Management and analysis of physics datasets, Part. 1

Second Laboratory

Antonio Bergnoli bergnoli@pd.infn.it

9/11/2021

Laboratory Introduction

Goals

- Exploit the simulation tool.
- Become familiar with components and $port\ map$.

VHDL naming convention

Signals/components	Name
Clock	clk
Reset	rst
Input Port	$port\_in$
Output Port	$port\_out$
VHDL file name	$entityname.vhd$
Test bench file name	$tb\_entityname.vhd$
Signal between 2 comps	$sign\_cmp1\_cmp2$
...	...

Simulation - Testbenches

Simulation - reasons

- Once written the code describing the component, it is necessary check its working.
- Regarding on not trivial projects, simulation is a crucial and mandatory step in the development of the project: testing any firmware on hardware without simulation has to be avoided because it is unsafe and very time consuming.
- So with the simulation we are going to stimulate the inputs and check the accuracy of the outputs.

Simulation and VHDL(1)

What VHDL is designed for:

- Documentation: Formal description of a digital design;
- Synthesis: to **Infer** digital logic and digital sequential structures specifying a formal description;
- Verification: to **assert** that a design will behave correctly **before** implement it.

Simulation and VHDL(2)

- Only a **subset** of the language constructs are synthesizables: (IEEE 1076.6)
- All the language contstructs are allowed in simulation.

Check the following examples.

Testbench (1)

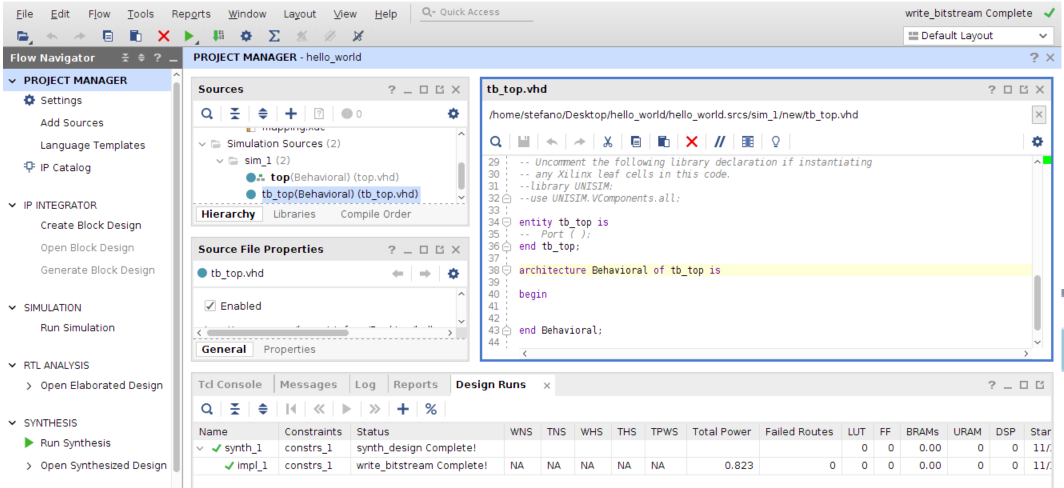
- To simulate the behavior of the code written, we have to make a VHDL file, in order to define the input stimuluses and so check the output evolution.
- This file is a **testbench**.
- The component to be tested (the ”Hello World” top) has to be instantiated and the processes where the input signals values are described have to be written.
- The testbench dœs not have input and output ports in its entity declaration.

Testbench (2)

1. Add sources;
2. Check ”Add or create simulation sources”;
3. Next $\rightarrow$;
4. Check ”Create File”;
5. File name: ”tb_top”;
6. OK $\rightarrow$.
7. Finish $\rightarrow$;
8. OK $\rightarrow$.
9. Yes $\rightarrow$.

Testbench (3) Simulation Sources → sim_1 →

tb_top.

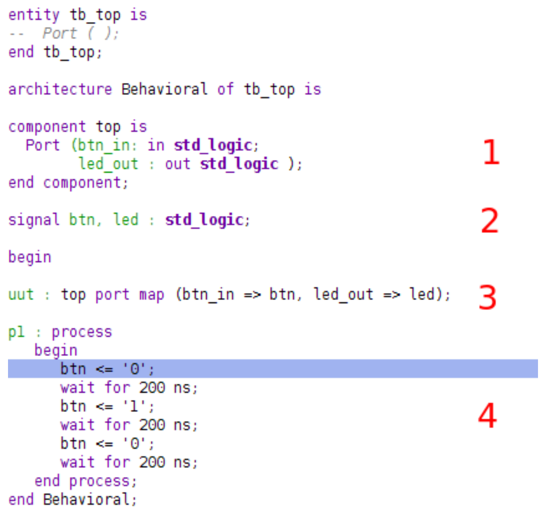


/home/stefano/Desktop/hello_world/hello_world.srcs/sim_1/new/tb_top.vhd

```vhdl
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity tb_top is
--  Port ( );
end tb_top;

architecture Behavioral of tb_top is

begin


end Behavioral;
```

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Star
✓ synth_1	constrs_1	synth_design Complete!								0	0	0.00	0	0	11/
✓ impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA	NA	0.823	0	0	0	0.00	0	0	11/

Testbench (4)



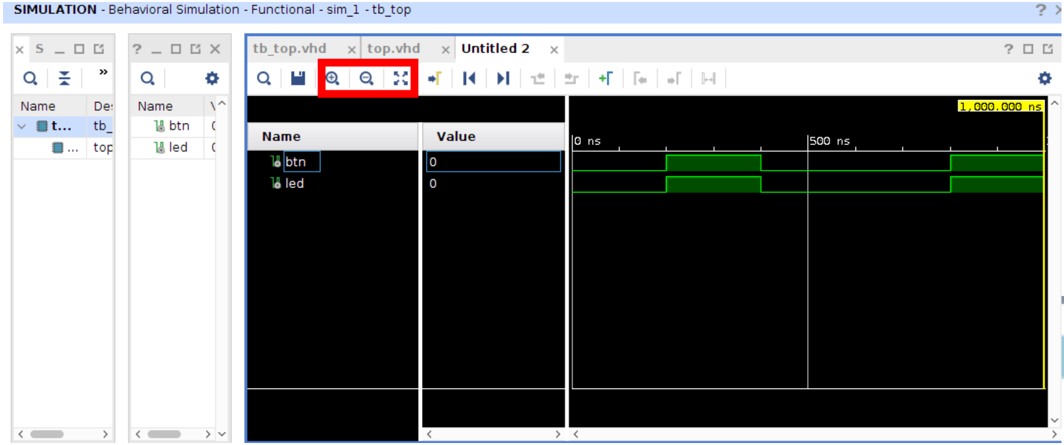
```
entity tb_top is
--  Port ( );
end tb_top;

architecture Behavioral of tb_top is

component top is
  Port (btn_in: in std_logic;
        led_out : out std_logic );                   1
end component;

signal btn, led : std_logic;                         2

begin

uut : top port map (btn_in => btn, led_out => led);  3

pl : process
   begin
      btn <= '0';
      wait for 200 ns;
      btn <= '1';                                    4
      wait for 200 ns;
      btn <= '0';
      wait for 200 ns;
   end process;
end Behavioral;
```

Testbench (5)

1. Declaration of the component to test. Suggest: copy/paste from top.vhd the piece of the code between *entity* and *end top*; and substitute *component* for *entity*.
2. Declaration of the internal signals. They are used to connect the component under test (**U**nit **U**nder **T**est) to the stimulus.
3. Instantiation of the component. Each port is connected through the internal signals.
4. Definition of the input stimuluses. It is very very very good practice check each input combination.

Simulation (1)

SIMULATION $\rightarrow$ Run Simulation $\rightarrow$ Run Behavioral Simulation.


SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_top

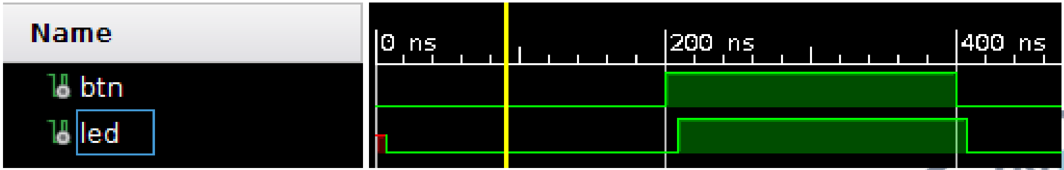
Inside the red rectangle there are the zoom options

Simulation (2)

- With the Behavioral Simulation we check that the *component* works as expected.
- The output changes happen simultaneously to the input changes. This kind of simulation check the behavior of the code written without considering the delays of a real electronic system.

Simulation (3)

- With the same testbench used before we are going to check the behavior of the component afterward the implementation step. Here the simulation evaluates the logic gates delay and the interconnections delays;
- SIMULATION → Run Simulation → Run Post-Implementation Timing Simulation → Yes;



If this simulation is successfully, almost always the project works in the hardware too.

Combinational Logic Circuits

Reusability of the code

- The next examples show you as re-use the code already written.
- The code in this course is reused in the form of **components**.
- The components are then located in the top code.
- You'll can see a construction of a hierarchical design.

How to use a component?

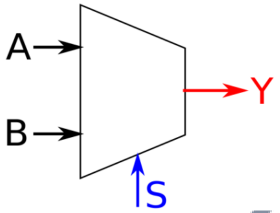
1. Declaration;
2. Instantiation.

Multiplexer

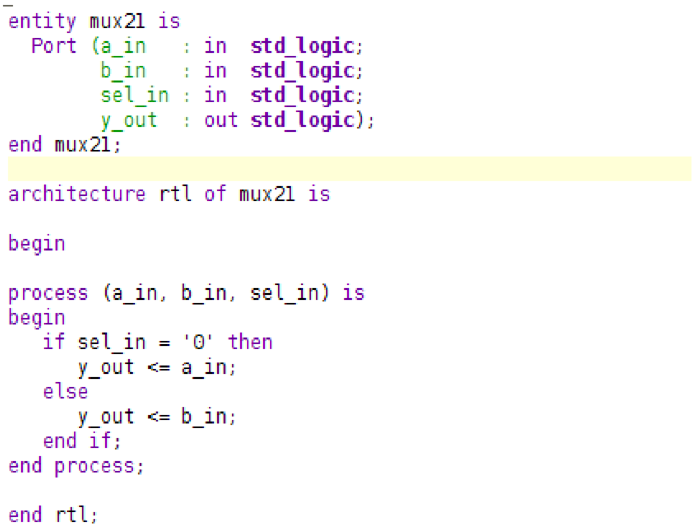
Multiplexer 2-1

A	B	S	Y
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

S	Y
0	A
1	B

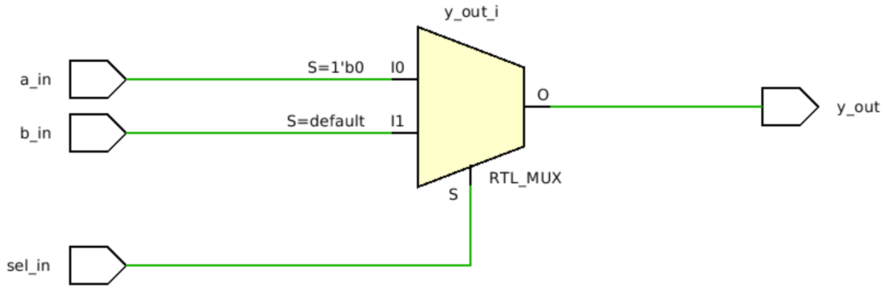


Mux 2-1 Code

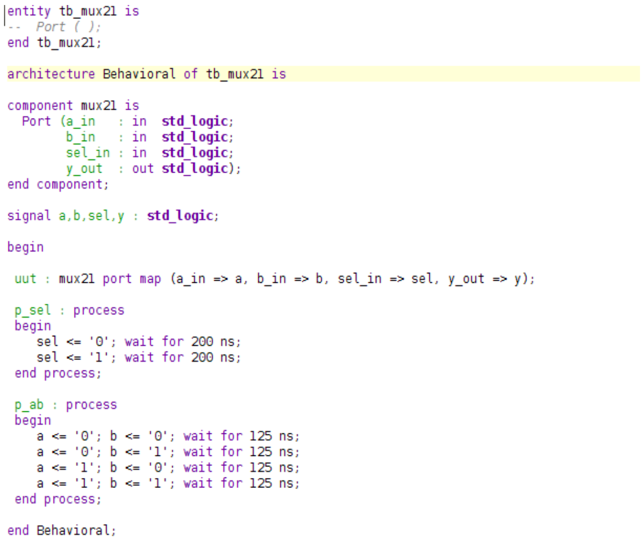


```vhdl
entity mux21 is
  Port (a_in   : in  std_logic;
        b_in   : in  std_logic;
        sel_in : in  std_logic;
        y_out  : out std_logic);
end mux21;

architecture rtl of mux21 is

begin

process (a_in, b_in, sel_in) is
begin
   if sel_in = '0' then
      y_out <= a_in;
   else
      y_out <= b_in;
   end if;
end process;

end rtl;
```

Mux 2-1 Schematic

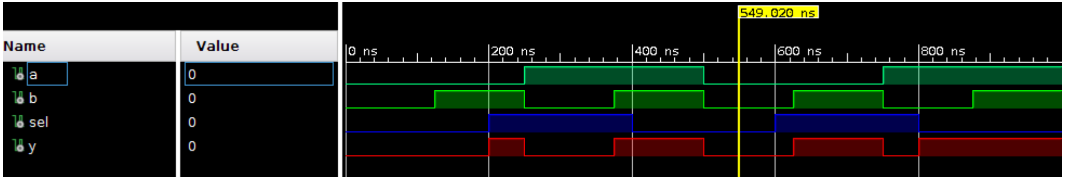


Mux 2-1 Testbench



```vhdl
entity tb_mux21 is
--  Port ( );
end tb_mux21;

architecture Behavioral of tb_mux21 is

component mux21 is
  Port (a_in   : in  std_logic;
        b_in   : in  std_logic;
        sel_in : in  std_logic;
        y_out  : out std_logic);
end component;

signal a,b,sel,y : std_logic;

begin

 uut : mux21 port map (a_in => a, b_in => b, sel_in => sel, y_out => y);

 p_sel : process
 begin
    sel <= '0'; wait for 200 ns;
    sel <= '1'; wait for 200 ns;
 end process;

 p_ab : process
 begin
    a <= '0'; b <= '0'; wait for 125 ns;
    a <= '0'; b <= '1'; wait for 125 ns;
    a <= '1'; b <= '0'; wait for 125 ns;
    a <= '1'; b <= '1'; wait for 125 ns;
 end process;

end Behavioral;
```

Mux 2-1 Behavioral Simulation

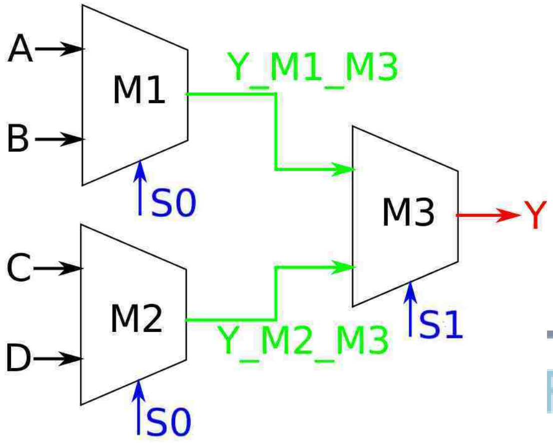


Multiplexer 4-1 (1)

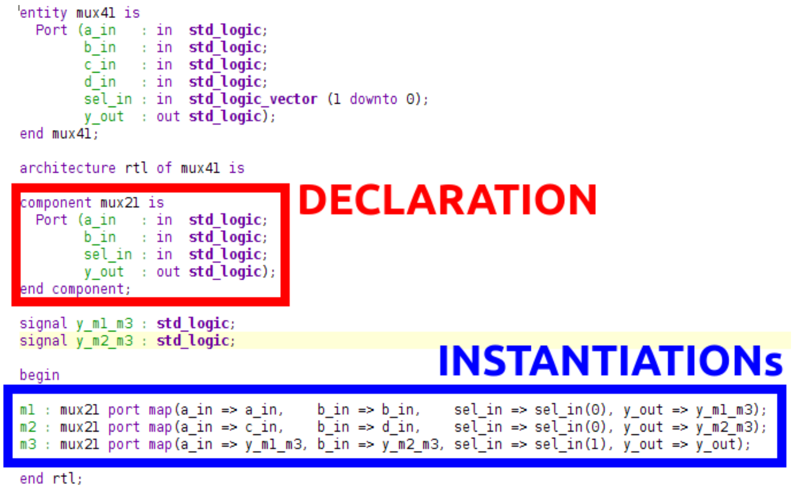
A	B	C	D	S0	S1	Y	A	B	C	D	S0	S1	Y
0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	1	1	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0	0	1	0
1	1	0	0	0	0	1	1	1	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1	0	0	1	1
1	0	1	0	0	0	1	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0	1	1	0	0	1	1
1	1	1	0	0	0	1	1	1	1	0	0	1	1
0	0	0	1	0	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	1	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0	1	0	1	0	1	0
1	1	0	1	0	0	1	1	1	0	1	0	1	0
0	0	1	1	0	0	0	0	0	1	1	0	1	1
1	0	1	1	0	0	1	1	0	1	1	0	1	1
0	1	1	1	0	0	0	0	1	1	1	0	1	1
1	1	1	1	0	0	1	1	1	1	1	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	1	0	0	1	0	0	0	1	1	0
0	1	0	0	1	0	1	0	1	0	0	1	1	0
1	1	0	0	1	0	1	1	1	0	0	1	1	0
0	0	1	0	1	0	0	0	0	1	0	1	1	0
1	0	1	0	1	0	0	1	0	1	0	1	1	0
0	1	1	0	1	0	1	0	1	1	0	1	1	0
1	1	1	0	1	0	1	1	1	1	0	1	1	0
0	0	0	1	1	0	0	0	0	0	1	1	1	1
1	0	0	1	1	0	0	1	0	0	1	1	1	1
0	1	0	1	1	0	1	0	1	0	1	1	1	1
1	1	0	1	1	0	1	1	1	0	1	1	1	1
0	0	1	1	1	0	0	0	0	1	1	1	1	1
1	0	1	1	1	0	0	1	0	1	1	1	1	1

Multiplexer 4-1 (2)

S0	S1	Y
0	0	A
1	0	B
0	1	C
1	1	D

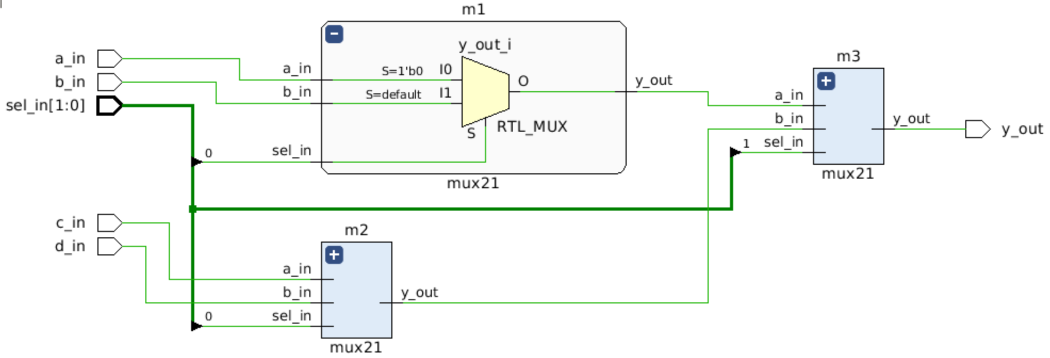


Mux 4-1 Code

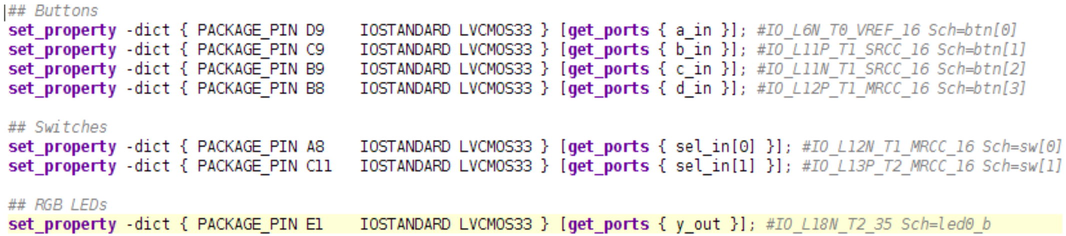


```vhdl
entity mux41 is
  Port (a_in   : in  std_logic;
        b_in   : in  std_logic;
        c_in   : in  std_logic;
        d_in   : in  std_logic;
        sel_in : in  std_logic_vector (1 downto 0);
        y_out  : out std_logic);
end mux41;

architecture rtl of mux41 is

component mux21 is
  Port (a_in   : in  std_logic;
        b_in   : in  std_logic;
        sel_in : in  std_logic;
        y_out  : out std_logic);
end component;

signal y_m1_m3 : std_logic;
signal y_m2_m3 : std_logic;

begin

m1 : mux21 port map(a_in => a_in,    b_in => b_in,    sel_in => sel_in(0), y_out => y_m1_m3);
m2 : mux21 port map(a_in => c_in,    b_in => d_in,    sel_in => sel_in(0), y_out => y_m2_m3);
m3 : mux21 port map(a_in => y_m1_m3, b_in => y_m2_m3, sel_in => sel_in(1), y_out => y_out);

end rtl;
```

Mux 4-1 Schematic



Mux 4-1 Constraints

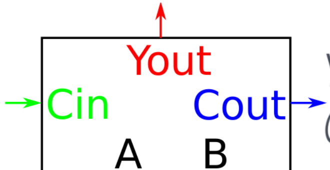


```
## Buttons
set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { a_in }]; #IO_L6N_T0_VREF_16 Sch=btn[0]
set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports { b_in }]; #IO_L11P_T1_SRCC_16 Sch=btn[1]
set_property -dict { PACKAGE_PIN B9    IOSTANDARD LVCMOS33 } [get_ports { c_in }]; #IO_L11N_T1_SRCC_16 Sch=btn[2]
set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33 } [get_ports { d_in }]; #IO_L12P_T1_MRCC_16 Sch=btn[3]

## Switches
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { sel_in[0] }]; #IO_L12N_T1_MRCC_16 Sch=sw[0]
set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 } [get_ports { sel_in[1] }]; #IO_L13P_T2_MRCC_16 Sch=sw[1]

## RGB LEDs
set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 } [get_ports { y_out }]; #IO_L18N_T2_35 Sch=led0_b
```

Adder

Adder 1 bit

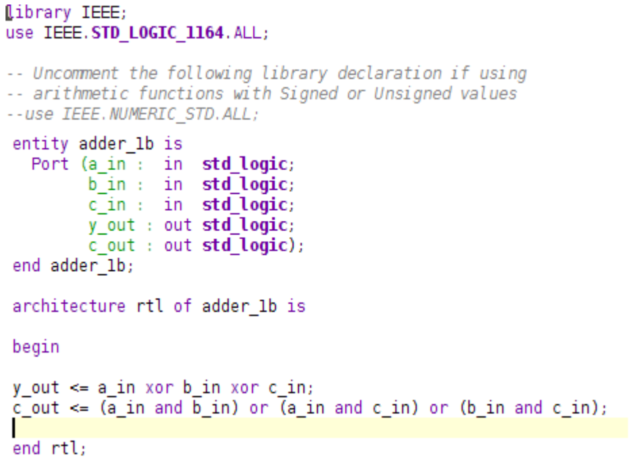
A	B	Cin	Yout	Cout	
0	0	0	0	0	*0+0+0 = 0*
1	0	0	1	0	*1+0+0 = 1*
0	1	0	1	0	*0+1+0 = 1*
1	1	0	0	1	*1+1+0 = 2*
0	0	1	1	0	*0+0+1 = 1*
1	0	1	0	1	*1+0+1 = 2*
0	1	1	0	1	*0+1+1 = 2*
1	1	1	1	1	*1+1+1 = 3*



$$Y_{OUT} = A \text{ xor } B \text{ xor } C_{in}$$

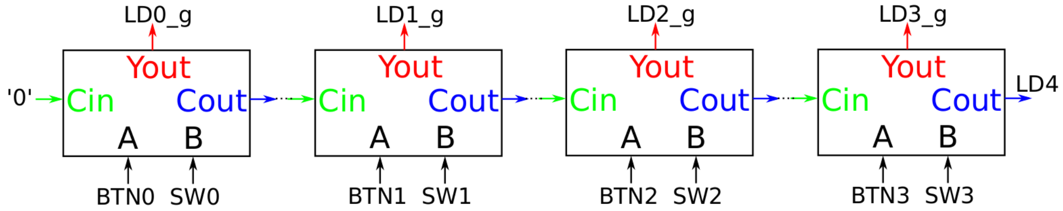
$$C_{OUT} = (A \text{ and } B) \text{ or } (A \text{ and } C_{in}) \text{ or } (B \text{ and } C_{in})$$

Add1b code



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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity adder_1b is
  Port (a_in :  in  std_logic;
        b_in :  in  std_logic;
        c_in :  in  std_logic;
        y_out : out std_logic;
        c_out : out std_logic);
end adder_1b;

architecture rtl of adder_1b is

begin

y_out <= a_in xor b_in xor c_in;
c_out <= (a_in and b_in) or (a_in and c_in) or (b_in and c_in);

end rtl;
```

Adder 4 bit


LD0_g
LD1_g
LD2_g
LD3_g
Yout
Yout
Yout
Yout
'0'
Cin
Cout
Cin
Cout
Cin
Cout
Cin
Cout
LD4
A
B
A
B
A
B
A
B
BTN0
SW0
BTN1
SW1
BTN2
SW2
BTN3
SW3

Add4b Code

```
entity adder_4b is
  Port (a_in :  in  std_logic_vector(3 downto 0);
        b_in :  in  std_logic_vector(3 downto 0);
        y_out : out std_logic_vector(4 downto 0) );
end adder_4b;

architecture rtl of adder_4b is

component adder_1b is
  Port (a_in :  in  std_logic;
        b_in :  in  std_logic;
        c_in :  in  std_logic;
        y_out : out std_logic;
        c_out : out std_logic);
end component;
```

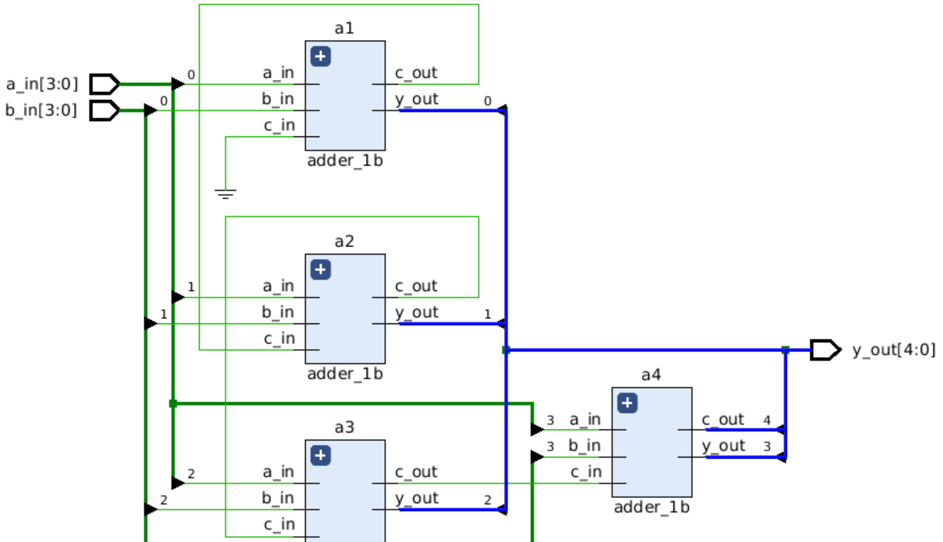
DECLARATION

```
signal y_a1_a2, y_a2_a3, y_a3_a4 : std_logic;

begin
```

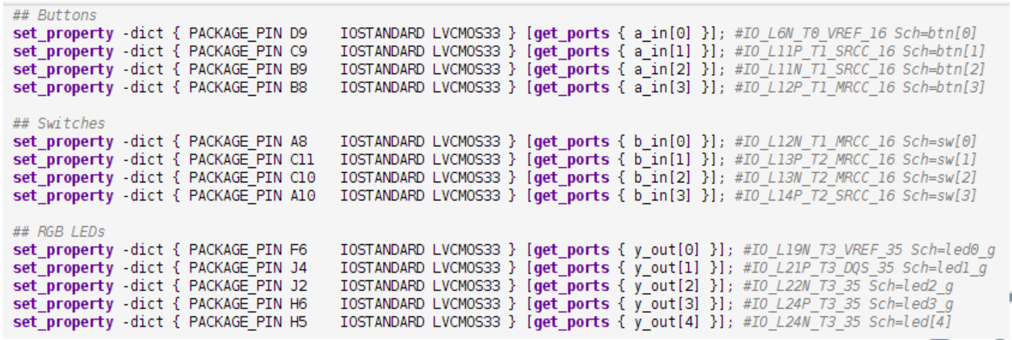
INSTANTIATION

```
a1: adder_1b port map (a_in => a_in(0), b_in => b_in(0), c_in => '0',     y_out => y_out(0), c_out => y_a1_a2);
a2: adder_1b port map (a_in => a_in(1), b_in => b_in(1), c_in => y_a1_a2, y_out => y_out(1), c_out => y_a2_a3);
a3: adder_1b port map (a_in => a_in(2), b_in => b_in(2), c_in => y_a2_a3, y_out => y_out(2), c_out => y_a3_a4);
a4: adder_1b port map (a_in => a_in(3), b_in => b_in(3), c_in => y_a3_a4, y_out => y_out(3), c_out => y_out(4));


end rtl;
```

Add4b Schematic



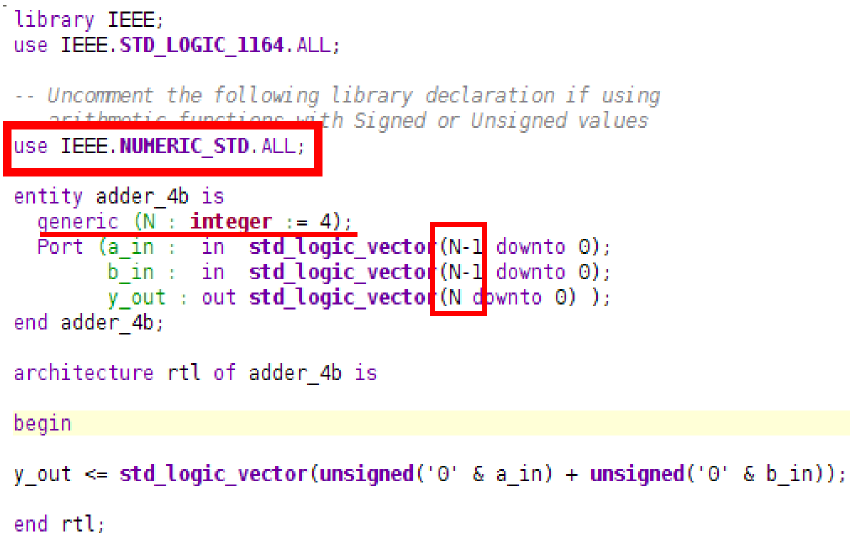
Add4b Constraints



```
## Buttons
set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { a_in[0] }]; #IO_L6N_T0_VREF_16 Sch=btn[0]
set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports { a_in[1] }]; #IO_L11P_T1_SRCC_16 Sch=btn[1]
set_property -dict { PACKAGE_PIN B9    IOSTANDARD LVCMOS33 } [get_ports { a_in[2] }]; #IO_L11N_T1_SRCC_16 Sch=btn[2]
set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33 } [get_ports { a_in[3] }]; #IO_L12P_T1_MRCC_16 Sch=btn[3]

## Switches
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { b_in[0] }]; #IO_L12N_T1_MRCC_16 Sch=sw[0]
set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 } [get_ports { b_in[1] }]; #IO_L13P_T2_MRCC_16 Sch=sw[1]
set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33 } [get_ports { b_in[2] }]; #IO_L13N_T2_MRCC_16 Sch=sw[2]
set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33 } [get_ports { b_in[3] }]; #IO_L14P_T2_SRCC_16 Sch=sw[3]

## RGB LEDs
set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 } [get_ports { y_out[0] }]; #IO_L19N_T3_VREF_35 Sch=led0_g
set_property -dict { PACKAGE_PIN J4    IOSTANDARD LVCMOS33 } [get_ports { y_out[1] }]; #IO_L21P_T3_DQS_35 Sch=led1_g
set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33 } [get_ports { y_out[2] }]; #IO_L22N_T3_35 Sch=led2_g
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { y_out[3] }]; #IO_L24P_T3_35 Sch=led3_g
set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports { y_out[4] }]; #IO_L24N_T3_35 Sch=led[4]
```

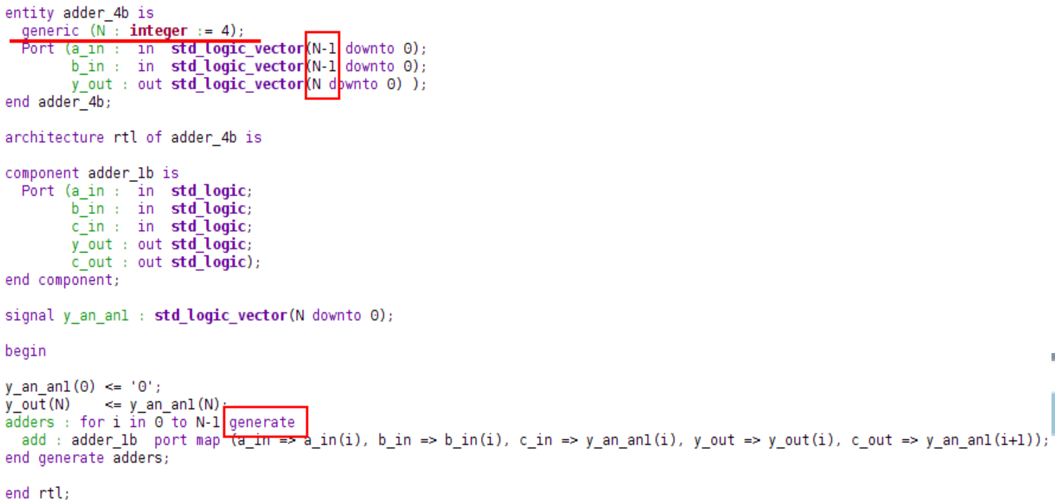
Adder: common solutions

Adder: most common solution



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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

entity adder_4b is
  generic (N : integer := 4);
  Port (a_in :  in  std_logic_vector(N-1 downto 0);
        b_in :  in  std_logic_vector(N-1 downto 0);
        y_out : out std_logic_vector(N downto 0) );
end adder_4b;

architecture rtl of adder_4b is

begin

y_out <= std_logic_vector(unsigned('0' & a_in) + unsigned('0' & b_in));

end rtl;
```

Adder: more elegant solution



```
entity adder_4b is
  generic (N : integer := 4);
  Port (a_in :  in  std_logic_vector(N-1 downto 0);
        b_in :  in  std_logic_vector(N-1 downto 0);
        y_out : out std_logic_vector(N downto 0) );
end adder_4b;

architecture rtl of adder_4b is

component adder_1b is
  Port (a_in :  in  std_logic;
        b_in :  in  std_logic;
        c_in :  in  std_logic;
        y_out : out std_logic;
        c_out : out std_logic);
end component;

signal y_an_an1 : std_logic_vector(N downto 0);

begin

y_an_an1(0) <= '0';
y_out(N)    <= y_an_an1(N);
adders : for i in 0 to N-1 generate
  add : adder_1b  port map (a_in => a_in(i), b_in => b_in(i), c_in => y_an_an1(i), y_out => y_out(i), c_out => y_an_an1(i+1));
end generate adders;

end rtl;
```

What remember ...

- In the first example you see the power of the VHDL code. Just with a code line you can replace the instantiation of a lot of logic gates and components. The work is done by the ”compiler”.
- The use of the **generic** is very common. It is essential in order to write a code reusable.
- **generic** together with **generate** statement is very useful.

Homework

Suggested exercises

- Redo 1,2,3, ..., N times the exercises. Or finish them.
- Get the code more complicated.
  Use a mux 2-1 and a 3 bit adder in order to implement an adder-subtractor.
  If BTN0 is ‘0’ the circuit works as an adder else as a subtractor.
  a_in is drived by BTN1, BTN2 and BTN3.
  b_in is drived by SW1,SW2 and SW3.
  The output is represented in LD4, LD3, LD2, LD1 and LD0.
  LD4 represent the sign bit, that is LD4 = ‘1’ means negative number else positive number.
  Hint : exploit the two’s complement properties.