lem Stats

Problem	Success	Incorrect	Compile error	Simulation error	Total attempts	Success rate (%)
1 step_one	82733	5309	95121	3197	186772	44%
2 zero	69758	2166	47215	607	119846	58%
3 wire	74089	10736	65896	1853	152815	48%
4 wire4	68922	5523	30295	156	104944	66%
5 notgate	67445	3461	22047	325	93345	72%
6 andgate	68156	7128	14687	211	90217	76%
7 norgate	64460	42886	30568	104	138061	47%
8 xnorgate	64145	32880	20241	49	117353	55%
9 wire_decl	63633	16757	27716	219	108370	59%
10 7458	60005	12913	21729	70	94765	63%
11 vector0	59831	7263	25866	254	93285	64%
12 vector1	54970	7053	29918	163	92162	60%
13 vector2	52709	13700	43563	104	110117	48%
14 vectorgates	53135	68854	44942	204	167162	32%
15 gates4	51801	19816	21868	178	93707	55%
16 vector3	51100	27120	54138	112	132546	39%
17 vectorr	54190	14962	72913	183	142303	38%
18 vector4	50210	26426	95745	67	172534	29%
19 vector5	50727	21427	59687	124	132051	38%
20 module	55329	10738	80852	218	147255	38%
21 module_pos	43889	11563	60881	39	116411	38%
22 module_name	42329	10214	26939	15	79502	53%
23 module_shift	44507	8673	51984	94	105297	42%
24 module_shift8	48204	37524	104768	175	190721	25%
dule_add	46067	27577	80933	98	154729	30%

Problem Statement

We're going to start with a small bit of HDL to get familiar with the interface used by HDLBits. Here's the description of the circuit you need to build for this exercise:

Build a circuit with no inputs and one output. That output should always drive 1 (or logic high).

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module( output one );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
1  module top_module( output one );
2
3  // Insert your code here
4      assign one = 1;
5
6  endmodule
7
```

Submit Submit (new window)

step_one — Compile and simulate

Running Quartus synthesis. Show Quartus messages...
Running ModelSim simulation. Show Modelsim messages...

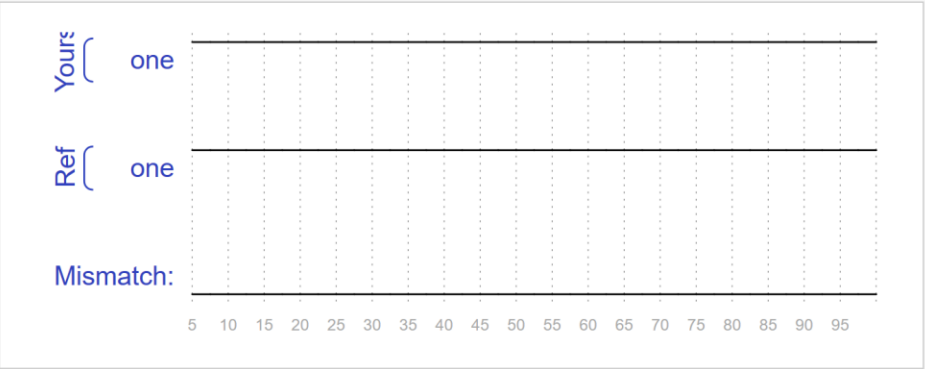
Status: Success!

You have solved 13 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

Output should be 1



Warning messages that may be important

Quartus messages (Show all)

Warning (13024): Output pins are stuck at VCC or GND

This warning says that an output pin never changes (is "stuck"). This can sometimes indicate a bug if the output pin shouldn't be a constant. If this pin is not supposed to be constant, check for bugs that cause the value being assigned to never change (e.g., `assign a = x & ~x;`)

- Getting Started
  - Getting Started
  - Output Zero
- Verilog Language
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

Zero

step_one Previous | Next wire

Build a circuit with no inputs and one output that outputs a constant `0`

Now that you've worked through the previous problem, let's see if you can do a simple problem without the hints.

HDLBits uses Verilog-2001 ANSI-style port declaration syntax because it's easier to read and reduces typos. You may use the older Verilog-1995 syntax if you wish. For example, the two module declarations below are acceptable and equivalent:

```
module top_module ( zero );
    output zero;
    // Verilog-1995
endmodule
```

```
module top_module ( output zero );

    // Verilog-2001
endmodule
```

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module(
    output zero
);
```

Hint...

Module Declaration

```
module top_module(
    output zero
);
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    output zero
);// Module body starts after semicolon

    assign  zero = 0;
endmodule

```

Submit | Submit (new window)

Upload a source file...

Solution

Show solution

zero — Compile and simulate

Running Quartus synthesis. Show Quartus messages...
Running ModelSim simulation. Show Modelsim messages...

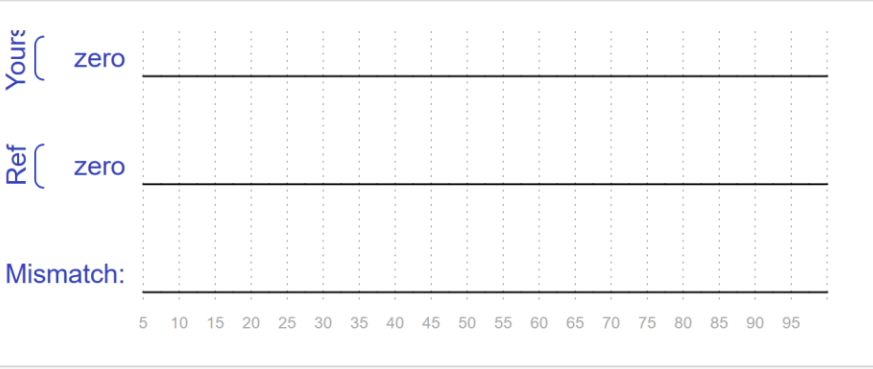
Status: Success!

You have solved 14 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

Output should 0



Warning messages that may be important

Quartus messages (Show all)

Warning (13024): Output pins are stuck at VCC or GND

This warning says that an output pin never changes (is "stuck"). This can sometimes indicate a bug if the output pin shouldn't be a constant. If this pin is not supposed to be constant, check for bugs that cause the value being assigned to never change (e.g., `assign a = x & ~x;`)

- Getting Started
- **Verilog Language**
  - **Basics**
    - **Simple wire**
    - Four wires
    - Inverter
    - AND gate
    - NOR gate
    - XNOR gate
    - Declaring wires
    - 7458 chip
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

Wire

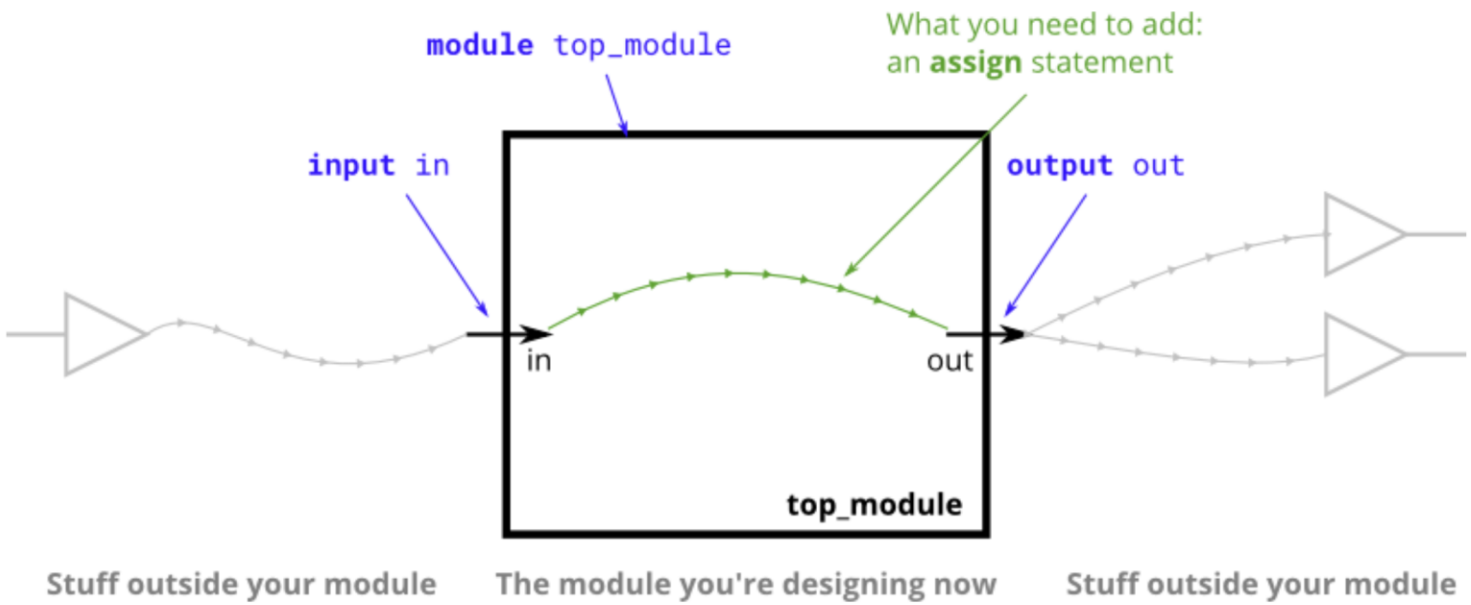
zero Previous Next wire4

Create a module with one input and one output that behaves like a wire.

Unlike physical wires, wires (and other signals) in Verilog are *directional*. This means information flows in only one direction, from (usually one) *source* to the *sinks* (The source is also often called a *driver* that *drives* a value onto a wire). In a Verilog "continuous assignment" (`assign left_side = right_side;`), the value of the signal on the right side is driven onto the wire on the left side. The assignment is "continuous" because the assignment continues all the time even if the right side's value changes. A continuous assignment is not a one-time event.

The ports on a module also have a direction (usually input or output). An input port is *driven by* something from outside the module, while an output port *drives* something outside. When viewed from inside the module, an input port is a driver or source, while an output port is a sink.

The diagram below illustrates how each part of the circuit corresponds to each bit of Verilog code. The module and port declarations create the black portions of the circuit. Your task is to create a wire (in green) by adding an `assign` statement to connect `in` to `out`. The parts outside the box are not your concern, but you should know that your circuit is tested by connecting signals from our test harness to the ports on your `top_module`.



In addition to continuous assignments, Verilog has three other assignment types that are used in procedural blocks, two of which are synthesizable. We won't be using them until we start using procedural blocks.

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module( input in, output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
1  module top_module( input in, output out );
2
3      assign out = in;
4  endmodule
5
```

Submit | Submit (new window)

Upload a source file...

wire — Compile and simulate

Running Quartus synthesis. Show Quartus messages...
Running ModelSim simulation. Show Modelsim messages...

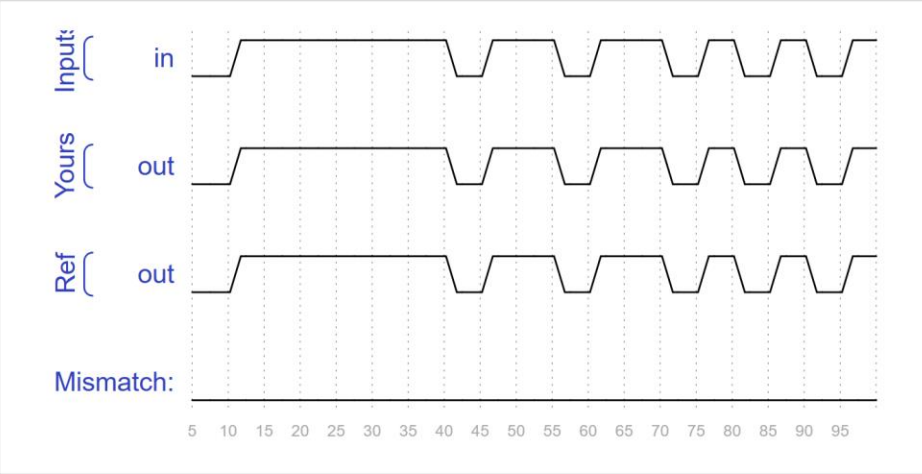
Status: Success!

You have solved 15 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

Output should follow input



- Basics
  - Simple wire
  - Four wires
  - Inverter
  - AND gate
  - NOR gate
  - XNOR gate
  - Declaring wires
  - 7458 chip
- Vectors
- Modules: Hierarchy
- Procedures
- More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

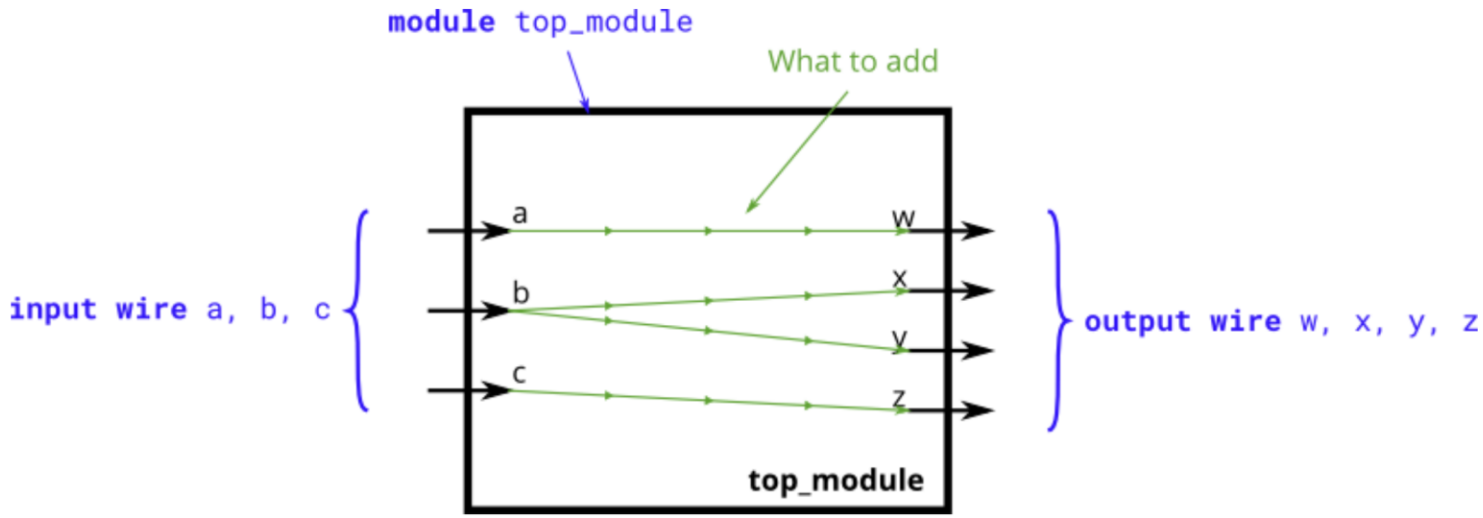
Create a module with 3 inputs and 4 outputs that behaves like wires that makes these connections:

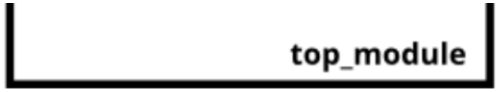
```
a -> w
b -> x
b -> y
c -> z
```

The diagram below illustrates how each part of the circuit corresponds to each bit of Verilog code. From outside the module, there are three input ports and four output ports.

When you have multiple `assign` statements, the **order** in which they appear in the code **does not matter**. Unlike a programming language, `assign` statements ("continuous assignments") describe *connections* between things, not the *action* of copying a value from one thing to another.

One potential source of confusion that should perhaps be clarified now: The green arrows here represent connections *between* wires, but are not wires in themselves. The module itself *already* has 7 wires declared (named a, b, c, w, x, y, and z). This is because `input` and `output` declarations actually declare a wire unless otherwise specified. Writing `input wire a` is the same as `input a`. Thus, the `assign` statements are not creating wires, they are creating the connections between the 7 wires that already exist.





Expected solution length: *Around 4 lines.*

Module Declaration

```
module top_module(
    input a,b,c,
    output w,x,y,z );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    input a,b,c,
    output w,x,y,z );
    assign w = a;
    assign x = b;
    assign y = b;
    assign z = c;
endmodule

```

Submit | Submit (new window)

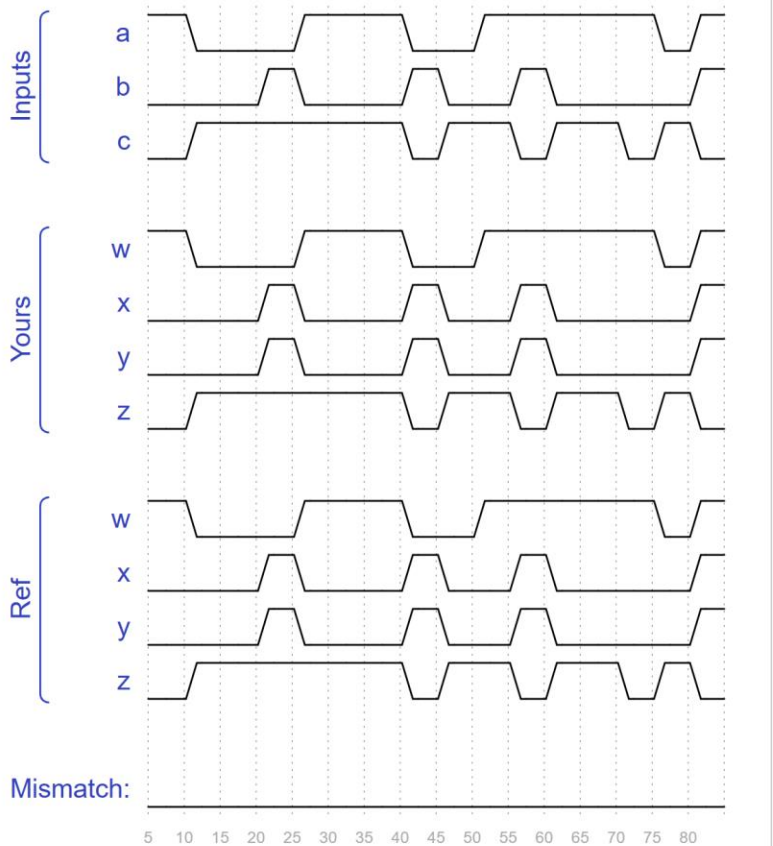
Upload a source file...

Status: Success!

You have solved 16 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).



- Getting Started
- **Verilog Language**
  - **Basics**
    - Simple wire
    - Four wires
    - **Inverter**
    - AND gate
    - NOR gate
    - XNOR gate
    - Declaring wires
    - 7458 chip
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

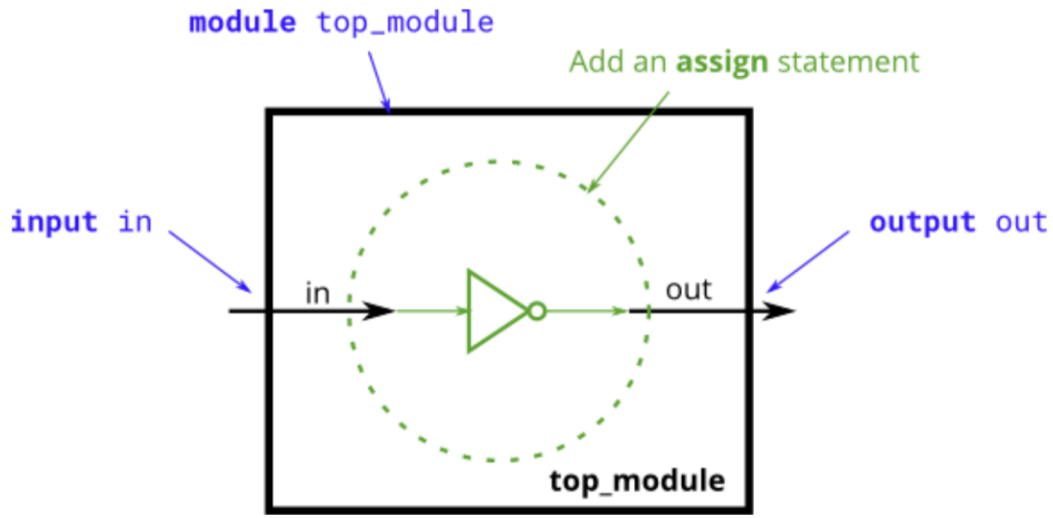
Notgate

wire4 Previous Next andgate

Create a module that implements a NOT gate.

This circuit is similar to wire, but with a slight difference. When making the connection from the wire `in` to the wire `out` we're going to implement an inverter (or "NOT-gate") instead of a plain wire.

Use an assign statement. The `assign` statement will *continuously drive* the inverse of `in` onto wire `out`.



Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module( input in, output out );
```

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module( input in, output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module( input in, output out );

    assign out = ~in;
endmodule

```

Submit | Submit (new window)

Upload a source file...

Solution

Show solution

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module( input in, output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module( input in, output out );

    not(out, in);
endmodule

```

Submit | Submit (new window)

Upload a source file...

Solution

Show solution

notgate — Compile and simulate

Running Quartus synthesis. Show Quartus messages...
Running ModelSim simulation. Show Modelsim messages...

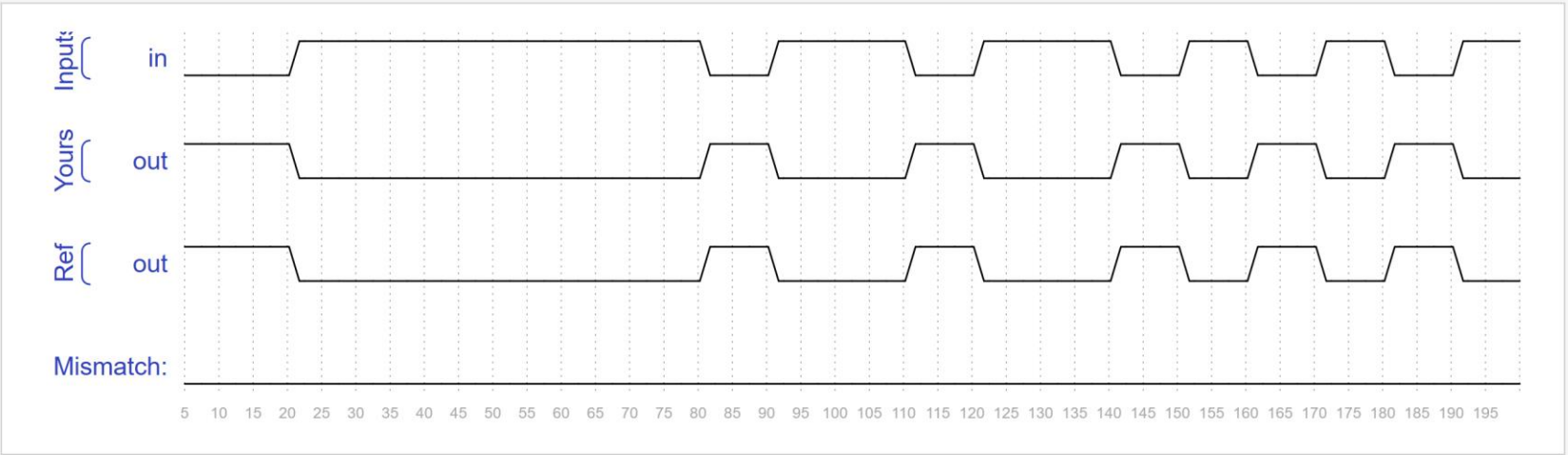
Status: Success!

You have solved 17 problems. See my progress...

Timing diagrams for selected test cases

These are timing diagrams from some of the test cases we used. They may help you debug your circuit. The diagrams show inputs to the circuit, outputs from your circuit, and the expected reference outputs. The "Mismatch" trace shows which cycles your outputs don't match the reference outputs (0 = correct, 1 = incorrect).

Inversion



- Getting Started
- **Verilog Language**
  - **Basics**
    - Simple wire
    - Four wires
    - Inverter
    - **AND gate**
    - NOR gate
    - XNOR gate
    - Declaring wires
    - 7458 chip
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

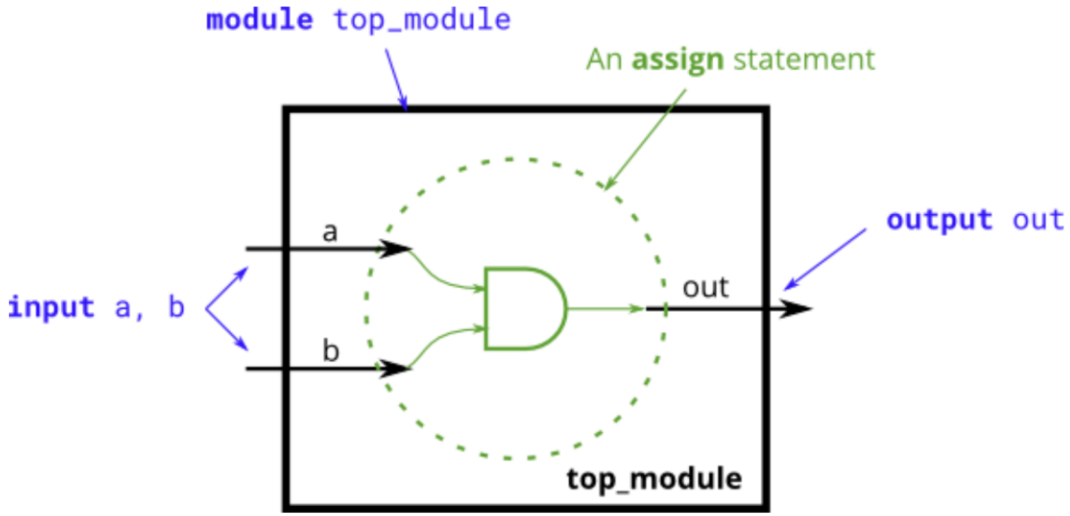
Andgate

notgate Previous Next norgate

Create a module that implements an AND gate.

This circuit now has three wires (a, b, and `out`). Wires a and b already have values driven onto them by the input ports. But wire `out` currently is not driven by anything. Write an `assign` statement that drives `out` with the AND of signals a and b.

Note that this circuit is very similar to the NOT gate, just with one more input. If it sounds different, it's because I've started describing signals as being *driven* (has a known value determined by something attached to it) or *not driven* by something. `Input wires` are driven by something outside the module. `assign` statements will drive a logic level onto a wire. As you might expect, a wire cannot have more than one driver (what is its logic level if there is?), and a wire that has no drivers will have an undefined value (often treated as 0 when synthesizing hardware).



Expected solution length: *Around 1 line.*

Module Declaration

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
1  module top_module(
2      input a,
3      input b,
4      output out );
5
6      and(out, a, b);
7  endmodule
8
```

Submit | Submit (new window)

Upload a source file...

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

Last success: 2022/12/27 21:23:55 Load

```
1  module top_module(
2      input a,
3      input b,
4      output out );
5
6      assign out = a & b;
7  endmodule
8
```

Submit Submit (new window)

Upload a source file...

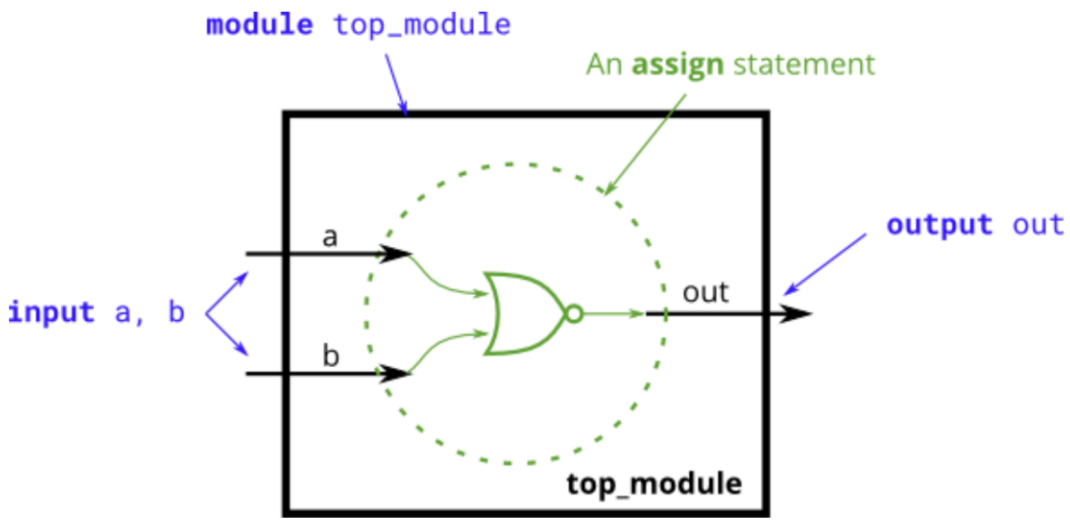
- Getting Started
- **Verilog Language**
  - **Basics**
    - Simple wire
    - Four wires
    - Inverter
    - AND gate
    - **NOR gate**
    - XNOR gate
    - Declaring wires
    - 7458 chip
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

Norgate

andgate Previous | Next xnorgate

Create a module that implements a NOR gate. A NOR gate is an OR gate with its output inverted. A NOR function needs two operators when written in Verilog.

An `assign` statement drives a wire (or "net", as it's more formally called) with a value. This value can be as complex a function as you want, as long as it's a *combinational* (i.e., memory-less, with no hidden state) function. An `assign` statement is a *continuous assignment* because the output is "recomputed" whenever any of its inputs change, forever, much like a simple logic gate.



Expected solution length: *Around 1 line.*

Module Declaration

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    input a,
    input b,
    output out );

    nor(out, a, b);
endmodule

```

Submit | Submit (new window)

Upload a source file...

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    input a,
    input b,
    output out );

    wire or_ab;

    or(or_ab, a, b);
    not(out, or_ab);

endmodule

```

Submit Submit (new window)

Upload a source file...

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    input a,
    input b,
    output out );

    assign out = ~(a|b);
endmodule

```

Submit Submit (new window)

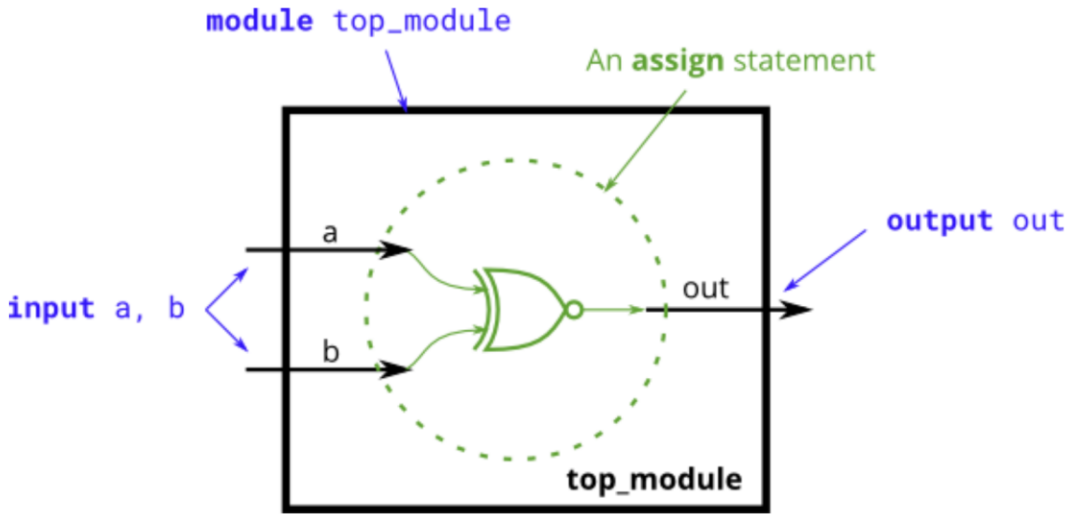
Upload a source file...

- Getting Started
- **Verilog Language**
  - **Basics**
    - Simple wire
    - Four wires
    - Inverter
    - AND gate
    - NOR gate
    - **XNOR gate**
    - Declaring wires
    - 7458 chip
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

Xnorgate

norgate Previous Next wire_decl

Create a module that implements an XNOR gate.



Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    input a,
    input b,
    output out );

    wire xor_ab;
    xor(xor_ab, a, b);
    not(out, xor_ab);
endmodule

```

Submit | Submit (new window)

Upload a source file...

Expected solution length: *Around 1 line.*

Module Declaration

```
module top_module(
    input a,
    input b,
    output out );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module(
    input a,
    input b,
    output out );

    assign out = ~(a ^ b);
endmodule

```

Submit Submit (new window)

Upload a source file...

- Getting Started
- **Verilog Language**
  - **Basics**
    - Simple wire
    - Four wires
    - Inverter
    - AND gate
    - NOR gate
    - XNOR gate
    - **Declaring wires**
    - 7458 chip
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

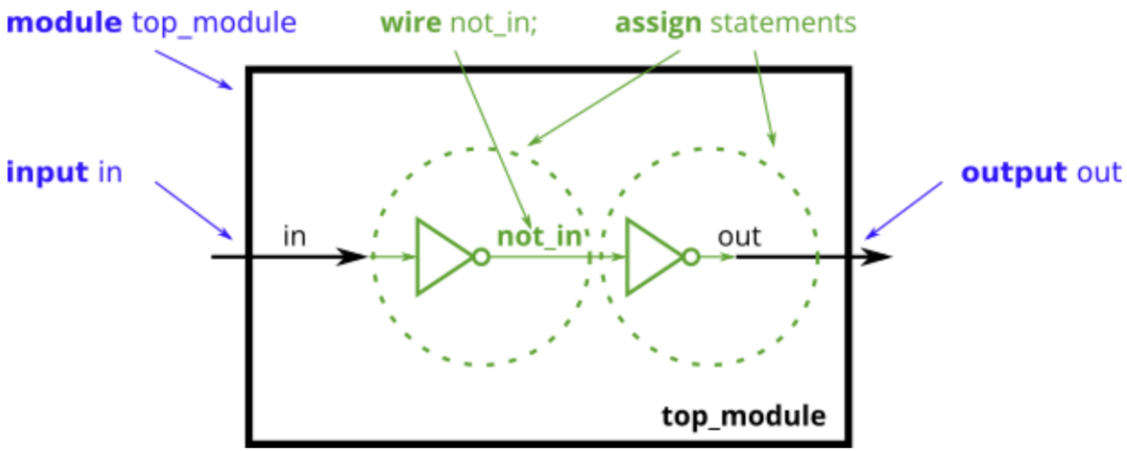
Wire decl

xnorgate Previous | Next 7458

Declaring wires

The circuits so far have been simple enough that the outputs are simple functions of the inputs. As circuits become more complex, you will need wires to connect internal components together. When you need to use a wire, you should declare it in the body of the module, somewhere before it is first used. (In the future, you will encounter more types of signals and variables that are also declared the same way, but for now, we'll start with a signal of type `wire`).

Example

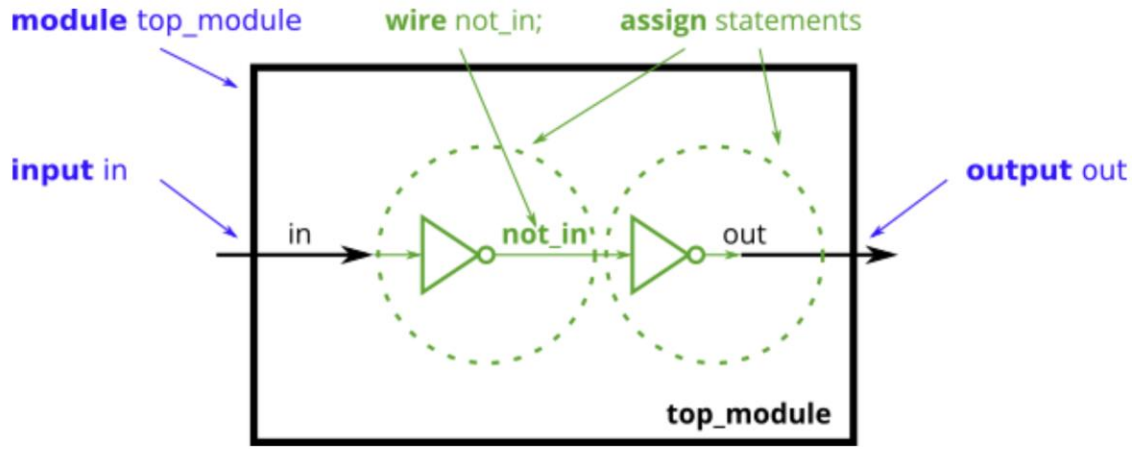



```
module top_module (
    input in,              // Declare an input wire named "in"
```

- vectors
- Modules: Hierarchy
- Procedures
- More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

signal of type `wire`).

Example




```
module top_module (
    input in,               // Declare an input wire named "in"
    output out              // Declare an output wire named "out"
);

    wire not_in;            // Declare a wire named "not_in"

    assign out = ~not_in;   // Assign a value to out (create a NOT gate).
    assign not_in = ~in;    // Assign a value to not_in (create another NOT gate).

endmodule    // End of module "top_module"
```

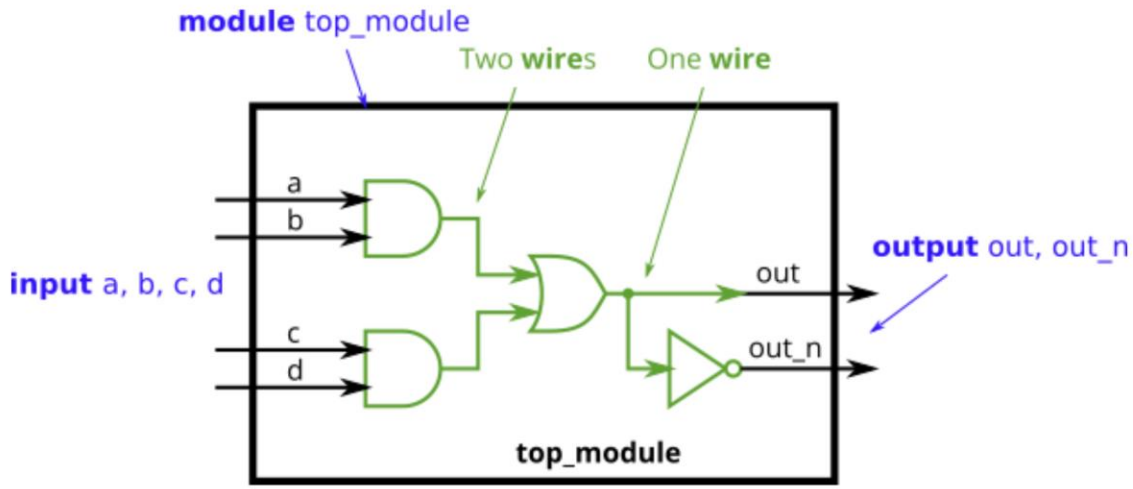
In the above module, there are three wires (`in`, `out`, and `not_in`), two of which are already declared as part of the module's input and output ports (This is why you didn't need to declare any wires in the earlier exercises). The wire `not_in` needs to be declared inside the module. It is not visible from outside the module. Then, two NOT gates are created using two `assign` statements. Note that it doesn't matter which of the NOT gates you create first: You still end up with the same circuit.

Practice

Implement the following circuit. Create two intermediate wires (named anything you want) to connect the AND and OR gates together. Note that the wire that feeds the NOT gate is really wire `out`, so you do not necessarily need to declare a third wire here. Notice how wires are driven by exactly one source (output of a gate), but can feed multiple inputs.

If you're following the circuit structure in the diagram, you should end up with four assign statements, as there are four signals that need a value assigned.

(Yes, it is possible to create a circuit with the same functionality without the intermediate wires.)



Expected solution length: *Around 5 lines.*

Module Declaration

```
`default_nettype none
module top_module(
```

```
    input d,
    output out,
    output out_n   );
```

Write your solution here

[Load a previous submission] Load

```
`default_nettype none
module top_module(
    input a,
    input b,
    input c,
    input d,
    output out,
    output out_n   );

    wire ab, cd, abcd;
    assign ab = a & b;
    assign cd = c & d;
    assign out = ab | cd;
    assign out_n = ~out;

endmodule

```

Submit | Submit (new window)

Upload a source file...

Solution

Show solution

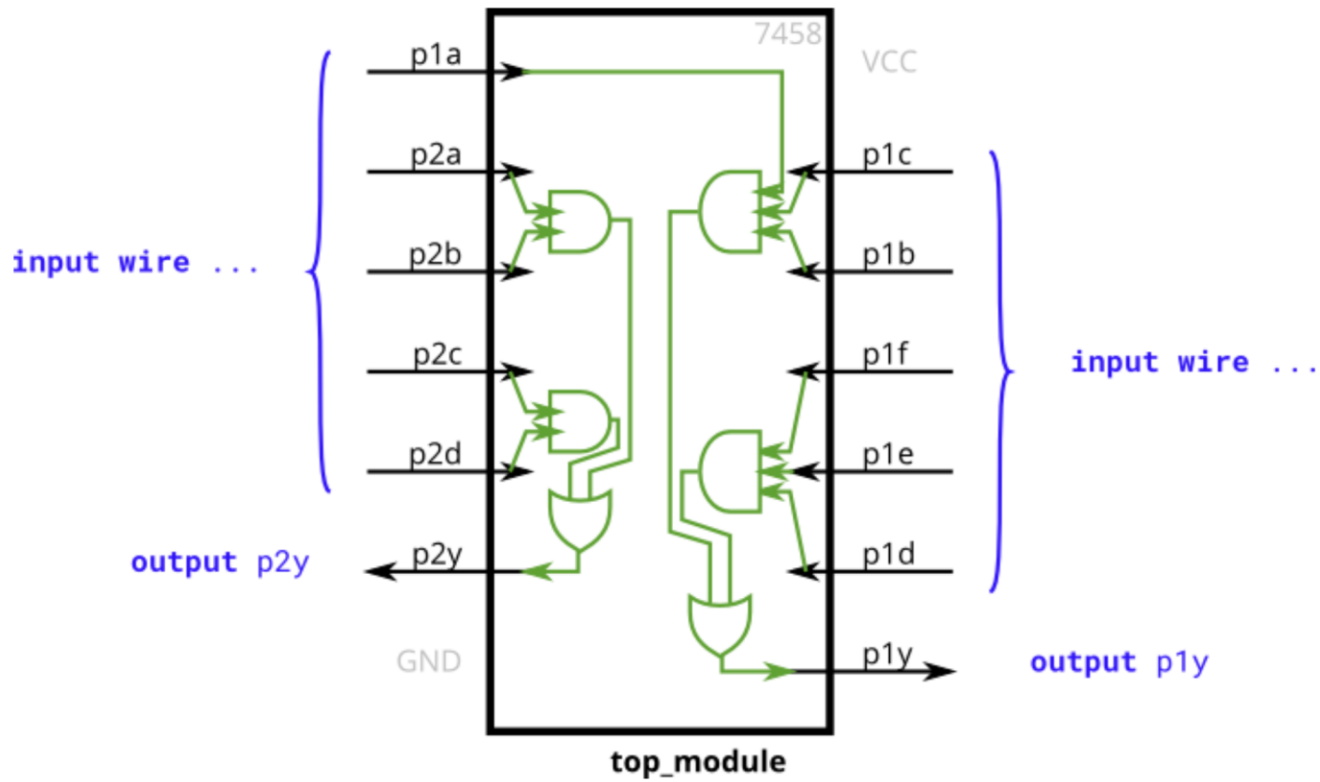
- Getting Started
- **Verilog Language**
  - **Basics**
    - Simple wire
    - Four wires
    - Inverter
    - AND gate
    - NOR gate
    - XNOR gate
    - Declaring wires
    - **7458 chip**
  - Vectors
  - Modules: Hierarchy
  - Procedures
  - More Verilog Features
- Circuits
- Verification: Reading Simulations
- Verification: Writing Testbenches
- CS450

7458

wire_decl Previous Next vector0

The 7458 is a chip with four AND gates and two OR gates. This problem is slightly more complex than 7420.

Create a module with the same functionality as the 7458 chip. It has 10 inputs and 2 outputs. You may choose to use an `assign` statement to drive each of the output wires, or you may choose to declare (four) wires for use as intermediate signals, where each internal wire is driven by the output of one of the AND gates. For extra practice, try it both ways.



Expected solution length: *Around 2–10 lines.*

```
    output p2y );
```

Hint...

Write your solution here

[Load a previous submission] Load

```
module top_module (
    input p1a, p1b, p1c, p1d, p1e, p1f,
    output p1y,
    input p2a, p2b, p2c, p2d,
    output p2y );

    wire ab, cd, abc, def;
    assign ab = p2a & p2b;
    assign cd = p2c & p2d;
    assign p2y = ab | cd;

    assign abc = p1a & p1b & p1c;
    assign def = p1d & p1e & p1f;
    assign p1y = abc | def;
endmodule

```

Submit | Submit (new window)

Upload a source file...

2.1 组合逻辑与布尔代数

组合逻辑电路的输出可以表示为瞬时输入变量的布尔函数形式，也就是说，在图 2.1 中任意时刻 t 输出 y_1、y_2 与 y_3 仅仅取决于该时刻 t 的输入值 a、b、c 与 d，组合逻辑电路在任意时刻 t 的输出仅为该时刻输入变量的函数。当电路的输出与时刻 t 之前的历史输入有关时，则称这一类电路为时序逻辑电路。时序逻辑电路在硬件上需要采用存储单元来实现。

逻辑电路中的变量为二进制变量——其值为 0 或 1，逻辑电路的硬件实现可以采用正逻辑，即用高电平（如 5 V）表示逻辑 1，用低电平（如 0 V）表示逻辑 0；也可以用负逻辑，用低电平表示逻辑 1，用高电平表示逻辑 0。

图 2.1　4 输入 3 输出的组合逻辑方框图

一些常见的逻辑门如图 2.2 所示，图中还给出了各逻辑门相应的布尔方程，可用于确定门电路在给定输入时的输出函数值。表 2.1 列出了基于硬件的布尔逻辑运算的常用符号。①。

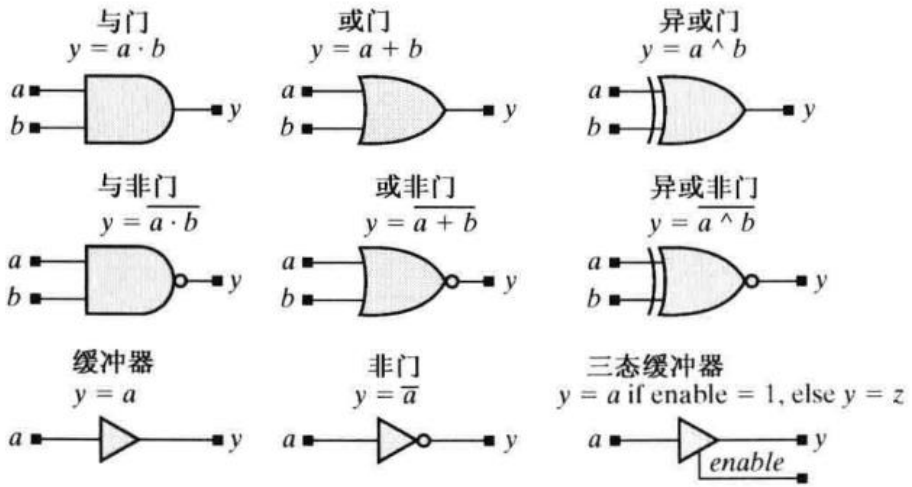


图 2.2　常用逻辑门电路的原理图符号与布尔关系式

表 2.1　常用布尔逻辑符号与运算符

符　号	逻辑操作
+	逻辑“或”
·	逻辑“与”
⊕	异或
∧	异或
′	逻辑非
—	逻辑非（上画线）

输入		输出	
a	*b*	*c_out*	*sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a)

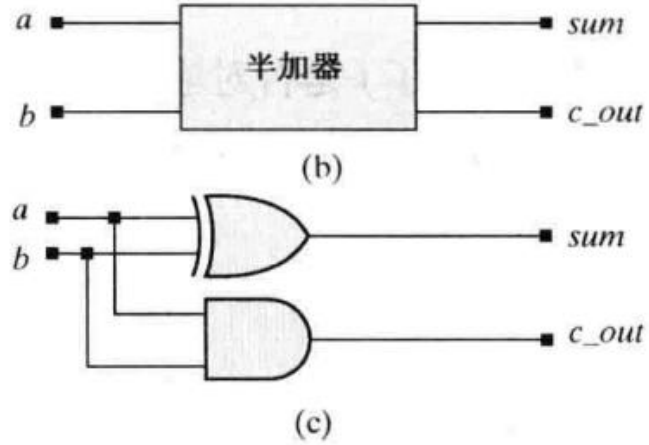


图 2.13　半加器：(a)真值表；(b)方框图符号；(c)原理图

例 2.2　全加器有两个数据输入位、一个进位输入位、一个和输出位及一个进位输出位。全加器组合逻辑的真值表如图 2.14 所示。

输入			输出	
a	*b*	*c_in*	*c_out*	*sum*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a)

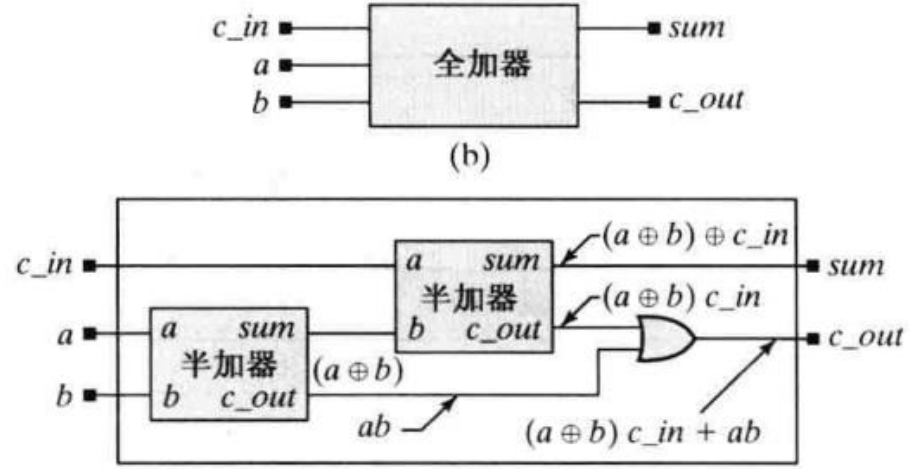


图 2.14　全加器：(a)真值表；(b)方框图符号；(c)由半加器和附加逻辑组成的全加器原理图

2.6.2 多路复用器

多路复用器(multiplexer)电路常用于控制数据通过计算机和其他数字系统的功能单元。例如，它可以控制某个特定寄存器的内容传输到算术逻辑单元(ALU)的输入端，并且把 ALU 的输出数据传输到该寄存器或另一个寄存器。双通道多路复用器的门级原理图如图 2.52 所示。当 $sel=0$ 时，输入端 a 的数据经过该电路(有一定的传播时延)到达 y_out；同样，当 $sel=1$ 时，输入端 b 的数据会达到 y_out，描述该电路功能的布尔表达式可表示为：$y\_out = sel' \cdot a + sel \cdot b$。

通常，多路复用器有 n 条数据输入通道和一条数据输出通道，m 位的地址线决定了哪个输入通道与输出通道相连。图 2.53 中用符号表示的多路复用器所选择的输入通道由下式确定：$Data\_Out = Data\_In[Address[k]]$，其中 k 为地址的标号。

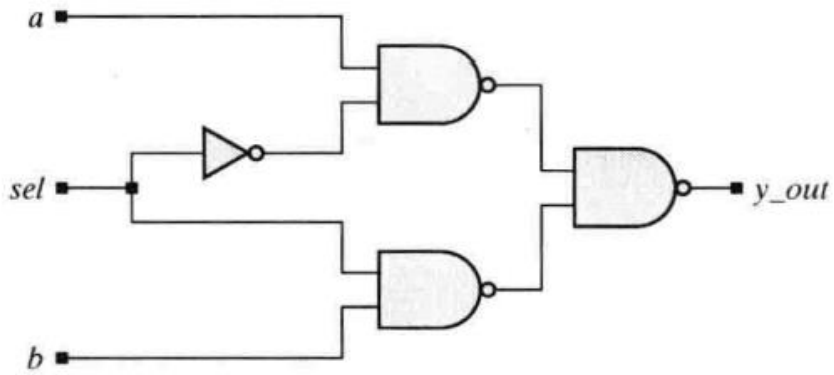


图 2.52 双通道多路复用器的门级原理图

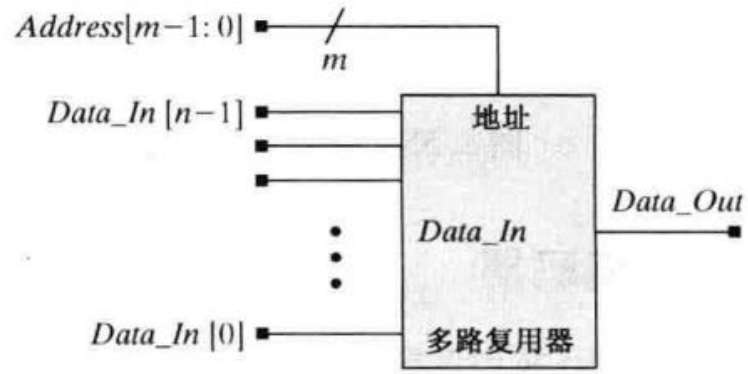


图 2.53 具有 m 位通道选择地址线的 n 个通道多路复用器原理符号

多路复用器也能够用于实现组合逻辑，可以将布尔函数的值赋给多路复用器的输入端，并用选择线进行译码。因为采用多路复用器必须将所有输入位的真值表进行完全译码，所以这种实现方式的效率不高。

2.6.6 译码器

二进制译码器对输入位组合进行译码，形成仅有 1 位为真的唯一输出码字。译码器通常用于从数字计算机的指令中提取操作码。行列地址译码器可根据地址码确定存储器中的码字位置。图 2.59 给出了译码器的方框图符号表示，二进制译码器有 m 个输入，n 个输出，且 $n=2^m$。在输入码字和输出码字之间存在许多种不同的可能映射，因此可以用输入-输出映射的组合逻辑来构建译码器(时序编码器和译码器广泛应用于通信和视频传输电路中)。

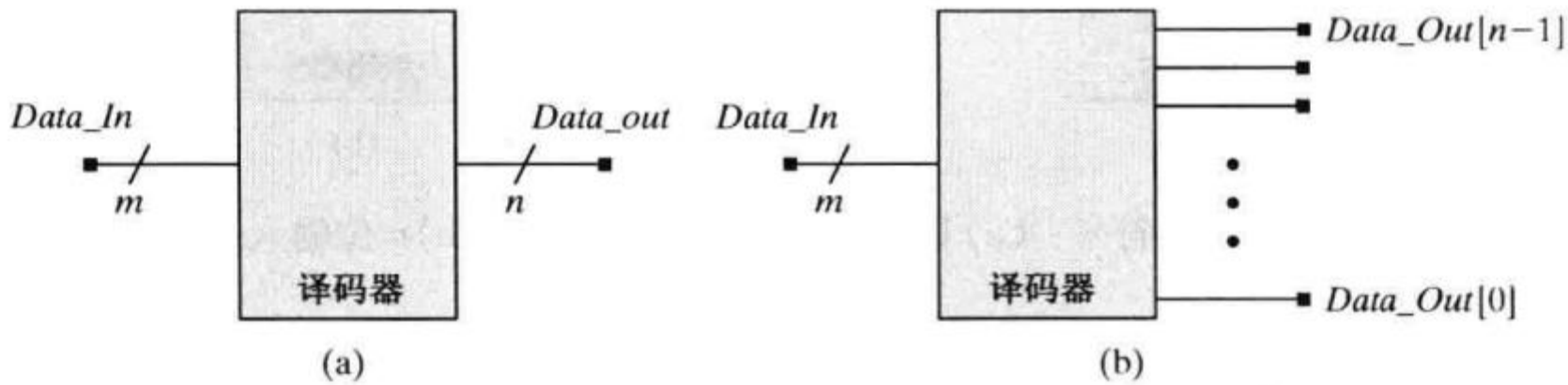


图 2.59　译码器方框图符号：(a)输入/输出总线标识的译码器；(b)输出分列标识的译码器

3.2.1 D 触发器

D 触发器是一种简单的触发器，在每个时钟的有效沿存储 D 输入端的当前值，这个值与之前已存储数据值无关。D 触发器的方框图及其真值表分别如图 3.4(a)和图 3.4(b)所示。真值表中包括触发器当前状态(Q)和时钟信号 clk 下一个有效沿处对应数据输入 D 的输出状态(Q_{next})。图 3.4(c)所示的波形说明了 D 数据的当前值在 clk 的上升沿(本例中上升沿有效)是如何存储的，以及在 clk 的两个有效沿之间 D 数据的变化是如何被忽略的。然而，D 信号必须在时钟有效沿之前的一段时间内保持稳定，否则器件将不能正常操作。描述 D 触发器的布尔逻辑表达式也称为特征方程[2]：$Q_{next} = D$。D 触发器也可以有其他(电平敏感)输入信号，如置位和复位信号，优先于同步操作并对输出进行初始化。

---

① 这里只考虑触发器的基本模式，即任一时刻只有一个输入发生变化。

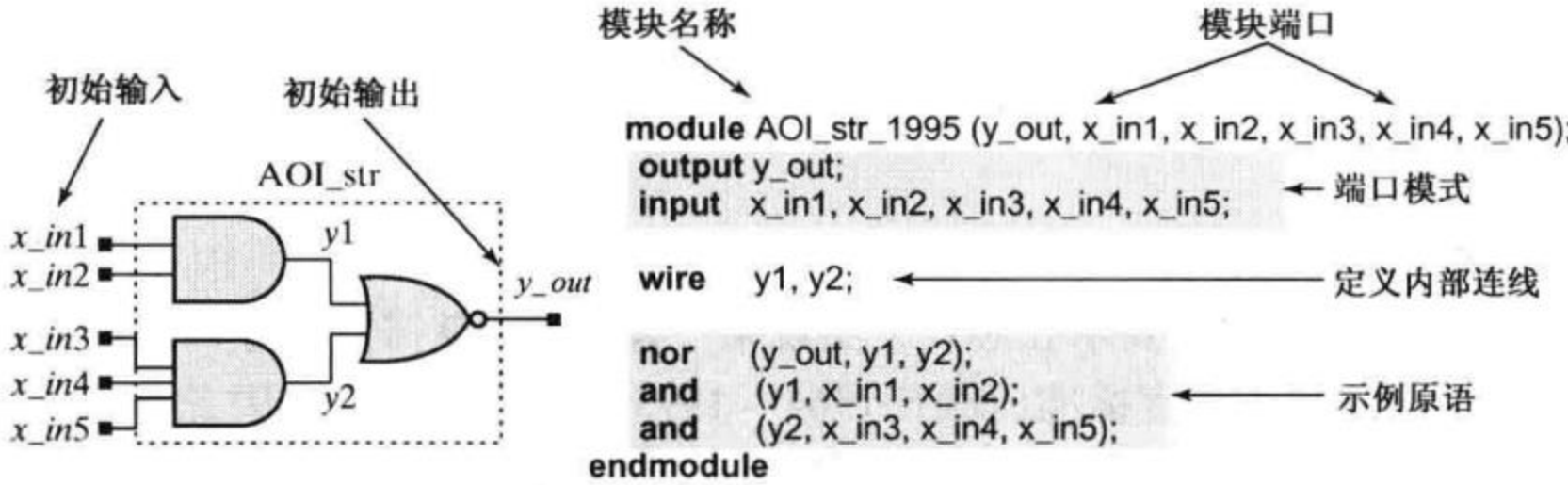


图 4.5　AOI 电路及其 Verilog 模型：(a) IEEE1346-1995 语法；(b) IEEE1364-2001，2005 语法

1.2　模块端口

例4.1 如图4.6(a)所示的二进制全加器电路由两个半加器和一个或门组合而成。Verilog层次化模型的划分设计中[参见图4.6(b)]包含两个半加器模块Add_half。

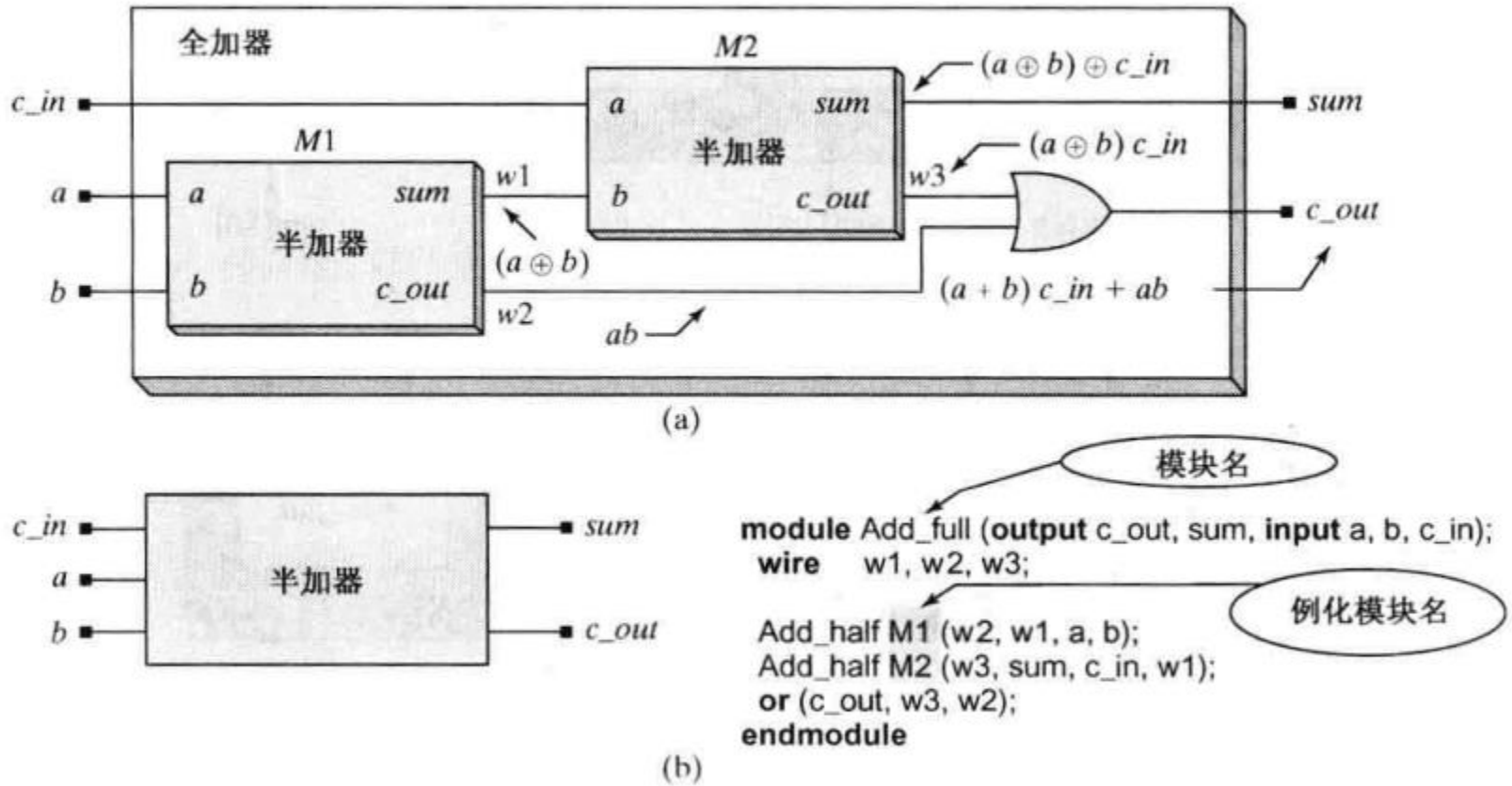
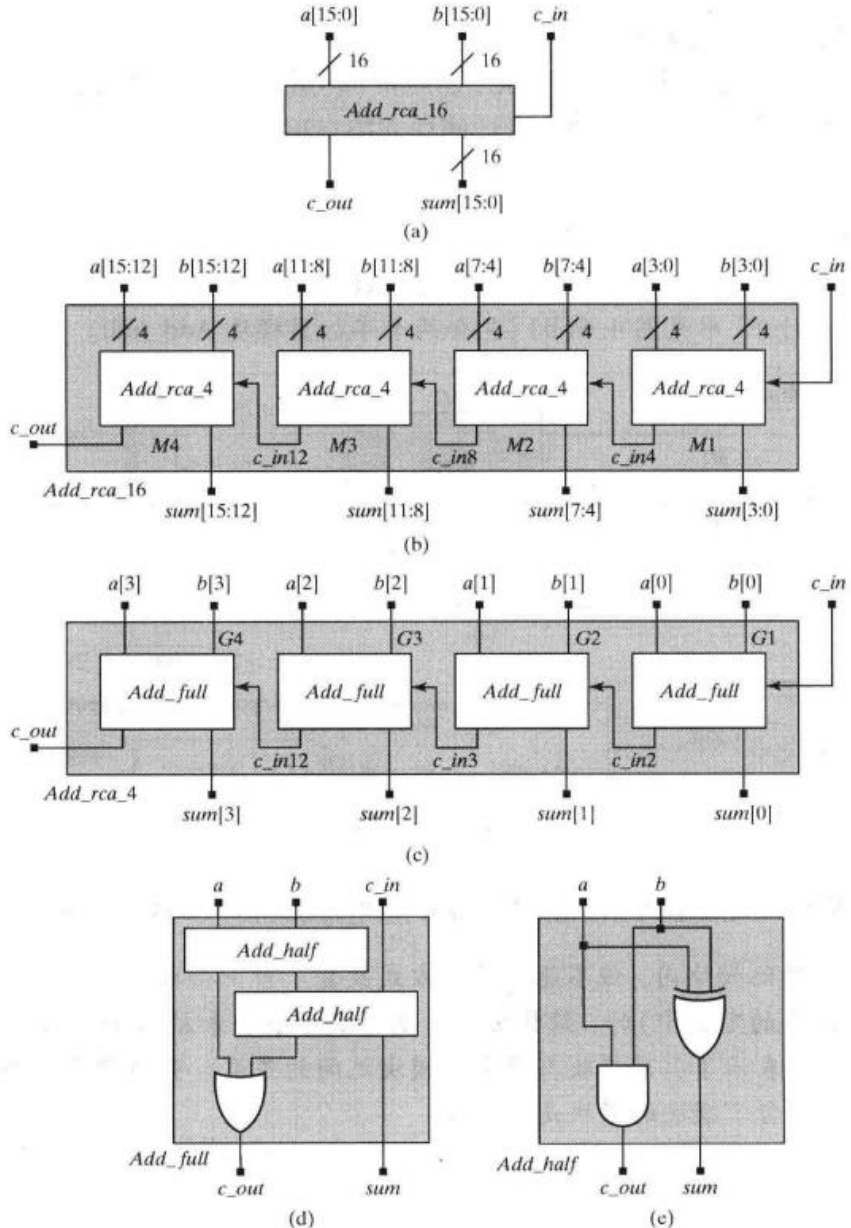


图4.6 全加器的层次化分解：(a)门级电路原理图；(b)Verilog模型

例4.2 一个16位行波进位(ripple-carry)加法器可由4个4位行波进位加法器级联而成，每个单元所产生的进位从最低位开始逐次传递至下一级的进位输入端。每个4位加法器都可视为全加器的级联。图4.7说明了一个零延时16位行波进位加法器 *Add_rca_16_0_delay* 的层次划分和端口信号连接关系。



Add_rca_16 的完整描述如下：

```
module Add_rca_16 (output c_out, output [15: 0] sum, input [15: 0], a, b, input c_in);
  wire          c_in4, c_in8, c_in12, c_out;
  Add_rca_4 M1  (c_in4,    sum[3: 0],   a[3:0],    b[3:0],    c_in);
  Add_rca_4 M2  (c_in8,    sum[7:4],    a[7:4],    b[7:4],    c_in4);
  Add_rca_4 M3  (c_in12,   sum[11:8],   a[11:8],   b[11:8],   c_in8);
  Add_rca_4 M4  (c_outsum, sum[15:12],  a[15:12],  b[15:12],  c_in12);
                c_out, a, b,
                c_in);
endmodule

module Add_rca_4 (output c_out, output [3: 0] sum, input [3 0] a, b, input c_in);
  wire          c_in2, c_in3, c_in4;
  Add_full      M1       (c_in2,    sum[0],    a[0], b[0], c_in);
  Add_full      M2       (c_in3,    sum[1],    a[1], b[1], c_in2);
  Add_full      M3       (c_in4,    sum[2],    a[2], b[2], c_in3);
  Add_full      M4       (c_out,    sum[3],    a[3], b[3], c_in4);
endmodule

module Add_full (output c_out, sum, input a, b, c_in)
  wire          w1, w2, w3;
  Add_half      M1 (w2, w1, a, b);
  Add_half      M2 (w3, sum, c_in, w1);
  or            M3 (c_out, w2, w3);
endmodule

module Add_half (output c_out, sum, input a, b);
  xor           M1 (sum, a, b);
  and           M2 (c_out, a, b);
endmodule
```