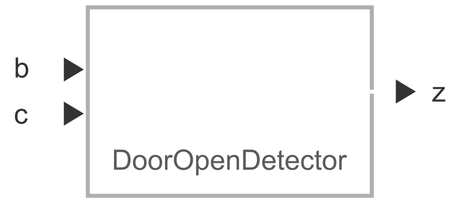
3.25 Combinational logic (Verilog)

In Verilog, a ***module*** defines a new component. A module definition starts with the module name, and a list of the module's named input or output of a module. Each module input and output must also appear in an input or output declaration.

PARTICIPATION ACTIVITY 3.25.1: DoorOpenDetector module definition.

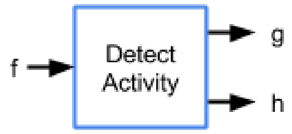
Start 2x speed

```
module DoorOpenDetector(b, c,z) ;
   input b, c;
   output reg z;

   // Module description goes here

endmodule
```

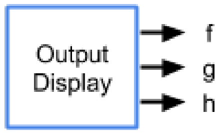


PARTICIPATION ACTIVITY 3.25.2: Module definitions.

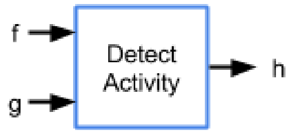
Match the module definition to the appropriate circuit.



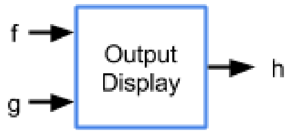
(A)



(B)



(C)



(D)

(B) (A) (D) (C)

```
module DetectActivity(f, g, h);
   input f;
   output reg g, h;
   ...
endmodule
```

```
module DetectActivity(f, g, h);
   input f, g;
   output reg h;
   ...
endmodule
```

```
module OutputDisplay(f, g, h);
   output reg f, g, h;
   ...
endmodule
```

```
module OutputDisplay(f, g, h);
   input f, g;
   output reg h;
   ...
endmodule
```

Reset

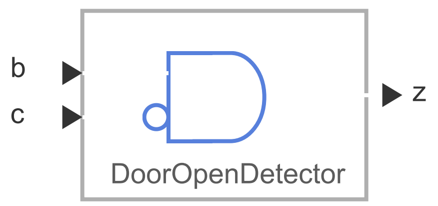
A designer can describe a combinational circuit's function using an always procedure. An ***always procedure*** defines a state
that executes throughout simulation. An always procedure's ***sensitivity list*** defines the inputs and variables whose value ch
procedure to execute. A combinational circuit is sensitive to all of the circuit's inputs.

begin and ***end*** surround an always procedure's statements.

PARTICIPATION ACTIVITY 3.25.3: Always procedure describing a combinational circuit.

Start 2x speed

```
module DoorOpenDetector(b, c, z);
    input b, c;
    output reg z;

    always @(b, c) begin
        z = b & ~c;
    end

endmodule
```



PARTICIPATION ACTIVITY 3.25.4: Always procedures for combinational circuits.

Complete the always procedure for the provided combinational equations.

1) z = ab

```
module GetSensorReading(a, b,
z);
   input a, b;
   output reg z;

   [          ] @(a, b) begin
      z = a & b;
   end
endmodule
```

Check Show answer

2) m = j'
   n = jk

```
module UpdateDisplay(j, k, m,
n);
   input j, k;
   output reg m, n;

   always @([          ]) begin
      m = ~j;
      n = j & k;
   end
endmodule
```

Check Show answer

3) t = v's'm'

```
module SetTimer(s, t, m, v);
   input v, s, m;
   output reg t;

   always @(s,          ) begin
      t = ~s & ~v & ~m;
   end
endmodule
```

Check Show answer

4) m = f
q = f'
p = f'

```
module GetSensorReading(f, m,
p, q);
   input f;
   output reg m, p, q;

                  begin
      m = f;
      q = ~f;
      p = ~f;
   end
endmodule
```

Check Show answer

An ***assignment statement*** like y = a & b assigns the left-side variable with the result of the right-side expression. A ***variable*** with the value being updated by assignment statements, and is declared using the reg keyword. An output can be declared in: `output reg y;`

An ***expression*** is a combination of items, like inputs, variables, literals, and operators, that evaluates to a value. Ex: a & b, wh
bitwise AND operator. If a is 0 and b is 1, then the expression evaluates to 0 & 1, which is 0.

An operator is a symbol for a built-in calculation like & for AND. ***Bitwise operators*** perform the specified Boolean operation
the operands.

Table 3.25.1: Bitwise operators.

Operator	Description
&	Bitwise AND operation.
|	Bitwise OR operation.
~	Bitwise NOT operation.
^	Bitwise XOR operation.
^~	Bitwise XNOR operation.

PARTICIPATION ACTIVITY 3.25.5: Boolean expression in Verilog.

Directly convert each given Boolean expression to Verilog.
Note: This activity requires answers that directly match the given expressions. Ex: For ab, type a & b. Variations like b & a are not accounted for.

1) z = a + b

```
z = ;
```

Check Show answer

2) z = a'b

```
z = [        ] & b;
```

Check Show answer

3) z = (ab) + c

```
z = [        ];
```

Check Show answer

4) z = (ab') + (ac)

```
z = [        ];
```

Check Show answer

Forgetting to include all inputs of combinational logic in the always procedure's sensitivity list is a common error. While the procedure is legal Verilog, the procedure does not describe combinational logic.

PARTICIPATION ACTIVITY 3.25.6: Detect the error in defining a module for combinational logic.

Click the erroneous code.

1)
```
module CloseDoor(a, b, c, z);
  input a, b, c;
  output reg z;
  always @ (a, b) begin
    z = ~a | ~b | ~c;
  end
endmodule
```

2)
```
module SoundAlarm(a, b, y, z);
```

```
  input a, b;
  output reg y, z;
  always @(a, b)
     y = a & b;
     z = a;
endmodule
```

3)
```
module LightController(a, b, z);
  input a, b;
  output z;
  always @ (a, b) begin
    z = a | b;
  end
endmodule
```

4)
```
module UnlockDoor(y, z, c, d);
  input c, d ;
  output reg y, z;
  always @(c, d) begin
    y = ~c;
    z = c ^ d;
  end
end module
```

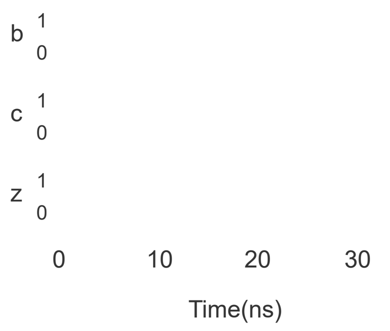
5)
```
module StartEngine(a, b, z)
  input a, b;
  output reg z;
  always @(a, b) begin
    z = ~(a & b);
  end
endmodule
```

PARTICIPATION ACTIVITY 3.25.7: HDL simulator: Always procedure.

Start 2x speed

```
module DoorOpenDetector(b, c, z);
  input b, c;
  output reg z;

  always @(b, c) begin
    z = b & ~c;
  end
endmodule
```

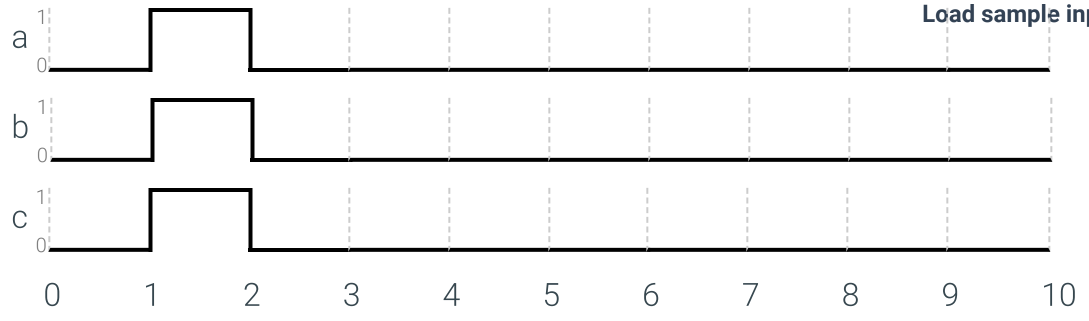
Timing diagram



PARTICIPATION ACTIVITY 3.25.8: Combinational logic simulator: Designer provides input values; running the simulator generates output values.

```
module CombinationalLogicExample(a, b, c, z);
        input a, b, c;
        output reg z;
        always @(a, b, c) begin
            z = a & b & c;
        end
      endmodule
```

Inputs: *click waveform to edit*



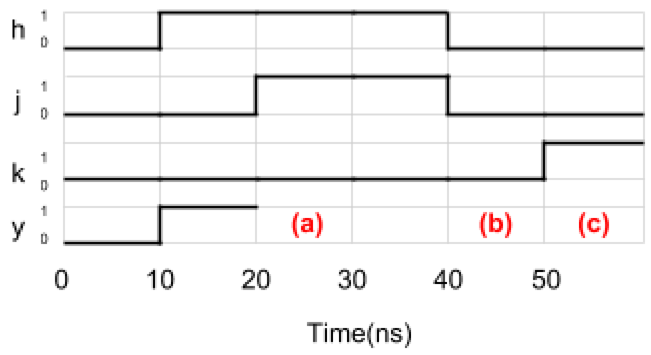
Run

Output:

PARTICIPATION ACTIVITY 3.25.9: Sensitivity list.

```
module AirController(h, j, k, y);
   input h, j, k;
   output reg y;

   always@(h, j, k) begin
      y = h | (~j & k);
   end
endmodule
```

Timing diagram



1) Value of y at (a).

○ 0

○ 1

2) The always procedure executes at 30 ns.

○ True

○ False

3) Value of y at (b).

○ 0

○ 1

4) The always procedure executes at 50 ns.

○ True

○ False

5) Value of y at (c).

○ 0

○ 1

Provide feedback on this section