Universidad
Carlos III de Madrid

Design of combinational circuits in VHDL

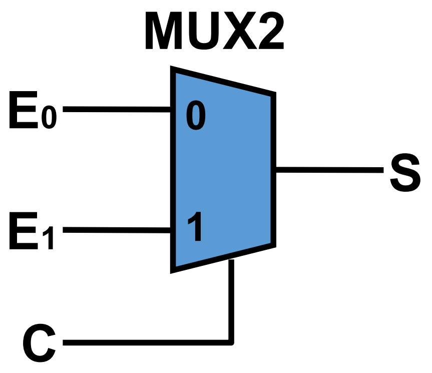
***Autores: Luis Entrena, Celia López, Mario García,
Enrique San Millán, Marta Portela, Almudena Lindoso***

Outline

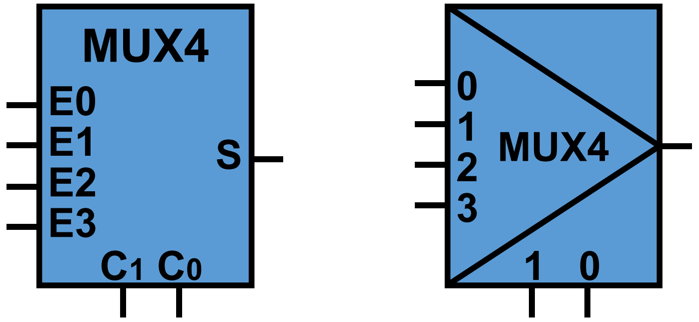
- Conditional statements
- Rules for the design of combinational circuits
- Examples and exercises

Digital Electronics Review: Multiplexers

- Definition:
  - Combinational circuit that selects one input according to a control signal and passes its value to the output
  - Named by the number of data inputs: MUX2, MUX4, …
  - N=data inputs, n=control inputs $\Rightarrow 2^n = N$



C0	S
0	E0
1	E1



C1	C0	S
0	0	E0
0	1	E1
1	0	E2
1	1	E3

$$S = \overline{C_1}\,\overline{C_0}E_0 + \overline{C_1}C_0E_1 + C_1\overline{C_0}E_2 + C_1C_0E3_0$$

Conditional statements

Sequential statements

- IF...THEN...ELSE
- CASE...IS...WHEN

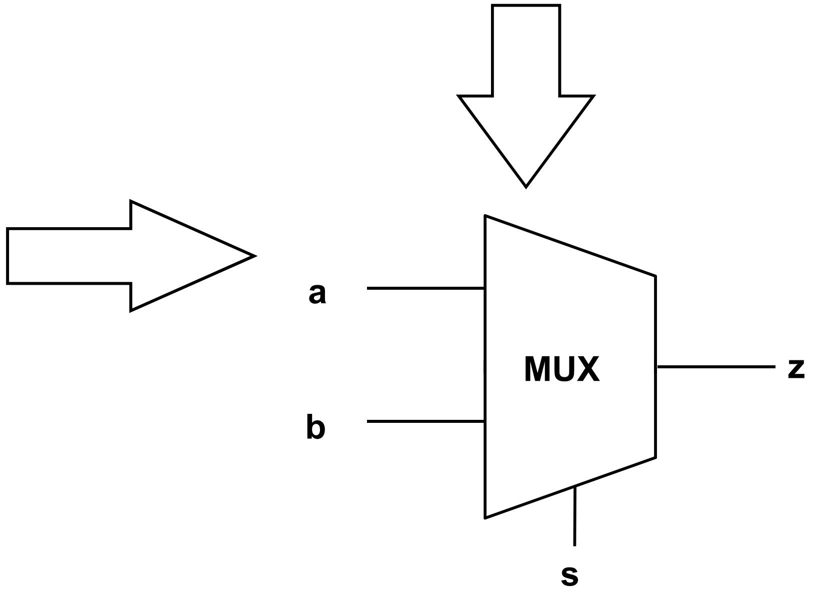
Concurrent statements

- ... WHEN ...
  (conditional assignment)
- WITH ... SELECT
  (selected assignment)

A 2:1 multiplexer

```
-- Sequential description
PROCESS (a, b, s)
BEGIN
          IF s = '0' THEN
                    z <= a;
          ELSE
                    z <= b;
          END IF;
END PROCESS;
```

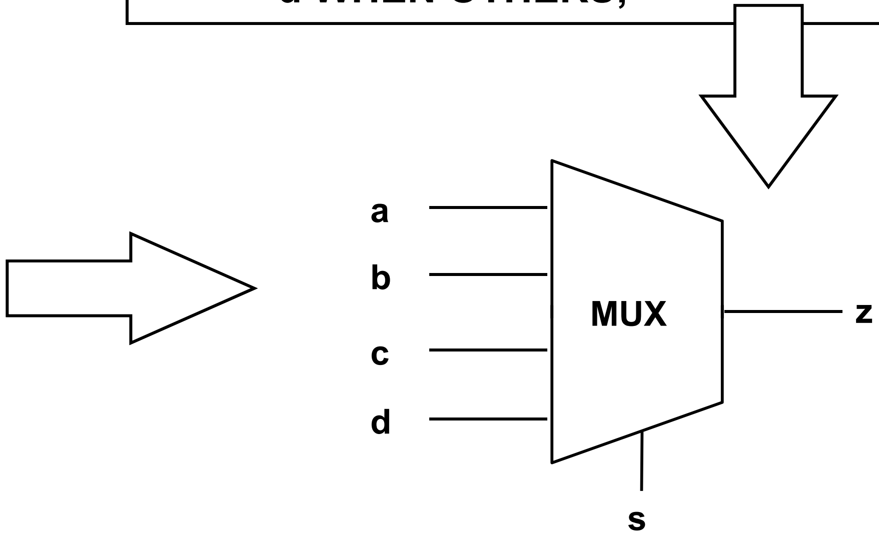
```
-- Concurrent description
z <= a WHEN s = '0'
        ELSE b;
```



A 4:1 multiplexer

```
-- Sequential description
PROCESS (a, b, c, d, s)
BEGIN
        CASE s IS
                WHEN "00" =>        z <= a;
                WHEN "01" =>        z <= b;
                WHEN "10" =>        z <= c;
                WHEN OTHERS => z <= d;
                END CASE;
END PROCESS;
```

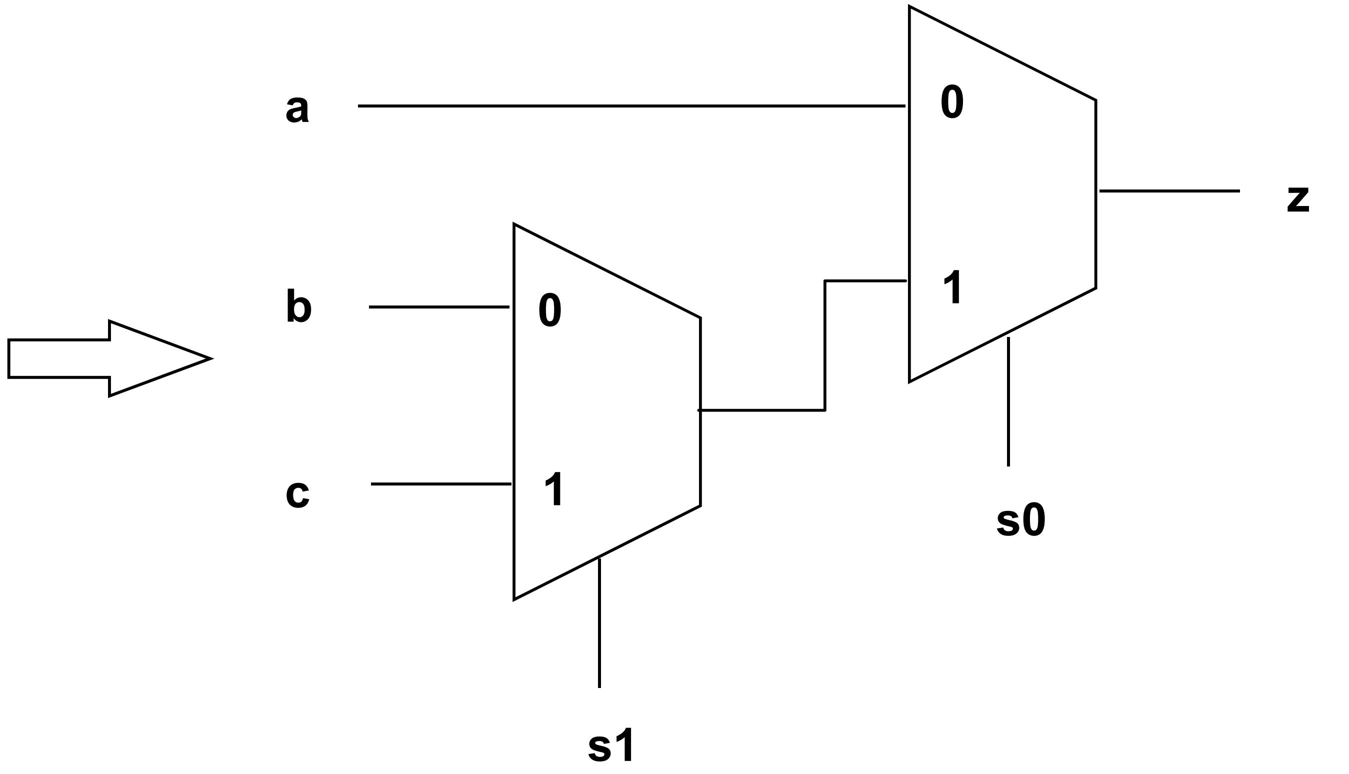
```
-- Concurrent description
WITH s SELECT
z <=      a WHEN "00",
          b WHEN "01",
          c WHEN "10",
          d WHEN OTHERS;
```



Differences between if and case

- IF...THEN...ELSE imposes priorities in the selection inputs

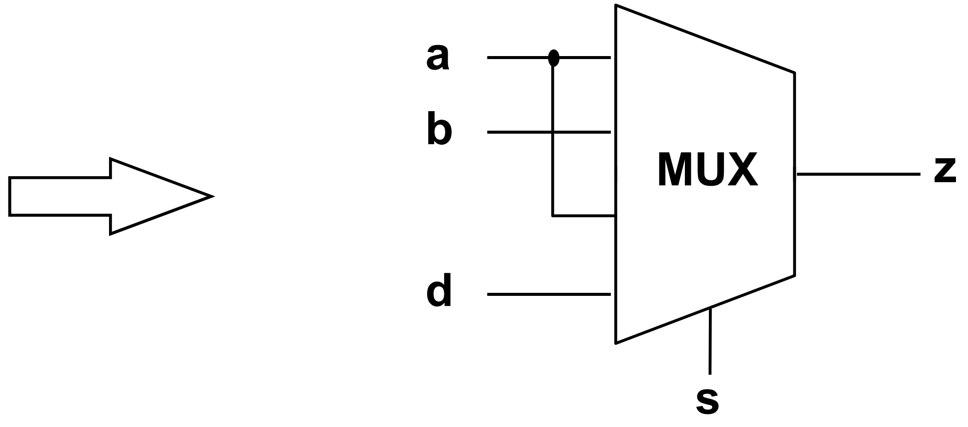
```
IF s0 = '0' THEN
            z <= a;
ELSIF s1 = '0' THEN
            z <= b;
ELSE
            z <= c;
END IF;
```



Differences between if and case

- CASE does not impose any priority

```
PROCESS (a, b, c, d, s)
BEGIN
        CASE s IS
                WHEN "00" | "10" =>      z <= a;
                WHEN "01" =>             z <= b;
                WHEN OTHERS =>           z <= d;
                END CASE;
END PROCESS;
```



Truth tables

```
PROCESS (a, b, c)
        VARIABLE inputs: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
        inputs  := a & b & c;
        CASE inputs IS
                WHEN "000" =>       f <= '1';
                WHEN "001" =>       f <= '-';
                WHEN "010" =>       f <= '0';
                WHEN "011" =>       f <= '1';
                WHEN "100" =>       f <= '0';
                WHEN "101" =>       f <= '1';
                WHEN OTHERS =>      f <= '-';
        END CASE;
END PROCESS;
```

Rules for the design of combinational circuits

- Conditions that are necessary for a process to infer a correct combinational circuit:
  - *If a signal is assigned in a process or a conditional concurrent statement, then it must be assigned a value for all the possible branches or conditions.*
  - *The sensitivity list of a process must contain all the signals that are read in the process, i.e. all the signals whose value is used in the process, and therefore are inputs in the synthesized circuit.*
  - *Variables used in processes have to be used as intermediate variables, i.e. they have to be written before being read.*

Rules for the design of combinational circuits

- Some consequences of the previous rules:
  - To cover all the possible cases, it is convenient that every IF sentence or conditional assignment is followed by an ELSE clause, and every CASE or selection assignment is followed by a WHEN OTHERS clause.
  - Circular references should not be created (e.g., outputs that depend on themselves), because they will create asynchronous feedback loops.
  - Use of signals is recommended over variables whenever possible.

Examples (mistakes and correct solution)

⊘

```
PROCESS( s, a)
BEGIN
        IF s = '0' THEN
                z <= a;
        END IF;
END PROCESS;
```

⊘

```
PROCESS( s, a)
BEGIN
        IF s = '0' THEN
                z <= a;
        ELSE
                z <= z;
        END IF;
END PROCESS;
```

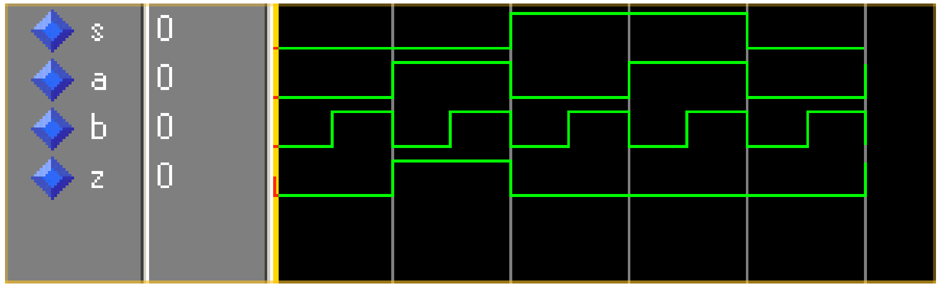
⊘

```
PROCESS( s, a)
BEGIN
        IF s = '0' THEN
                z <= a;
        ELSE
                z <= b;
        END IF;
END PROCESS;
```

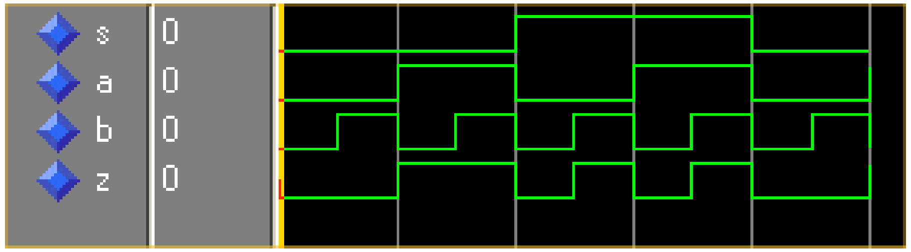
```
PROCESS( s, a, b)
BEGIN
        IF s = '0' THEN
                z <= a;
        ELSE
                z <= b;
        END IF;
END PROCESS;
```

Effect of the sensitivity list in the simulation

- Incorrect sensitivity list (s, a)

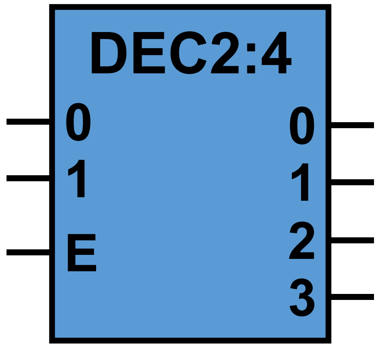


- Correct sensitivity list (s, a, b)



Digital Electronics Review: Decoders

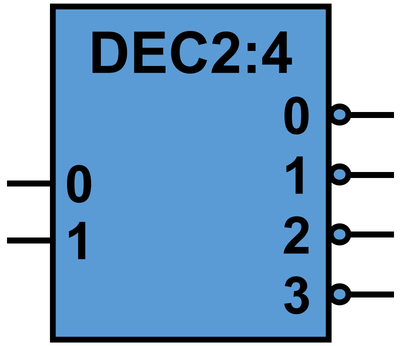
- Definition:
  - Combinational circuit that transforms an encoded value into the activation of an output that corresponds to that value



E	E_1	E_0	S_3	S_2	S_1	S_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Digital Electronics Review: Decoders

- Active-low decoder



E_1	E_0	S_3	S_2	S_1	S_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Exercise

- Design a 2:4 Decoder
  - Using sequential statements
  - Using concurrent statements
- Entity declaration

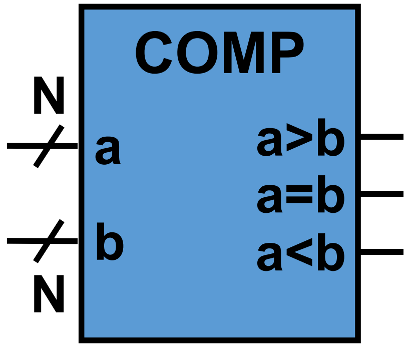
```
ENTITY decoder IS
          PORT (  a: IN STD_LOGIC_VECTOR (1 DOWNTO 0);

                  z: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));

END decoder;
```

Digital Electronics Review: Comparators

- ❑ Definition: circuit that determines if an input is equal, smaller or greater than another input.
- ❑ N is the data number of bits



Comparator 1-bit

a	b	a=b	a>b	a<b
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

$$f_{a=b} = \overline{a \oplus b}$$

$$f_{a>b} = a\overline{b}$$

$$f_{a<b} = \overline{a}b$$

Exercise

- Design a 4-bit comparator
  - Using sequential statements
  - Using concurrent statements
- Entity declaration

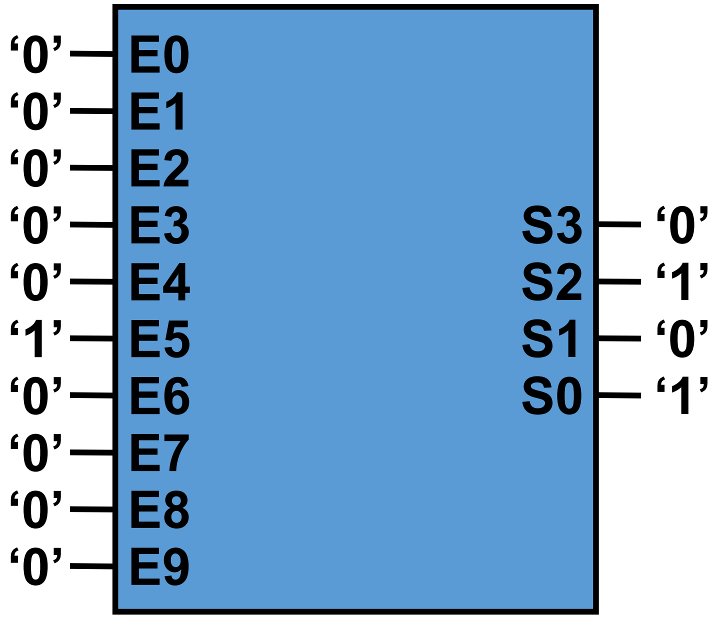
```
ENTITY comparator IS
        PORT (  a, b: IN SIGNED(3 DOWNTO 0);

                greater, smaller, equal: OUT STD_LOGIC);

END comparator;
```

Digital Electronics Review: Encoders

- Definition:
  - A combinational circuit allows to translate and active level of an input into an encoded value
- Example: numeric keypad
  - Inputs: digits 0-9
  - Outputs: binary encoding (4 bits)



E5 active => S=“0101” (=5)

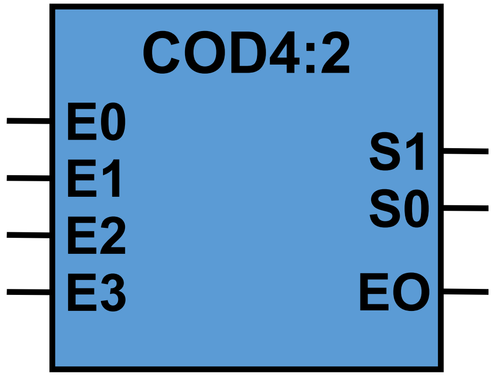
Digital Electronics Review: Encoders

- Limitations and problems
  - Only one input can be active at the same time
  - Several active inputs may lead to erroneus output
    - E1 and E4 active result in output 5
  - No active inputs result in output 0, same as E0 active

Active Input	$S_3S_2S_1S_0$
E_0	0000
E_1	0001
E_2	0010
E_3	0011
E_4	0100
E_5	0101
E_6	0110
E_7	0111
E_8	1000
E_9	1001

Digital Electronics Review: Encoders

- M:N ⇒ ‘M’ inputs, ‘N’ outputs
- EO: “Enable Output”
  - Used to distinguish the “no active inputs” and “E0 active” input cases.
  - Used also to chain several encoders
- Limitations
  - Any multiple activation
  - Don’t care output values



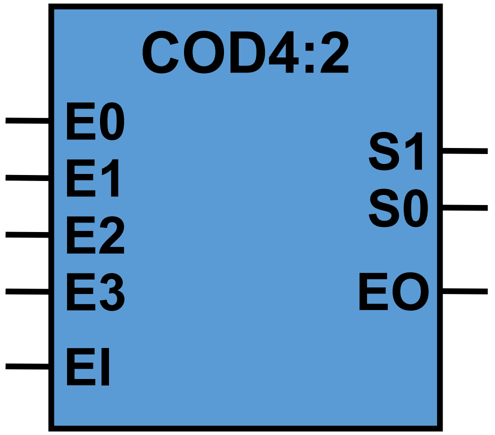
E_3	E_2	E_1	E_0	S_1	S_0	EO
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	0	1	1	0
0	0	0	0	0	0	1
Other cases				X	X	X

Digital Electronics Review: Priority Encoders

- Characteristics
  - Prioritize multiple input activation
  - Priority:
    - To the most significant bit (MSB): priority is given to the MSB
      E1 and E5 active, result is 5
    - To the least significant bit (LSB): priority is given to the LSB
      E1 and E5 active, result is 1

Digital Electronics Review: Priority Encoders

- M:N ⇒ 'M' inputs, 'N' outputs
- EO: "Enable Output"
  - EI or E: "Enable Input" or "Enable": Enables the circuit:
    - '0' (disabled): outputs are tied to '0'
    - '1' (enabled): normal operation
  - EI and OE are used together to chain encoders
- GS: "Group Signal"
  - Indicates at least one input is active



EI	E3	E2	E1	E0	S1	S0	EO
0	X	X	X	X	0	0	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	X	0	1	0
1	0	1	X	X	1	0	0
1	1	X	X	X	1	1	0

Exercise

- Design a 2-bit encoder with priority
  - Using sequential sentences
  - Using concurrent sentences
- Declaration of the entity:

```
ENTITY encoder IS
          PORT (  a: IN STD_LOGIC_VECTOR (3 DOWNTO 0);

                  z: OUT UNSIGNED(1 DOWNTO 0);

                  gs: OUT STD_LOGIC);

END encoder;
```