Describing circuits with VHDL

OBJECTIVES

- Designing a VHDL module:
  - Entity
  - Architecture
  - Libraries
- Declaring and using the object "Signal"
- Working with composite types STD_LOGIC_VECTOR
- Complex designs: Structural description
- Arithmetic operations: Types Signed y Unsigned

Describing a digital circuit (Old-fashion style)

Two-inputs multiplexer.

Truth table

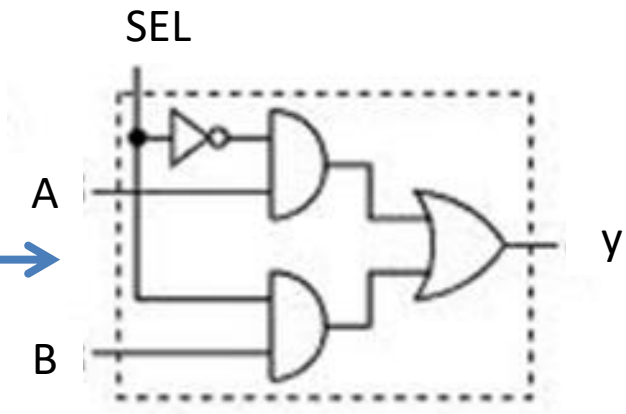
Entradas Salida

A	B	SEL	Y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Boolean expression

$$Y = A * \overline{sel} + B * sel$$

Logic gates circuit



1. Describing digital circuits with VHDL.

Concurrent Statements

- **Assignment**
  - ✓ Simple or unconditional (<=)
  - ✓ Conditional (when-else)
  - ✓ Selected (with –select- when)
- **Process**

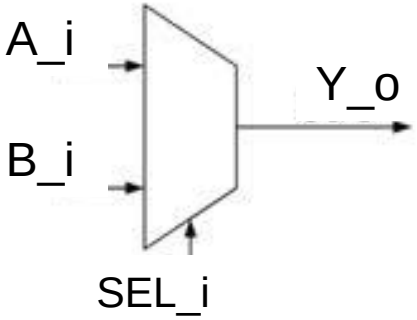
Sequential Statements (inside a process)

- **Signal simple assignment**
- **Control structures**
  - ✓ If-then-else
  - ✓ case

1.1. VHDL concurrent signal assignment statements

Project_04. Designing a two-inputs multiplexer (MUX2_1bit).

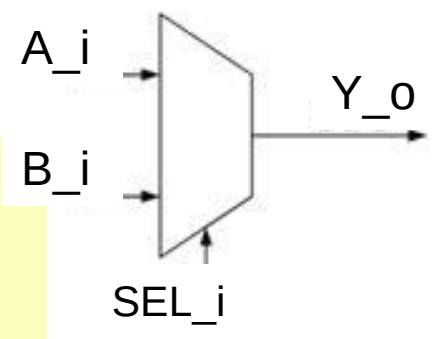
Entidad




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

entity MUX2_1bit is
Port ( A_i : in  STD_LOGIC;
       B_i : in STD_LOGIC;
       SEL_i : in  STD_LOGIC;
       Y_o : out STD_LOGIC);
end MUX2_1bit ;
```

Describing a digital circuit with VHDL.

Project_04. Designing a two-inputs multiplexer (MUX2_1bit).



- VHDL allows a natural description of the circuit
- Boolean equations are no needed

"Y_o" is "A_i" when SEL_i='0' otherwise is "B_i"

```
architecture behavioral of MUX2 is
begin
    -- Multiplexer description by means of
    -- a conditional concurrent assignment statement

    Y_o <= A_i when SEL_i = '0' else B_i;

end behavioral ;
```

Project_04. Designing a two-inputs multiplexer (MUX2_1bit).

Now, in order to test the described circuit :

1. A constraints file (XDC) should be added:
   - Data inputs: A_i and B_i → Two switches
   - Select input: Sel_i → One pushbutton
   - Data output: Y_o → One LED
2. Run the **synthesis** and **implementation** processes
3. Generate the **bitstream** file and programm the FPGA
4. **Check** that the circuit behaviours as you expected

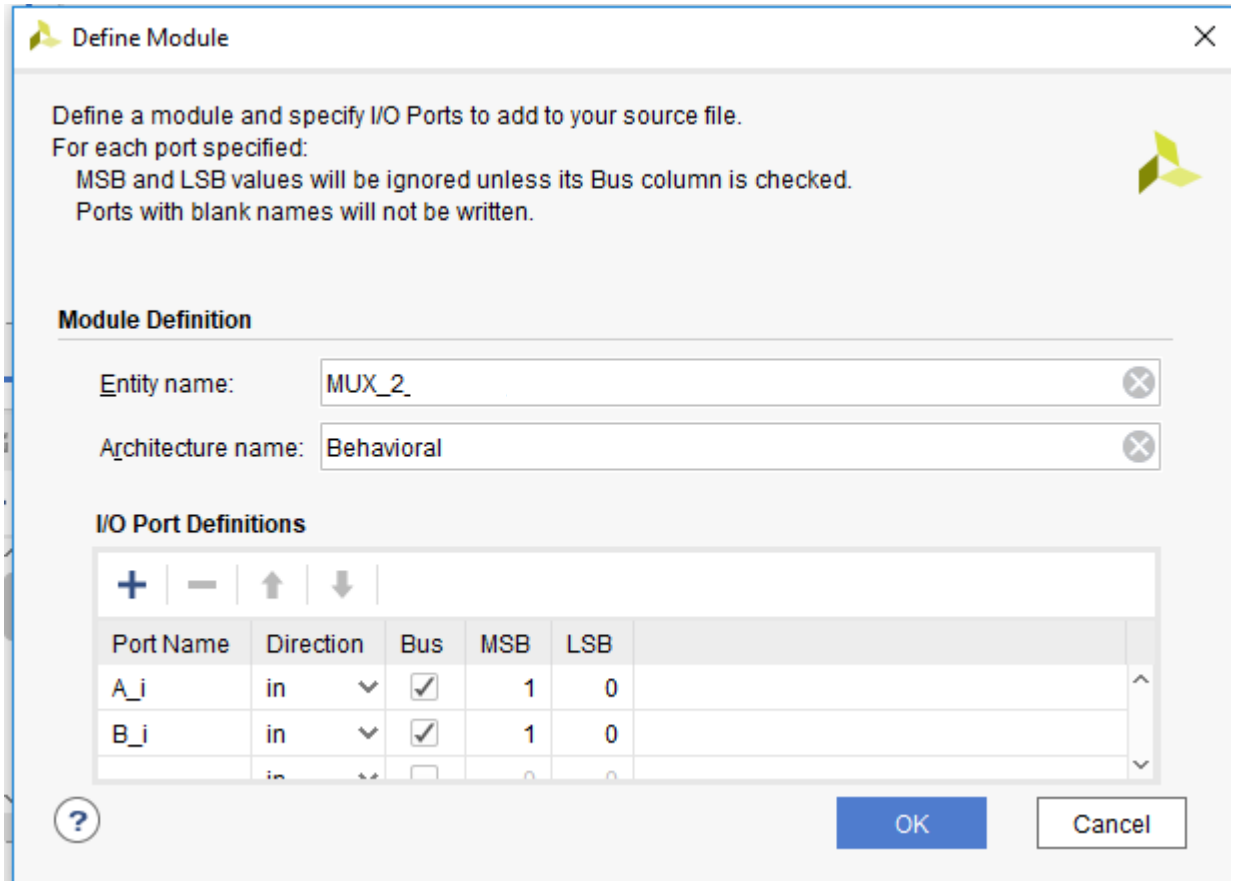
1.2. VHDL sequential signal assignment statements

Project_04. Multiplexor, "**MUX2**".

Now, add a new file to this Project.

Parameterized hardware will be constructed by means a "generic". The data ports size will be modified to achieve a reusable code.

In the new file declare A_i, B_i and Y_o as bus type.



Project_04. Multiplexor, "**MUX2**".

Set the generic with a initial value 2

```
entity MUX2 is
      generic(width: integer:=2);
      Port ( A_i : in STD_LOGIC_VECTOR (width-1 downto 0);
             B_i : in STD_LOGIC_VECTOR (width-1 downto 0);
             SEL_i : in STD_LOGIC;
             Y_o : out STD_LOGIC_VECTOR (width-1 downto 0));
end MUX2;
```

Project_04. Multiplexor, "**MUX2**".

```
architecture behavioral of MUX2 is
begin
-- A description with a process

 process (A_i,B_i,SEL_i)
      begin
            if Sel='0' then
                  Y_o <= A_i;
            else
                  Y_o <= B_i;
            end if;
 end process;

end behavioral;
```

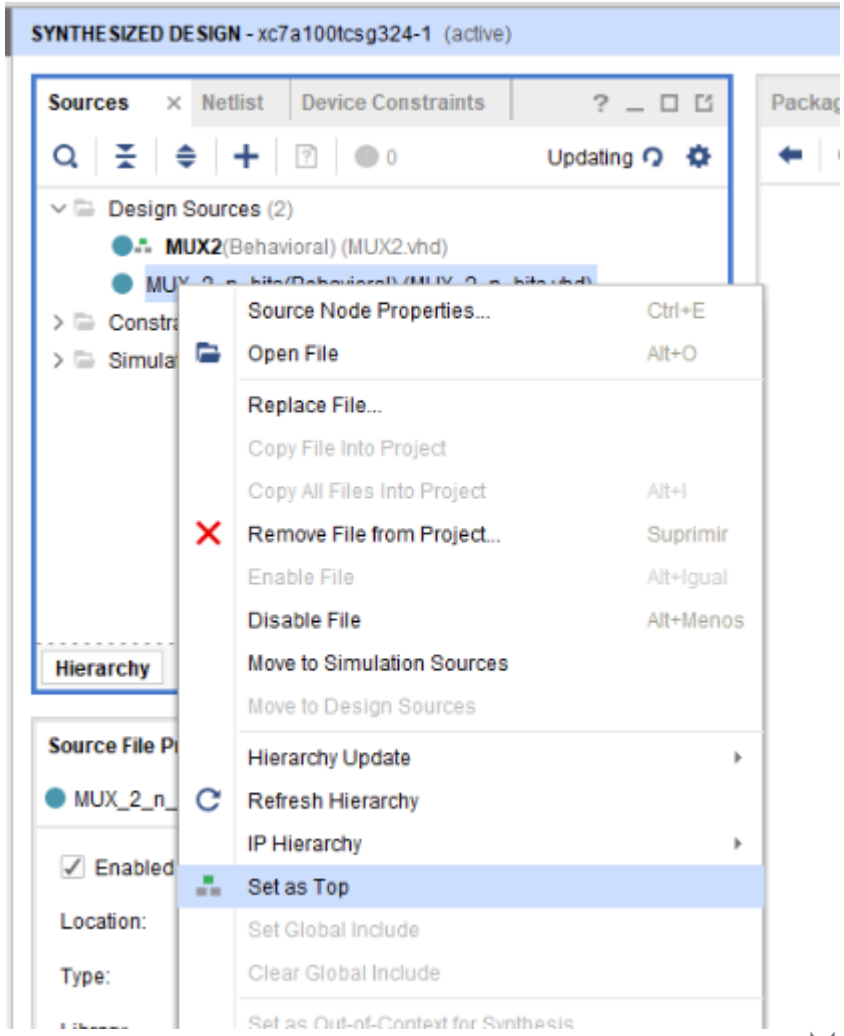
The sensitivity list must include all the input ports. (VHDL'93)

Sequential statements look like a high level language as C

- Staments are executed sequentially in a process
- A process runs concurrently with other processes

Project_04. Multiplexor, "**MUX2**"

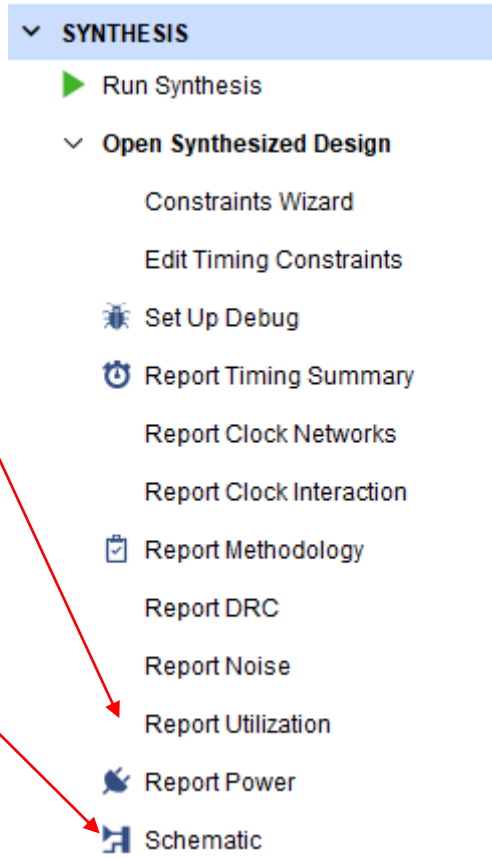
A project has one top module that is the root of the design hierarchy for the purpose of synthesis and implementation.
"Set as a Top" the new file by means of its contextual menu.

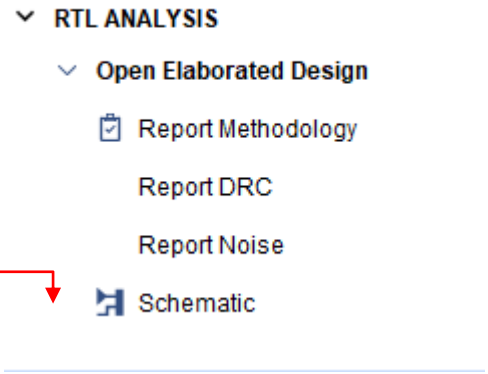


Describing a digital circuit with VHDL

5. Find out the resource usage. (IOBs and *Slices*).

6. Open the schematic in "Open Elaborated Design" and you can see the inferred macro. 7.
7. Open the schematic in "Open Synthesized Design" and see the technology resources used. (LUTs, FF, etc)





Describing a digital circuit with VHDL

8. Open the previously added XDC file and notice the arrays syntax

```
##Switches

#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
```

Port’s name

Port’s index

9. Is the circuit working as expected? Check it on Nexys4

1. 3. Suggested exercises

Exercises

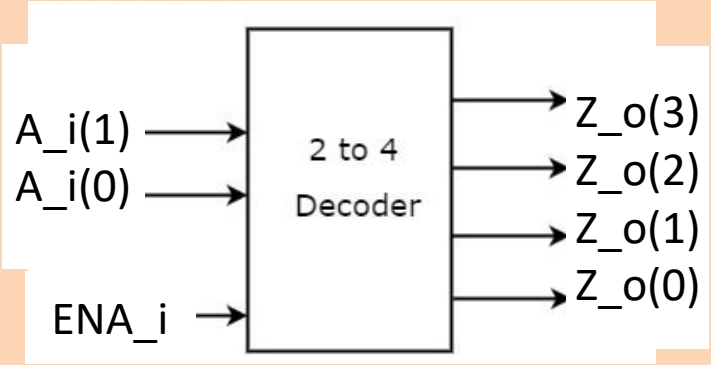
1. Describe a 2 to 4 DECODER with a high-level ENABLE.

   Projects's name: **Project_05**

   Module's name: **DEC_2to4**

   Port's name and resources on NEXYS4

   - **A_i (1:0) → Switches**
   - **ENA_i → Push button**
   - **Z_o (3:0) → LEDs**



Enable	Inputs		Outputs			
ENA_i	A_i(1)	A_i(0)	Z_o(3)	Z_o(2)	Z_o(1)	Z_o(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

Exercises

2. Describe a 1-bit FULL ADDER. Use the boolean expression to describe the architecture:

   SUM_o = (A_i xor B_i) xor CARRY_i

   Carry_o = (A_i and B_i) or (A_i and CARRY_i) or (B_i and CARRY_i)

   Projects's name: Project_06

   Module's name: FULL_ADDER

   Port's name and resources on NEXYS4

   - A_i → Switch
   - B_i → Switch
   - CARRY_i → Switch

   - CARRY_o → LED
   - SUM_o → LEDs

3. Describe a generic 4-to-1 Multiplexer

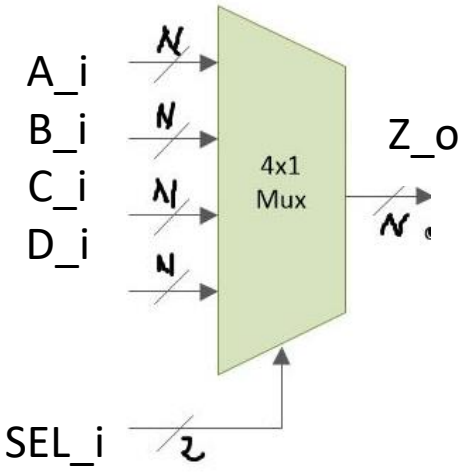
Projects's name: Project_07

Module's name: MUX4

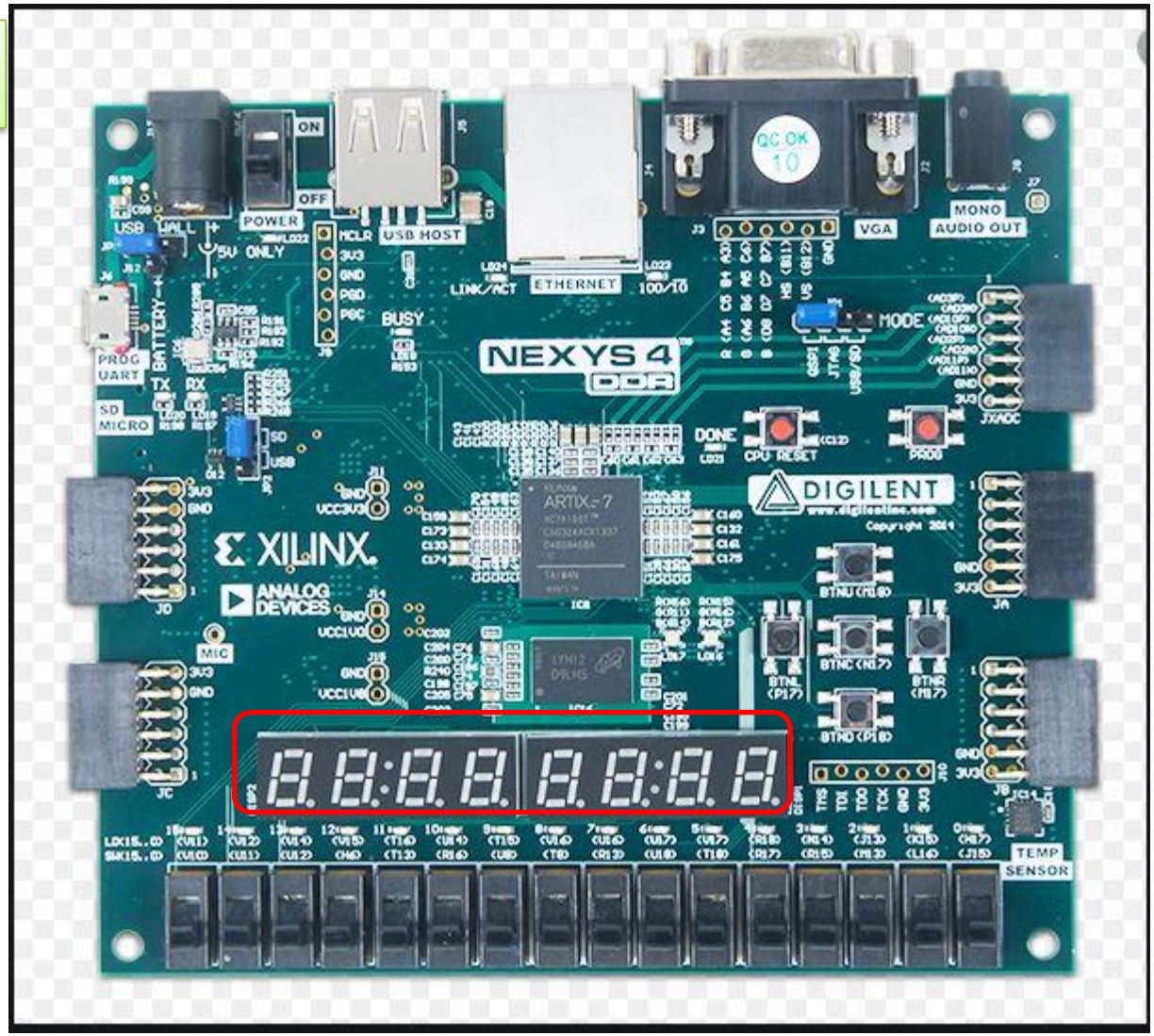
Generic's name: WIDTH

Port's name and resources on NEXYS4

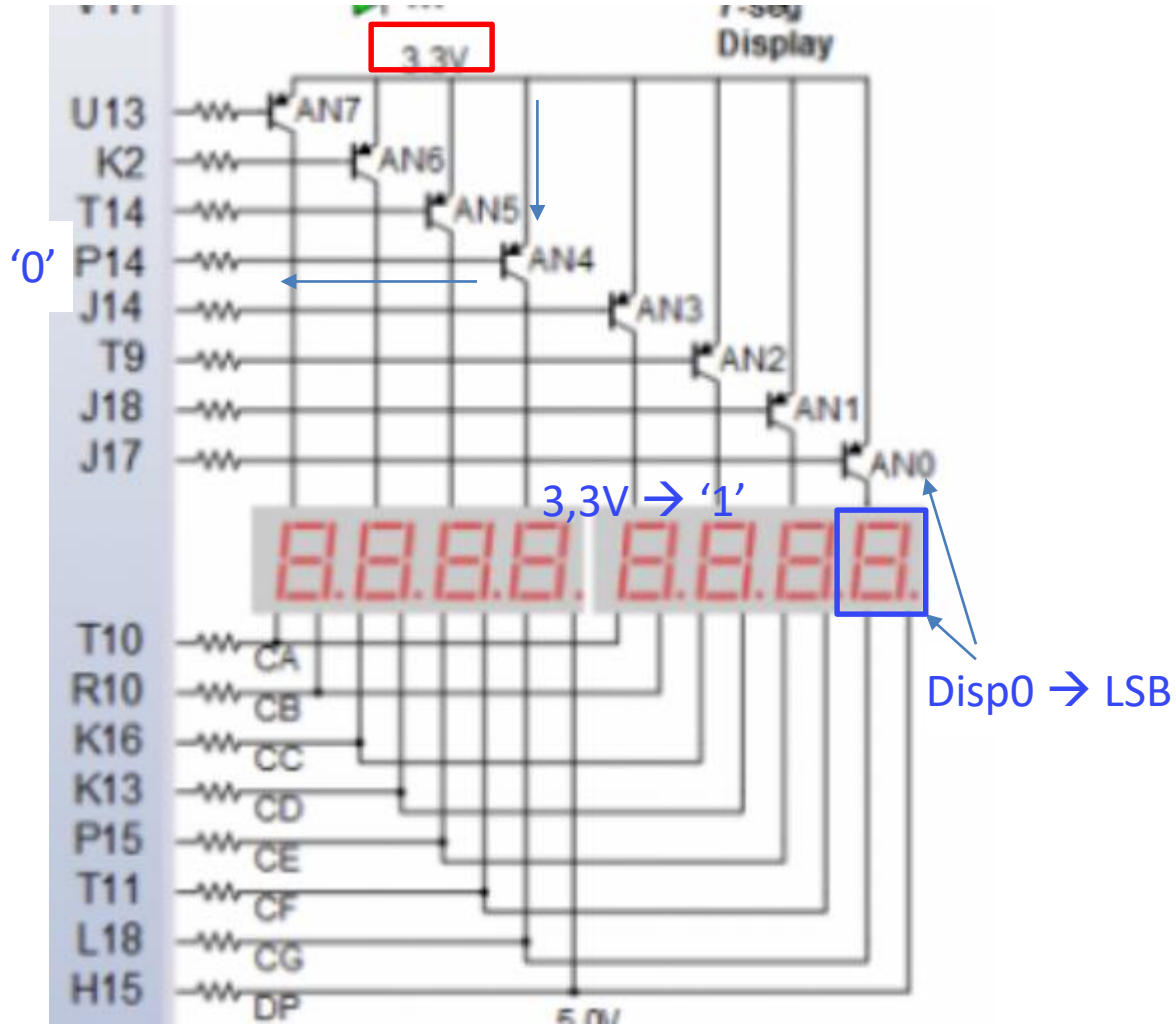
- Inputs : A_i, B_i, C_i, D_i **(**WIDTH-1 **: 0)** → **Switches**
- Outputs Z_o **(**WIDTH-1 **: 0)** → **2 LED**
- Input **SEL_i (1:0)** → Two pushbuttons



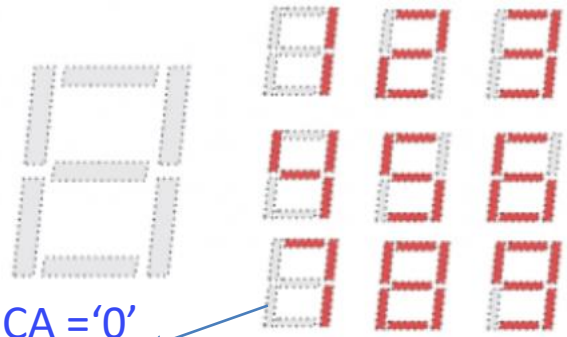
7-segment display



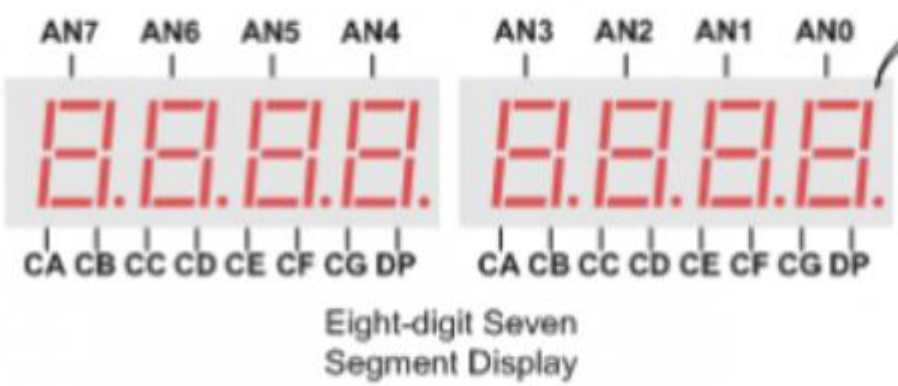
7-segment display

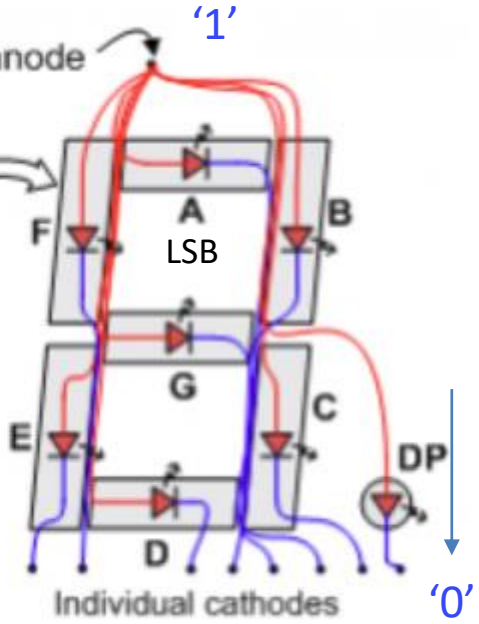


7-segment display



CA ='0'
CB ='0'
CC ='0'





Exercises

4. Design a circuit to show a number according to the hexadecimal value by means of four switches. Two additional switches should allow choosing among the 4 right-most displays.

Projects's name: Project_08

Module's name: DISP7SEG

Port's name and resources on NEXYS4:

- Inputs : BINARY_i, DISPLAY_i → Switches
- Outputs: CATHODE_o (6:0) → 7-segment display Cathodes
  ANODE_o (7:0) → 7-segment display Anodes

Notes:

- Firstly, draw the black-box corresponding to the Entity
- Only concurrent staments should be used
- Get help from the template editor (Following slide)

Exercises

Project Summary | DISP7SEG.vhd *

E:/Nube/OneDrive/Docencia_actual/TDC/TDC_PROJECTS_VIVADO/PROJECT_08/PROJECT_08.srcs/sources_1/new/DISP7SEG.vhd

```
38  --   LED:   out   STD_LOGIC_VECTOR (6 downt
39  --
40  -- segment encoinputg
41  --       0
42  --      ---
43  --  5 |    | 1
44  --      ---    <- 6
45  --  4 |    | 2
46  --      ---
47  --       3
48
49  with BINARY_i SELect
50     CATHODE_O <=  "1111001" when "0001",
51                   "0100100" when "0010",
52                   "0110000" when "0011",
53                   "0011001" when "0100",
54                   "0010010" when "0101",
55                   "0000010" when "0110",
56                   "1111000" when "0111",
57                   "0000000" when "1000",
58                   "0010000" when "1001",
59                   "0001000" when "1010",
60                   "0000011" when "1011",
61                   "1000110" when "1100",
62                   "0100001" when "1101",
63                   "0000110" when "1110",
64                   "0001110" when "1111",
65                   "1000000" when others;
66  end Behavioral;
```

Language Templates

Select a language template

CATHODE_o (6 downto 0)

76543210
GFEDCBA

Templates

- Coding Examples
  - Accumulators
  - Arithmetic
  - Basic Gates
  - Bi-directional I/O
  - Comparators
  - Counters
  - Decoders
  - DSP
  - Encoders
  - Flip Flops
  - Logical Shifters
  - Misc
    - 7-Segment Display Hex Conv
    - Asynchronous Input Synchron
    - Barrel Shifter

Preview

```
13  --
14
15    with HEX SELect
16   LED<= "1111001" when "0001",   --1
17         "0100100" when "0010",   --2
18         "0110000" when "0011",   --3
19         "0011001" when "0100",   --4
20         "0010010" when "0101",   --5
21         "0000010" when "0110",   --6
22         "1111000" when "0111",   --7
23         "0000000" when "1000",   --8
24         "0010000" when "1001",   --9
25         "0001000" when "1010",   --A
26         "0000011" when "1011",   --b
27         "1000110" when "1100",   --C
28         "0100001" when "1101",   --d
29         "0000110" when "1110",   --E
30         "0001110" when "1111",   --F
31         "1000000" when others;   --0
32
33
34
35
```

Copy & Paste

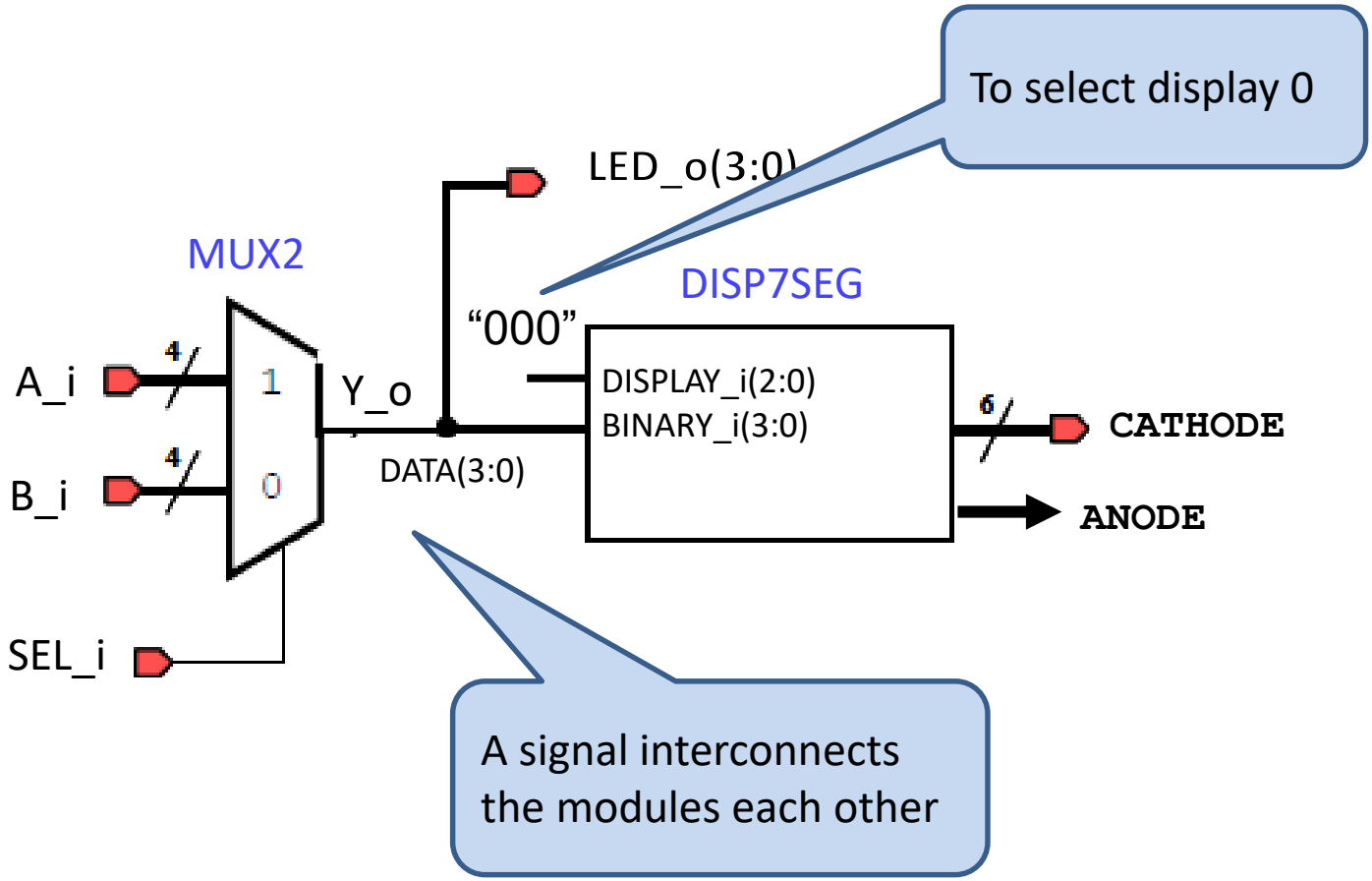
Close

TNS | WHS | THS | TPWS | Total Power | Failed Routes | LUT | FF | BRAMs | URAM | DSP | LUTRAM | IO | GT | BUFG | MMCM | PLL | PCIe | Start | Elapsed | Run Strategy

1.4. Structural design

Structural design

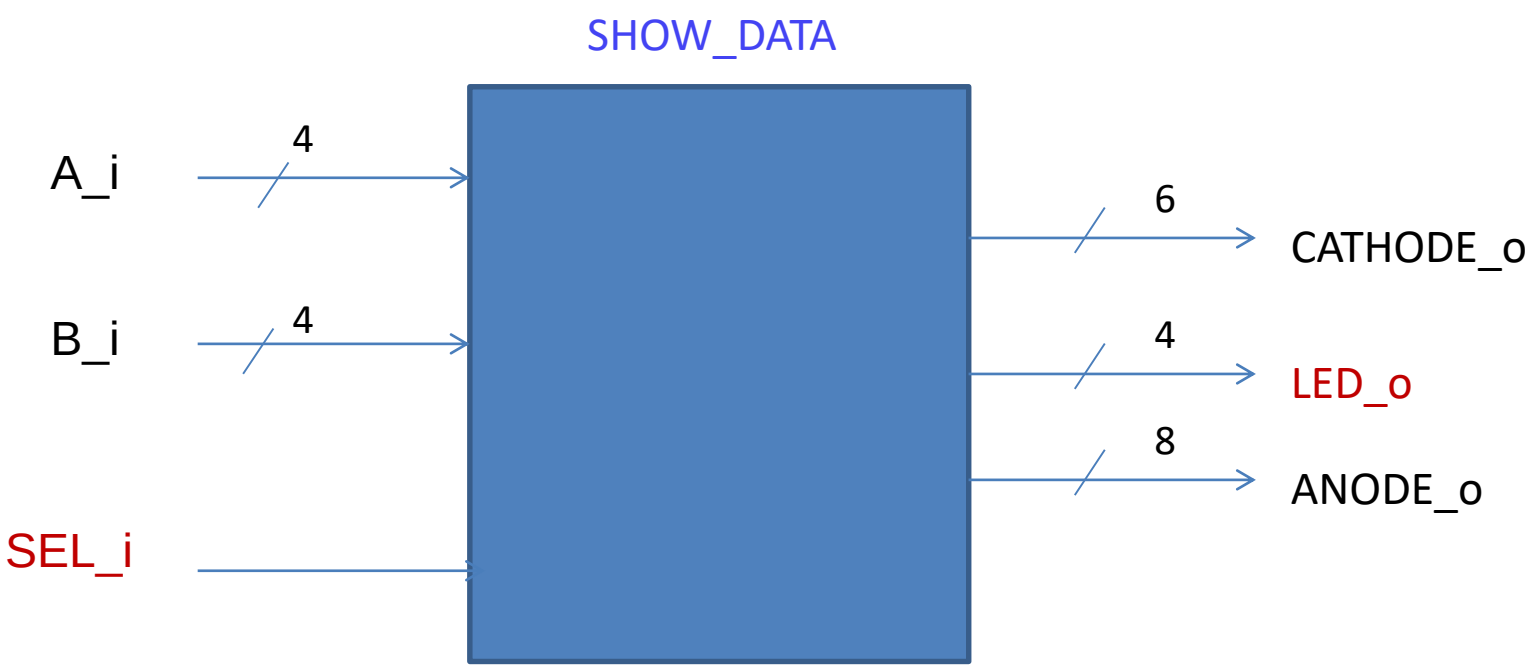
Project_09. (SHOW_DATA). Design a circuit to display a data selected by a MUX onto a 7-segment display and in a set of LED set. The data width is 4. Notice that the SEL_i input only chooses what input be shown, not the display to show it. Only the right-most display will be used.



Structural design

Project_09. (SHOW_DATA). Design a circuit to display a data selected by a MUX onto a 7-segment display and in a set of LED. The data width is 4.

Step 1. Draw a black-box to depicte the entity.



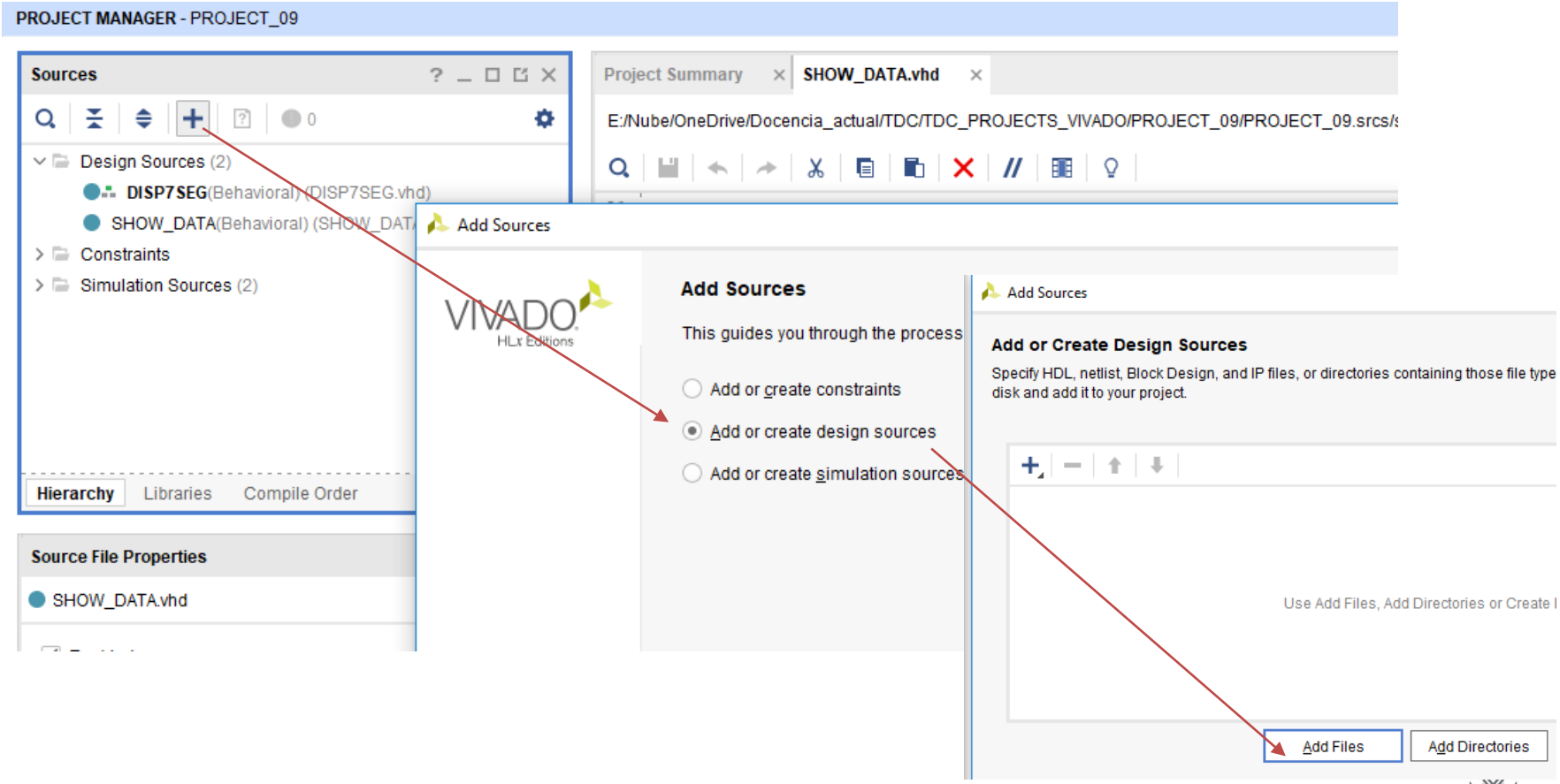
Project_09. Design a circuit to display a data selected by a MUX onto a 7-segment display and in a set of LED. The data width is 4. (SHOW_DATA)

Step 1. By means VHDL describe the entity

```
entity SHOW_DATA is
 generic (WIDTH_TOP: integer:=4);
  Port ( A_i : in STD_LOGIC_VECTOR (WIDTH_TOP-1 downto 0);
         B_i : in STD_LOGIC_VECTOR (WIDTH_TOP-1 downto 0);
         SEL_O : in STD_LOGIC;
         CATHODE_o : out STD_LOGIC_VECTOR (6 downto 0);
         ANODE_O : out STD_LOGIC_VECTOR (3 downto 0);
         LED_o : out STD_LOGIC_VECTOR (WIDTH_TOP-1 downto 0));
       end SHOW_DATA;
```

Step 2. Add the modules to reuse in the top design:

- MUX2
- DISP7SEG



Structural design

Step 3. Declare the added modules as components in the SHOW_DATA architecture.

The modules to reuse must be declare as components in the architecture of the top level module, i.e. “SHOW_DATA”, between “architecture” and “begin” words.

```
architecture Behavioral of SHOW_DATA is
-- List of components
component MUX2
    generic(width: integer:=2);
    Port ( A_i : in STD_LOGIC_VECTOR (width-1 downto 0);
           B_i : in STD_LOGIC_VECTOR (width-1 downto 0);
           SEL_i : in STD_LOGIC;
           Y_o : out STD_LOGIC_VECTOR (width-1 downto 0));
end component;

entity DISP7SEG is
    Port ( BINARY_i : in STD_LOGIC_VECTOR (3 downto 0);
           DISPLAY_I : in STD_LOGIC_VECTOR (2 downto 0);
           CATHODE_O : out STD_LOGIC_VECTOR (6 downto 0);
           ANODE_O : out STD_LOGIC_VECTOR (7 downto 0));
end DISP7SEG;

begin


end Behavioral;
```

TIP: Copy the entity an replace/delete the underlined words as you can see at the image.

Structural design

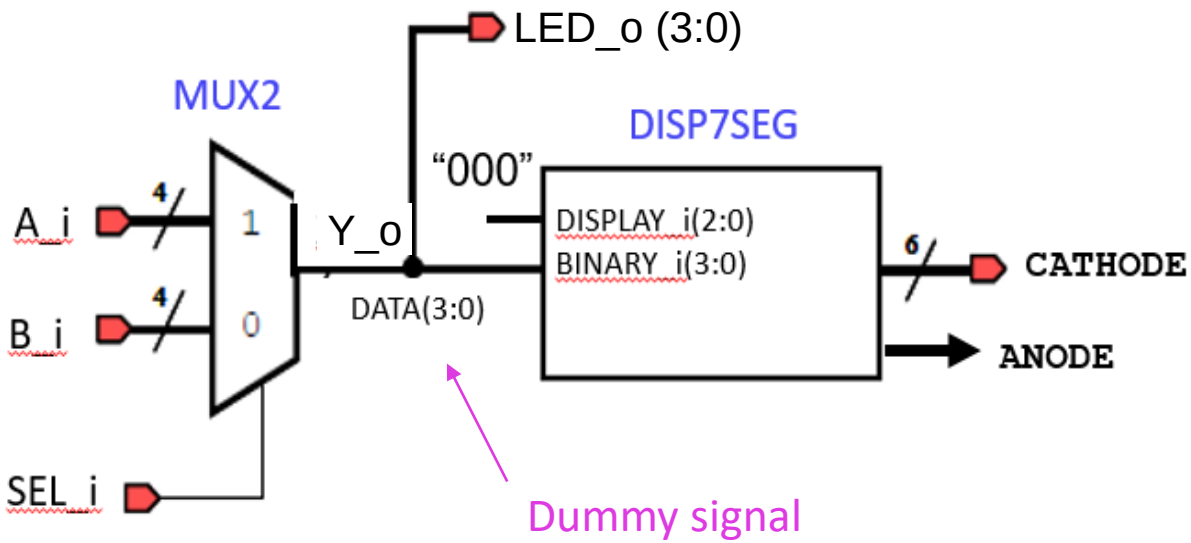
```
architecture Behavioral of SHOW_DATA is
-- List of components
component DISP7SEG
Port ( BINARY_i : in STD_LOGIC_VECTOR (3 downto 0);
       DISPLAY_I : in STD_LOGIC_VECTOR (2 downto 0);
       CATHODE_O : out STD_LOGIC_VECTOR (6 downto 0);
       ANODE_O : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component MUX2
 generic(WIDTH: integer);
Port ( A_i : in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
       B_i : in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
       SEL_i : in STD_LOGIC;
       Y_o : out STD_LOGIC_VECTOR (WIDTH-1 downto 0));
end component;
```

Structural design

Project_09. Design a circuit to display a data selected by a MUX onto a 7-segment display and in a set of LED. The data width is 4. (SHOW_DATA)

Step 4. Declare the signals to interconnect the modules

```
-- Signal to connect both modules
signal DATA: STD_LOGIC_VECTOR(WIDTH_TOP-1 downto 0);
```



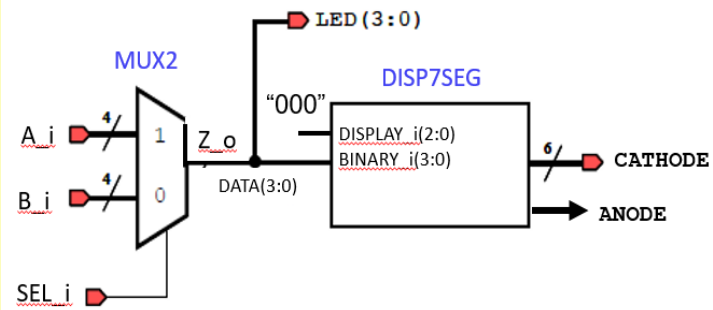
Structural design

Step 5. Instance and map the components as times as you want. The instances are added into the architecture, after "begin".

```
begin
MUX2_0 : MUX2
 generic map ( WIDTH => WIDTH_TOP )
 port map (
      A_i => A_i,
      B_i => B_i,
      SEL_i => SEL_i,
      Y_o => DATA
);
DISP7SEG_0 : DISP7SEG
 port map (
      BINARY_i => DATA,
      DISPLAY_I => "000",
      CATHODE_O => CATHODE_o,
      ANODE_O => ANODE_o
 );
end Behavioral;
```

Ports of the component

Ports and signals in the new structural design



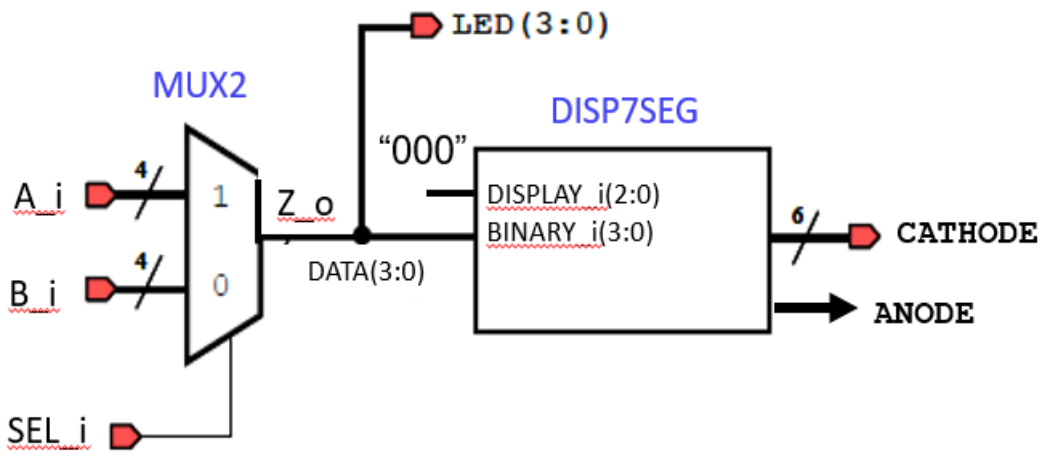
Signal to connect both modules

Select a specific DISPLAY

Structural design

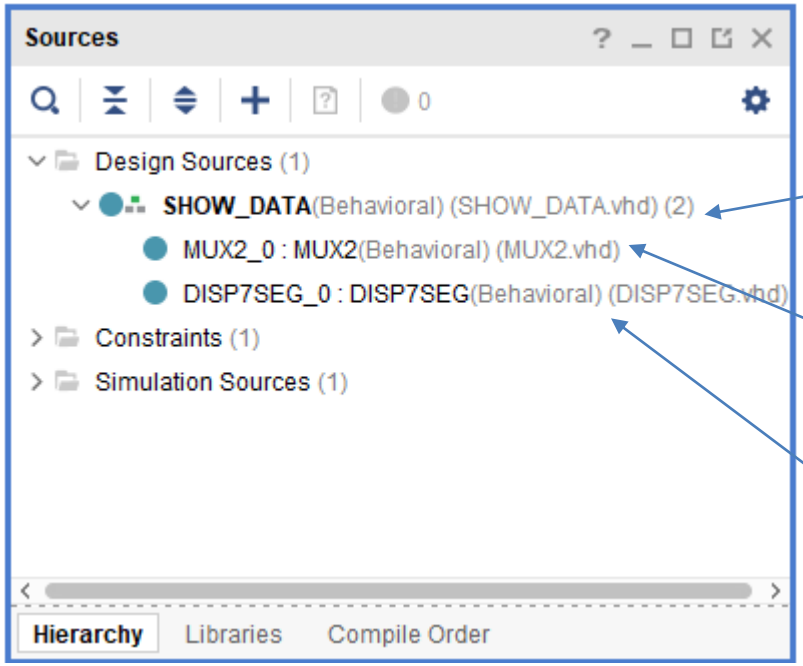
Step 6: If it is necessary, assign signals to output ports.

```
…

LED_o <= DATA;

end Behavioral;
```



Structural design

Step 7: After the synthesis the sources pane will show the components hierarchy



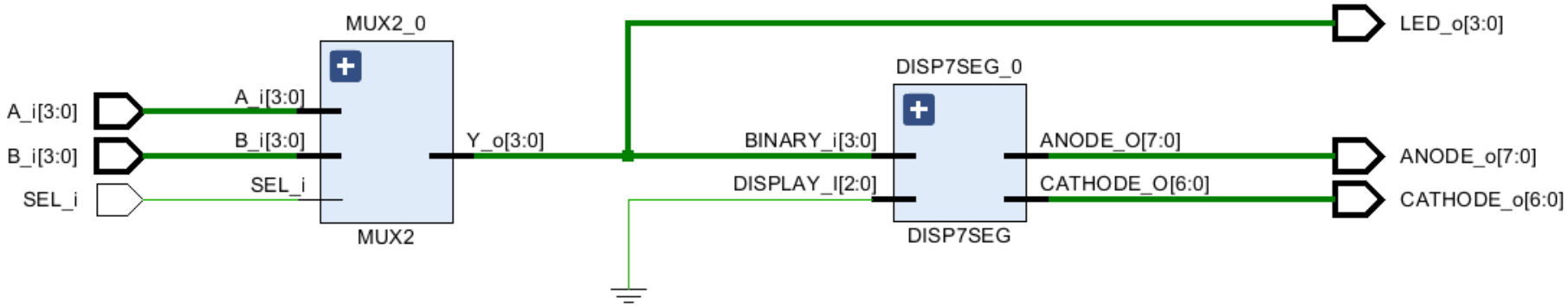
Top Module

Instance of component MUX

Instance of DISP7SEG

Structural design

<u>Step 8</u>: By means of "Open Elaborated Dsign → Schematics" check the structural schematic



Structural design

- **PROJECT_09.** Add the XDC file and check the performance is correct on Nexys4.
- **PROJECT_09.** Find out how many LUTs and Slices are used.
- **PROJECT_09.** Open the Schematic after Synthesis and watch the LUTs and Slices in the design

1.4.1. Structural design exercises

Exercises

PROJECT_10. Design a 1-bit ALU in a structural style.
Operations: AND, XOR, Full ADD, Move A.

To ease the design create a new user library following the tutorial included at Virtual Campus.

Project's name: **PROJECT_10**
Top Module's name: **ALU_1bit**
Resources from Nexys4:

- Inputs: **A_i, B_i, CARRY_i → Switches**
- Operation select input: **OP_i(1:0) → Switches**
- Outputs: **CARRY_o, RESULT_o → LED**

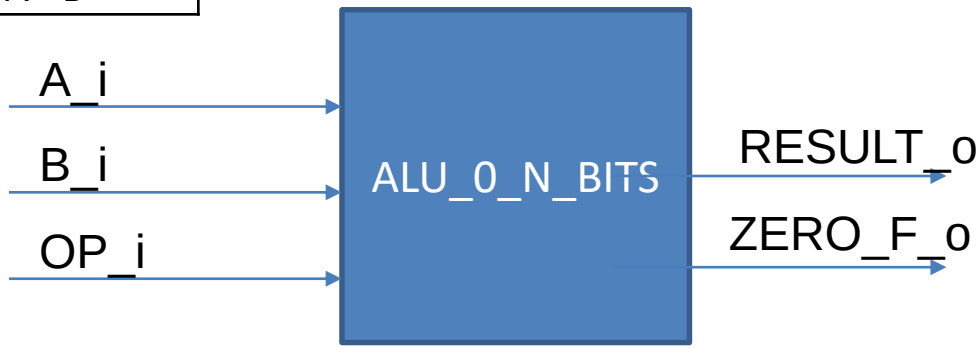
1.5. Unsigned and Signed datatypes. Arithmetic operations

Exercises: n-bits ALU

PROJECT_11. Design a generic ALU to perform the operations shown in the below table (ALU_0_N_BITS). A zero-flag must be asserted with a zero result.

Describe the ALU in **behavioral** style, therefore, only one VHDL file should exist in the project. Use **concurrent** statements.

OP(1)	OP(0)	Function
0	0	MOV A
0	1	INC A
1	0	A + B
1	1	A - B



Exercises: n-bits ALU

PROJECT_11. Add a new file to the project. Design a generic ALU to perform the operations shown in the below table (ALU_1_N_BITS). A **Zero-flag** must be asserted with a zero result, also a **Carry-flag** should be included.

- Describe the ALU in **behavioral** style, only one VHDL file
- Use **sequential** statements (Process and CASE).

OP(2)	OP(1)	OP(0)	Function
0	0	0	INC A
0	0	1	INC B
0	1	0	SRL A 1 bit
0	1	1	SLL A 1 bit
1	0	0	A+B
1	0	1	A-B
1	1	0	MovA
1	1	1	MovB

Exercises: n-bits ALU

PROJECT_12. Design a generic DECODER **N** to **2**N**. Describe the circuit in a **behavioral** style with sequential statements and use constants.