



University of San Carlos | Department of  
**COMPUTER ENGINEERING**

CPE 3101L – Introduction to HDL

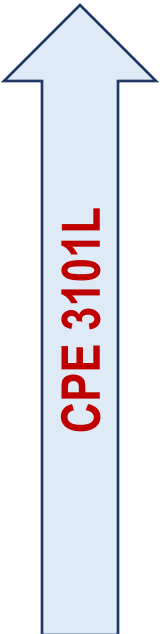
# Unit 3: Structural Modeling of Combinational Circuits

Lecture 1

# Outline

- Abstraction Levels in Digital Design
- Modeling Styles in Verilog
- Structural Modeling of Combinational Circuits
- Simple Testbenches for Combinational Logic

# Abstraction Levels in Digital Design



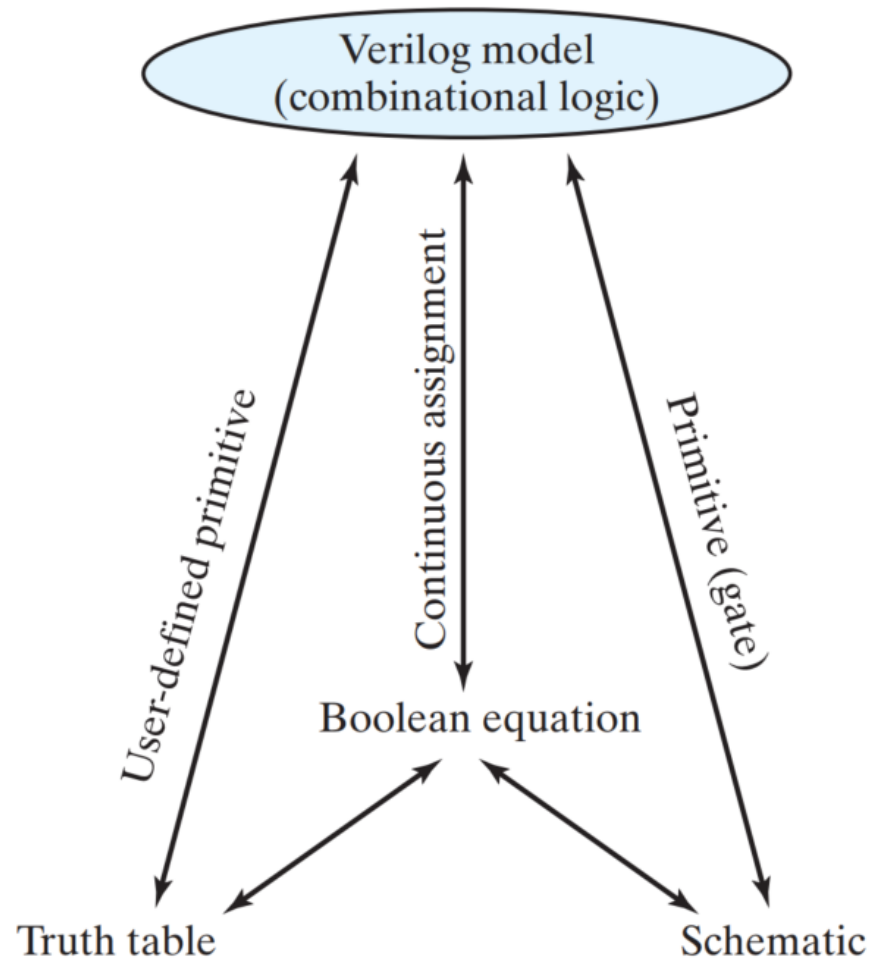
LEVEL	COMPONENTS	REPRESENTATION
Behavioral Level	Algorithms Finite State Machines (FSMs)	Algorithmic representation of system functions
System Level ( <i>Architectural</i> )	Processors, memories, functional subsystems	Interconnection of major functional units
Register Transfer Level (RTL)	Registers, adders, ALUs, counters, controllers	Data movement and transformation Required sequential control
Logic Gate Level	Logic gates, flip-flops	Implementation of register level components using gates and flip-flops
Transistor Level ( <i>Switch</i> )	Transistors, resistors, capacitors	Implementation of gates and flip-flops using discrete components

# Modeling Styles in Verilog

- Structural modeling (*Units 2-3*)
  - Uses *instantiations* of predefined gate primitives, ~~user-defined primitives (UDPs)~~, and other HDL modules
  - Describes a circuit by specifying inner modules (or gates) and how they are connected with each other (*circuit diagram*)
- Dataflow modeling (*Unit 4*)
  - Uses *continuous assignment* statements (*assign* keyword)
  - Describes a circuit using its *Boolean equations*
- Behavioral modeling (*Units 5-6*)
  - Uses *procedural assignment* statements (*always* keyword)
  - Describes a circuit according to its *behavior or function*
  - Involves designing at a higher level of abstraction

# Combinational Logic in Verilog

- To create the HDL model of combinational circuits:
- Structural modeling (*Units 2-3*)
  - Gate primitives, UDPs, other HDL modules
  - Needs schematic or truth tables or other HDL modules
- Dataflow modeling (*Unit 4*)
  - Needs Boolean equations
- Behavioral modeling (*Unit 5*)
  - Needs behavior or functionality of the circuit
  - *More frequently used in sequential logic (Unit 6)*



# Structural Modeling of Combinational Circuits

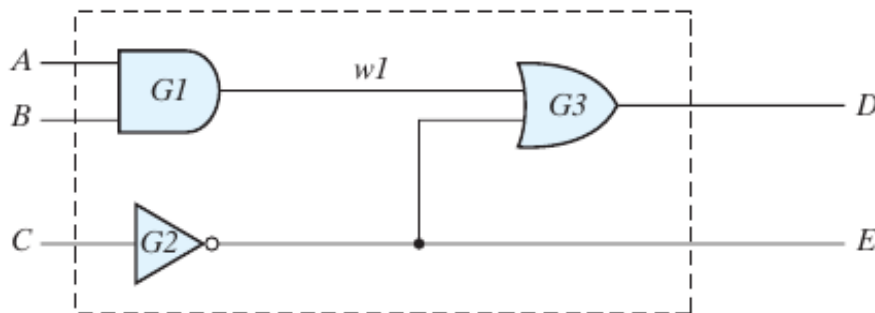
- Uses *instantiations* of predefined gate primitives and other HDL modules
  - Connect them through port maps (*implicit/explicit port mapping*)
- Describes a circuit by specifying inner modules (or gates) and how they are connected with each other
  - A *circuit diagram* is necessary for design entry
- Gate primitive instances and module instances are concurrent statements
  - **Concurrency** – Independent processes that are evaluated at the same time during simulation (*in parallel/not sequential*)
  - *Concurrent statements can be rearranged*

# Structural Model (Recap from Unit 2)

- Typical Contents:

- Gate primitives
- Module instantiations
- Port maps
- Nets (*wires*)

- Example:



- Module Declaration:

- Verilog-1995 (IEEE Standard 1364-1995)
- Ports are required to be declared at least twice:
  - Port list
  - Port direction

```
module GateLevel (A, B, C, D, E);
    input    A, B, C;
    output   D, E;
    wire     w1;

    and      G1 (w1, A, B);
    not      G2 (E, C);
    or       G3 (D, w1, E);

endmodule
```

# Module Declaration in Verilog-2001

- Verilog-2001 (IEEE Standard 1364-2001):
  - Ports are **declared only once** (*ANSI-C style port list*)

```
module [module_name] (
    port_declaration [port_name1], [port_name2], ... ,
    port_declaration [port_name1], [port_name2], ... );
    ...
endmodule
```

```
//
// Module declaration in Verilog-1995
//
module GateLevel (A, B, C, D, E);
    input      A, B, C;
    output    D, E;
    wire      w1;

    and       G1 (w1, A, B);
    not       G2 (E, C);
    or        G3 (D, w1, E);

endmodule
```



```
//
// Module declaration in Verilog-2001
//
module GateLevel (
    input      A, B, C,
    output    D, E
);

    wire      w1;

    and       G1 (w1, A, B);
    not       G2 (E, C);
    or        G3 (D, w1, E);

endmodule
```

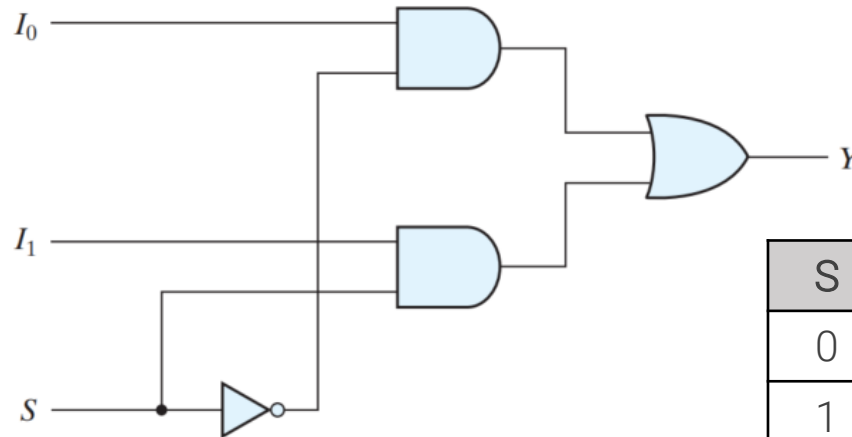
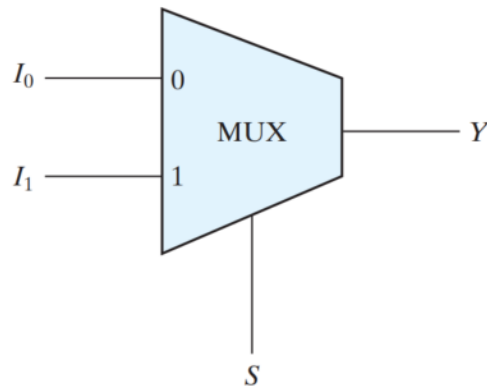


# Example 1: 2-to-1 Line Multiplexer

- Create a Verilog description of a 2-to-1 line mux using gate primitives.

- Specifications:**

- Inputs:  $I[1:0]$ ,  $S$
- Output:  $Y$



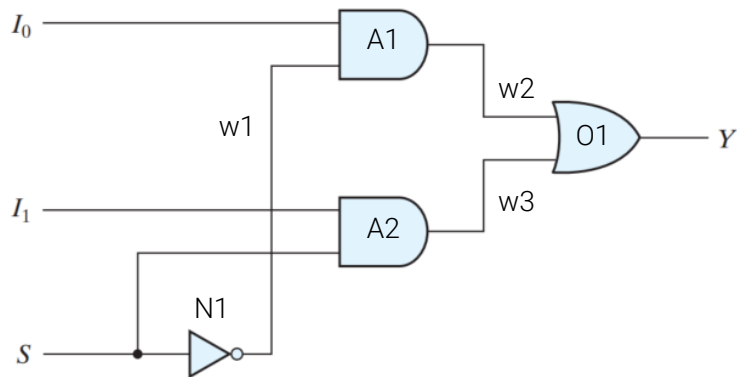
S	Y
0	$I_0$
1	$I_1$

# Example 1: Design Entry (2x1 Mux)

- Entity Diagram:



- Schematic Diagram:



- Verilog HDL Description:

```
// Structural modeling of 2x1 mux
module Mux_2x1 (
    input [1:0] I,
    input S,
    output Y
);

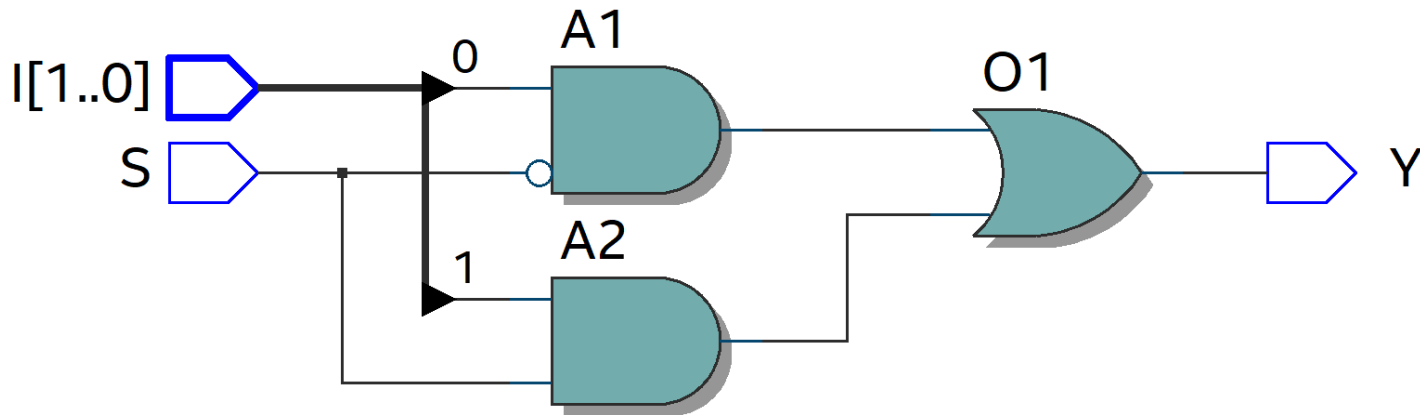
    // declare signals
    wire w1, w2, w3;

    // implement with gate primitives
    and A1 (w2, I[0], w1);
    and A2 (w3, I[1], S);
    not N1 (w1, S);
    or O1 (Y, w2, w3);

endmodule
```

# Example 1: Synthesized Design (2x1 Mux)

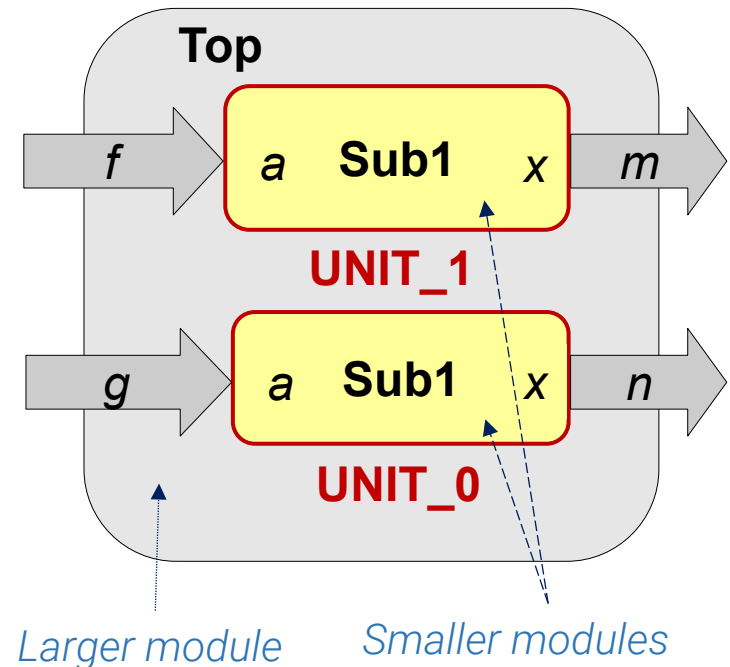
- RTL Schematic View:



Total logic elements	1
Total registers	0
Total pins	4

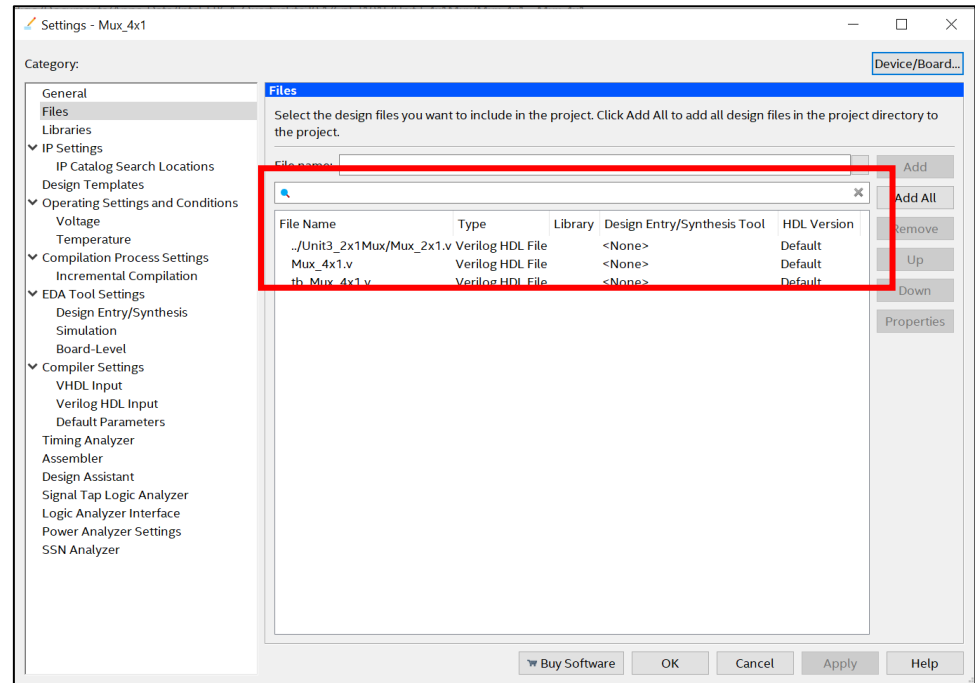
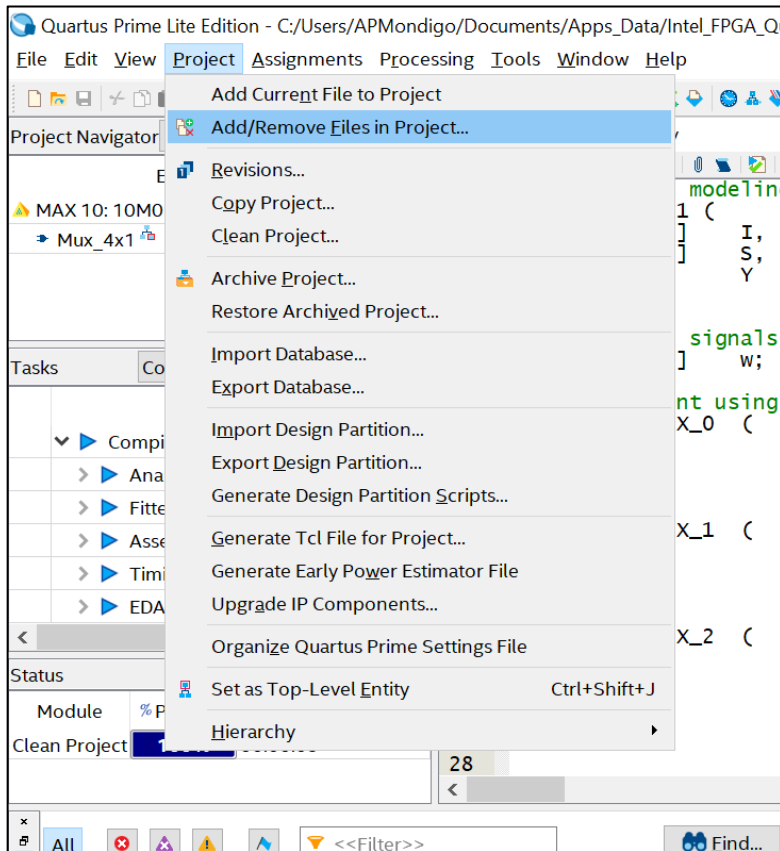
# Instantiated Modules in Structural Modeling

- Smaller modules (sub-modules) must already exist and should be “visible” to where it will be instantiated (top module)
- Two ways:
  - Use of Quartus Lite Prime
    - *We will use this for CPE 3101L*
  - Use of Verilog Macro ``include`



# Use of Quartus Lite Prime

- Project Menu > Add/Remove Files in Project > Settings: Category: Files
  - This will allow the existing modules' visibility within the project



## Example 2: 4-to-1 Line Multiplexer

- Create a Verilog structural description of a 4-to-1 line mux using 2-to-1 line muxes (*from Example 1*).

- Specifications:**

- Inputs:  $I[3:0]$ ,  $S[1:0]$
- Output:  $Y$

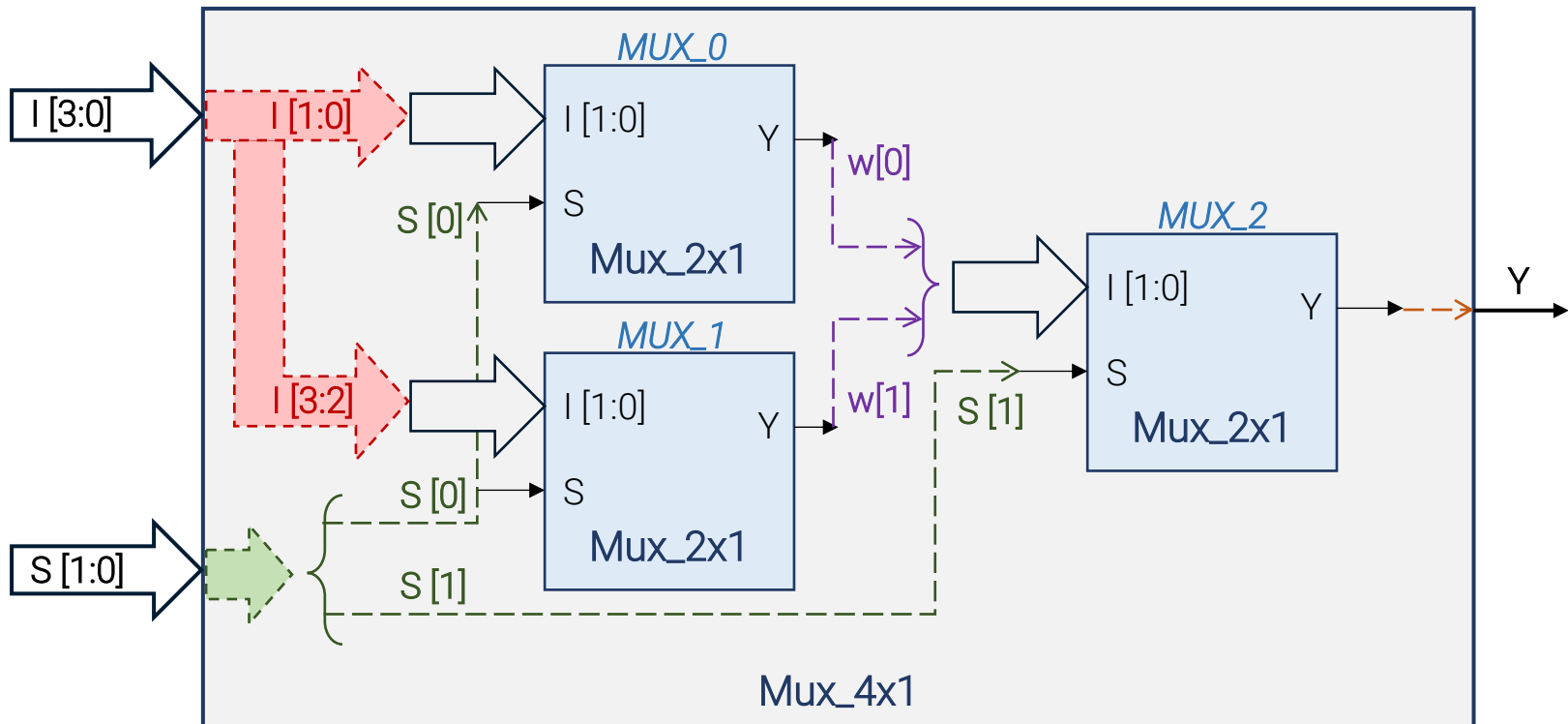
$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$



# Example 2: Block Diagram

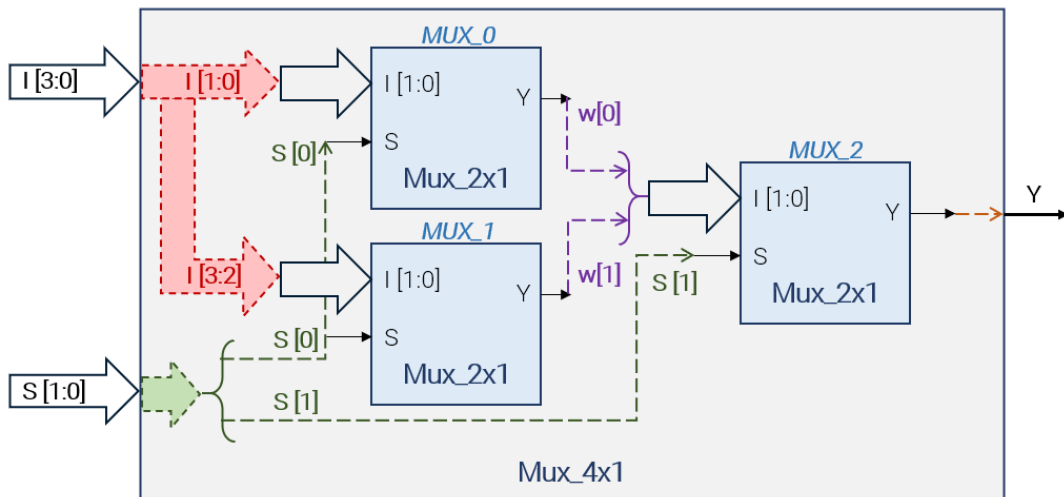
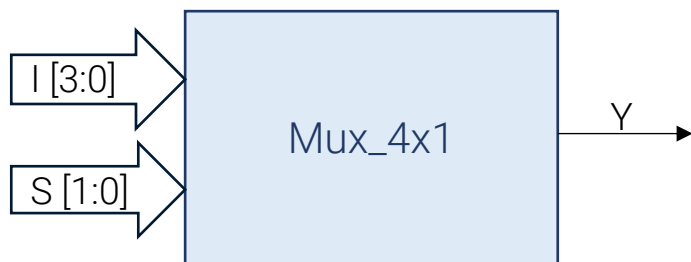
- Uses three 2x1 muxes

$S_1$	$S_0$	$Y$	
0	0	$I_0$	$MUX_0$
0	1	$I_1$	
1	0	$I_2$	$MUX_1$
1	1	$I_3$	



# Example 2: Design Entry (4x1 Mux)

- Verilog HDL Description:



```
// Structural modeling of 4x1 mux
module Mux_4x1 (
    input [3:0] I,
    input [1:0] S,
    output Y
);

    // declare signals
    wire [1:0] w;

    // implement using 2x1 muxes
    Mux_2x1 MUX_0 (
        .I (I[1:0]),
        .S (S[0]),
        .Y (w[0])
    );

    Mux_2x1 MUX_1 (
        .S (S[0]),
        .I (I[3:2]),
        .Y (w[1])
    );

    Mux_2x1 MUX_2 (
        .Y (Y),
        .I (w),
        .S (S[1])
    );

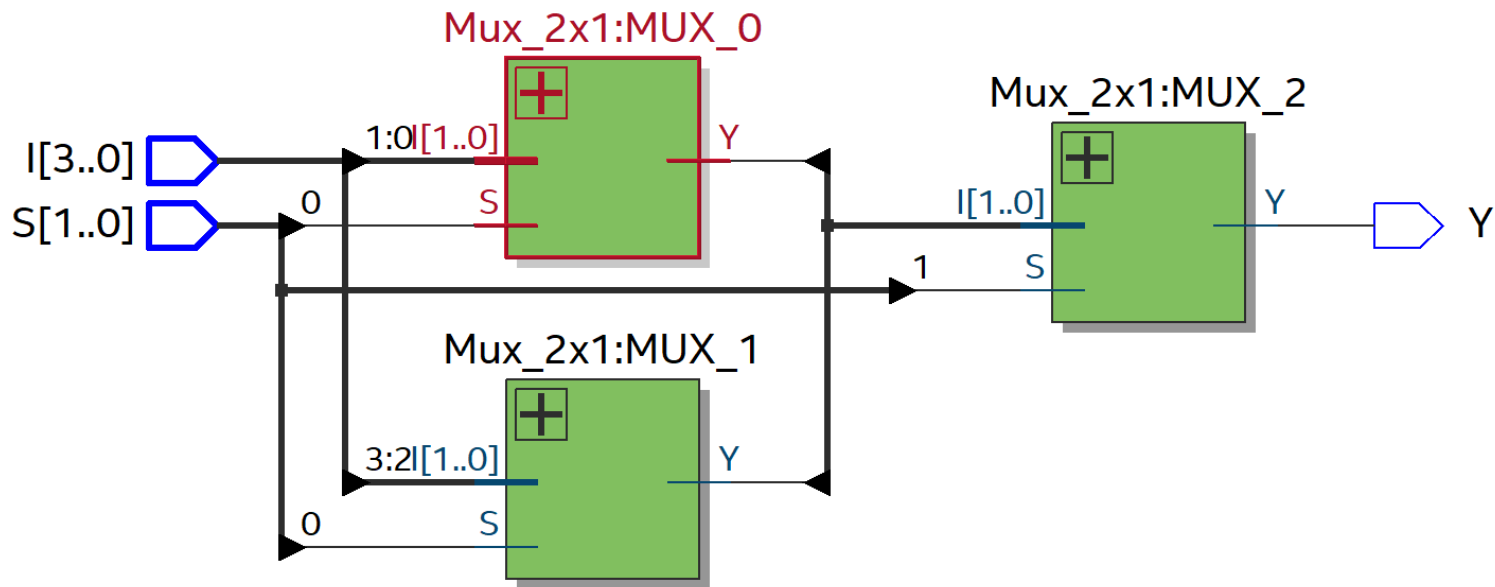
endmodule
```



# Example 2: Synthesized Design (4x1 Mux)

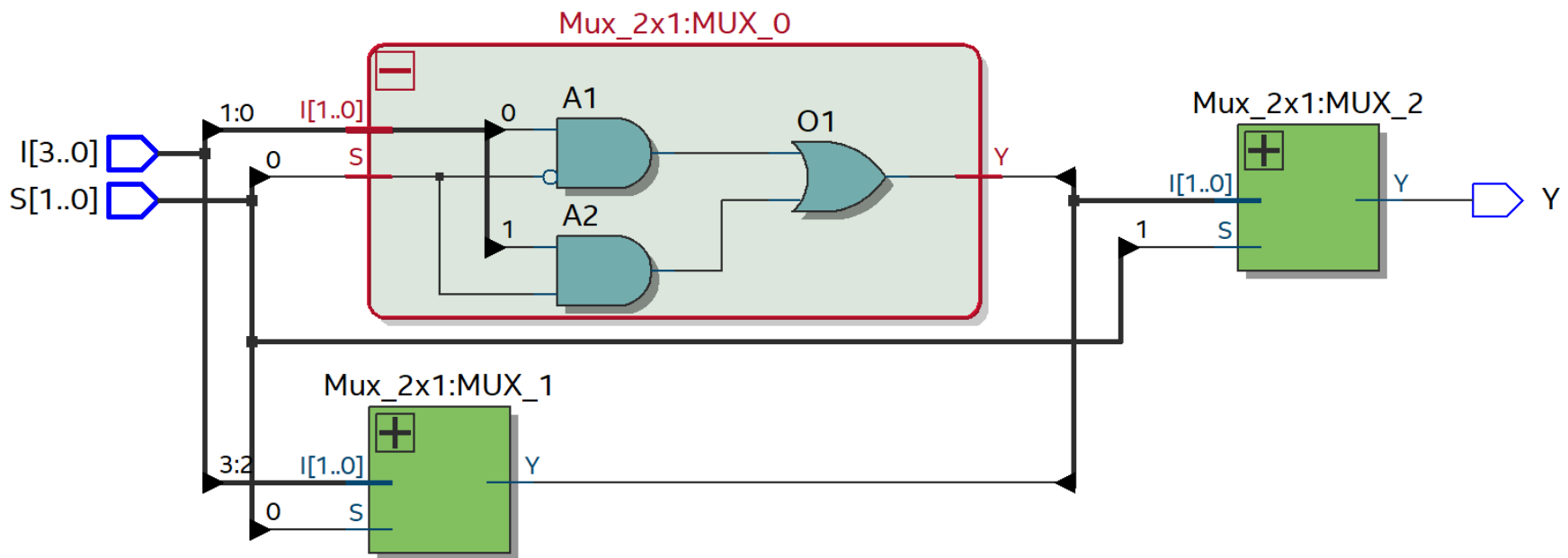
- RTL Schematic View:
  - Top Level View

Total logic elements	2
Total registers	0
Total pins	7



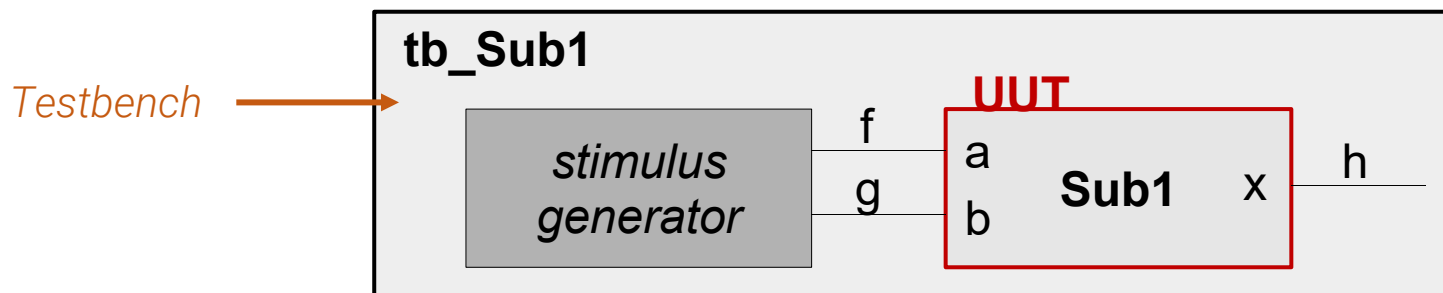
# Example 2: Synthesized Design (4x1 Mux)

- RTL Schematic View:
  - Inside `Mux_2x1` sub-module (*Instance MUX\_0*)



# Simple Testbenches for Combinational Logic

- A **testbench** is an HDL code used for supplying stimuli (*inputs*) to an HDL-based unit under test (UUT) in order to observe its behavior/response during simulation



```

`timescale 1 ns / 1 ps
module tb_Sub1 (); // Testbench module with no ports
    reg f, g; // all inputs to UUT are reg type
    wire h; // all outputs from UUT are wire type

    // Instantiate UUT and connect to external signals
    Sub1 UUT ( .a(f), .b(g), .x(h) );

    // Generate stimuli
endmodule

```

```

// Generate stimuli
initial
begin
    f = 0;    g = 0;    #10
    f = 0;    g = 1;    #10
    f = 1;    g = 0;    #10
    f = 1;    g = 1;    #10

    //system task to end simulation
    $stop;
end

```

# Simple Testbenches for Combinational Logic

- From Unit 2, the following were introduced:
  - Testbench has **no ports** (*no inputs/outputs*)
  - UUT is **instantiated** in the testbench file (*connected by a port map*)
  - Input signals to UUT are declared as **reg**
  - Output signals from UUT are declared as **wire**
  - Generate stimuli in the testbench inside an **initial** block
  - Use **gate delays (#)** to generate input stimuli timing sequence
  - System task **\$stop** is used to terminate the simulation

```
`timescale 1 ns / 1 ps
module tb_Sub1 ();    // Testbench module with no ports
    reg f, g;         // all inputs to UUT are reg type
    wire h;           // all outputs from UUT are wire type

    // Instantiate UUT and connect to external signals
    Sub1 UUT ( .a(f), .b(g), .x(h) );

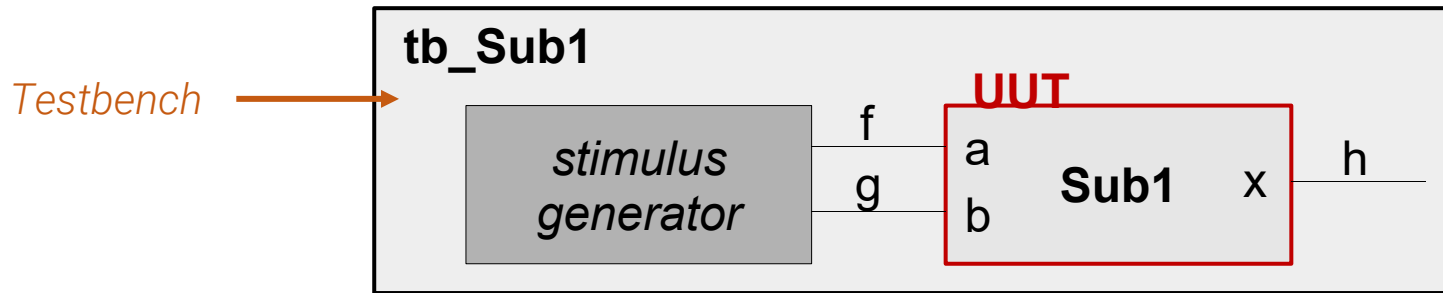
    // Generate stimuli
endmodule
```

```
// Generate stimuli
initial
begin
    f = 0;    g = 0;    #10
    f = 0;    g = 1;    #10
    f = 1;    g = 0;    #10
    f = 1;    g = 1;    #10

    //system task to end simulation
    $stop;
end
```

# Simple Testbenches for Combinational Logic

- For combinational circuits, *all input combinations* or *a sufficient number of test cases must be supplied to the UUT.*



```
`timescale 1 ns / 1 ps
module tb_Sub1 (); // Testbench module with no ports
    reg f, g; // all inputs to UUT are reg type
    wire h; // all outputs from UUT are wire type

    // Instantiate UUT and connect to external signals
    Sub1 UUT ( .a(f), .b(g), .x(h) );

    // Generate stimuli
endmodule
```

```
// Generate stimuli
initial
begin
    f = 0; g = 0; #10
    f = 0; g = 1; #10
    f = 1; g = 0; #10
    f = 1; g = 1; #10

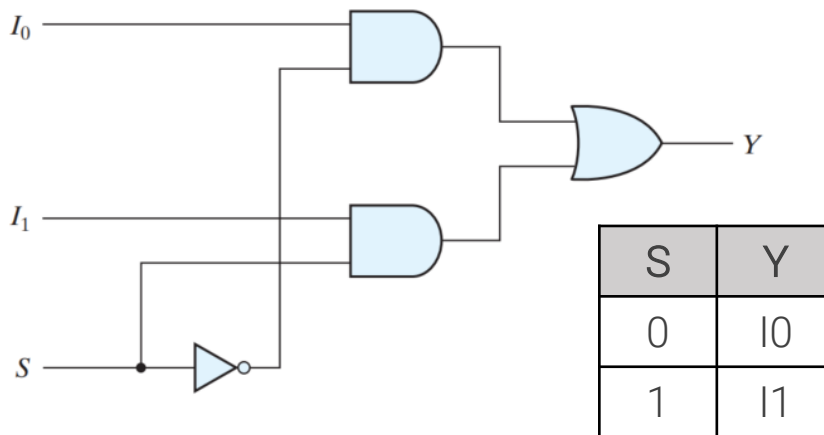
    //system task to end simulation
    $stop;
end
```

# Example 1: Simulation (2x1 Mux)

- Entity Diagram:



- Schematic Diagram with Truth Table:



- Testbench File for *Mux\_2x1.v*:

```
// Testbench file of 2x1 mux
`timescale 1 ns / 1 ps
module tb_Mux_2x1 ();

    // inputs as reg, outputs as wire
    reg    [1:0]    I;
    reg      S;
    wire     Y;

    // instantiate UUT
    Mux_2x1 UUT ( I, S, Y);

    // generate stimuli
    initial begin
        S  = 1'b0;  I  = 2'b00; #5
                I  = 2'b01; #5
                I  = 2'b10; #5
                I  = 2'b11; #5

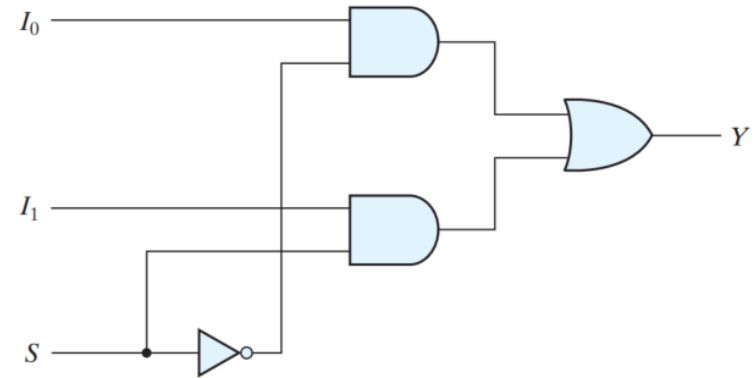
        S  = 1'b1;  I  = 2'b00; #5
                I  = 2'b01; #5
                I  = 2'b10; #5
                I  = 2'b11; #10

        // system task to stop simulation
        $stop;
    end
endmodule
```

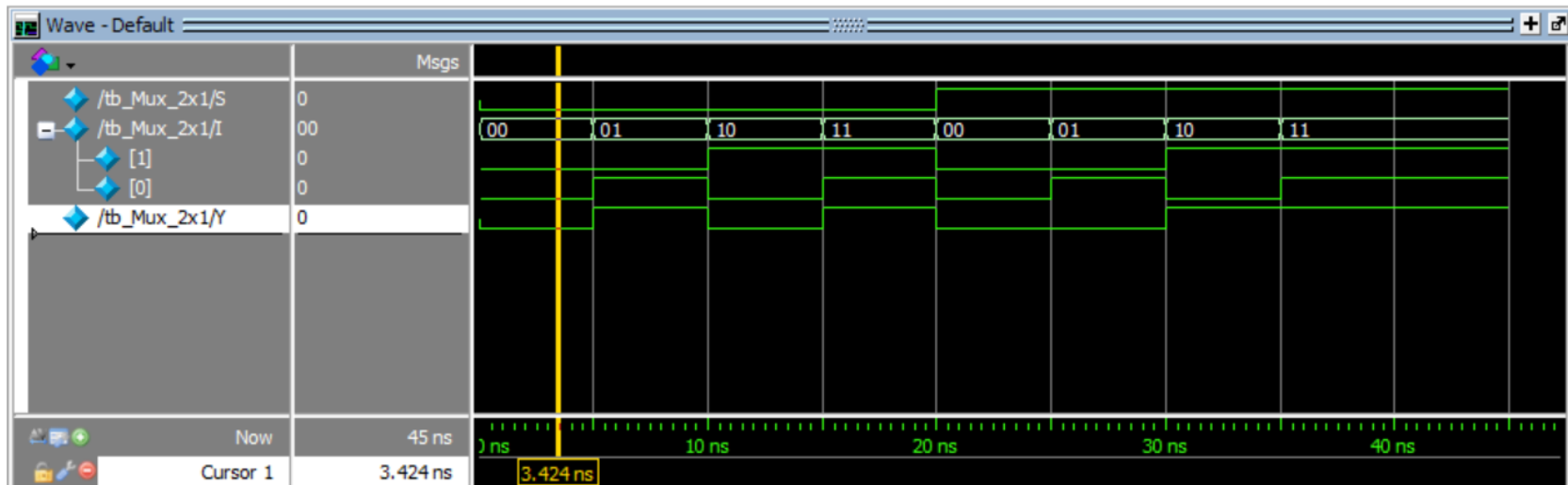
# Example 1: Simulation (2x1 Mux)

- Schematic Diagram with Truth Table:

S	Y
0	I <sub>0</sub>
1	I <sub>1</sub>



- Simulation Waveform:



# Simple Testbenches for Combinational Logic

- Response to the stimuli generated by the *initial* block will appear in **text format as standard output** and as **waveforms (timing diagrams)** in simulators with graphical output capability
- **Numerical outputs** are displayed using **system tasks**:
  - **\$display** – display a one-time value of variables or strings with an end-of-line return
  - **\$write** – same as **\$display** but without going to next line
  - **\$monitor** – display variables whenever a value changes during simulation run
  - **\$time** – display the simulation time
  - **\$finish** – terminate the simulation
  - **\$stop** – stops the simulation and goes to interactive mode



# Simple Testbenches for Combinational Logic

- Syntax for \$display, \$write, \$monitor:

**[task\_name]** *(format specification, argument list)*

- Examples:

- **\$display** ("%d %b %b", C, A, B);

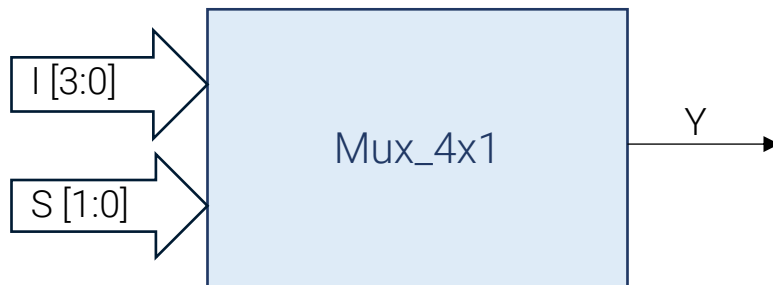
*// displays C in decimal, A and B in binary*

- **\$display** ("time = %0d A = %b B = %b", \$time, A, B);

*// displays text inputs, and system time in decimal without leading spaces*

# Example 2: Simulation (4x1 Mux)

- Entity Diagram:



- Truth Table:

$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

- Testbench File for *Mux\_4x1.v*:

```

// Testbench file of 4x1 mux
`timescale 1 ns / 1 ps
module tb_Mux_4x1 ();

    // declare inputs as reg types
    reg [3:0] I;
    reg [1:0] S;
    // outputs as wire types
    wire Y;

    // instantiate UUT
    Mux_4x1 UUT ( .I(I), .S(S), .Y(Y) );

    // generate stimuli
    initial
    begin
        // system task to print message with next line
        $display("Starting simulation at %0d ns...", $time);

        S = 2'b00;    I = 4'h0; #5
                     I = 4'h7; #5
                     I = 4'h8; #5
                     I = 4'hF; #5

        S = 2'b01;    I = 4'h1; #5
                     I = 4'h6; #5
                     I = 4'h9; #5
                     I = 4'hE; #5

        S = 2'b10;    I = 4'h2; #5
                     I = 4'h5; #5
                     I = 4'hA; #5
                     I = 4'hD; #5

        S = 2'b11;    I = 4'h3; #5
                     I = 4'h4; #5
                     I = 4'hB; #5
                     I = 4'hC; #5
    end

```

## Example 2: Simulation (4x1 Mux)

```
// system task to print message with next line
$display("Finished simulation at %0d ns.", $time);

// system task to stop simulation
$stop;
end

// response monitor
initial begin
    // system task to display values whenever a value changes
    $monitor("Time = %2d ns\t Select = %2b\t I = %h\t t = %4b\t Y = %b", $time, S, I, I, Y);
end

endmodule
```

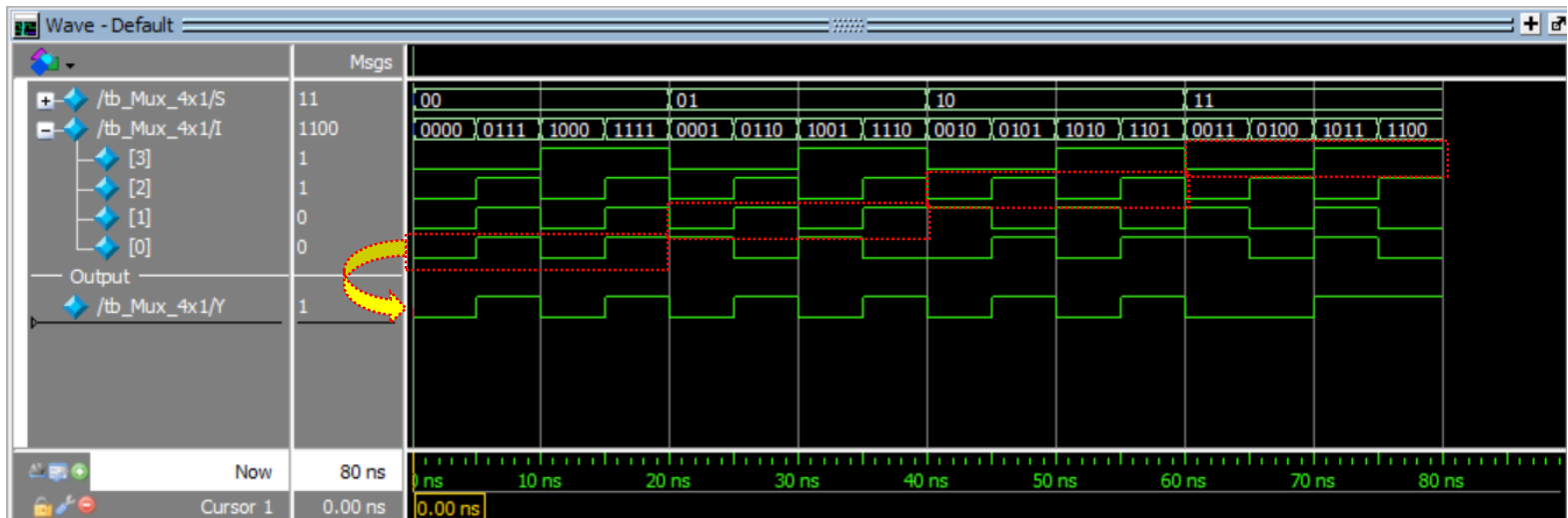
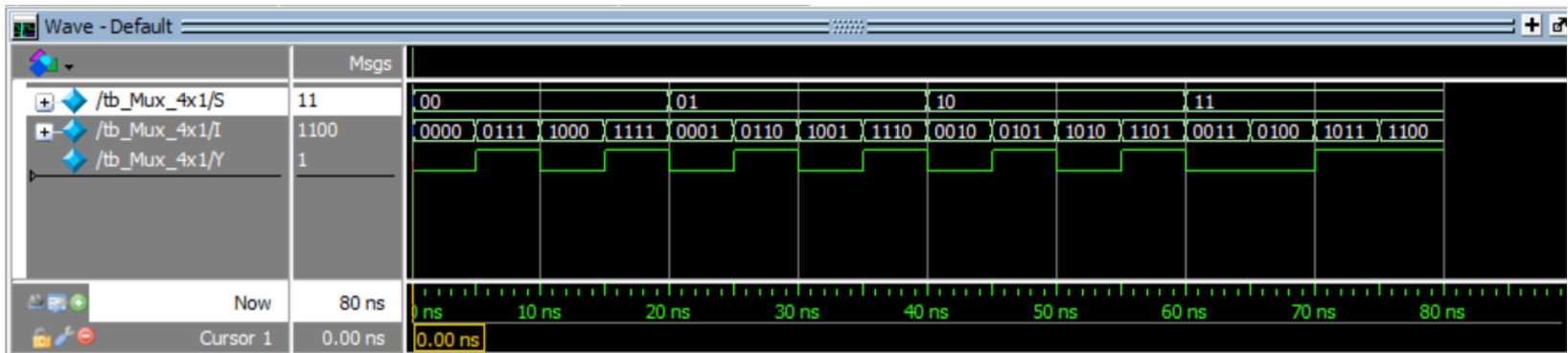
- Standard Output (*Text format*):

$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

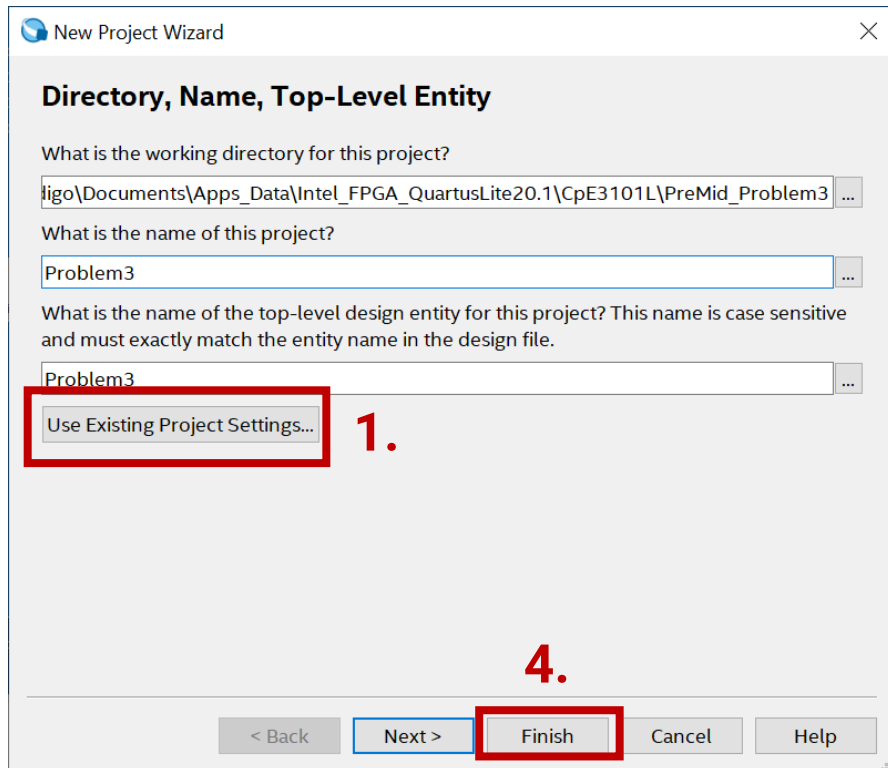
```
Transcript
# Starting simulation at 0 ns...
# Time = 0 ns   Select = 00   I = 0   = 0000   Y = 0
# Time = 5 ns   Select = 00   I = 7   = 0111   Y = 1
# Time = 10 ns  Select = 00   I = 8   = 1000   Y = 0
# Time = 15 ns  Select = 00   I = f   = 1111   Y = 1
# Time = 20 ns  Select = 01   I = 1   = 0001   Y = 0
# Time = 25 ns  Select = 01   I = 6   = 0110   Y = 1
# Time = 30 ns  Select = 01   I = 9   = 1001   Y = 0
# Time = 35 ns  Select = 01   I = e   = 1110   Y = 1
# Time = 40 ns  Select = 10   I = 2   = 0010   Y = 0
# Time = 45 ns  Select = 10   I = 5   = 0101   Y = 1
# Time = 50 ns  Select = 10   I = a   = 1010   Y = 0
# Time = 55 ns  Select = 10   I = d   = 1101   Y = 1
# Time = 60 ns  Select = 11   I = 3   = 0011   Y = 0
# Time = 65 ns  Select = 11   I = 4   = 0100   Y = 0
# Time = 70 ns  Select = 11   I = b   = 1011   Y = 1
# Time = 75 ns  Select = 11   I = c   = 1100   Y = 1
# Finished simulation at 80 ns.
```

# Example 2: Simulation (4x1 Mux)

- Simulation Waveform:



# Misc: A (Relatively) Faster Way in Creating a New Project in Quartus



**New Project Wizard**

**Directory, Name, Top-Level Entity**

What is the working directory for this project?  
C:\Users\ligo\Documents\Apps\_Data\Intel\_FPGA\_QuartusLite20.1\CpE3101L\PreMid\_Problem3 ...

What is the name of this project?  
Problem3 ...

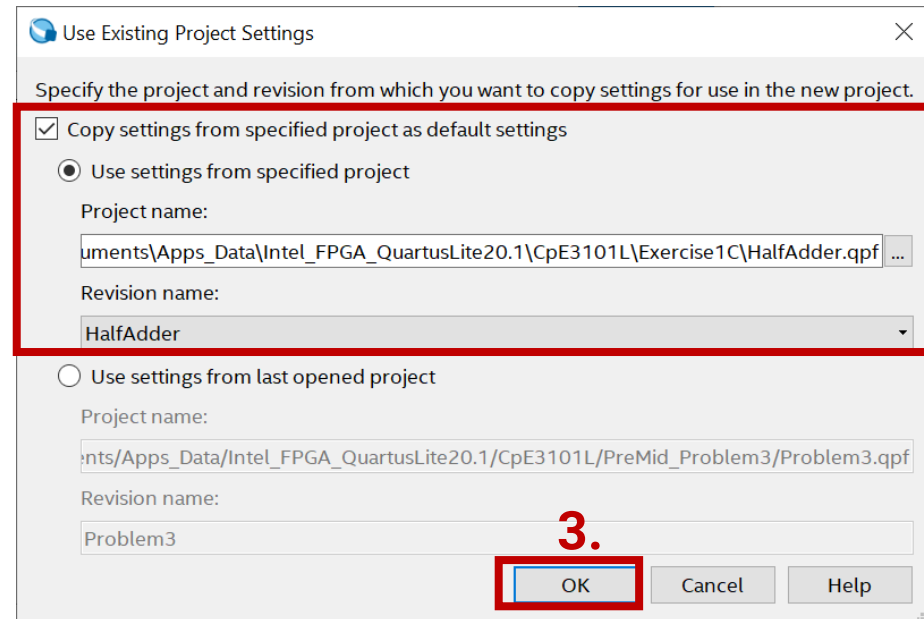
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.  
Problem3 ...

**1.** Use Existing Project Settings...

**4.** Finish

< Back Next > Cancel Help

**2. Choose a project from previous LEs (preferably from LE #2 to include ModelSim setup)**



**Use Existing Project Settings**

Specify the project and revision from which you want to copy settings for use in the new project.

☒ Copy settings from specified project as default settings

☒ Use settings from specified project

Project name:  
C:\Users\ligo\Documents\Apps\_Data\Intel\_FPGA\_QuartusLite20.1\CpE3101L\Exercise1C\HalfAdder.qpf ...

Revision name:  
HalfAdder

☐ Use settings from last opened project

Project name:  
C:\Users\ligo\Documents\Apps\_Data\Intel\_FPGA\_QuartusLite20.1\CpE3101L\PreMid\_Problem3\Problem3.qpf

Revision name:  
Problem3

**3.** OK

Cancel Help

# Misc: Adding Existing .v Files in a Quartus Project

- Create New Project

The image shows the 'New Project Wizard' and 'Use Existing Project Settings' dialog boxes in Quartus. Red boxes and numbers 1 through 5 highlight specific steps in the process:

- 1**: A red box highlights the 'What is the name of this project?' and 'What is the name of the top-level design entity for this project?' sections in the 'New Project Wizard'.
- 2**: A red box highlights the 'Copy settings from specified project as default settings' checkbox in the 'Use Existing Project Settings' dialog.
- 3**: A red box highlights the 'Use settings from last opened project' radio button in the 'Use Existing Project Settings' dialog.
- 4**: A red box highlights the 'OK' button at the bottom of the 'Use Existing Project Settings' dialog.
- 5**: A red box highlights the 'Finish' button at the bottom of the 'New Project Wizard'.

# Misc: Adding Existing .v Files in a Quartus Project

- Click **Project > Add/Remove Files in Project**

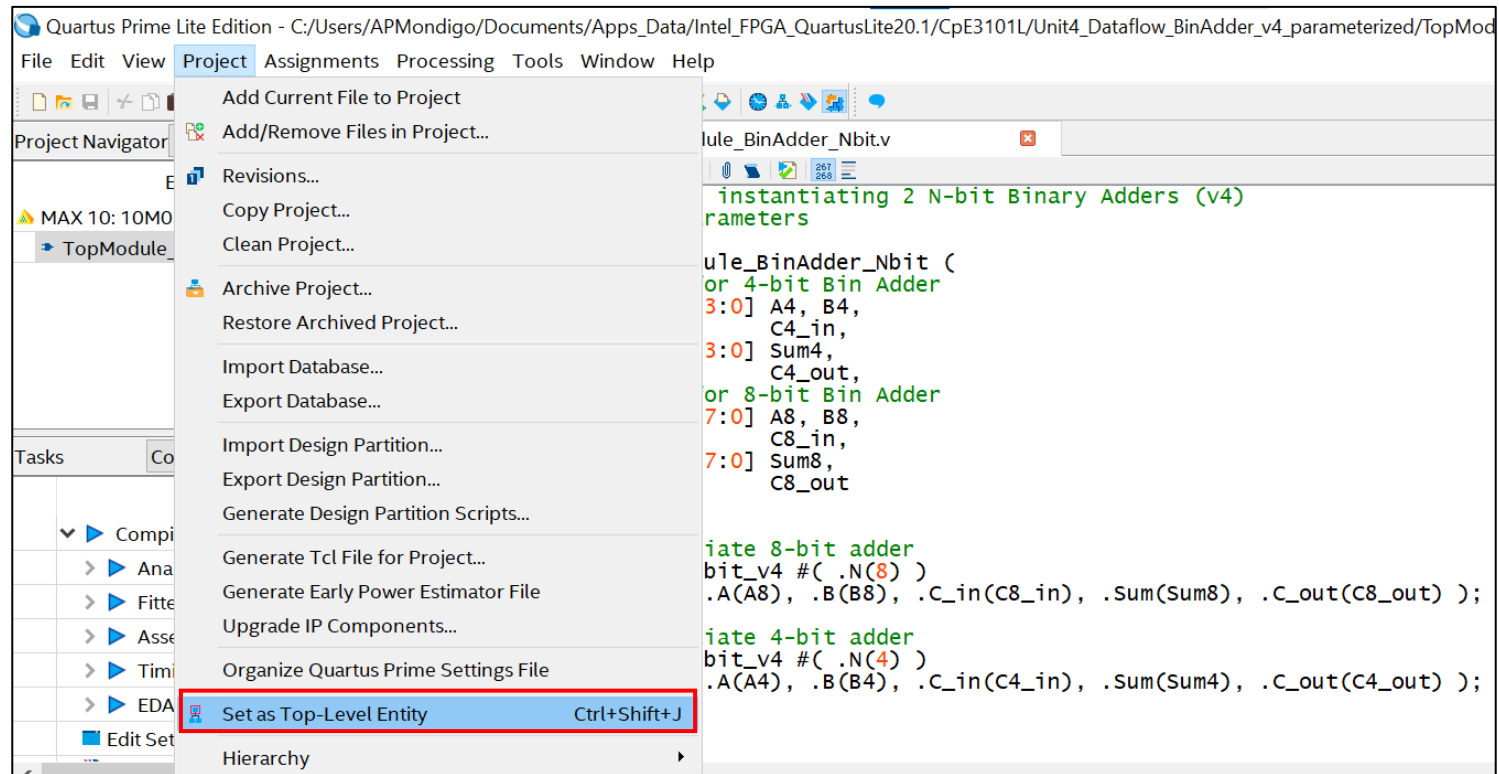
The screenshot illustrates the process of adding an existing Verilog file to a Quartus project. The steps are numbered 1 through 5:

- 1**: Click on **Project > Add/Remove Files in Project...** in the Quartus Project Manager menu.
- 2**: In the **Settings - TopModule\_BinAdder\_Nbit** dialog, select the **Files** category.
- 3**: Click the **...** button next to the **Add** button to open the file selection dialog.
- 4**: In the file selection dialog, navigate to the folder containing the file **BinAdder\_Nbit\_v4.v**.
- 5**: Click the **Open** button in the file selection dialog.

*Find sub-modules needed:  
"BinAdder\_Nbit\_v4.v"*

# Misc: Adding Existing .v Files in a Quartus Project

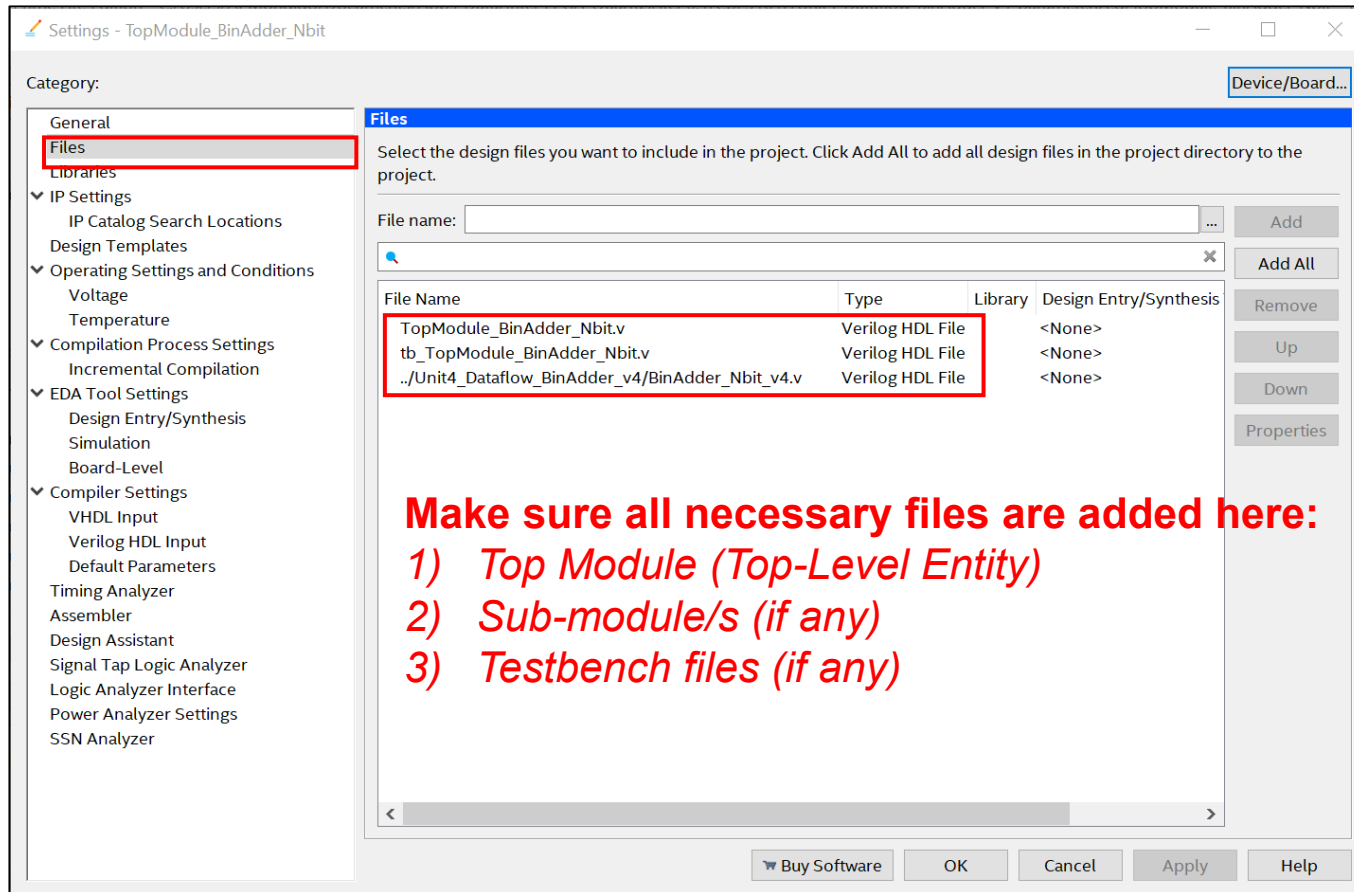
- “TopModule\_BinAdder\_Nbit.v” must be the Top Level Module
- Click **Project > Set as Top-Level Entity**





# Misc: Adding Existing .v Files in a Quartus Project

- Click **Project > Add/Remove Files in Project**





End of Unit 3