vector(3 downto 0);
        pwm_pulse: out std_logic
    );
end pwm;
architecture two_seg_arch of pwm is
    signal r_reg: unsigned(3 downto 0);
    signal r_next: unsigned(3 downto 0);
    signal buf_reg: std_logic;
    signal buf_next: std_logic;
begin
    — register & output buffer
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
            buf_reg <= '0';
        elsif (clk'event and clk='1') then
            r_reg <= r_next;
            buf_reg <= buf_next;
        end if;
    end process;
    — next-state logic
    r_next <= r_reg + 1;
    — output logic
    buf_next <=
        '1' when (r_reg<unsigned(w)) or (w="0000") else
        '0';
    pwm_pulse <= buf_reg;
end two_seg_arch;
```