CPE 462
VHDL: Simulation and Synthesis

Topic #03 - d) Introduction to test-benches

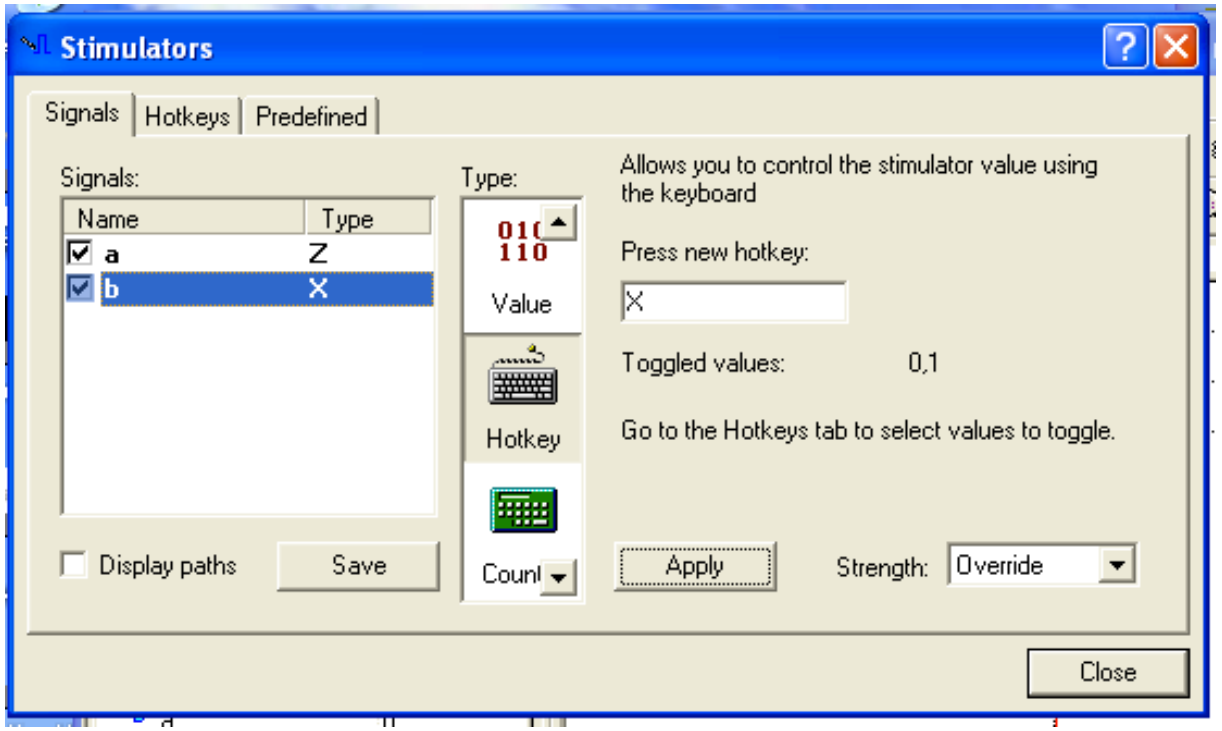
Simulating a circuit

- On our last class you learned a method of simulating the circuit using waveforms.

- You would specify a clock, and manually you would turn each symbol HIGH or LOW.

- **That is a mess.** We need to automate this process if we want to get anything done.

- We don't want to manually simulate a huge circuit!



Signals:

Name	Type
☑ a	Z
☑ b	X

Allows you to control the stimulator value using the keyboard

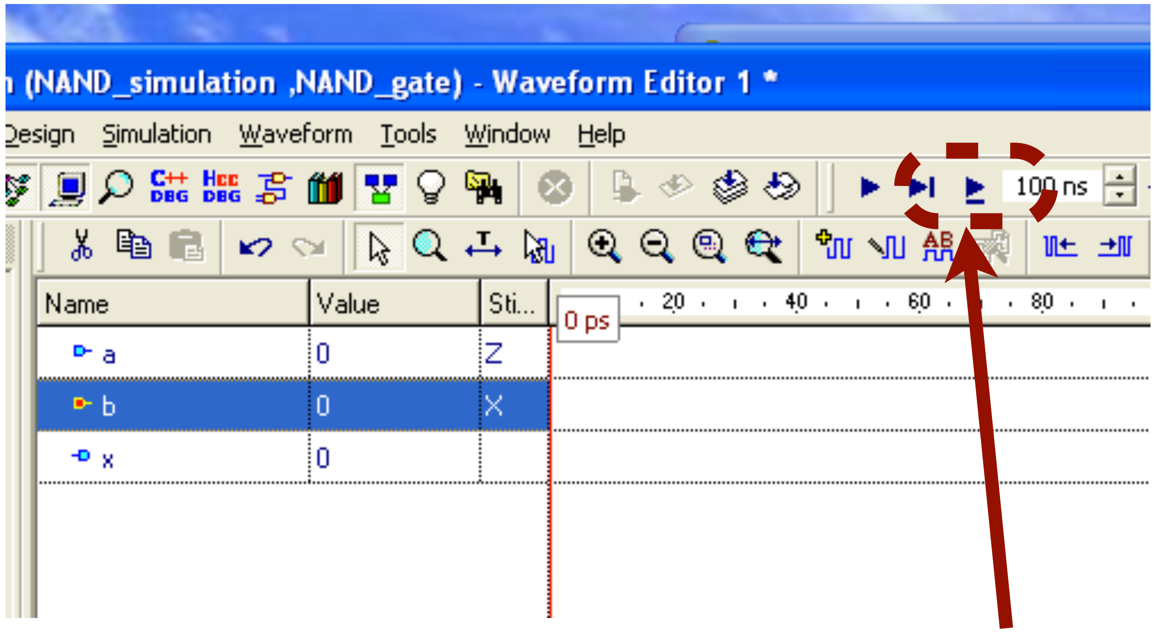
Press new hotkey: X

Toggled values: 0,1

Go to the Hotkeys tab to select values to toggle.

☐ Display paths

Strength: Override

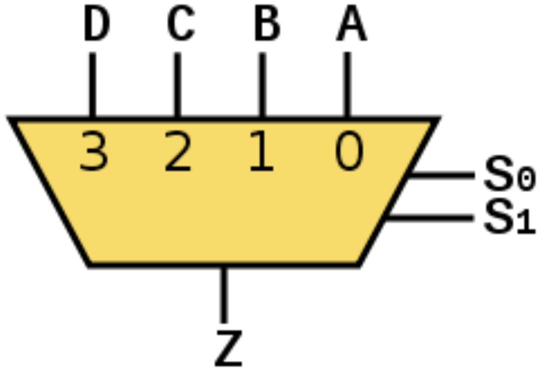


Name	Value	Sti...
a	0	Z
b	0	X
x	0	

Implementation of a 4-to-1 MUX

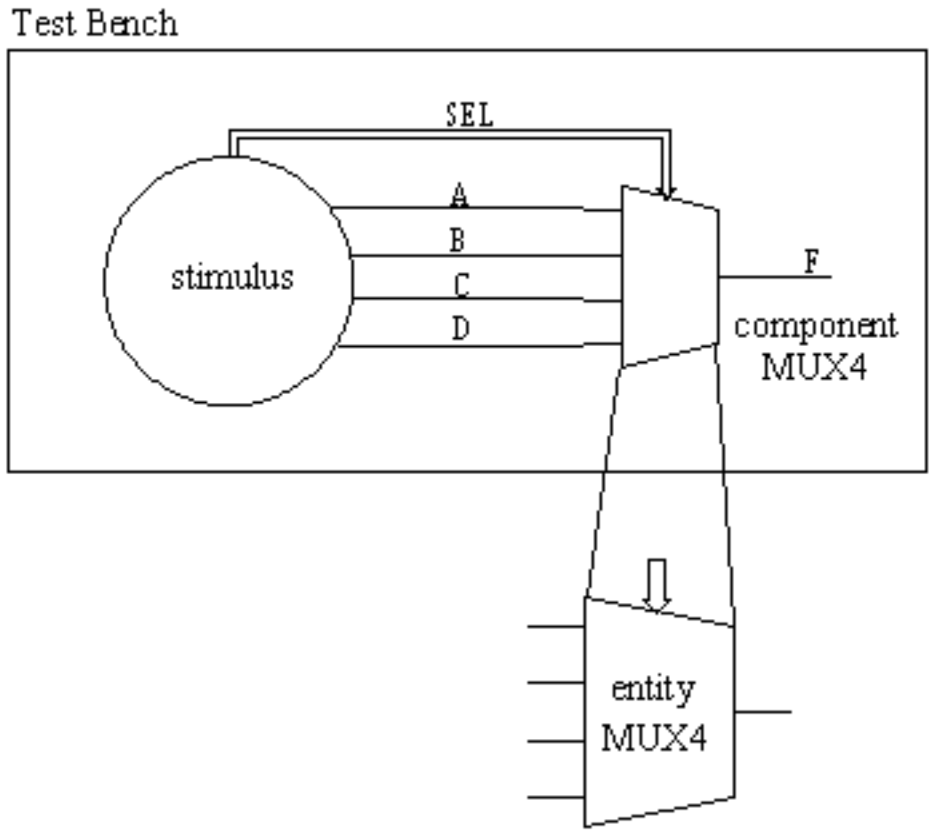
We are now going to automate the testing of a 4-to-1 MUX...
First we write the VHDL code for a 4-to-1 MUX.

```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```



$$F = (A \cdot \overline{S_0} \cdot \overline{S_1}) + (B \cdot S_0 \cdot \overline{S_1}) + (C \cdot \overline{S_0} \cdot S_1) + (D \cdot S_0 \cdot S_1)$$

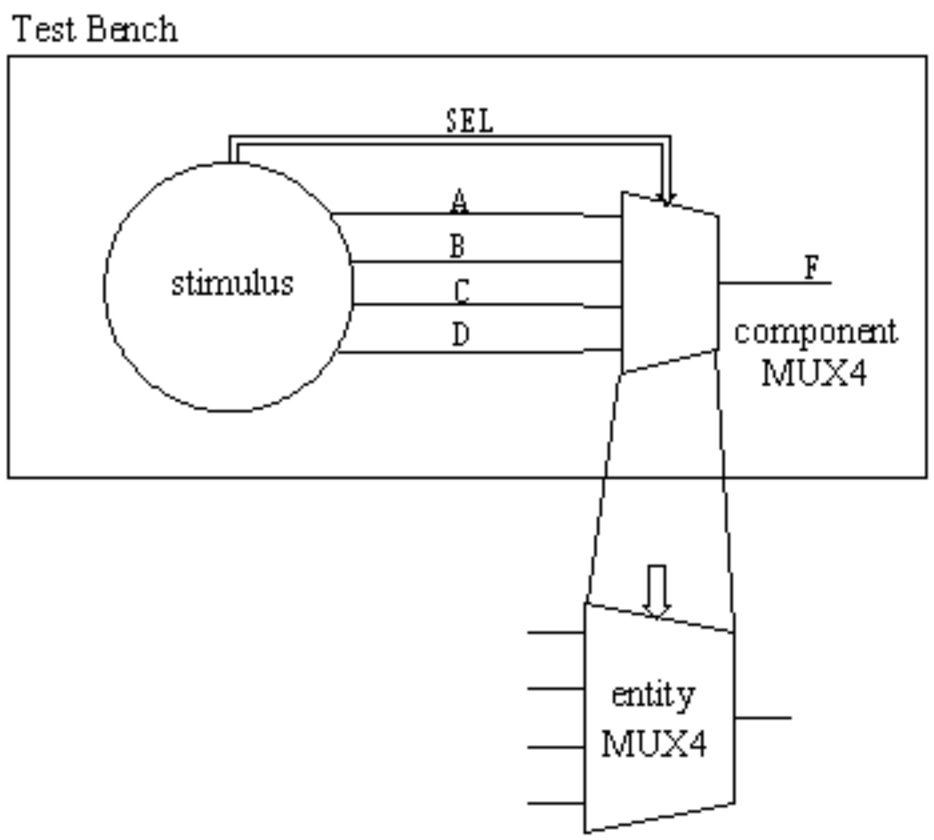
Concept of a VHDL test-bench



With VHDL, we can:

- Model the hardware.
- We can also model a test-bench to apply stimulus to the design and to analyze the results, or compare the results of two simulations.
- In effect, VHDL can be used as a stimulus definition language as well as a hardware description language.

Test-bench for MUX4



- The entity declaration for a test bench is usually empty.
- This is because the test bench itself does not have any inputs or outputs.
- Test vectors are generated and applied to the unit under test within the test bench.
- Keep in mind that it is illegal to have an architecture body without an entity declaration.

Concurrent signal assignment

- The 6 concurrent signal assignment statements within the test bench define the input test vectors

- These delays are relative to the time when the assignments execute

```
entity  test_mux4 is
end;

architecture bench of test_mux4 is
  component mux
    port (a, b, c, d, s0, s1: in bit;
    z :out bit);
  end component;

  signal a, b, c, d, z, s0, s1: bit;

begin
    s0 <= '0', '1' after 20 ns;
    s1 <= '0', '1' after 10ns, '0' after 20 ns, '1' after 30 ns;
    a <= '0', '1' after 5 ns;
    b <= '0', '1' after 20 ns, '0' after 30 ns;
    c <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 25 ns;
    d <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (a, b, c, d, s0, s1, z);

end bench;
```

we will learn about components in the future

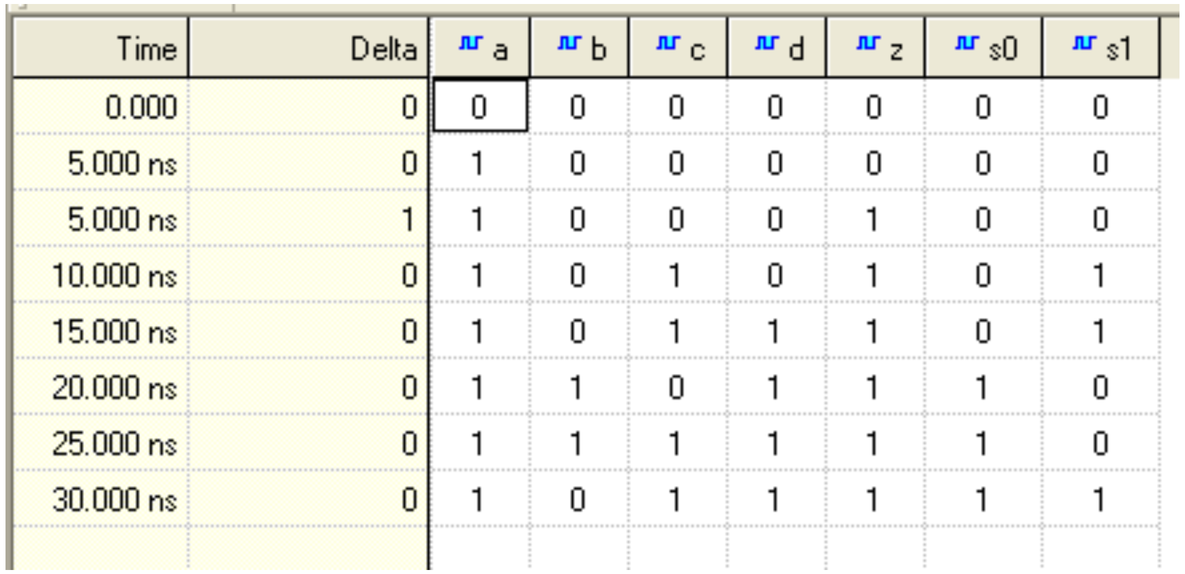
we will also learn about this map...

Outcome of the test-bench

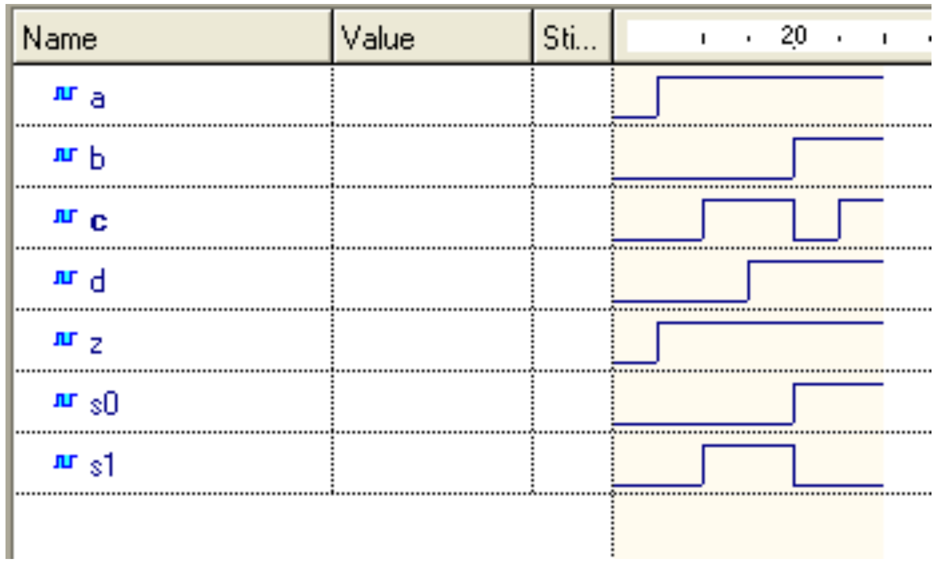
```
entity  test_mux4 is
(... some code is missing here: see previous slide ...)

begin
    s0 <= '0', '1' after 20 ns;
    s1 <= '0', '1' after 10ns, '0' after 20 ns, '1' after 30 ns;
   a <= '0', '1' after 5 ns;
   b <= '0', '1' after 20 ns, '0' after 30 ns;
   c <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 25 ns;
   d <= '0', '1' after 15 ns, '1' after 25 ns;

(... some code is missing here: see previous slide ...)

end bench;
```

changes value after 5ns

initial value

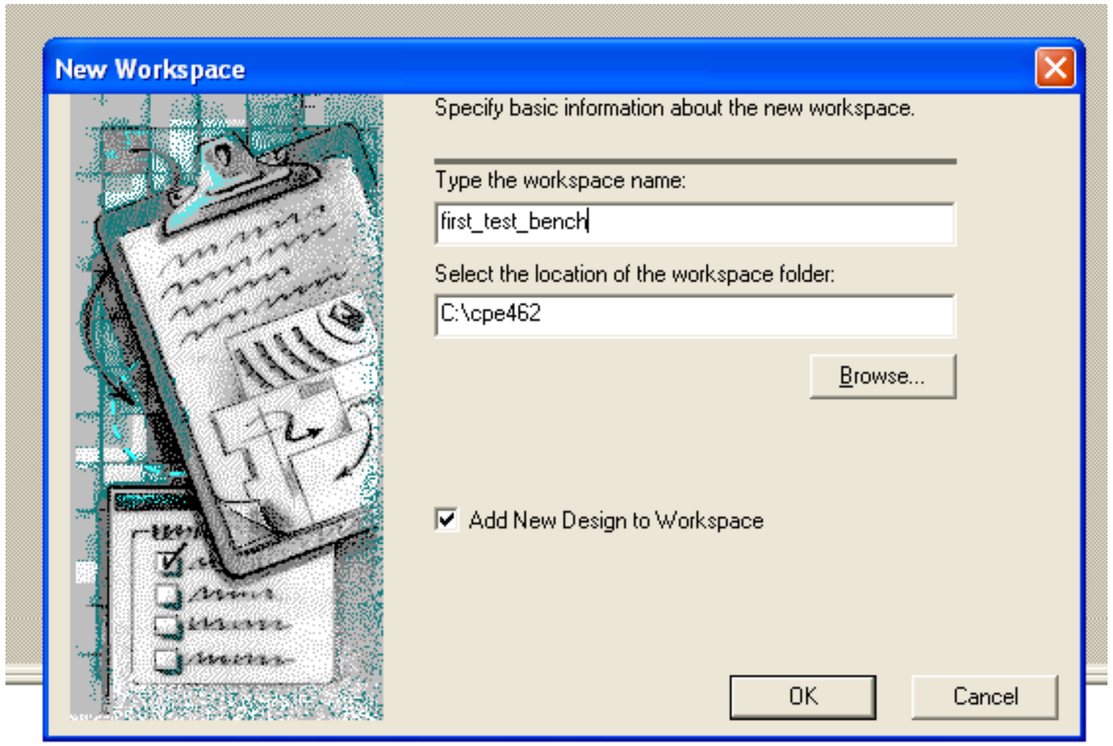


Time	Delta	a	b	c	d	z	s0	s1
0.000	0	0	0	0	0	0	0	0
5.000 ns	0	1	0	0	0	0	0	0
5.000 ns	1	1	0	0	0	1	0	0
10.000 ns	0	1	0	1	0	1	0	1
15.000 ns	0	1	0	1	1	1	0	1
20.000 ns	0	1	1	0	1	1	1	0
25.000 ns	0	1	1	1	1	1	1	0
30.000 ns	0	1	0	1	1	1	1	1



Step by step in active HDL

Create a new project like before



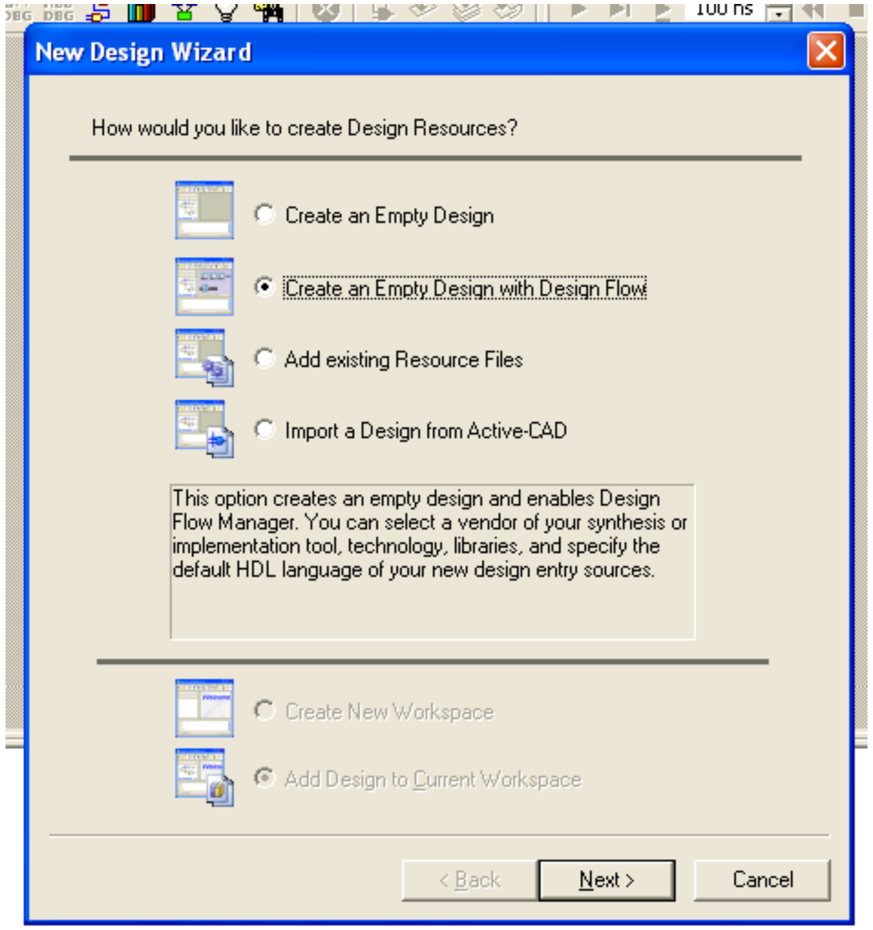
New Workspace

Specify basic information about the new workspace.

Type the workspace name: first_test_bench

Select the location of the workspace folder: C:\cpe462

☑ Add New Design to Workspace



New Design Wizard

How would you like to create Design Resources?

○ Create an Empty Design

● Create an Empty Design with Design Flow

○ Add existing Resource Files

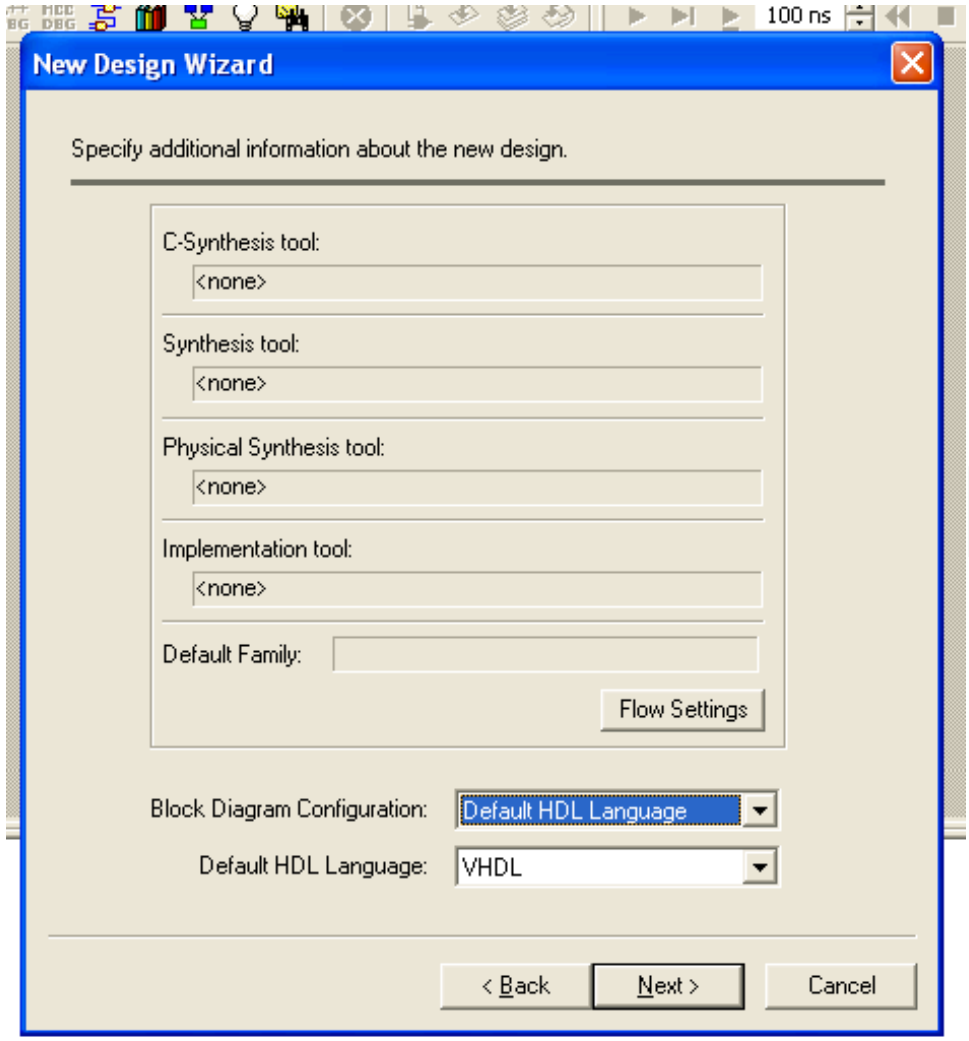
○ Import a Design from Active-CAD

This option creates an empty design and enables Design Flow Manager. You can select a vendor of your synthesis or implementation tool, technology, libraries, and specify the default HDL language of your new design entry sources.

○ Create New Workspace

● Add Design to Current Workspace

Still, same as before



Specify additional information about the new design.

C-Synthesis tool: <none>

Synthesis tool: <none>

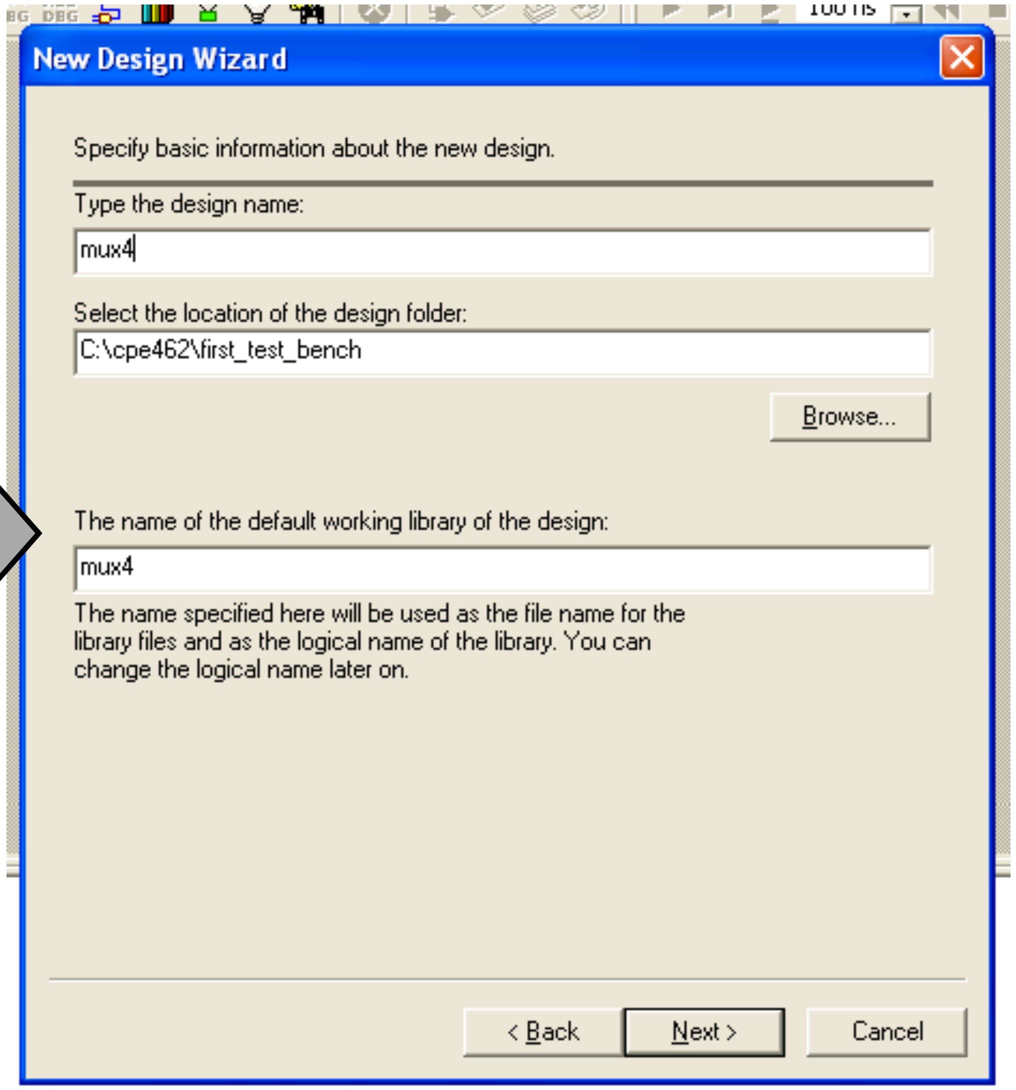
Physical Synthesis tool: <none>

Implementation tool: <none>

Default Family:

Block Diagram Configuration: Default HDL Language

Default HDL Language: VHDL



Specify basic information about the new design.

Type the design name: mux4

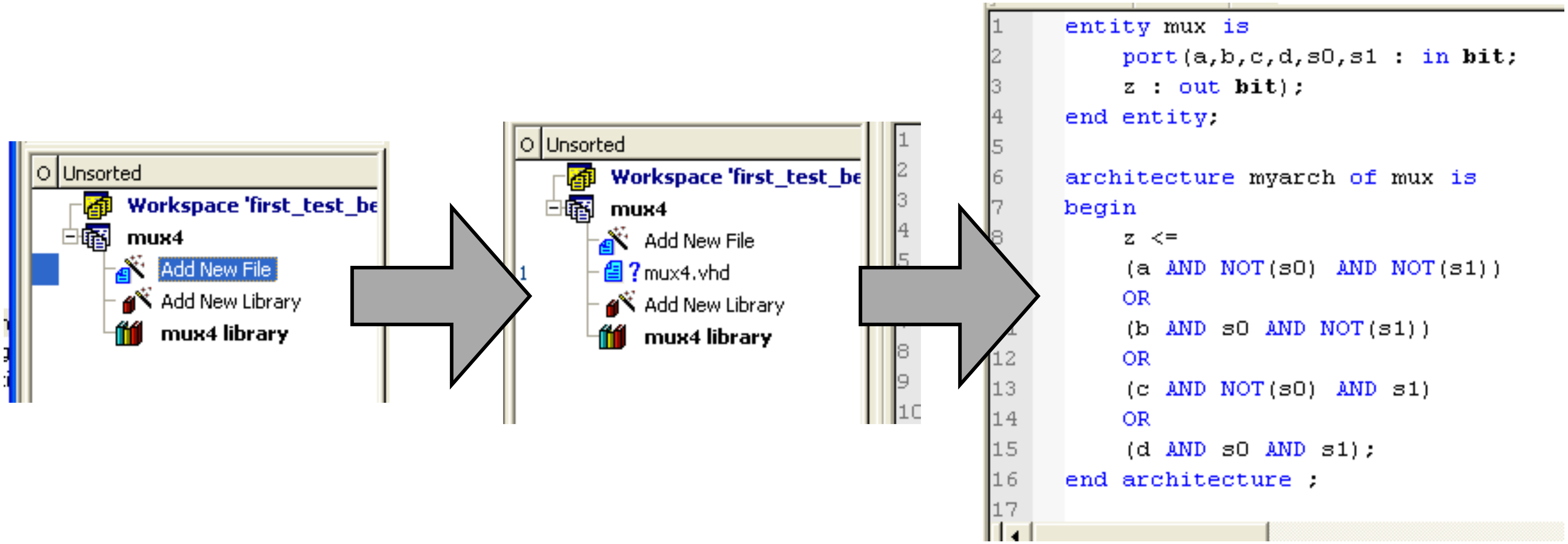
Select the location of the design folder: C:\cpe462\first_test_bench

The name of the default working library of the design: mux4

The name specified here will be used as the file name for the library files and as the logical name of the library. You can change the logical name later on.

Write the main code

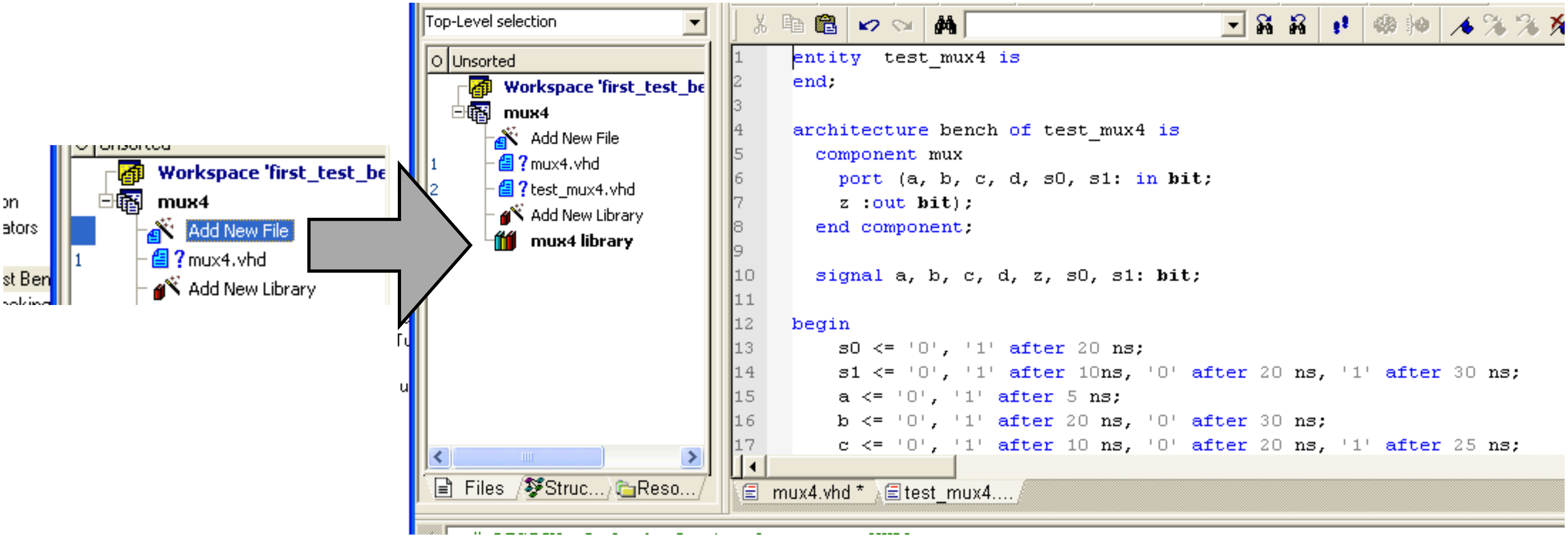
Add a new file, and write the MUX4 code into it.



```vhdl
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

Adding the test-bench code

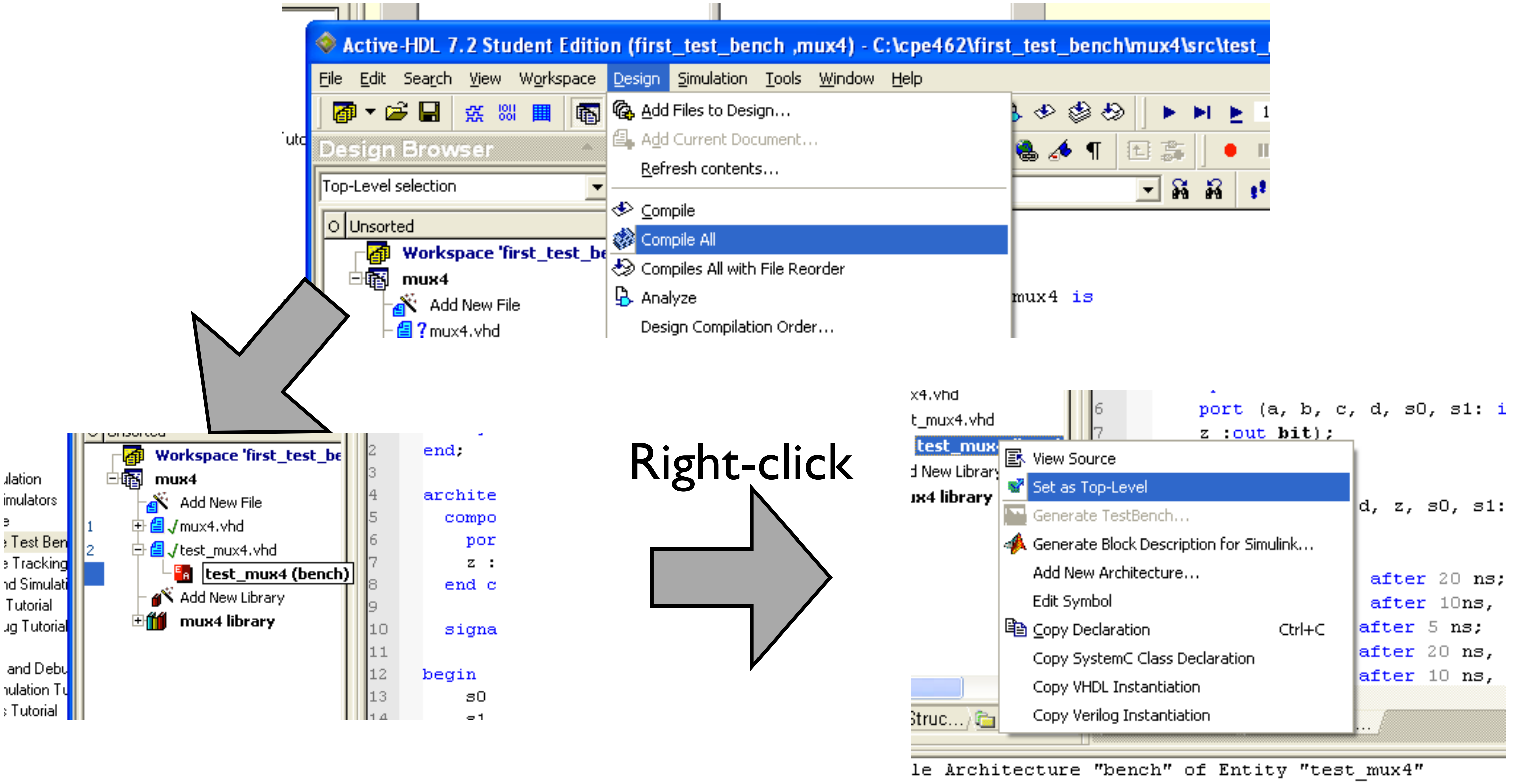
Add a new file, and write the MUX4 test-bench code into it.



```
entity  test_mux4 is
end;

architecture bench of test_mux4 is
  component mux
    port (a, b, c, d, s0, s1: in bit;
    z :out bit);
  end component;

  signal a, b, c, d, z, s0, s1: bit;

begin
    s0 <= '0', '1' after 20 ns;
    s1 <= '0', '1' after 10ns, '0' after 20 ns, '1' after 30 ns;
    a <= '0', '1' after 5 ns;
    b <= '0', '1' after 20 ns, '0' after 30 ns;
    c <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 25 ns;
```

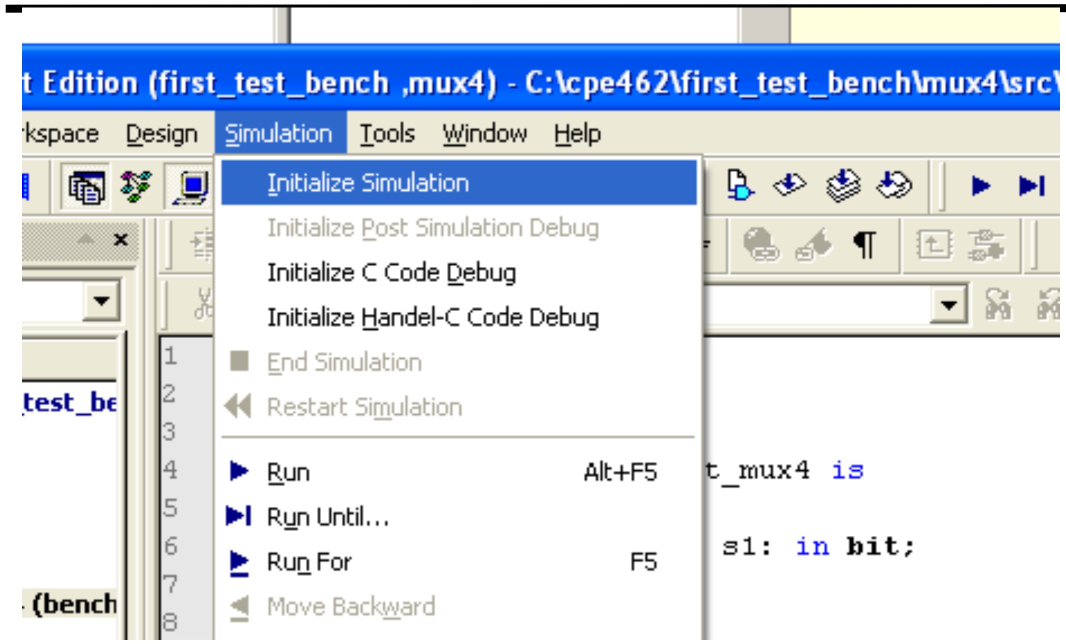
Compile and set simulation entry point

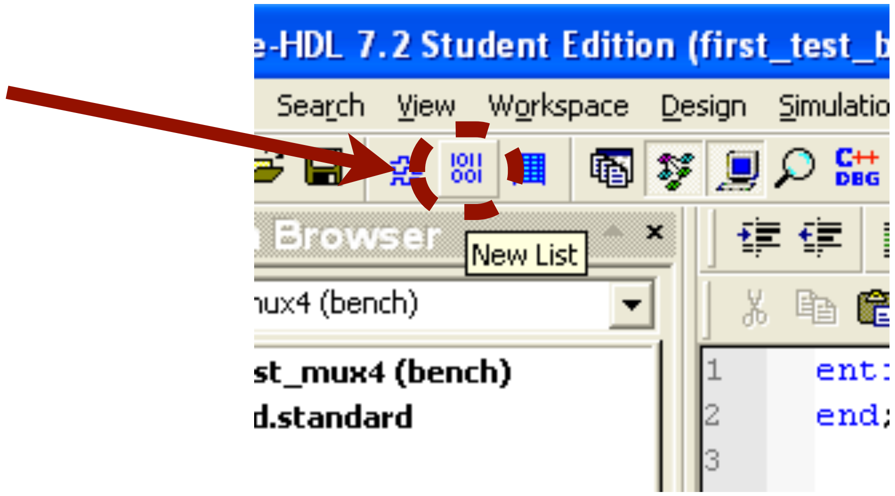
Compile everything, then make sure the test-bench is the simulation top-level.

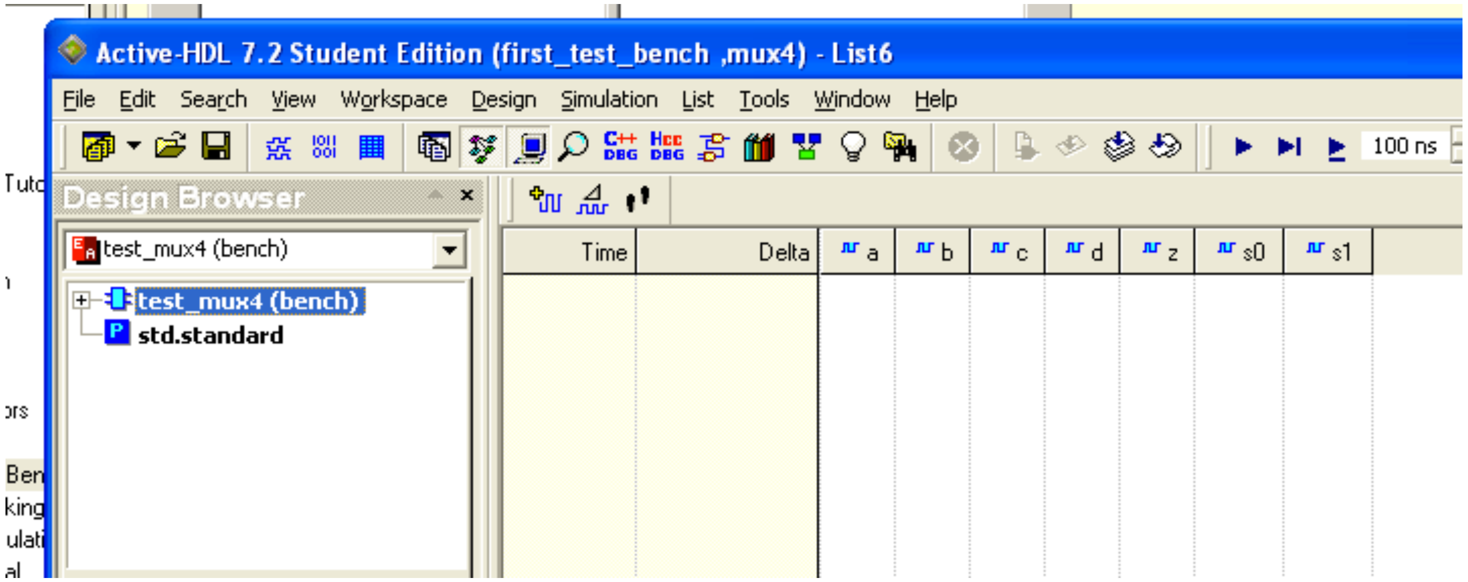


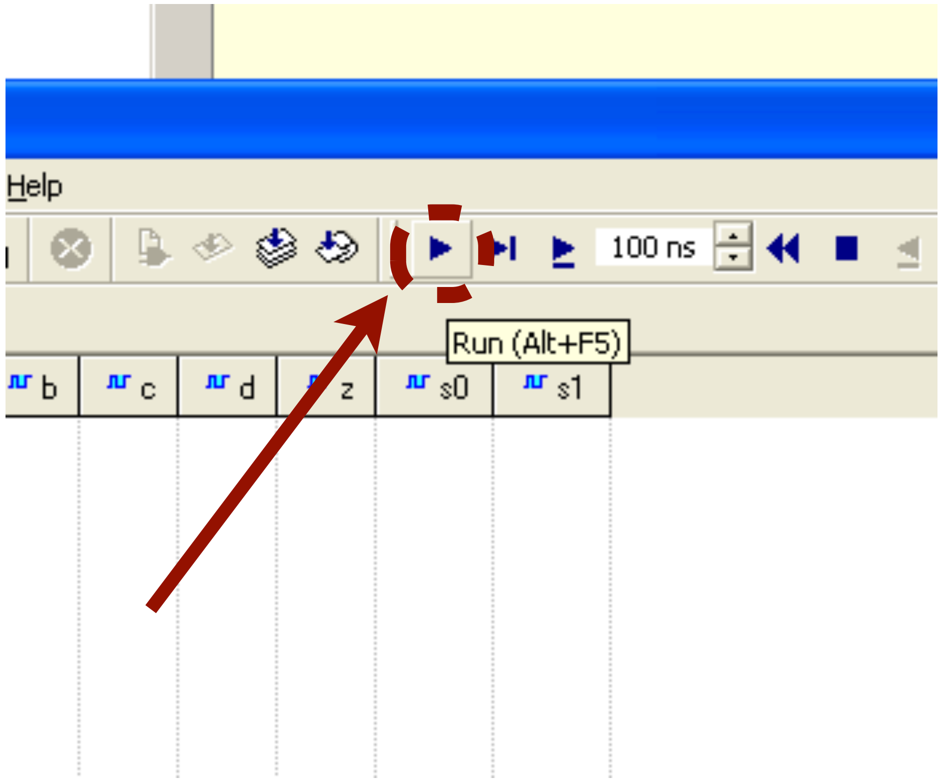
Right-click

Start simulation and run it


t Edition (first_test_bench ,mux4) - C:\cpe462\first_test_bench\mux4\src\
rkspace Design Simulation Tools Window Help
Initialize Simulation
Initialize Post Simulation Debug
Initialize C Code Debug
Initialize Handel-C Code Debug
End Simulation
Restart Simulation
Run Alt+F5
Run Until...
Run For F5
Move Backward
test_be
(bench
t_mux4 is
s1: in bit;

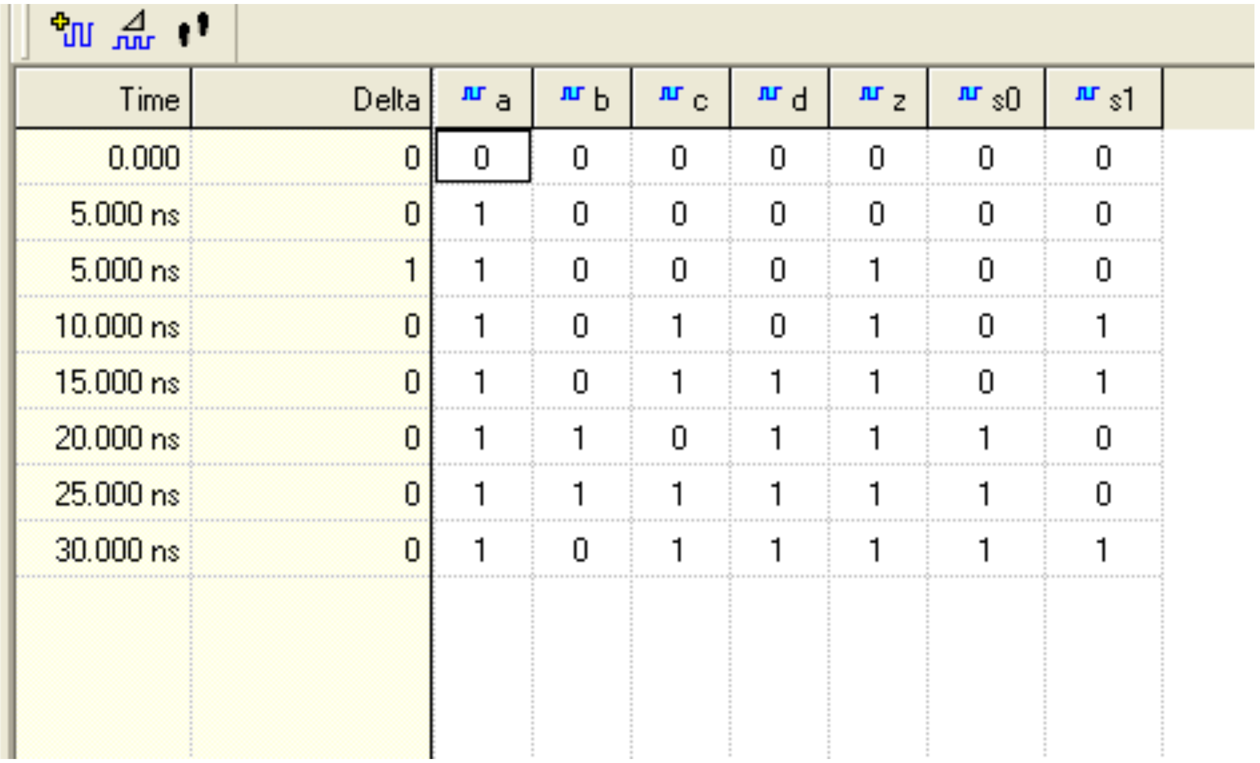

e-HDL 7.2 Student Edition (first_test_b
Search View Workspace Design Simulatio
Browser
New List
nux4 (bench)
st_mux4 (bench)
d.standard
ent:
end;


Active-HDL 7.2 Student Edition (first_test_bench ,mux4) - List6
File Edit Search View Workspace Design Simulation List Tools Window Help
100 ns
Design Browser
test_mux4 (bench)
test_mux4 (bench)
std.standard
Time
Delta
a
b
c
d
z
s0
s1

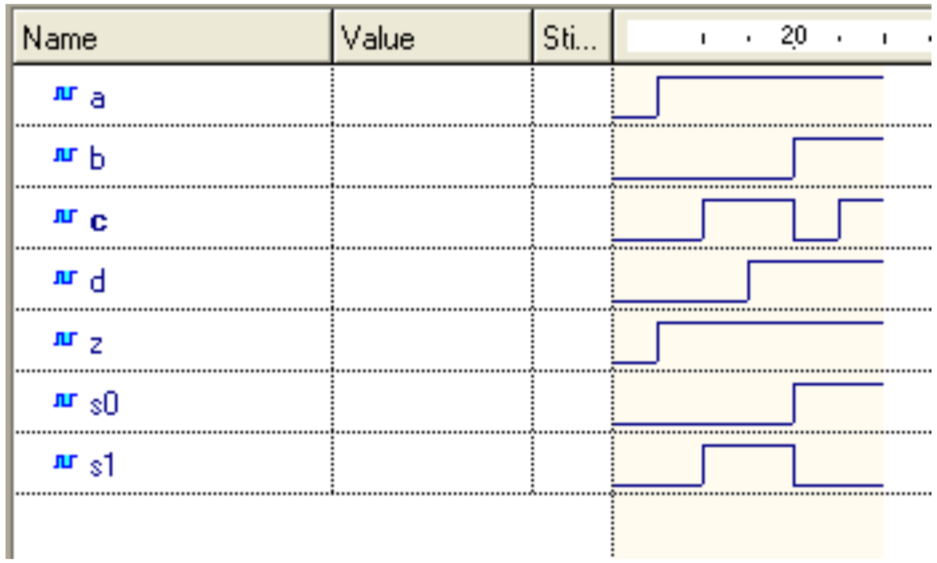


Visualizing output

Now you pick whatever output you want to see. Waveforms or lists.

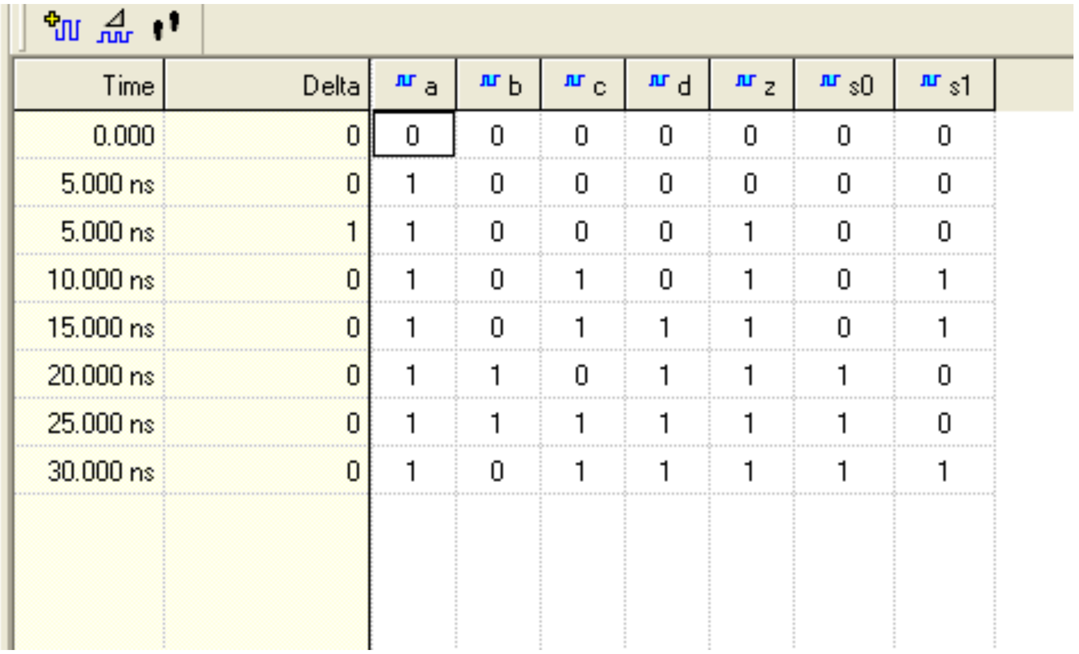


Time	Delta	a	b	c	d	z	s0	s1
0.000	0	0	0	0	0	0	0	0
5.000 ns	0	1	0	0	0	0	0	0
5.000 ns	1	1	0	0	0	1	0	0
10.000 ns	0	1	0	1	0	1	0	1
15.000 ns	0	1	0	1	1	1	0	1
20.000 ns	0	1	1	0	1	1	1	0
25.000 ns	0	1	1	1	1	1	1	0
30.000 ns	0	1	0	1	1	1	1	1

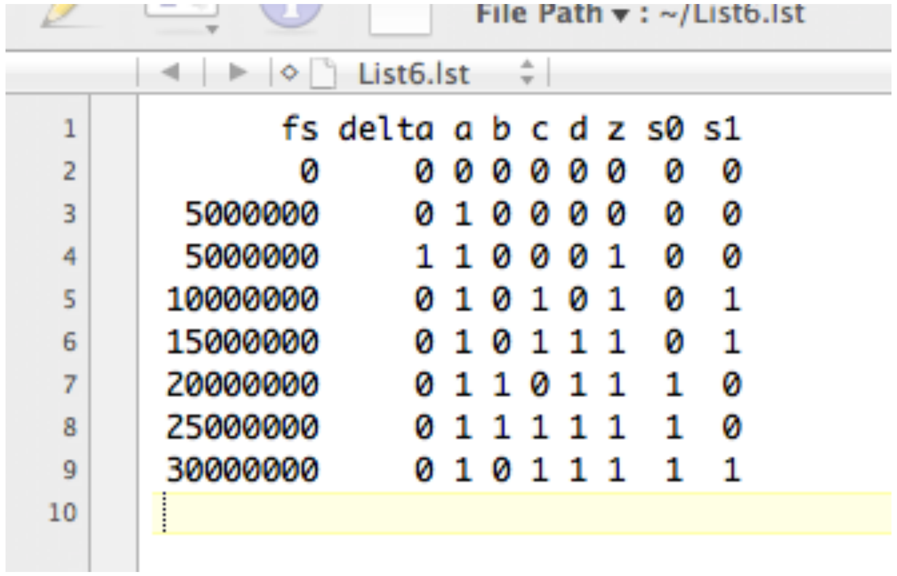


Isn’t this still a manual process?

- Not quite...
- Since test-benches are just plain VHDL files (in text), I can generate test-benches automatically with a scripting language such as Perl or Python.
- I can export the output list as a text file, and use a scripting language to confirm that my VHDL code is doing what it is supposed to do.



Time	Delta	a	b	c	d	z	s0	s1
0.000	0	0	0	0	0	0	0	0
5.000 ns	0	1	0	0	0	0	0	0
5.000 ns	1	1	0	0	0	1	0	0
10.000 ns	0	1	0	1	0	1	0	1
15.000 ns	0	1	0	1	1	1	0	1
20.000 ns	0	1	1	0	1	1	1	0
25.000 ns	0	1	1	1	1	1	1	0
30.000 ns	0	1	0	1	1	1	1	1



```
      fs delta a b c d z s0 s1
       0       0 0 0 0 0 0  0  0
 5000000       0 1 0 0 0 0  0  0
 5000000       1 1 0 0 0 1  0  0
10000000       0 1 0 1 0 1  0  1
15000000       0 1 0 1 1 1  0  1
20000000       0 1 1 0 1 1  1  0
25000000       0 1 1 1 1 1  1  0
30000000       0 1 0 1 1 1  1  1
```

Test-benches are NOT synthesizable!

- You can **NOT** add a test bench to an FPGA board
- The purpose of test-benches is to simulate the functional behavior of the circuit
- For example, simulate pressing *buttons* or *activating switches*.

Summary of test-benches

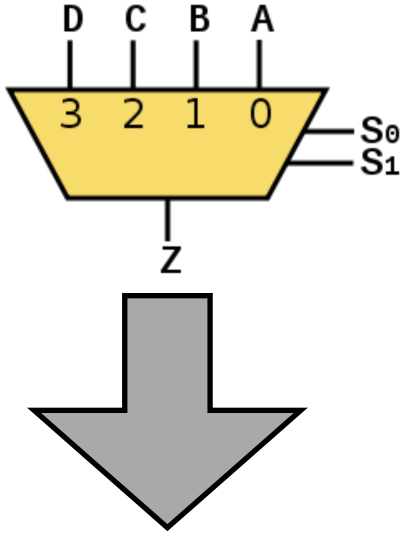
there are two separate files...

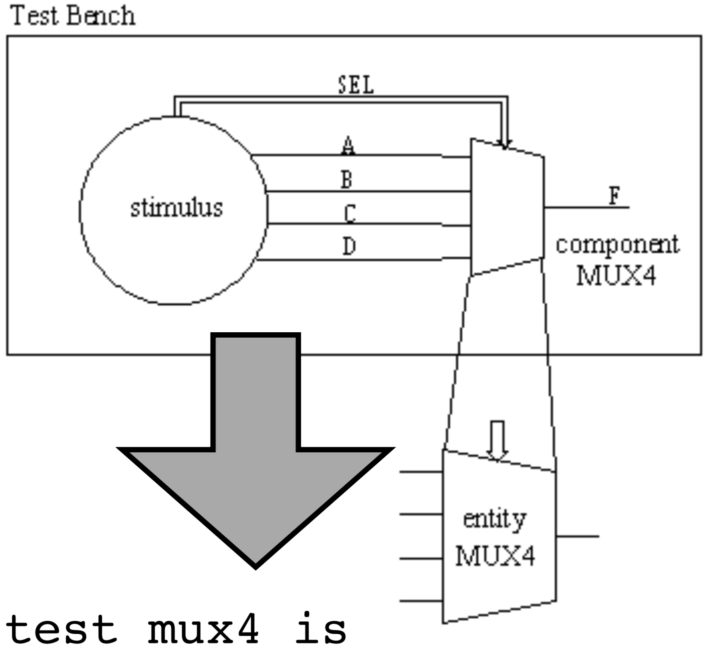
main code

```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

test-bench code

```
entity  test_mux4 is
end;

architecture bench of test_mux4 is
  component mux
    port (a, b, c, d, s0, s1: in bit;
    z :out bit);
  end component;

  signal a, b, c, d, z, s0, s1: bit;

begin
   s0 <= '0', '1' after 20 ns;
   s1 <= '0', '1' after 10ns;
   a  <= '0', '1' after 5 ns;
   b  <= '0', '1' after 20 ns, '0' after 30 ns;
   c  <= '0', '1' after 10 ns, '0' after 20 ns;
   d  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (a, b, c, d, s0, s1, z);

end bench;
```





```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

```
entity  test_mux4 is
end;

architecture bench of test_mux4 is
  component mux
    port (a, b, c, d, s0, s1: in bit;
    z :out bit);
  end component;

  signal a, b, c, d, z, s0, s1: bit;

begin
   s0 <= '0', '1' after 20 ns;
   s1 <= '0', '1' after 10ns;
   a  <= '0', '1' after 5 ns;
   b  <= '0', '1' after 20 ns, '0' after 30 ns;
   c  <= '0', '1' after 10 ns, '0' after 20 ns;
   d  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (a, b, c, d, s0, s1, z);

end bench;
```

A test-bench is just a set of inputs to the circuit

Lets crete a test-bench with a template

```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

```
entity <<TEST-BENCH NAME>> is
end;

architecture bench of <<TEST-BENCH NAME>> is
  component <<ENTITY NAME TO BE TESTED>>

     <<COPY IN/OUT PORT NAMES>>

  end component;

  signal <<INSERT IN/OUT PORT NAMES & SIGNAL TYPES>>;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: <<ENTITY NAME TO BE TESTED>> port map (<<INSERT IN/OUT
PORT NAMES>>);

end bench;
```

Lets crete a test-bench with a template

```
entity mux is
     port(a,b,c,d,s0,s1 : in bit;
     z : out bit);
end entity;

architecture myarch of mux is
begin
     z <=
     (a AND NOT(s0) AND NOT(s1))
     OR
     (b AND s0 AND NOT(s1))
     OR
     (c AND NOT(s0) AND s1)
     OR
     (d AND s0 AND s1);
end architecture ;
```

Step #1 - Change test-bench name

```
entity <<TEST-BENCH NAME>> is
end;

architecture bench of <<TEST-BENCH NAME>> is
  component <<ENTITY NAME TO BE TESTED>>

     <<COPY IN/OUT PORT NAMES>>

  end component;

  signal <<INSERT IN/OUT PORT NAMES & SIGNAL TYPES>>;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: <<ENTITY NAME TO BE TESTED>> port map (<<INSERT IN/OUT
PORT NAMES>>);

end bench;
```

Lets crete a test-bench with a template

Step #2 - Include entity to be tested

```
entity mux is
     port(a,b,c,d,s0,s1 : in bit;
     z : out bit);
end entity;

architecture myarch of mux is
begin
     z <=
     (a AND NOT(s0) AND NOT(s1))
     OR
     (b AND s0 AND NOT(s1))
     OR
     (c AND NOT(s0) AND s1)
     OR
     (d AND s0 AND s1);
end architecture ;
```

```
entity testMUX is
end;

architecture bench of testMUX is
  component <<ENTITY NAME TO BE TESTED>>

     <<COPY IN/OUT PORT NAMES>>

  end component;

  signal <<INSERT IN/OUT PORT NAMES & SIGNAL TYPES>>;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: <<ENTITY NAME TO BE TESTED>> port map (<<INSERT IN/OUT
PORT NAMES>>);

end bench;
```

Lets crete a test-bench with a template

Step #3 - Copy input output port names

```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
     z <=
     (a AND NOT(s0) AND NOT(s1))
     OR
     (b AND s0 AND NOT(s1))
     OR
     (c AND NOT(s0) AND s1)
     OR
     (d AND s0 AND s1);
end architecture ;
```

```
entity testMUX is
end;

architecture bench of testMUX is
  component mux

    <<COPY IN/OUT PORT NAMES>>

  end component;

  signal <<INSERT IN/OUT PORT NAMES & SIGNAL TYPES>>;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (<<INSERT IN/OUT PORT NAMES>>);

end bench;
```

Lets crete a test-bench with a template

```
entity mux is
     port(a,b,c,d,s0,s1 : in bit;
     z : out bit);
end entity;

architecture myarch of mux is
begin
     z <=
     (a AND NOT(s0) AND NOT(s1))
     OR
     (b AND s0 AND NOT(s1))
     OR
     (c AND NOT(s0) AND s1)
     OR
     (d AND s0 AND s1);
end architecture ;
```

Step #4 - Add all signal names (in order)

```
entity testMUX is
end;

architecture bench of testMUX is
  component mux
     port(a,b,c,d,s0,s1: in bit;
     z : out bit);
  end component;

  signal <<INSERT IN/OUT PORT NAMES & SIGNAL TYPES>>;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (<<INSERT IN/OUT PORT NAMES>>);

end bench;
```

Lets crete a test-bench with a template

```
entity mux is
     port(a,b,c,d,s0,s1 : in bit;
     z : out bit);
end entity;

architecture myarch of mux is
begin
     z <=
     (a AND NOT(s0) AND NOT(s1))
     OR
     (b AND s0 AND NOT(s1))
     OR
     (c AND NOT(s0) AND s1)
     OR
     (d AND s0 AND s1);
end architecture ;
```

```
entity testMUX is
end;

architecture bench of testMUX is
  component mux
     port(a,b,c,d,s0,s1: in bit;
     z : out bit);
  end component;

  signal a,b,c,d,s0,s1,z : bit;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (<<INSERT IN/OUT PORT NAMES>>);

end bench;
```

Step #5 - Copy the signal names to the port line...

Lets crete a test-bench with a template

```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

Step #6 - Add the circuit inputs here

```
entity testMUX is
end;

architecture bench of testMUX is
  component mux
     port(a,b,c,d,s0,s1: in bit;
     z : out bit);
  end component;

  signal a,b,c,d,s0,s1,z : bit;

begin
 <<INPUT #1 NAME>>  <= '0', '1' after 20 ns;
 <<INPUT #2 NAME>>  <= '0', '1' after 10ns;
 <<INPUT #3 NAME>>  <= '0', '1' after 5 ns;
 <<INPUT #4 NAME>>  <= '0', '1' after 20 ns, '0' after 30 ns;
 <<INPUT #5 NAME>>  <= '0', '1' after 10 ns, '0' after 20 ns;
 <<INPUT #6 NAME>>  <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (a,b,c,d,s0,s1,z);

end bench;
```

Lets crete a test-bench with a template

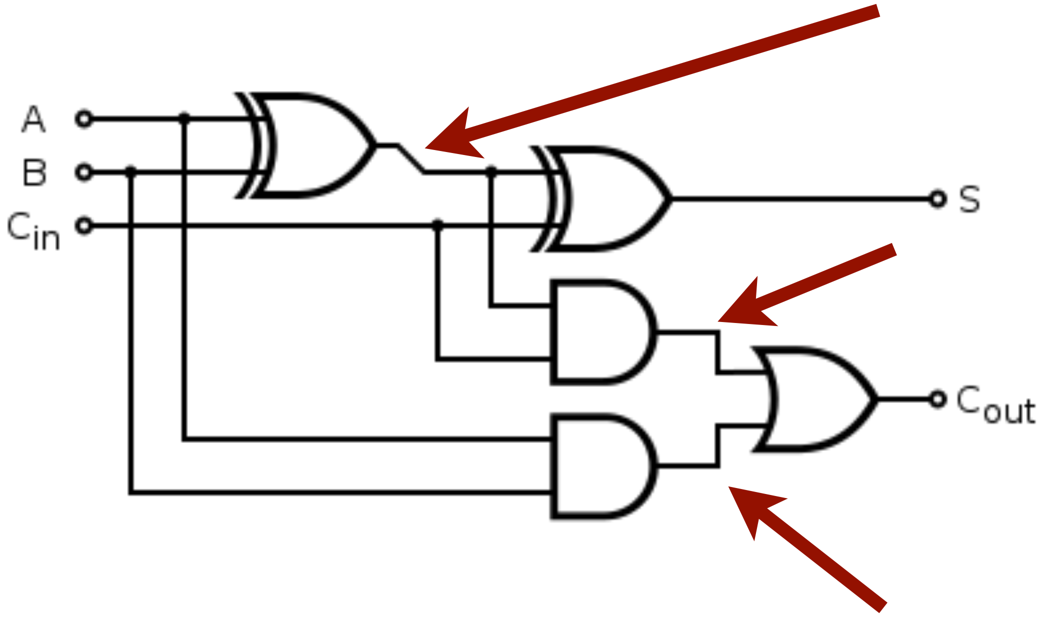
```
entity mux is
    port(a,b,c,d,s0,s1 : in bit;
    z : out bit);
end entity;

architecture myarch of mux is
begin
    z <=
    (a AND NOT(s0) AND NOT(s1))
    OR
    (b AND s0 AND NOT(s1))
    OR
    (c AND NOT(s0) AND s1)
    OR
    (d AND s0 AND s1);
end architecture ;
```

Step #7 - Change the circuit stimulus here.

```
entity testMUX is
end;

architecture bench of testMUX is
  component mux
     port(a,b,c,d,s0,s1: in bit;
     z : out bit);
  end component;

  signal a,b,c,d,s0,s1,z : bit;

begin
 s0   <= '0', '1' after 20 ns;
 s1   <= '0', '1' after 10ns;
 a    <= '0', '1' after 5 ns;
 b    <= '0', '1' after 20 ns, '0' after 30 ns;
 c    <= '0', '1' after 10 ns, '0' after 20 ns;
 d    <= '0', '1' after 15 ns, '1' after 25 ns;

    m: mux port map (a,b,c,d,s0,s1,z);

end bench;
```

Practice Exercises

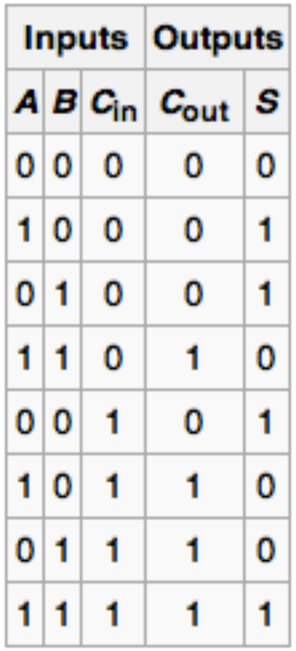
Exercise #1- Implement a full-adder and test it using a complete test-bench



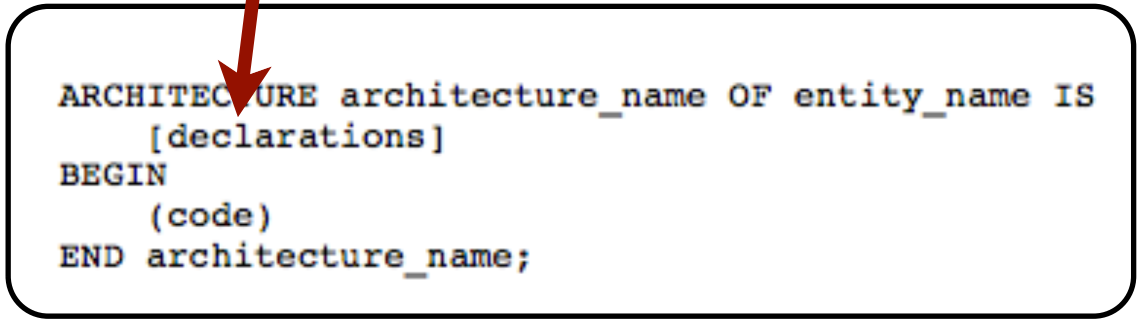
- As a reminder, this is how a full adder looks like

- Use an internal signal at the end of each gate

- Make sure you declare these internal signals



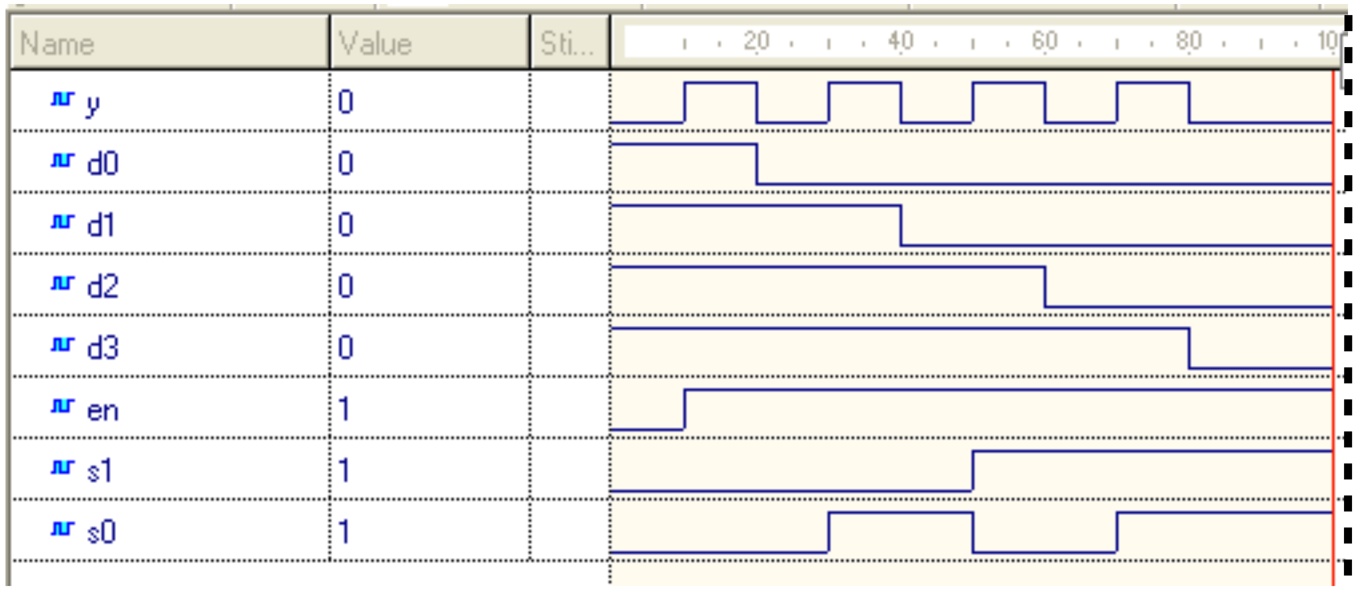
Inputs			Outputs	
A	*B*	*C*in	*C*out	*S*
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Exercise #2- Create a test-bench

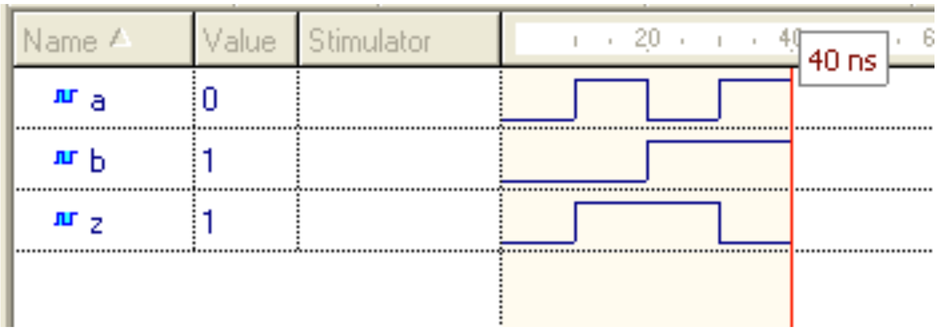
- What does the code do?
- Create an a test bench that will replicate the following waveforms



```
library IEEE;
use IEEE.std_logic_1164.all;

entity ex3 is
    port (
        en,d3,d2,d1,d0,s1,s0: in STD_LOGIC;
         y: out STD_LOGIC
         );
end entity ex3;

architecture arch of ex3 is
begin
y<='0' when (en='0') else
    d0 when (s1='0' and s0='0') else
    d1 when (s1='0' and s0='1') else
    d2 when (s1='1' and s0='0') else
    d3;
end architecture arch;
```

Exercise #3 - Write the main VHDL code from the test-bench + waveform

```
entity testmaincode is
end;

architecture bench of testmaincode is
  component maincircuit
    port(a,b: in bit;
   z : out bit);

  end component;

  signal a,b,z: bit;

begin
 a  <= '0', '1' after 10ns, '0' after 20ns, '1' after 30ns, '0' after 40ns;
 b  <= '0', '1' after 20ns;

    m: maincircuit port map (a,b,z);

end bench;
```



Name	Value	Stimulator
a	0	
b	1	
z	1	