Designing with VHDL and FPGA

Instructor:

Dr. Ahmad El-Banna

LAB# 7
FALL 2016

Agenda

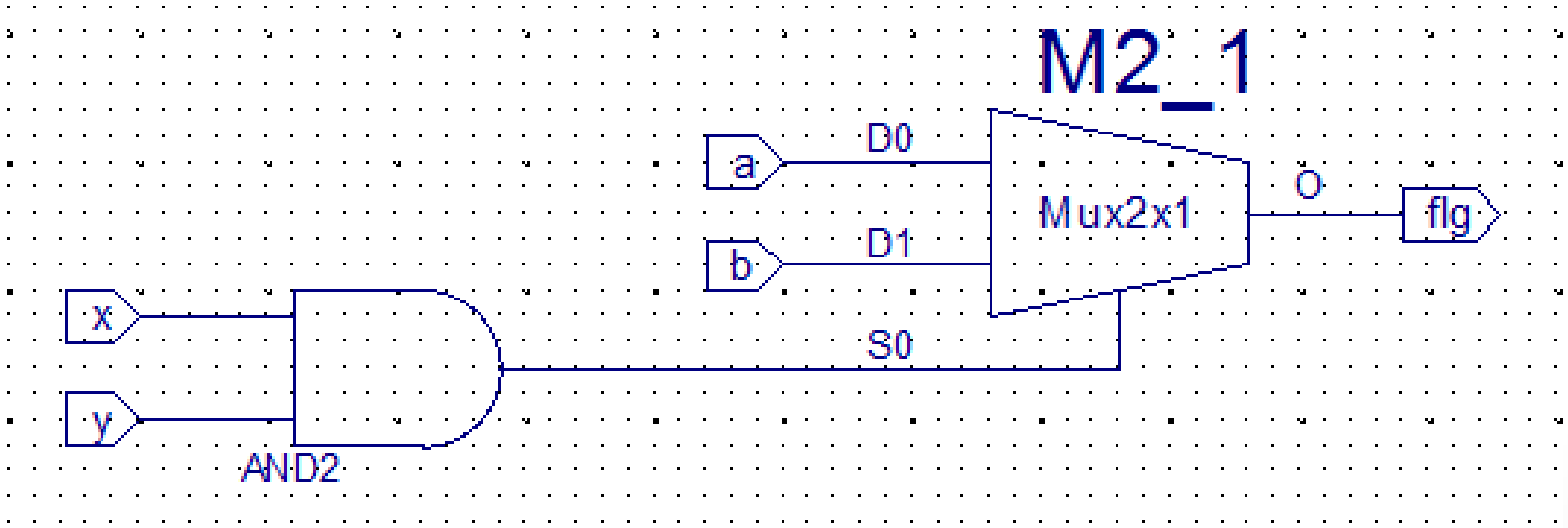
Schematic Level Design

- Create schematic project
- Add modules
- Add ports
- Simulate the design

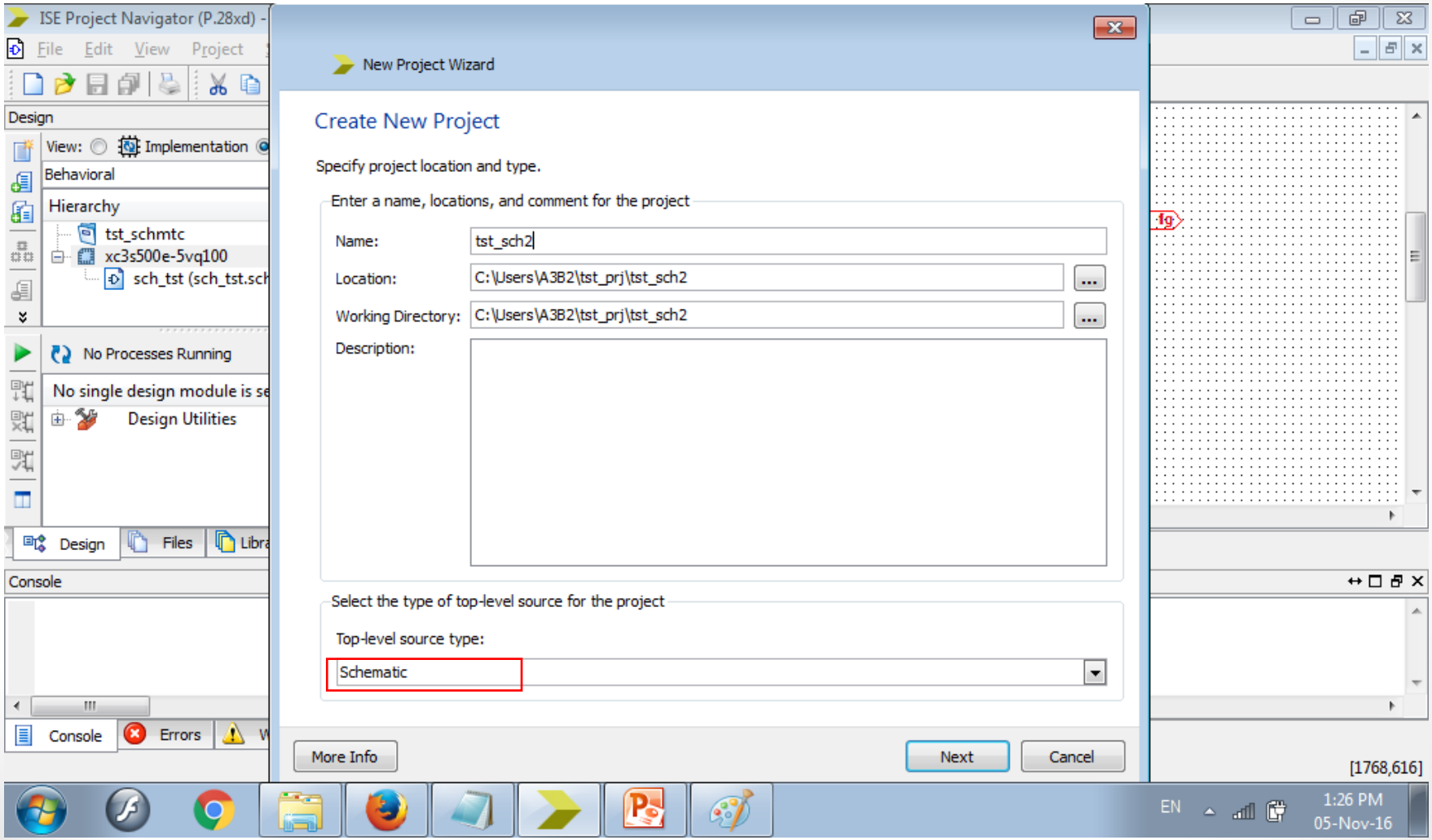
Example

Schematic Design

- Use the schematic way to implement the following module

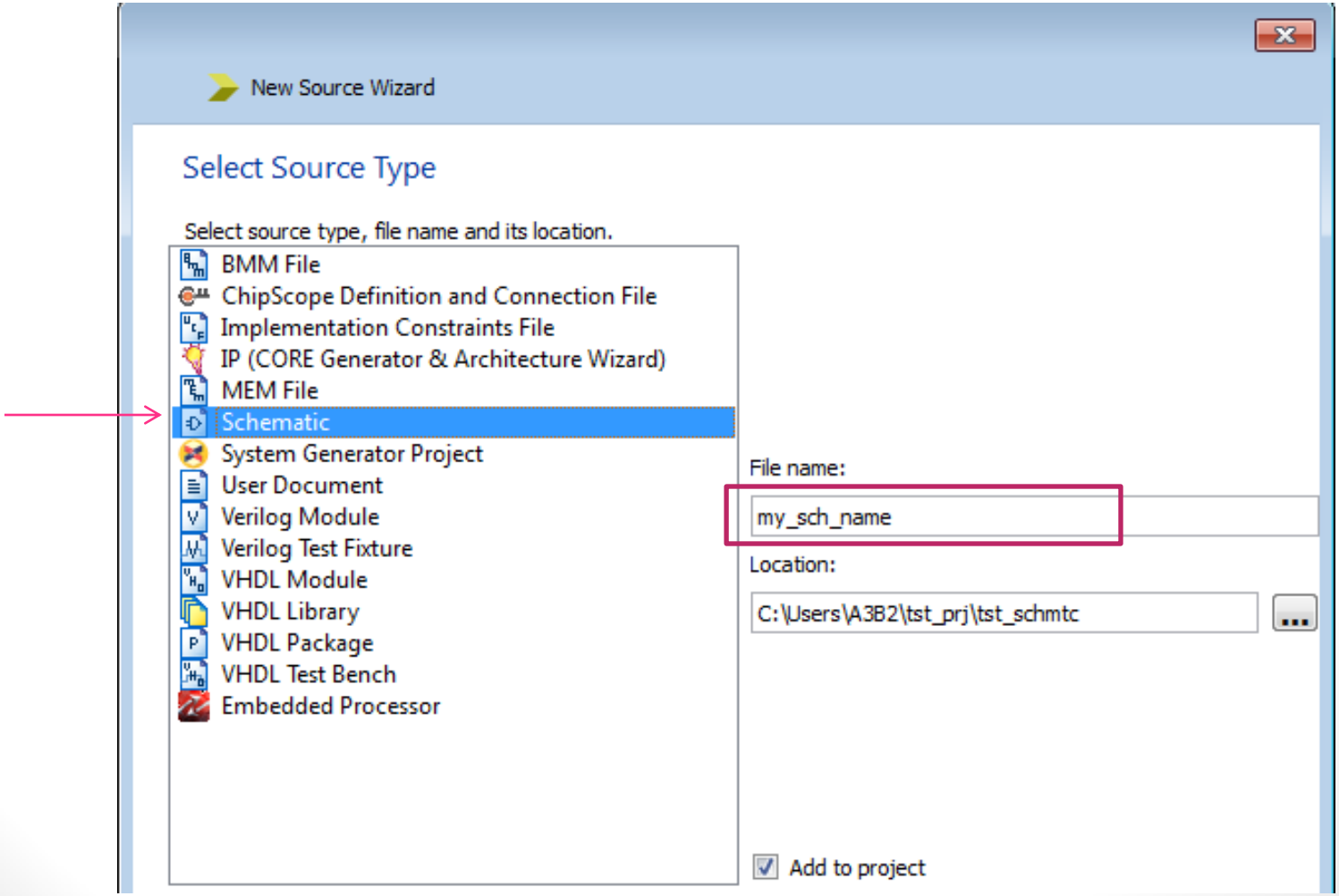


Create Project with top level as schematic

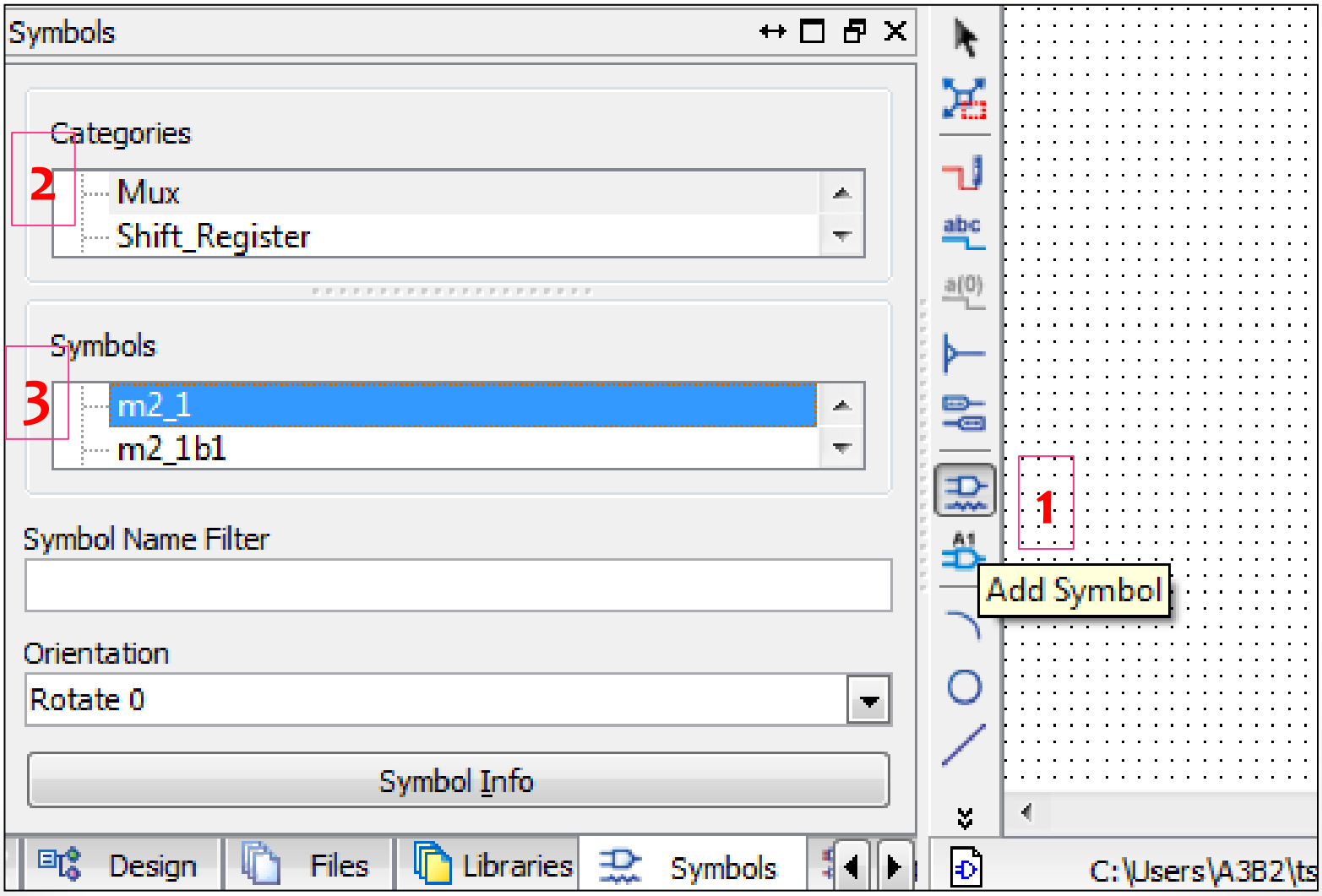


Add new schematic source

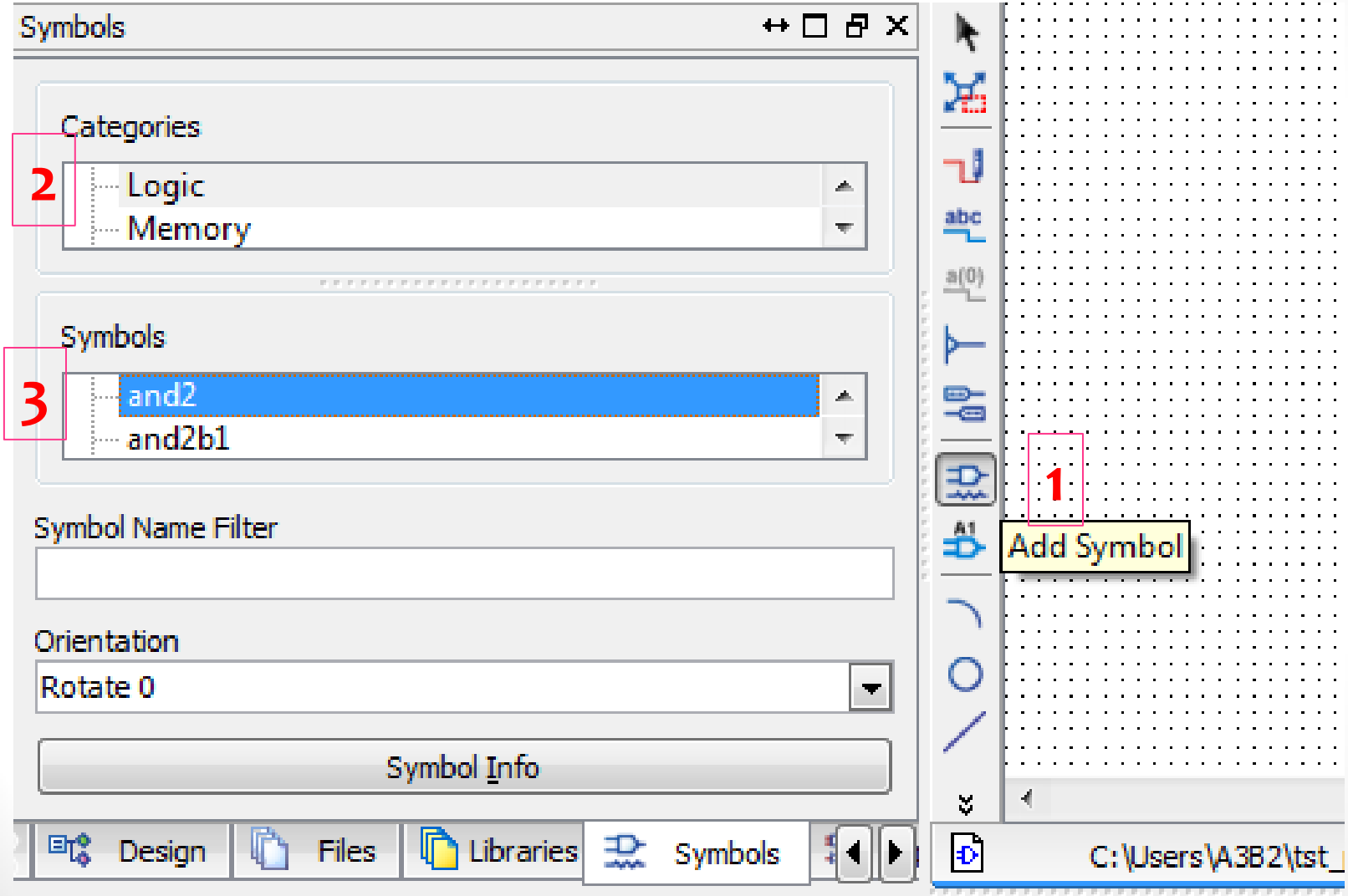
- Project → new source



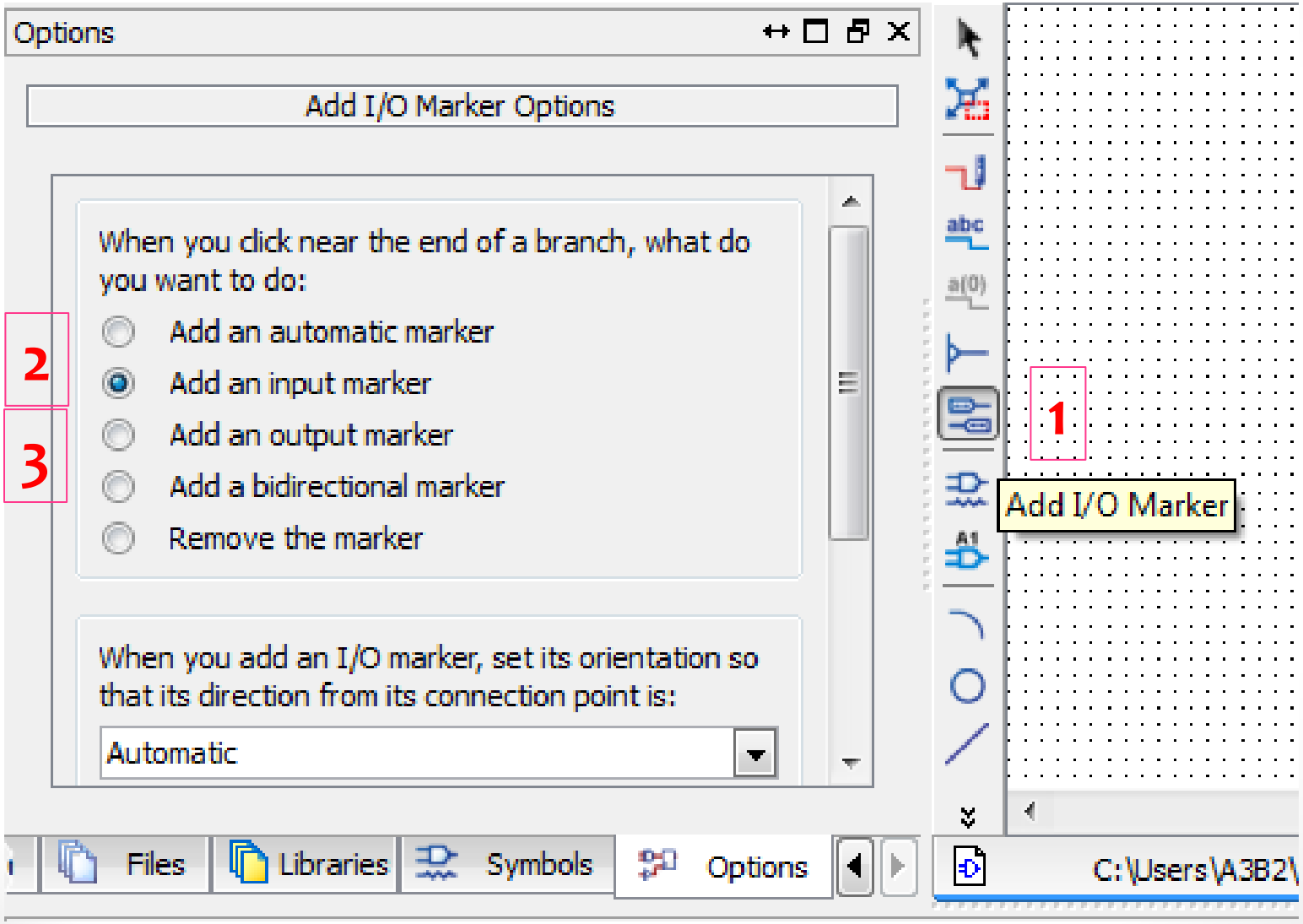
Add 2x1 Mux



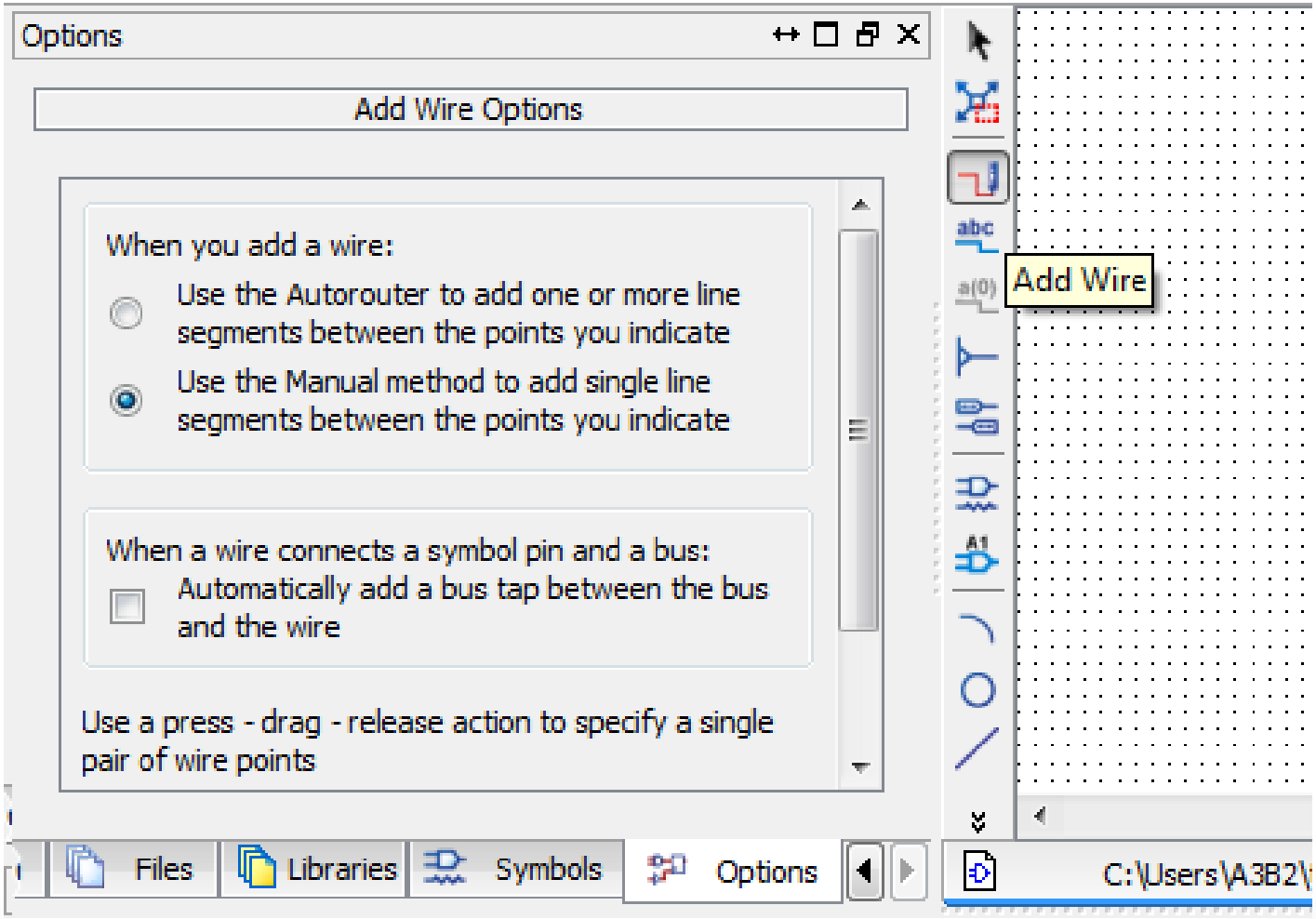
Add and gate



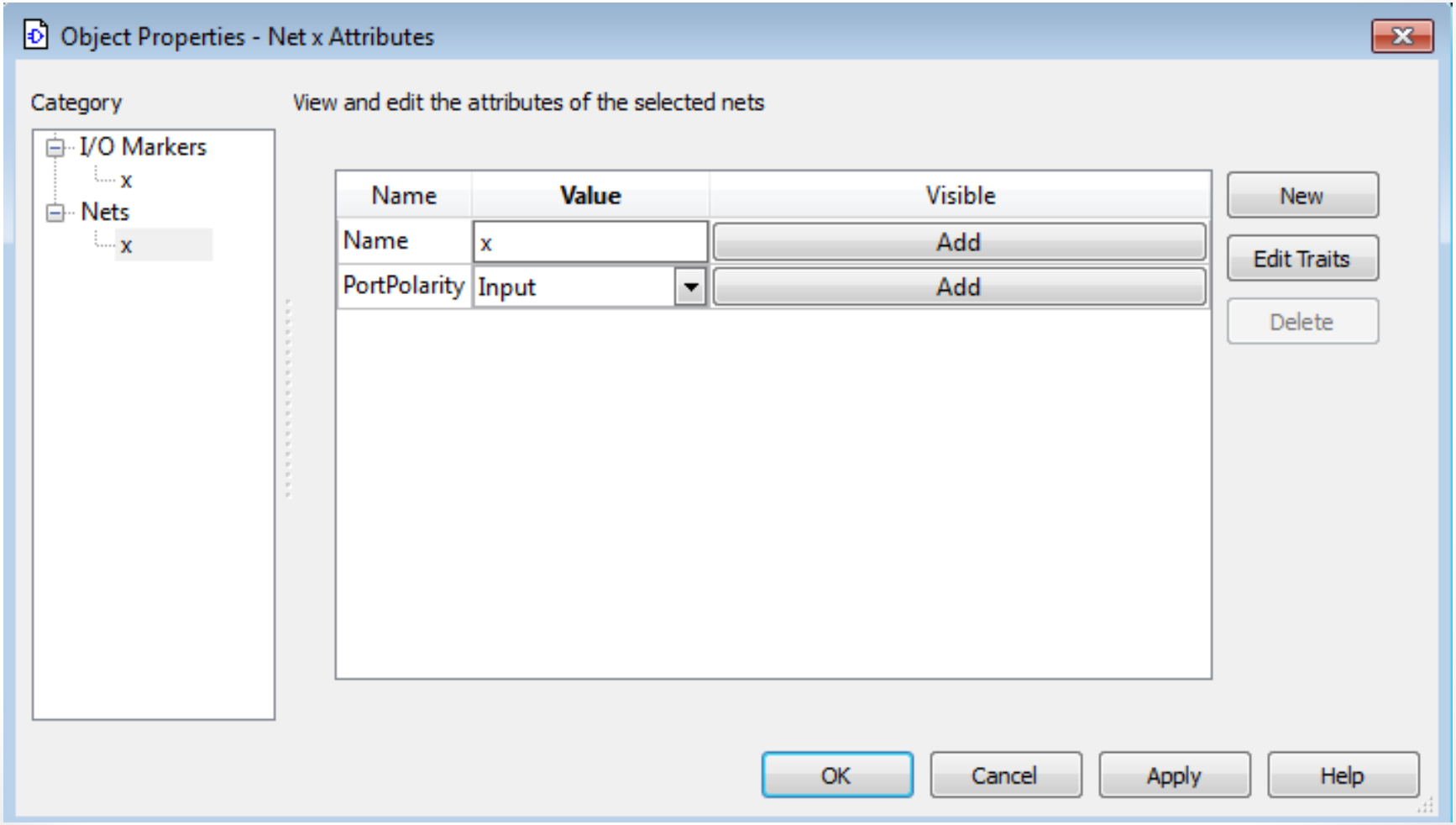
Add input & output ports



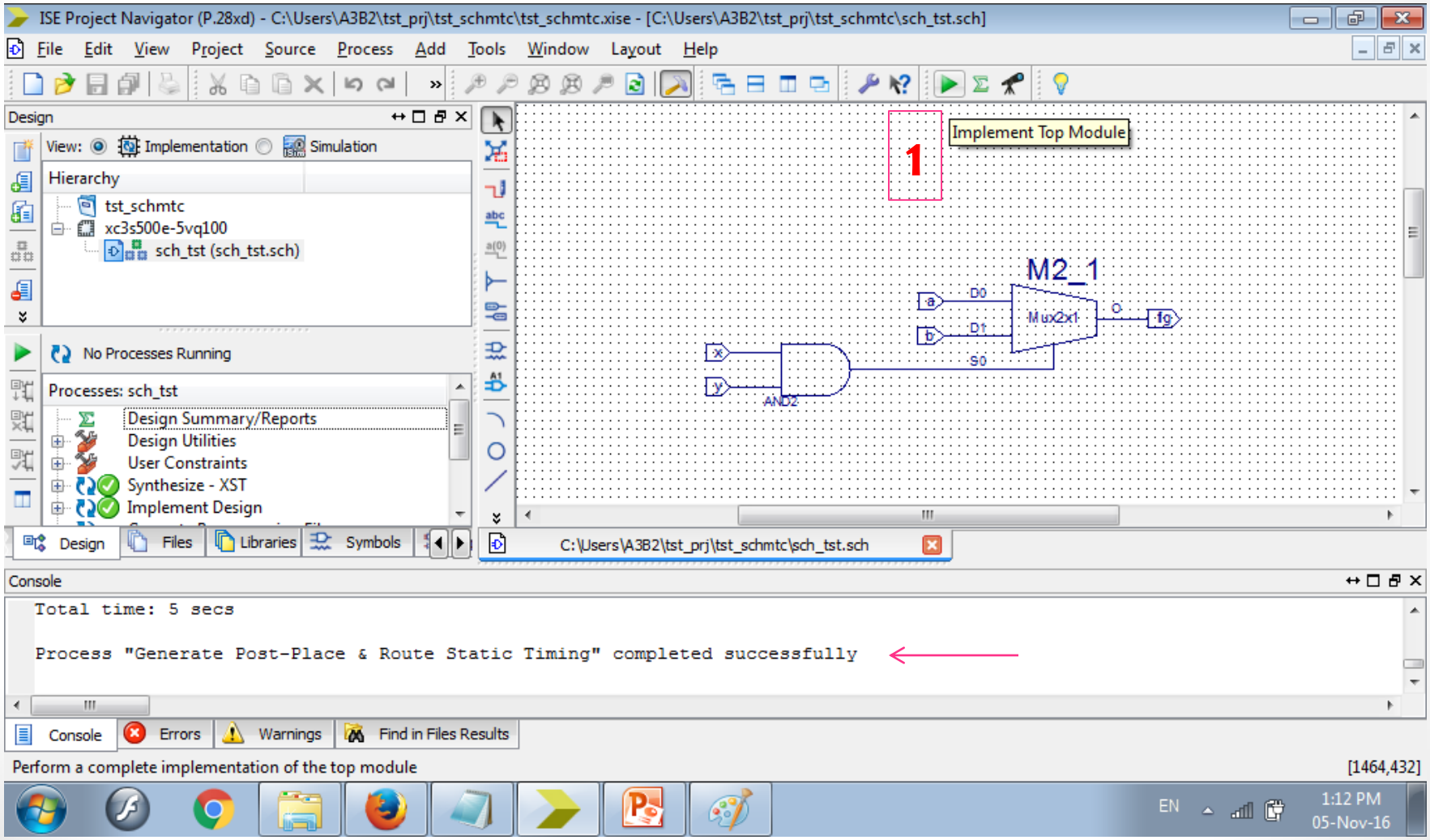
Add wires between blocks



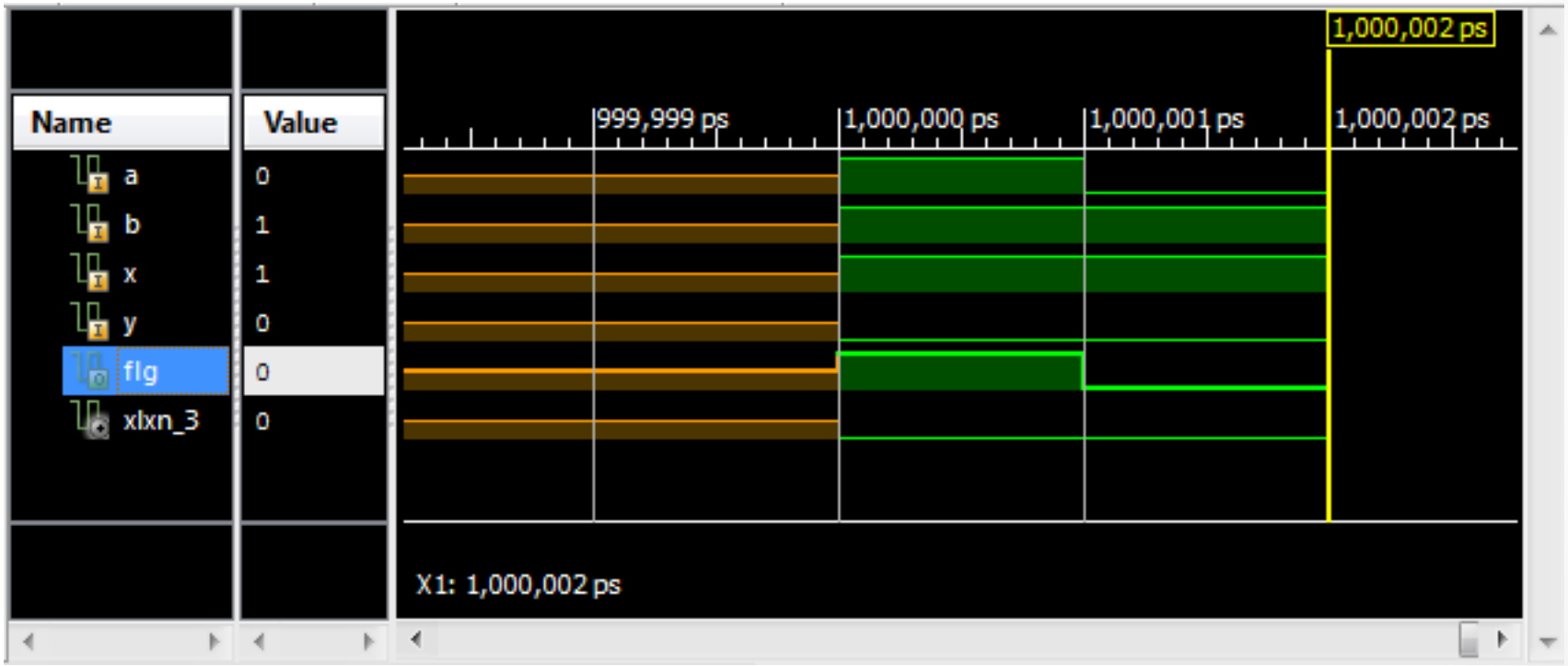
Rename the ports



Implement top module



Simulation Waveform

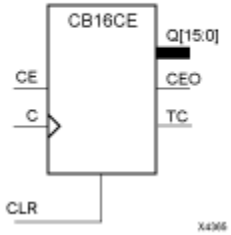


Example

- Use the schematic way to test the counter

CB16CE

Macro: 16-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear



Logic Table

Inputs			Outputs		
CLR	CE	C	Qz-Q0	TC	CEO
1	X	X	0	0	0
0	0	X	No change	No change	0
0	1	↑	Inc	TC	CEO

z = bit width - 1
TC = Qz•Q(z-1)•Q(z-2)•...•Q0
CEO = TC•CE

- For more details, refer to:
  - VHDL Tutorial: Learn by Example *by Weijun Zhang*
    - http://esd.cs.ucr.edu/labs/tutorial/
  - **VHDL GUIDELINES FOR SYNTHESIS**
    - http://www.utdallas.edu/~zxb107020/EE6306/Tutorial/VHDL.pdf
  - "**Introduction to VHDL**" presentation by Dr. Adnan Shaout, *The University of Michigan-Dearborn*
  - **The VHDL Cookbook**, *Peter J. Ashenden, 1st edition, 1990.*
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12135
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg

Designing with VHDL and FPGA

Instructor:

Dr. Ahmad El-Banna

LAB# 8
FALL 2016

Agenda

Packages and Libraries

- **Functions**
- **Packages**
- **Libraries**

FSM

- **Basics**
- **Examples**

Packages & Libraries

- Groups of procedures and functions that are related can be aggregated into a module that is called package.
- A package can be shared across many VHDL models.
- A package can also contains user defined data types and constants.
- A library is a collection of related packages.
- Packages and libraries serve as repositories for functions, procedures, and data types.

Functions

- A function computes and returns a value of specified type using the input parameters.
- Function declaration:
  - `function rising_edge(signal clock: in std_logic) return Boolean;`
- Parameters are used but not modified within the function.
  - the mode of input parameters is **in**.
  - functions do not have parameters of mode **out**.
  - In fact, we do not have to specify the mode.

Function Definition

```
function rising_edge(signal clock: std_logic)
return Boolean is
--
-- declarative region: declare variables local to the function
--
begin
--
-- body
--
return (value);
end function rising_edge;
```

formal parameters

- The function is called with the actual parameters.
- Example: `rising_edge(enable);`
- Types of formal and actual parameters must match.
- Actual parameters could be variable, signal, constant or an expression.

6

Functions

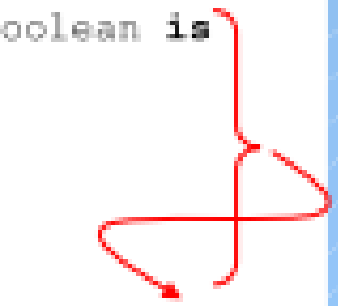
- When no class is specified, parameters are by default constants.
- `wait` statements cannot be used in functions' body.
  - functions execute in zero simulation time.
- Functions cannot call a procedure that has a `wait` statement in it.
- Parameters have to be mode `in`.
  - signals passed as parameters cannot be assigned values in the function body.

Pure vs. Impure Functions

- VHDL'93 supports two distinct types of functions:
- Pure functions
  - always return the same value when called with the same parameter values.
- Impure functions
  - may return different values even if they are called with the same parameter values at different times.
  - All the signals in the architecture is visible within the function body
  - Those signals may not appear in the function parameter list (e.g. ports of the entity)

Example

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dff is
  port(d, clock: in std_logic; q, qbar: out std_logic);
end entity dff;
architecture beh of dff is
  function rising_edge(signal clock:std_logic) return Boolean is
    variable edge: Boolean:=FALSE;
  begin
    edge:= (clock = '1' and clock'event);
    return (edge);
  end function rising_edge;
begin
  output: process is
  begin
    wait until (rising_edge(clock));
    q <= d after 5 ns;
    qbar <= not d after 5 ns;
  end process output;
end architecture beh;
```



Packages

- A package provides a convenient mechanism to store items that can be shared across distinct VHDL programs:
  - Those items are type definitions, functions, procedures, etc.
  - We group logically related sets of functions and procedures into a package.
  - When working with a large design project consisting of many small VHDL programs, it is convenient to have common procedures and functions in separate packages.
  - For example, a package containing definition of new types for registers, instructions, memories, etc. will certainly be useful for microprocessor design.

Package Declaration

- Package declaration
  - contains information about what is available in the package that we can use in our VHDL programs.
  - In other words, it contains interface or specifications of the functions and procedures that are available in the package.
  - For example,
  - list of functions and procedures,
  - what are the parameters they take
  - the type of input parameters
  - What it returns,
  - what is the type of the returning value
  - etc.

Package Declaration: Syntax

```
package package-name is
  package-item-declarations → these may be:
  -- subprogram declarations
  -- type declarations
  -- subtype declarations
  -- constant declarations
  -- signal declarations
  -- variable declarations
  -- file declarations
  -- alias declarations
  -- component declarations
  -- attribute declarations
  -- attribute specifications
  -- disconnection specifications
  -- use clauses
end [package] [package-name];
```

Package Declaration: Example

```
package synthesis_pack is
  constant low2high: time := 20 ns;
  type alu_op is (add, sub, mul, div, eql);
  attribute pipeline: Boolean;
  type mvl is ('U', '0', '1', 'Z');
  type mvl_vector is array (natural range <>) of mvl;
  subtype my_alu_op is alu_op range add to div;
  component nand2
    port(a, b: in mvl; c: out mvl);
  end component;
end synthesis_pack;
```

- Items declared in a package declaration can be accessed by other design units by using `library` and `use` clauses.

```
use work.synthesis_pack.all;
```

Package Body

- It basically contains the code that implements the subprograms
- Syntax:

```
package body package-name is
   package-body-item-declarations → These are:
   -- subprogram bodies
   -- complete constant declarations
   -- subprogram declarations
   -- type and subtype declarations
   -- file and alias declarations
   -- use clauses
end [package body] [package-name];
```

Package Body: Example

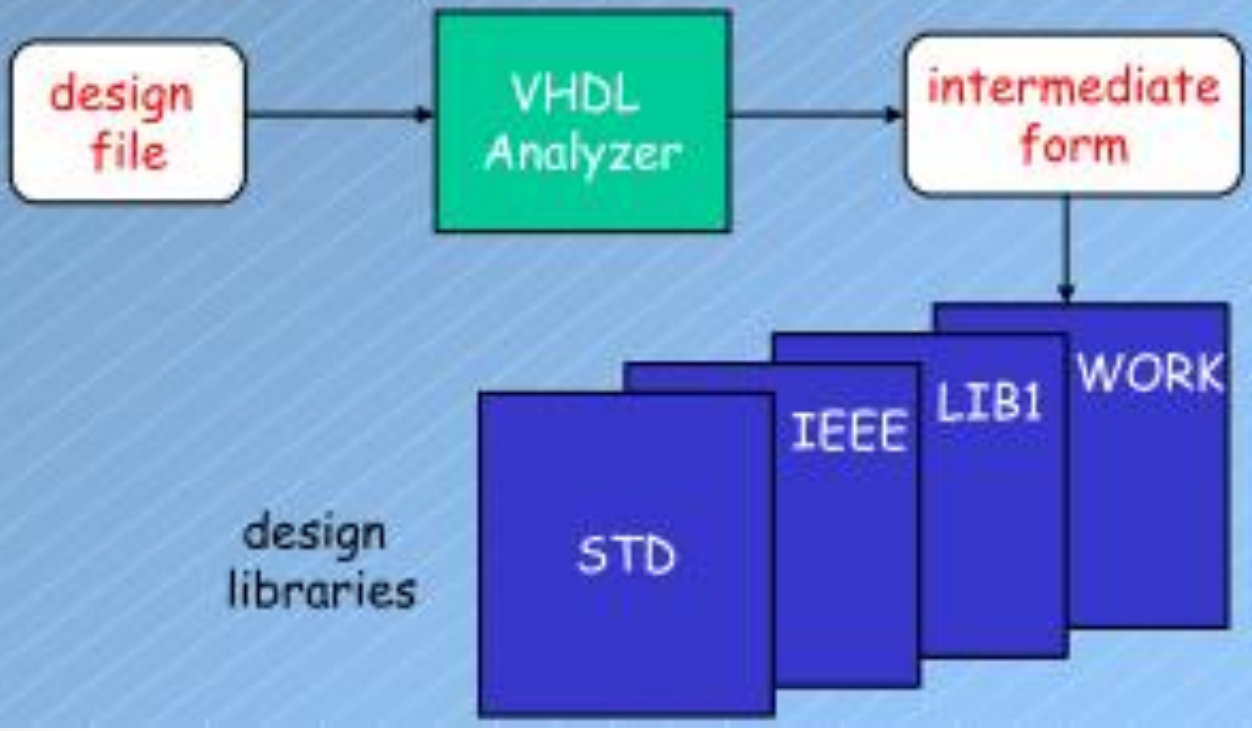
```
package body program_pack is
  use work.tables.all;
  constant prop_delay: time := 15 ns;
  function "and"(l, r: mvl) return mvl;
  begin
    return table_and(l, r);
    -- table_and is a 2-D constant defined in
    -- another package, called "tables" in the
    -- current working directory
  end "and";
  procedure load(signal array_name: inout mvl_vector;
     start_bit, stop_bit, int_value: in integer) is
  begin
    -- procedure behavior here
  end load;
end program_pack;
```

Libraries

- Design units (design files)
  - entity declaration,
  - architecture body,
  - configuration declaration,
  - package declaration,
  - package body
- Each design unit is analyzed (compiled) and placed in a design library.
  - recall that libraries are generally implemented as directories and are referenced by a logical name.
  - this logical name corresponds to physical path to the corresponding directory.

Compilation Process

- VHDL analyzer verify the syntactic and semantic correctness of the source
- then compiles each design unit into an intermediate form.
- Each intermediate form is stored in a design library called working library.



Libraries: Visibility

- Implicit visibility
  - In VHDL, the libraries STD and WORK are implicitly declared.
- Explicit visibility is achieved through
  - `library` clause
  - `use` clause
  - Example:

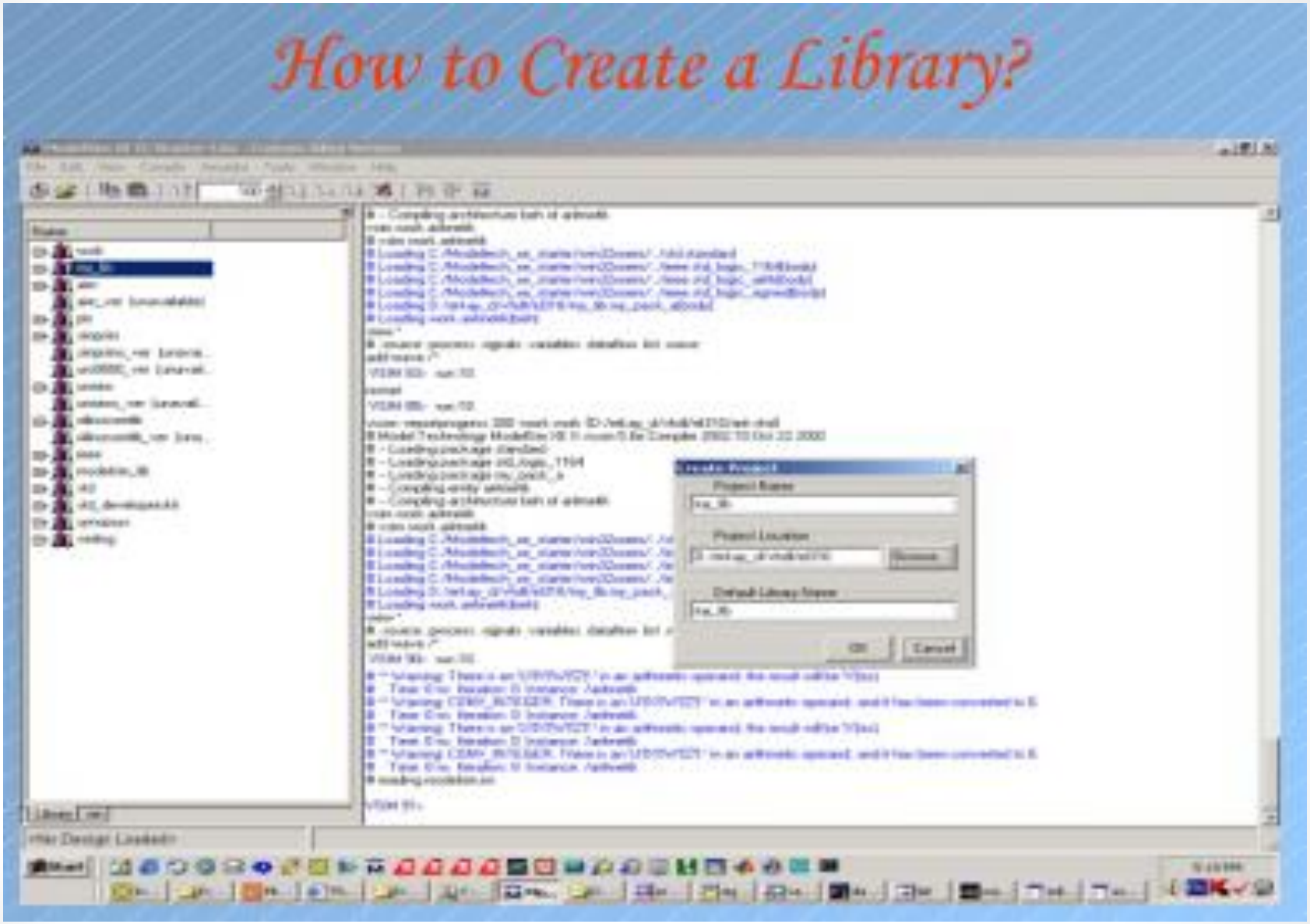
```
library IEEE;
use IEEE.std_logic_1164.all;
```

- Once a library is declared, all of the subprograms, type declarations in this library become visible to our programs through the use of `use` clause.

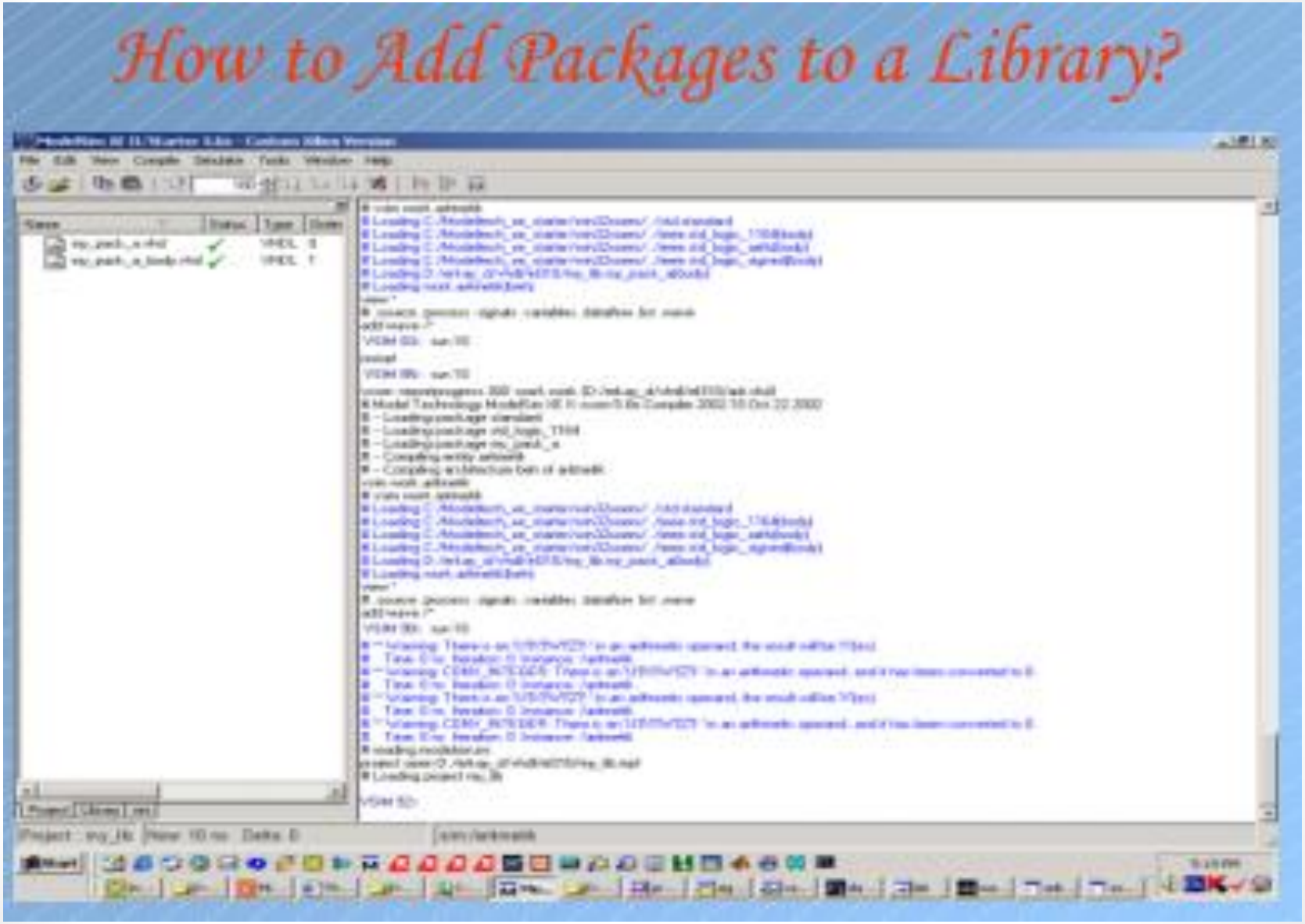
Context Clauses

- Examples for context clauses:
  -
```
library IEEE;
use IEEE.std_logic_1164.all;
```
- Context clauses only applies the following design entity
  - if a file contains more than one entity, context clauses must precede each of them to provide the appropriate visibility to each design entity.

How to Create a Library?



How to Add Packages to a Library?



Package Declaration

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

package my_pack_a is
   subtype word is std_logic_vector(15 downto 0);
   function "+" (op1, op2: word) return word;
   function "-" (op1, op2: word) return word;
   function "*" (op1, op2: word) return word;
end package my_pack_a;
```

Package Body

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;
package body my_pack_a is
  function "+" (op1, op2: word) return word is
    variable result: word;
    variable a, b, c: integer;
  begin
    a := conv_integer(op1);
    b := conv_integer(op2);
    c := a + b;
    result := conv_std_logic_vector(c, 16);
    return result;
  end function;
...
end package body my_pack_a;
```

Package Body

```
...
  function "-" (op1, op2: word) return word is
    variable result: word; variable a, b, c: integer;
  begin
    a := conv_integer(op1); b := conv_integer(op2);
    c := a - b;
    result := conv_std_logic_vector(c, 16);
    return result;
  end function;
  function "*" (op1, op2: word) return word is
    variable result: word; variable a, b, c: integer;
  begin
    a := conv_integer(op1);
    b := conv_integer(op2);
    c := a * b;
    result := conv_std_logic_vector(c, 16);
    return result;
  end function;
  end package body my_pack_a;
```

Example using this Package

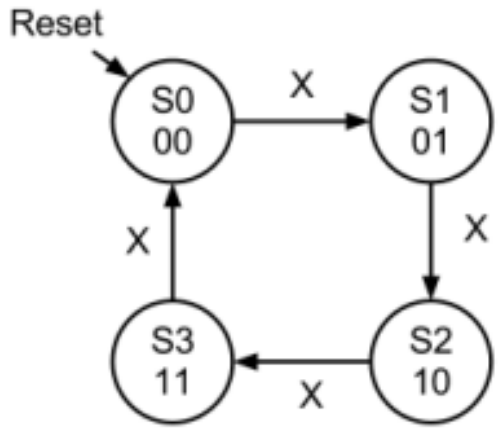
```
library IEEE;
library my_lib;
use IEEE.std_logic_1164.all;
use my_lib.my_pack_a.all;

entity aritmetik is
end entity;

architecture beh of aritmetik is
  signal a, b, c: word;
begin
 a <= x"abcd";
 b <= x"1347";
 c <= a + b;
end architecture beh;
```

FSM

FSM Implementation




```
architecture Behavioral of ColorFSM is

   signal state : STD_LOGIC_VECTOR(1 downto 0) := "00";

begin
```

add the reset condition

```
Color_FSM : process (clk, reset)
begin

   if (reset = '1') then
      state <= "00";
   end if;

end process;
```

FSM Implementation..

add a clock edge condition

```
if (reset = '1') then
   -- Reset state
   state <= "00";
elsif (clk'event and clk = '1') then
   -- Transitions will go inside here
end if;
```

add the state transitions!

```
if (reset = '1') then
   -- Reset state
   state <= "00";
elsif (clk'event and clk = '1') then
   -- Transitions go inside here
   if (state = "00" and X = '1') then
      -- Transition to S1 from S0 when X is 1
      state <= "01";
   end if;
end if;
```

FSM Implementation…

add the rest of the state transitions

```
Color_FSM : process (clk, reset)
begin

   if (reset = '1') then
      -- Reset state
      state <= "00";
   elsif (clk'event and clk = '1') then
      -- Transitions go inside here
      if (state = "00" and X = '1') then
         state <= "01";
      elsif (state = "01" and X = '1') then
         state <= "10";
      elsif (state = "10" and X = '1') then
         state <= "11";
      elsif (state = "11" and X = '1') then
         state <= "00";
      else
         -- Do nothing, hold state
      end if;
   end if;

end process;
```

assign the value of the FSM output.

```
   S <= state;

end Behavioral;
```

- Find FSM full examples at:
- https://www.usna.edu/ECE/ec262/notes/Lesson_18_FSM_VHDL_student.pdf
- https://www.seas.upenn.edu/~ese171/vhdl/VHDL_FSM_Tutorial.pdf

- For more details, refer to:
  - VHDL Tutorial: Learn by Example *by Weijun Zhang*
    - http://esd.cs.ucr.edu/labs/tutorial/
  - **VHDL GUIDELINES FOR SYNTHESIS**
    - http://www.utdallas.edu/~zxb107020/EE6306/Tutorial/VHDL.pdf
  - "**Introduction to VHDL**" presentation by Dr. Adnan Shaout, *The University of Michigan-Dearborn*
  - **The VHDL Cookbook**, *Peter J. Ashenden, 1st edition, 1990.*
  - *http://people.sabanciuniv.edu/erkays/el310/ch09.pdf*
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12135
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg