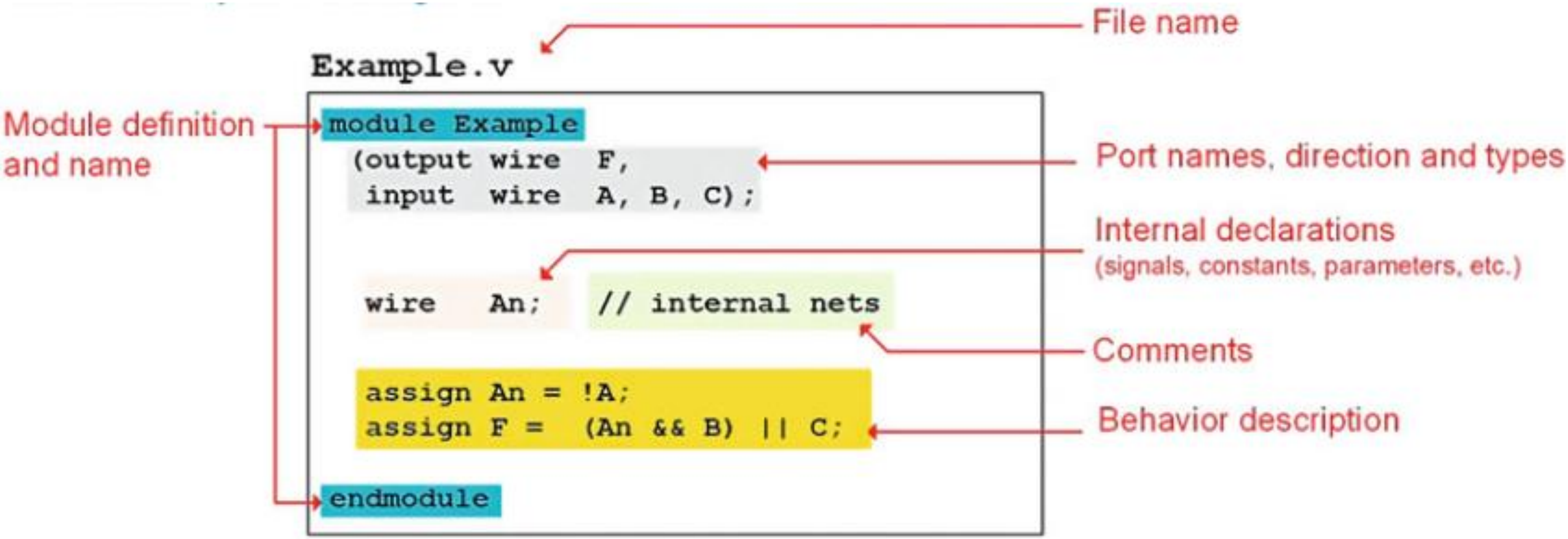
Verilog HDL: Module Getting Started

Pravin Zode

Outline

- Structure of Verilog Program
- Ports, Signals, Operators
- Continuous Signal Assignment
- Exercises

Structure of Verilog program



Example.v

```
module Example
 (output wire  F,
  input  wire  A, B, C);

  wire   An;   // internal nets

  assign An = !A;
  assign F =  (An && B) || C;

endmodule
```

File name

Module definition and name

Port names, direction and types

Internal declarations

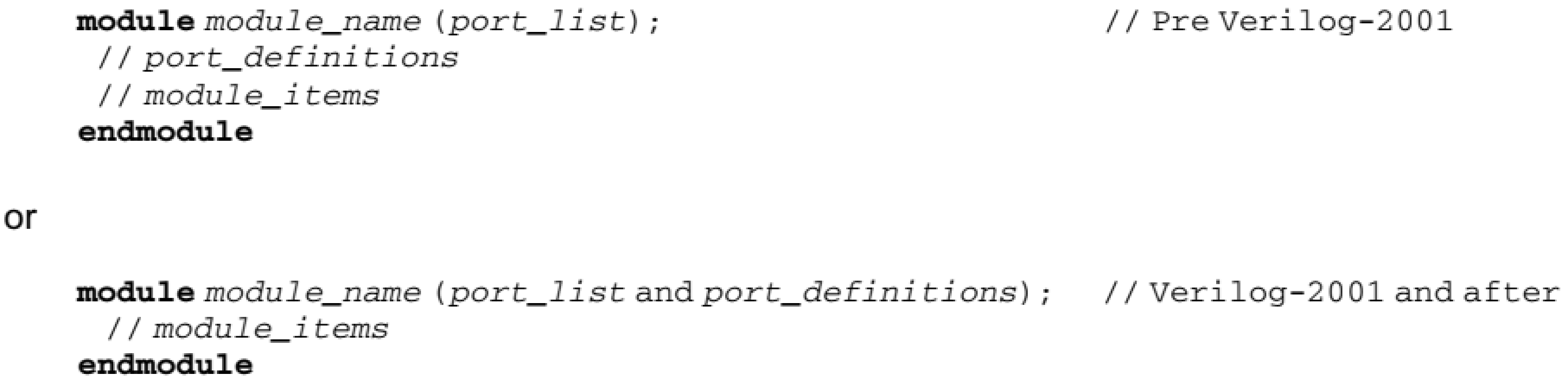
(signals, constants, parameters, etc.)

Comments

Behavior description

Structure of Verilog program

- All systems in Verilog are encapsulated inside a module



```
module module_name (port_list);                          // Pre Verilog-2001
 // port_definitions
 // module_items
endmodule
```

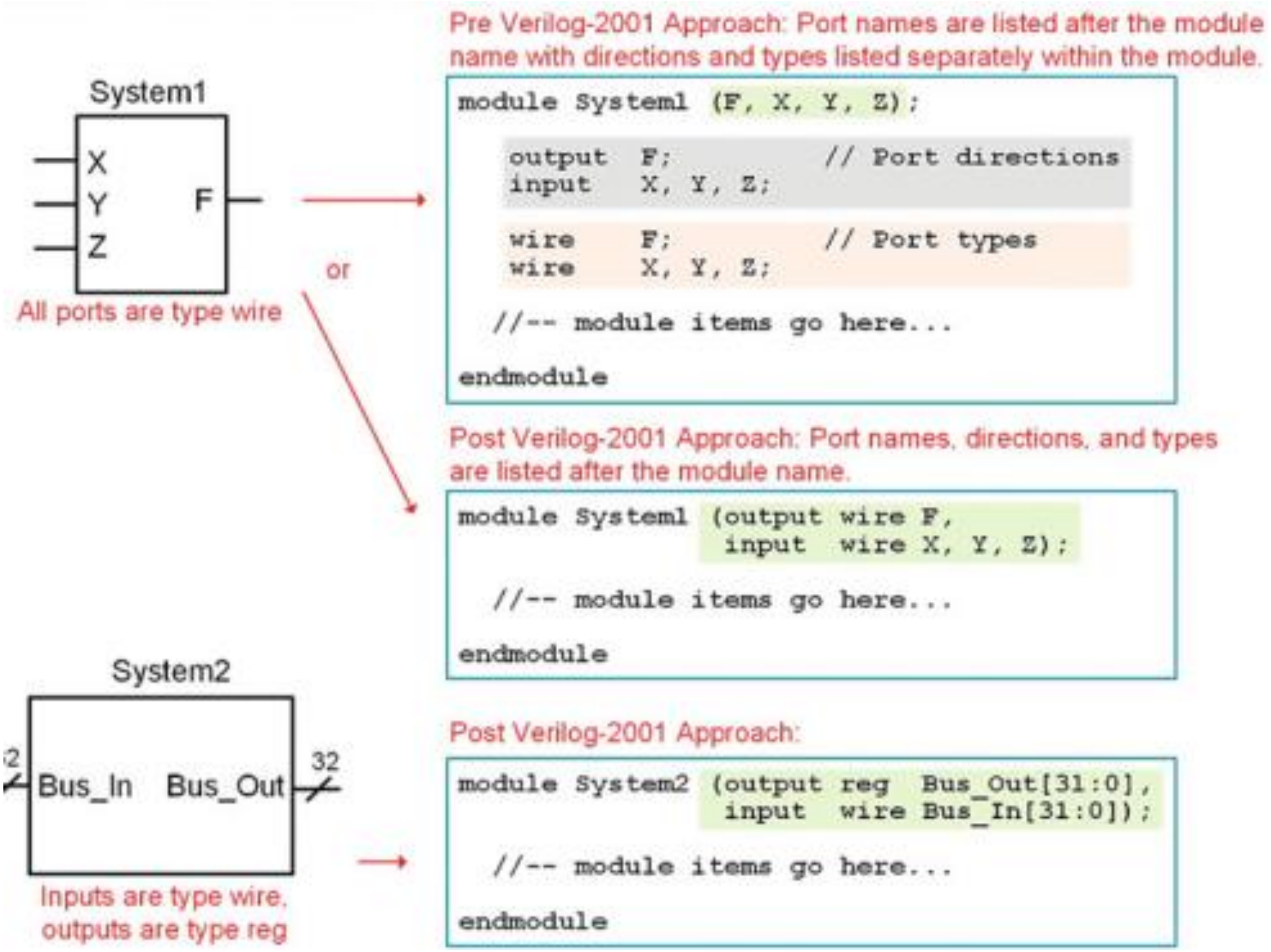
or

```
module module_name (port_list and port_definitions);   // Verilog-2001 and after
  // module_items
endmodule
```

Port Definitions

- The first item in a module is the definition of inputs and outputs (ports).
- Each port must have:
  - User-defined name (case-sensitive, must start with an alphabetic character).
  - Direction: input, output, or inout.
  - Type: Wires, Registers, or Integers (only these are synthesizable).
  - Multiple ports of the same type and direction can be listed on the same line, separated by commas.

Port Definitions



All ports are type wire

Pre Verilog-2001 Approach: Port names are listed after the module name with directions and types listed separately within the module.

```
module System1 (F, X, Y, Z);

   output  F;          // Port directions
   input   X, Y, Z;

   wire    F;          // Port types
   wire    X, Y, Z;

  //-- module items go here...

endmodule
```

Post Verilog-2001 Approach: Port names, directions, and types are listed after the module name.

```
module System1 (output wire F,
                 input  wire X, Y, Z);

  //-- module items go here...

endmodule
```

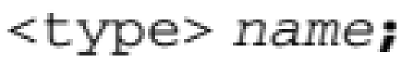
Inputs are type wire, outputs are type reg

Post Verilog-2001 Approach:

```
module System2 (output reg  Bus_Out[31:0],
                 input  wire Bus_In[31:0]);

  //-- module items go here...

endmodule
```

Signal Declarations

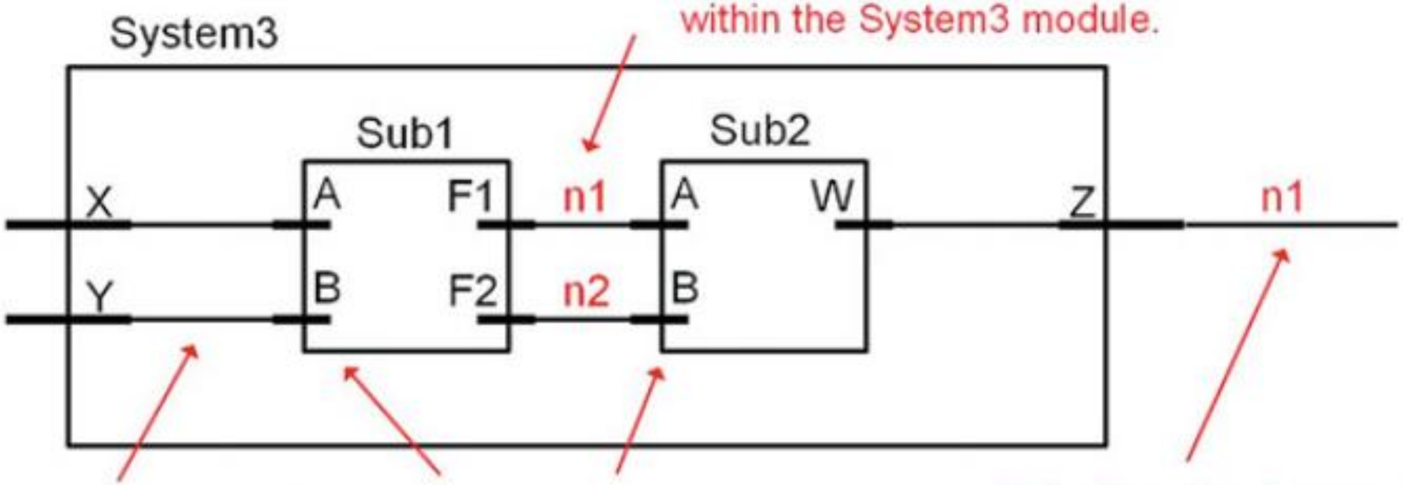
- Internal signals are used for connections within a module.
- Signals must be declared before their first use in the module.
- Declaration format: Type (e.g., wire, reg, integer)
- User-defined name (case-sensitive, must start with an alphabetic character)
- Multiple signals of the same type can be declared on the same line, separated by commas.
- Synthesizable signal types:
    - net (e.g., wire, tri)
    - Reg
    - integer



```
<type> name;
```

Signals and Systems

Signals n1 and n2 are declared within the System3 module.



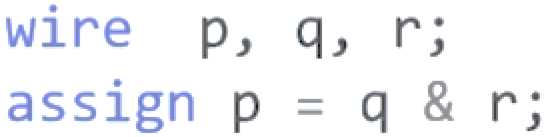
A new signal is not needed for these connections. The port names can be used to signify the connections instead.

The port names A and B are used in two sub-systems. This is legal since they are named within the lower-level sub-systems. They are not connected to each other implicitly and there is no conflict.

Using the signal name n1 is legal here. The signal does not “see” the duplicate signal name “n1” within the System3 module because they are at different levels of hierarchy.

Continuous Assignments

- Mainly used to assign values to vector and scalar nets
- Whenever the RHS is changed, the values are assigned
- Continuous assignments can enable modeling of combinational logic
- Through these assignments, the logical expressions can drive the nets



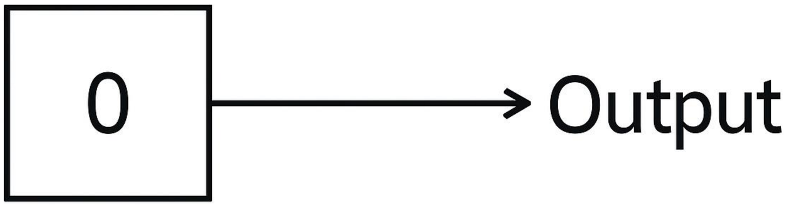
```
wire  p, q, r;
assign p = q & r;
```

Dataflow Modeling

- Describes flow of data from inputs to outputs
- Gate level structures are not used
- It uses several operators that act on operands to produce desired results
- Dataflow modeling is used to describe combinational circuits

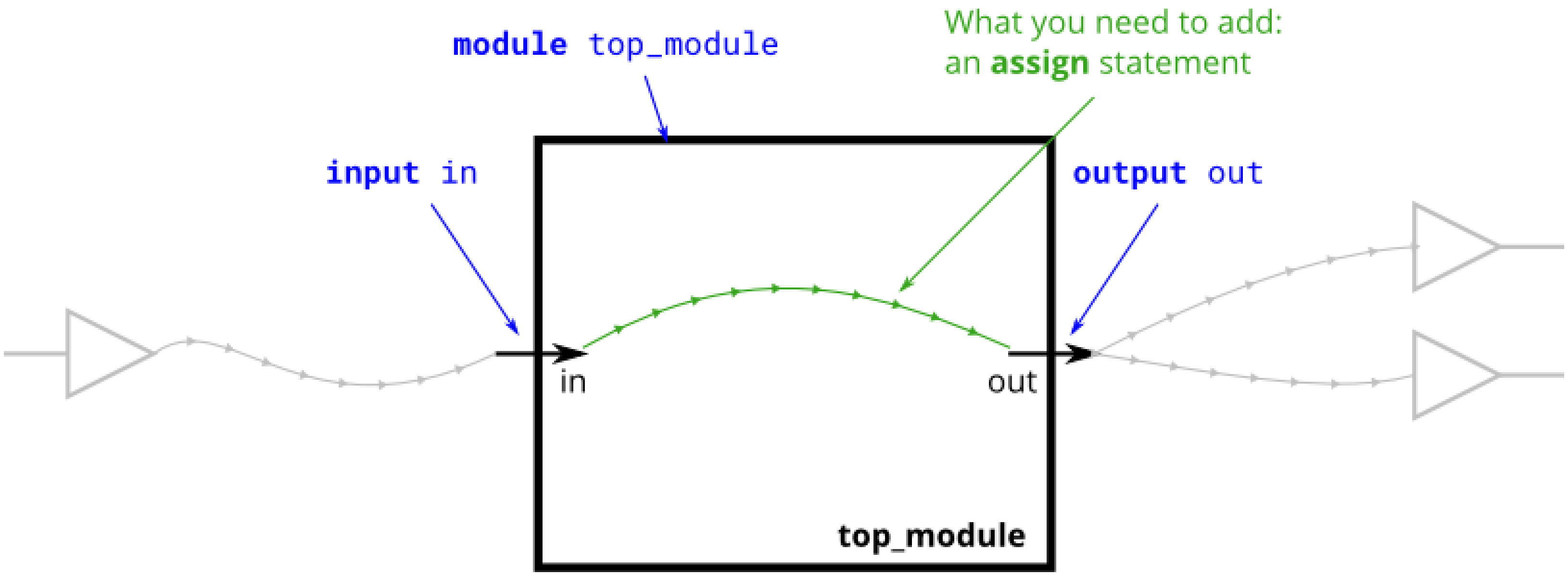
Exercise : 1

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



Exercise : 2

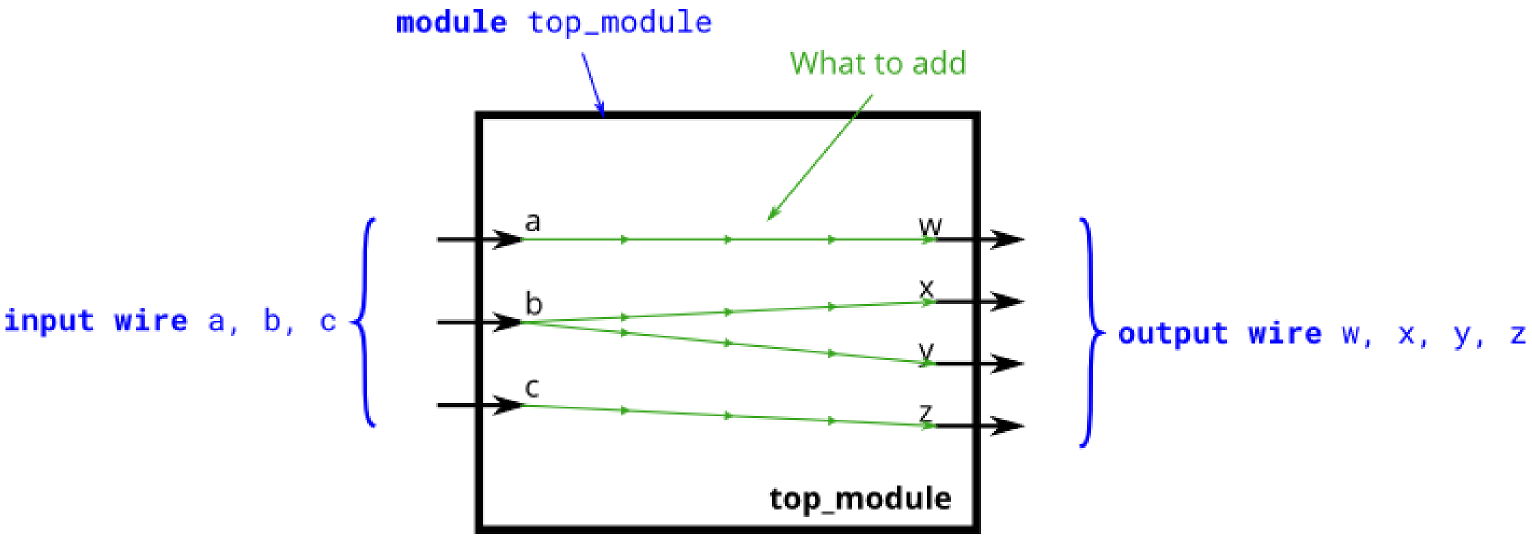
- Create a module with one input and one output that behaves like a wire.



Exercise : 2

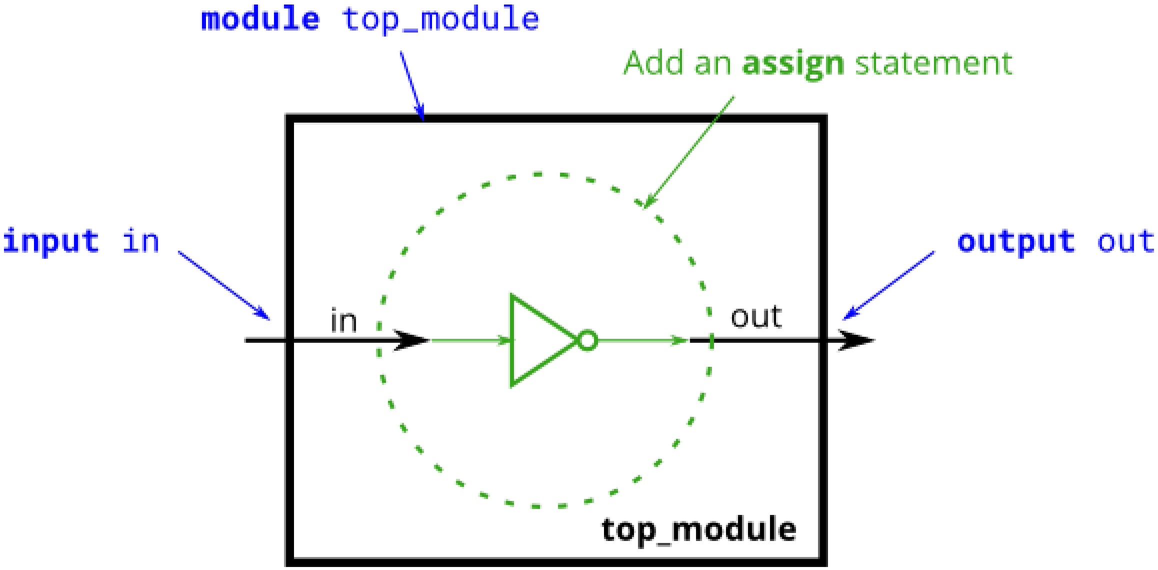
- Create a module with 3 inputs and 4 outputs that behaves like wires that makes these connections:

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



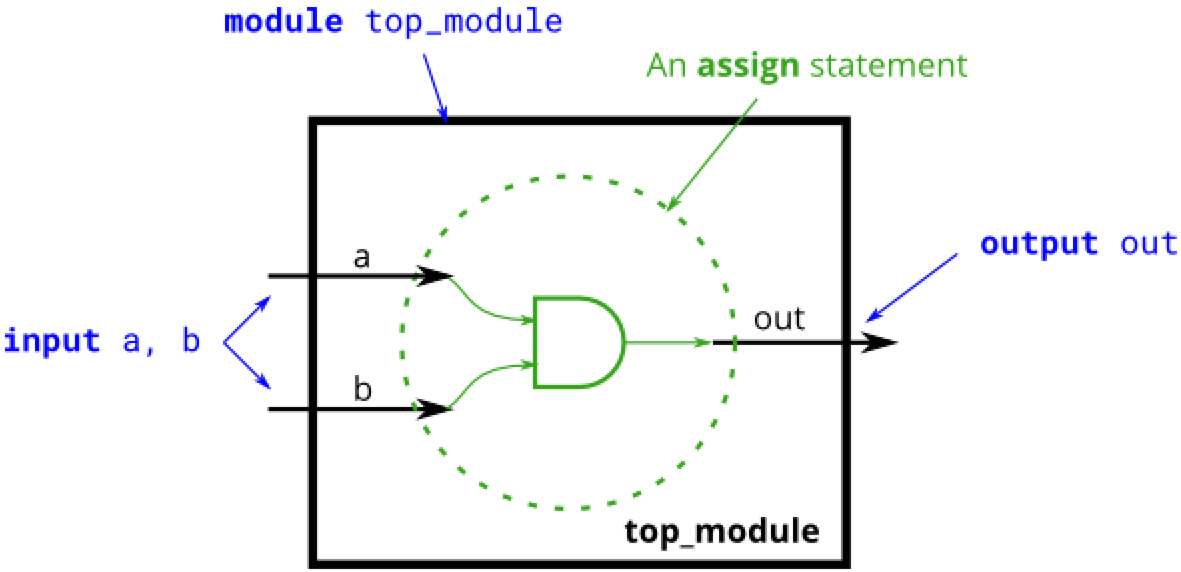
Exercise : 4

- Create a module that implements a NOT gate.



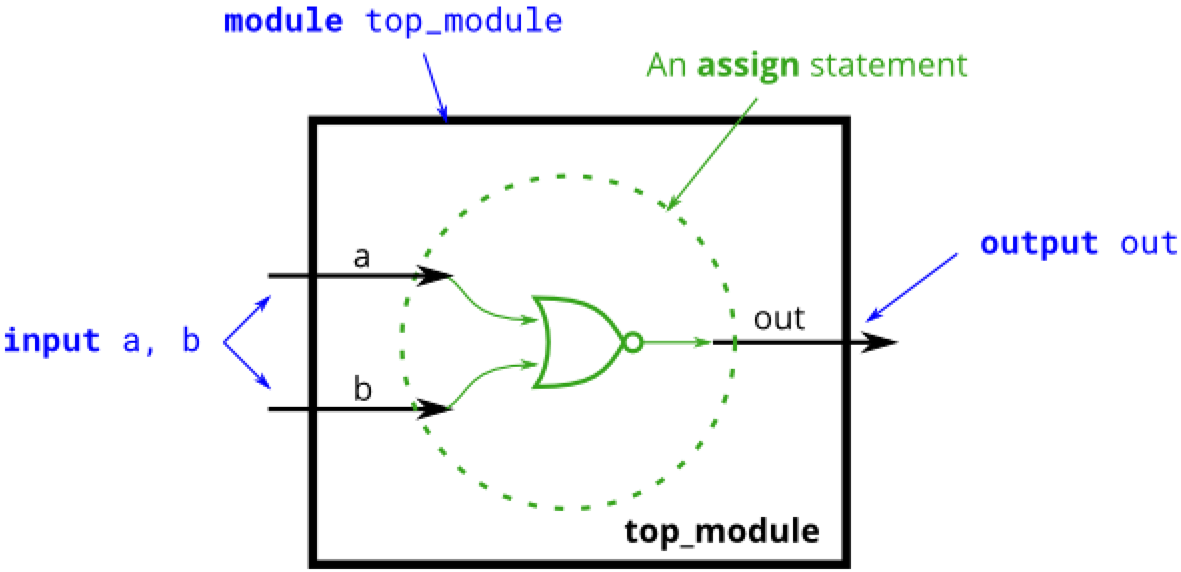
Exercise : 5

- Create a module that implements AND gate.



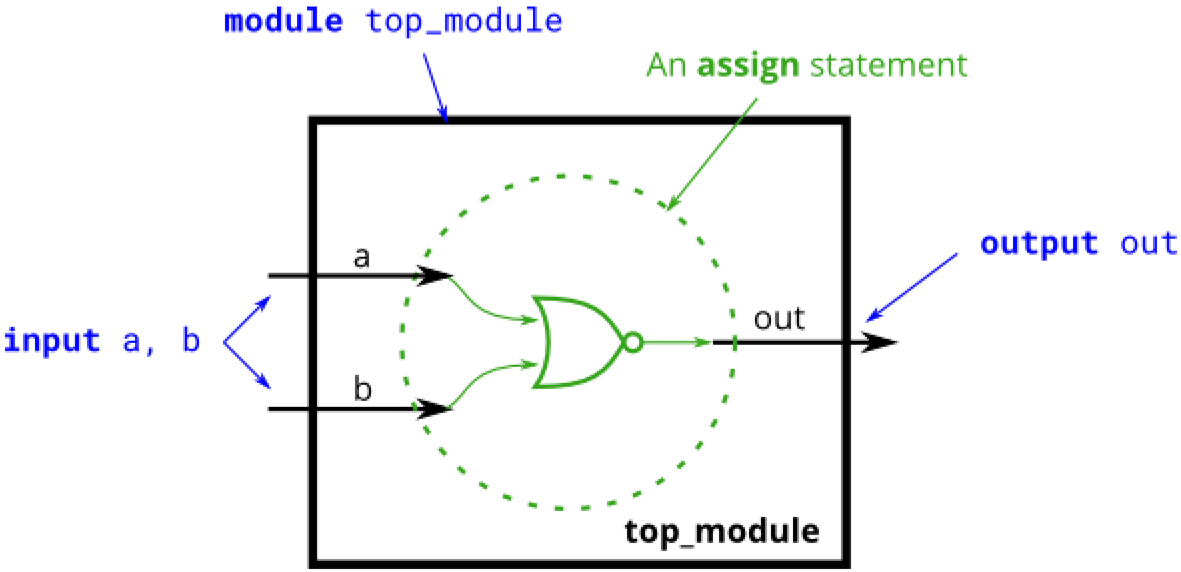
Exercise : 6

- Create a module that implements NOR gate.



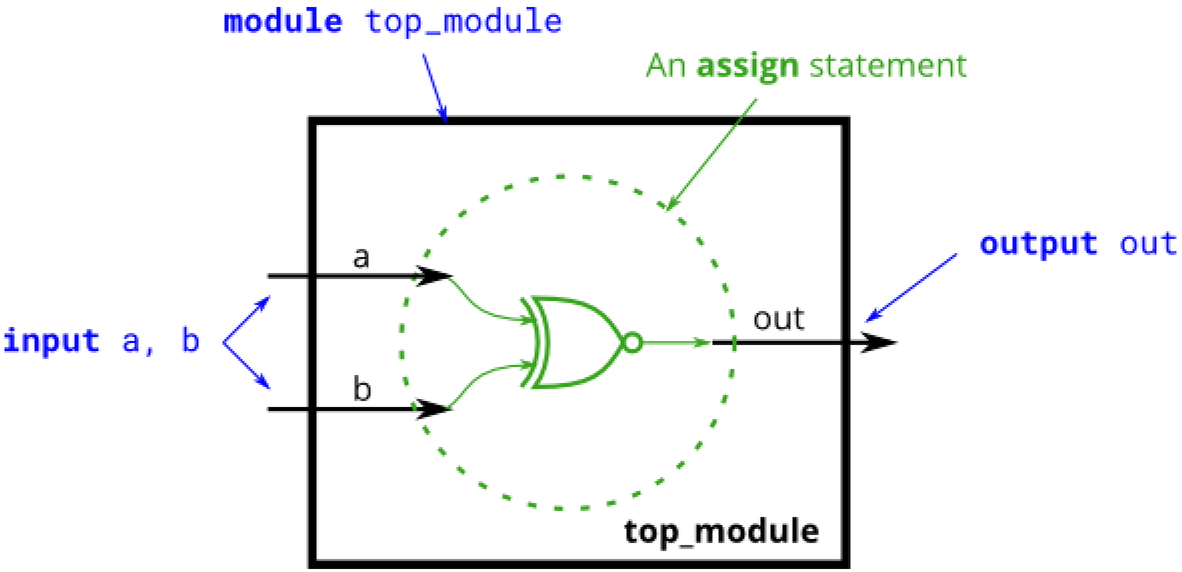
Exercise : 7

- Create a module that implements NOR gate.



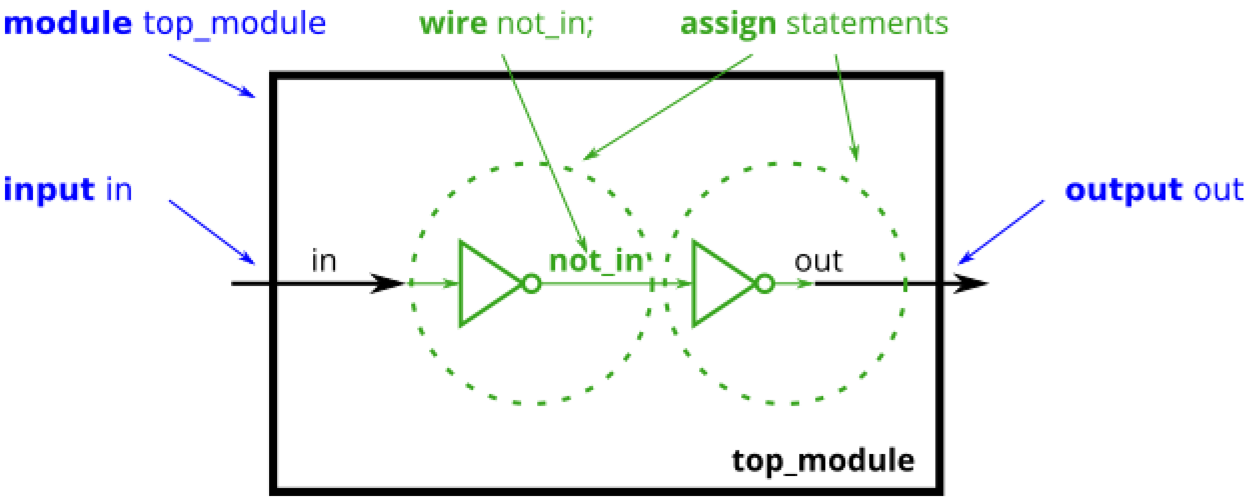
Exercise : 8

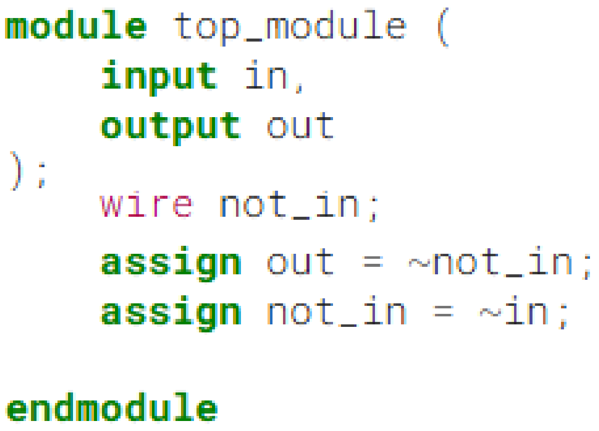
- Create a module that implements an XNOR gate.



Declaring Wire

- Wires are used to connect internal components

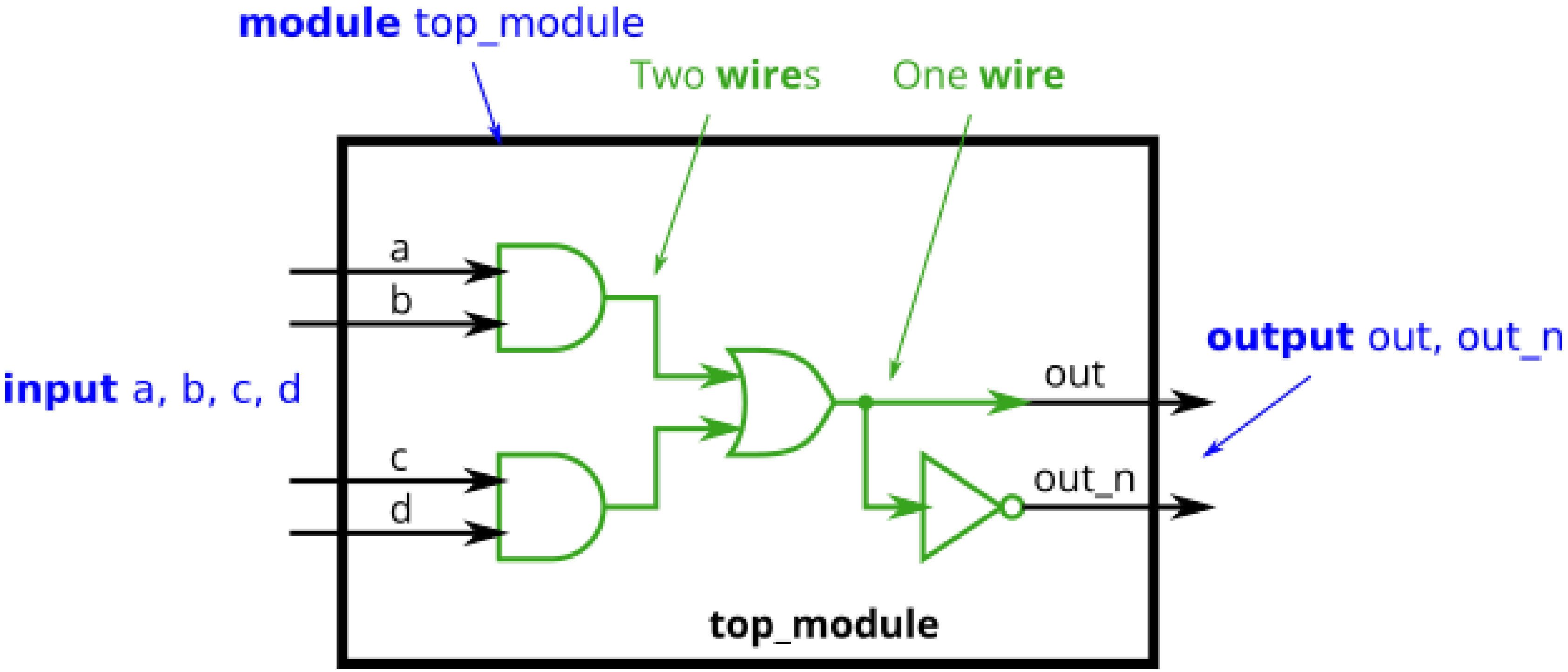




```
module top_module (
    input in,
    output out
);
    wire not_in;

    assign out = ~not_in;
    assign not_in = ~in;

endmodule
```

Exercise : 9

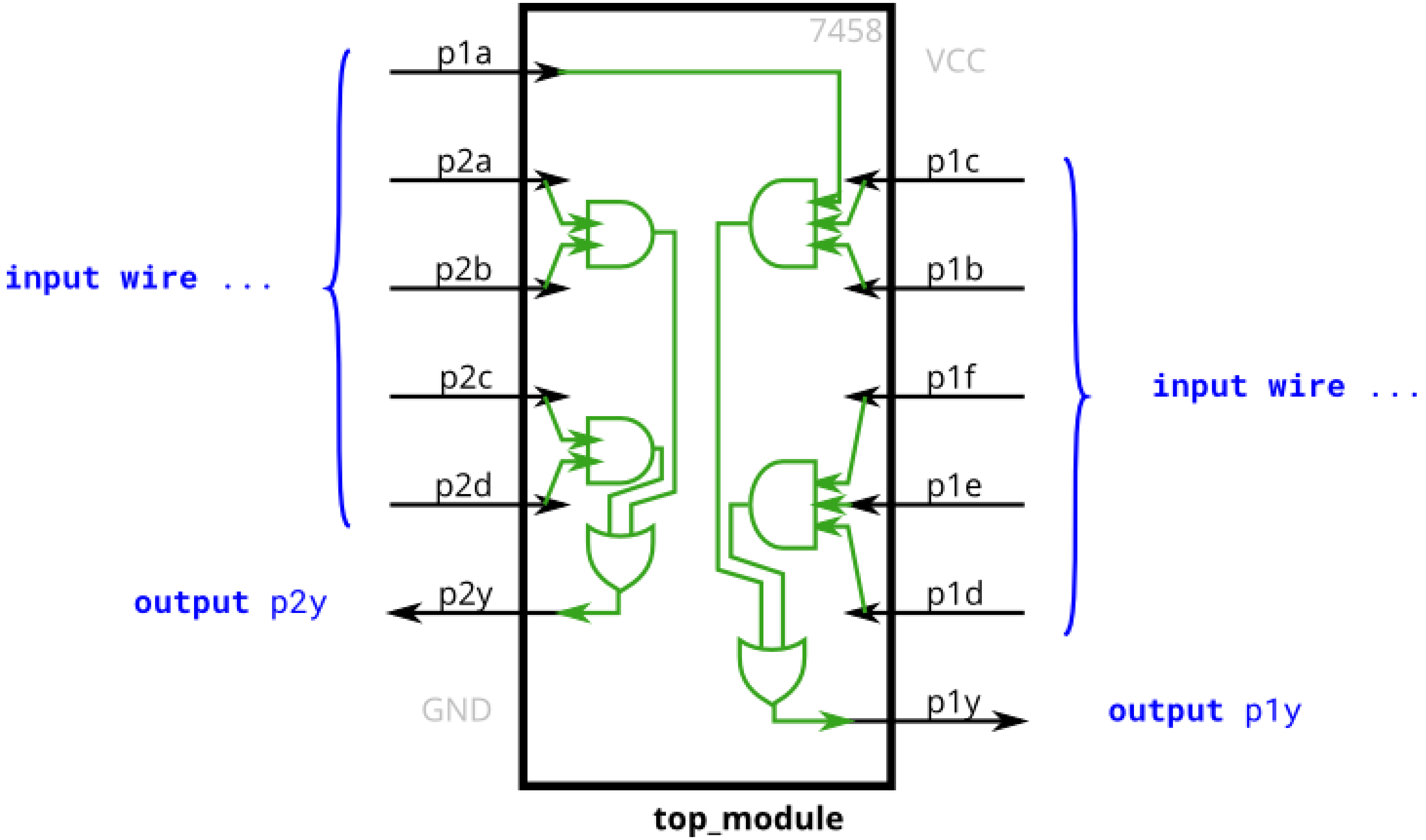
- Implement the following circuit



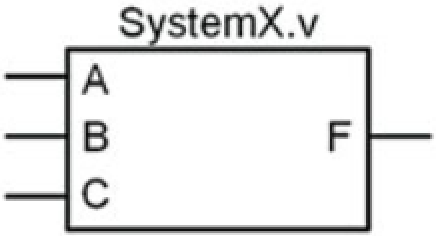
Exercise : 10

- Create a module with the same functionality as the 7458 chip.

  It has 10 inputs and 2 outputs

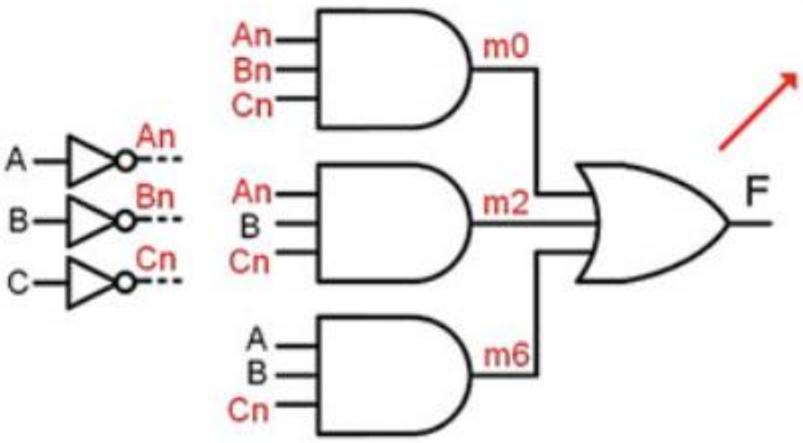


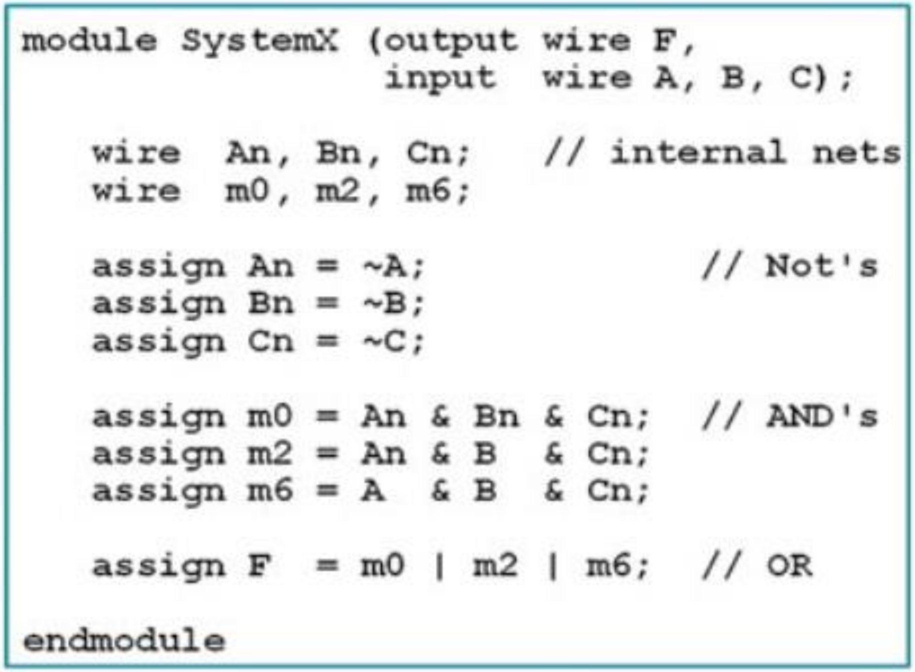
Continuous Assignment with Logical Operators



A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

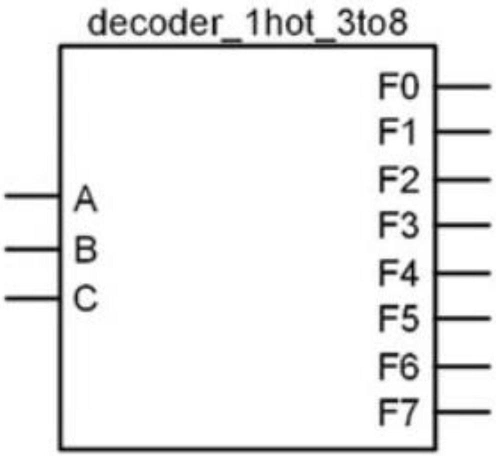
$$F = \Sigma_{A,B,C}(0,2,6) = A'\cdot B'\cdot C' + A'\cdot B\cdot C' + A\cdot B\cdot C'$$

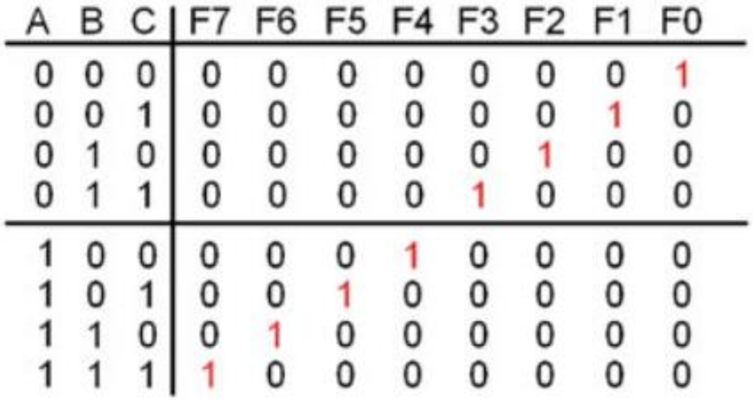




```
module SystemX (output wire F,
                input  wire A, B, C);

   wire  An, Bn, Cn;   // internal nets
   wire  m0, m2, m6;

   assign An = ~A;                 // Not's
   assign Bn = ~B;
   assign Cn = ~C;

   assign m0 = An & Bn & Cn;   // AND's
   assign m2 = An & B  & Cn;
   assign m6 = A  & B  & Cn;

   assign F  = m0 | m2 | m6;   // OR

endmodule
```

One-Hot Decoder





A	B	C	F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$$F0 = \Sigma_{A,B,C}(0) = A' \cdot B' \cdot C'$$

$$F1 = \Sigma_{A,B,C}(1) = A' \cdot B' \cdot C$$

$$F2 = \Sigma_{A,B,C}(2) = A' \cdot B \cdot C'$$

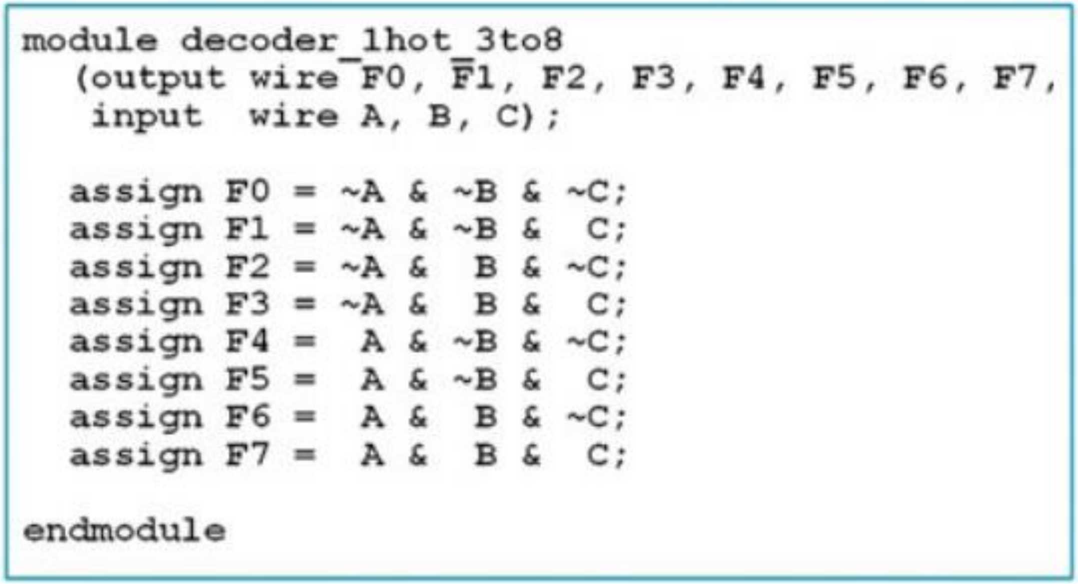
$$F3 = \Sigma_{A,B,C}(3) = A' \cdot B \cdot C$$

$$F4 = \Sigma_{A,B,C}(4) = A \cdot B' \cdot C'$$

$$F5 = \Sigma_{A,B,C}(5) = A \cdot B' \cdot C$$

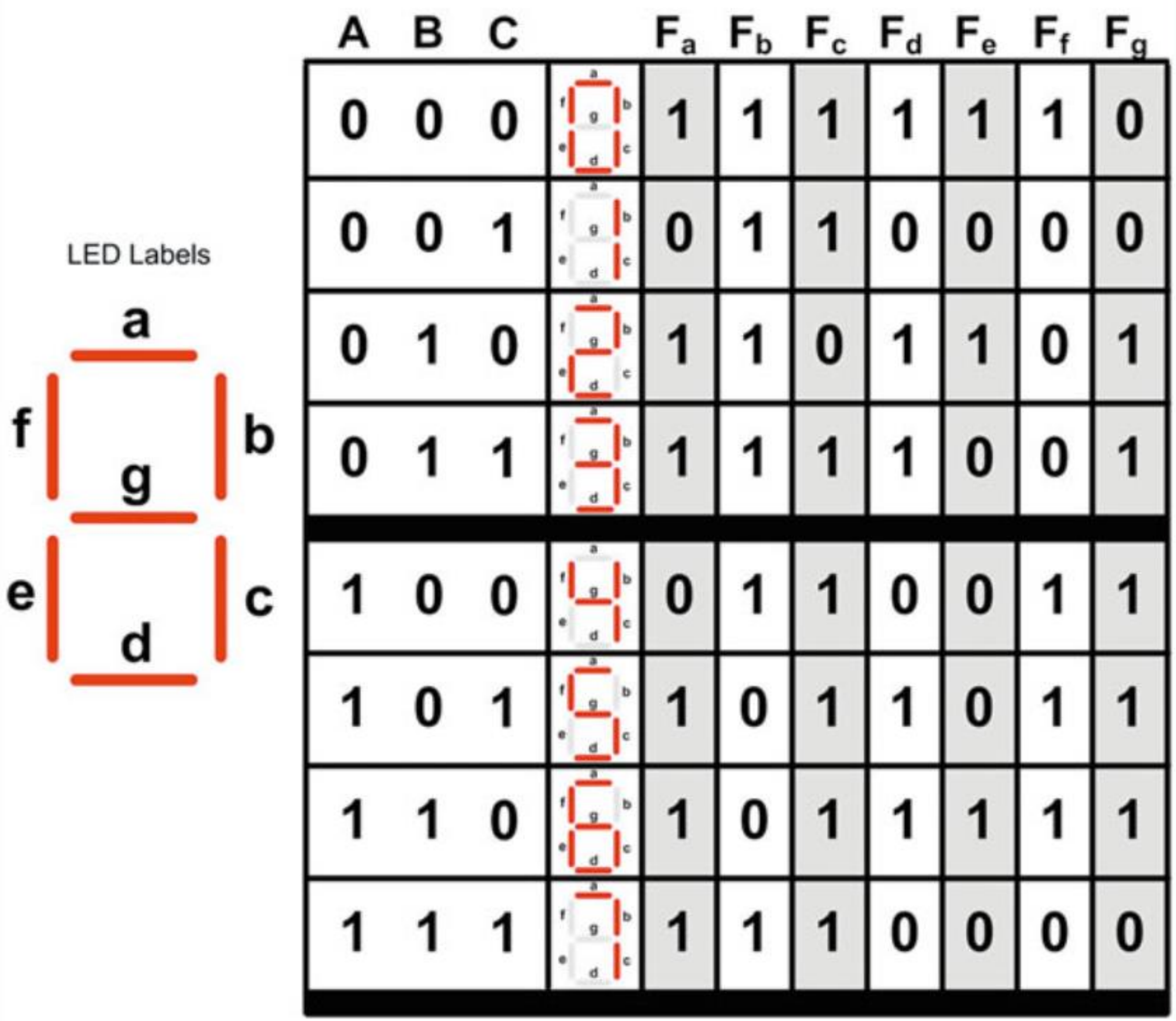
$$F6 = \Sigma_{A,B,C}(6) = A \cdot B \cdot C'$$

$$F7 = \Sigma_{A,B,C}(7) = A \cdot B \cdot C$$



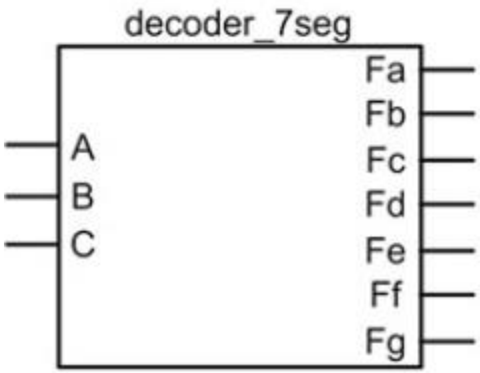
```
module decoder_1hot_3to8
   (output wire F0, F1, F2, F3, F4, F5, F6, F7,
    input  wire A, B, C);

  assign F0 = ~A & ~B & ~C;
  assign F1 = ~A & ~B &  C;
  assign F2 = ~A &  B & ~C;
  assign F3 = ~A &  B &  C;
  assign F4 =  A & ~B & ~C;
  assign F5 =  A & ~B &  C;
  assign F6 =  A &  B & ~C;
  assign F7 =  A &  B &  C;

endmodule
```

Exercise : 7 Segment Decoder



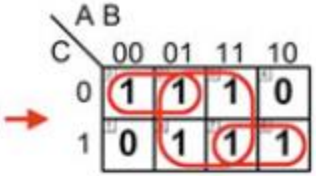
A	B	C		F_a	F_b	F_c	F_d	F_e	F_f	F_g
0	0	0		1	1	1	1	1	1	0
0	0	1		0	1	1	0	0	0	0
0	1	0		1	1	0	1	1	0	1
0	1	1		1	1	1	1	0	0	1
1	0	0		0	1	1	0	0	1	1
1	0	1		1	0	1	1	0	1	1
1	1	0		1	0	1	1	1	1	1
1	1	1		1	1	1	0	0	0	0

Exercise : 7 Segment Decoder



A	B	C	Fa	Fb	Fc	Fd	Fe	Ff	Fg
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

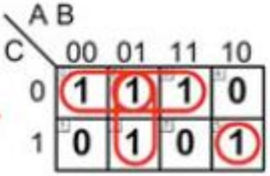
Fa →



C \ AB	00	01	11	10
0	1	1	1	0
1	0	1	1	1

→ Fa = A'·C' + B + A·C

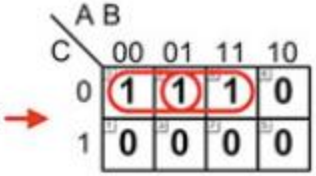
Fd →



C \ AB	00	01	11	10
0	1	1	1	0
1	0	1	0	1

→ Fd = A'·C' + A'·B + B·C' + A·B'·C

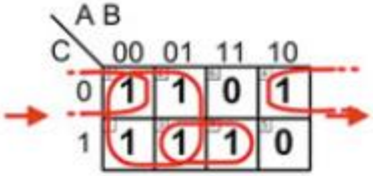
Fe →



C \ AB	00	01	11	10
0	1	1	1	0
1	0	0	0	0

→ Fe = A'·C' + B·C'

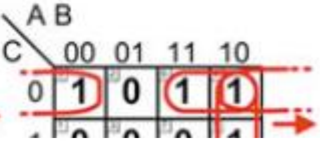
Fb →



C \ AB	00	01	11	10
0	1	1	0	1
1	1	1	1	0

→ Fb = B'·C' + A' + B·C

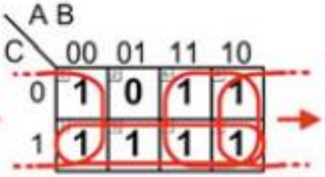
Ff →



C \ AB	00	01	11	10
0	1	0	1	1

→ Ff = B'·C' + A·C' + A·B'

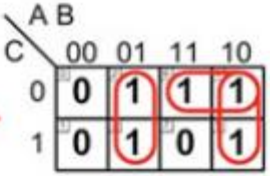
Fc →



C \ AB	00	01	11	10
0	1	0	1	1
1	1	1	1	1

→ Fc = A + B' + C

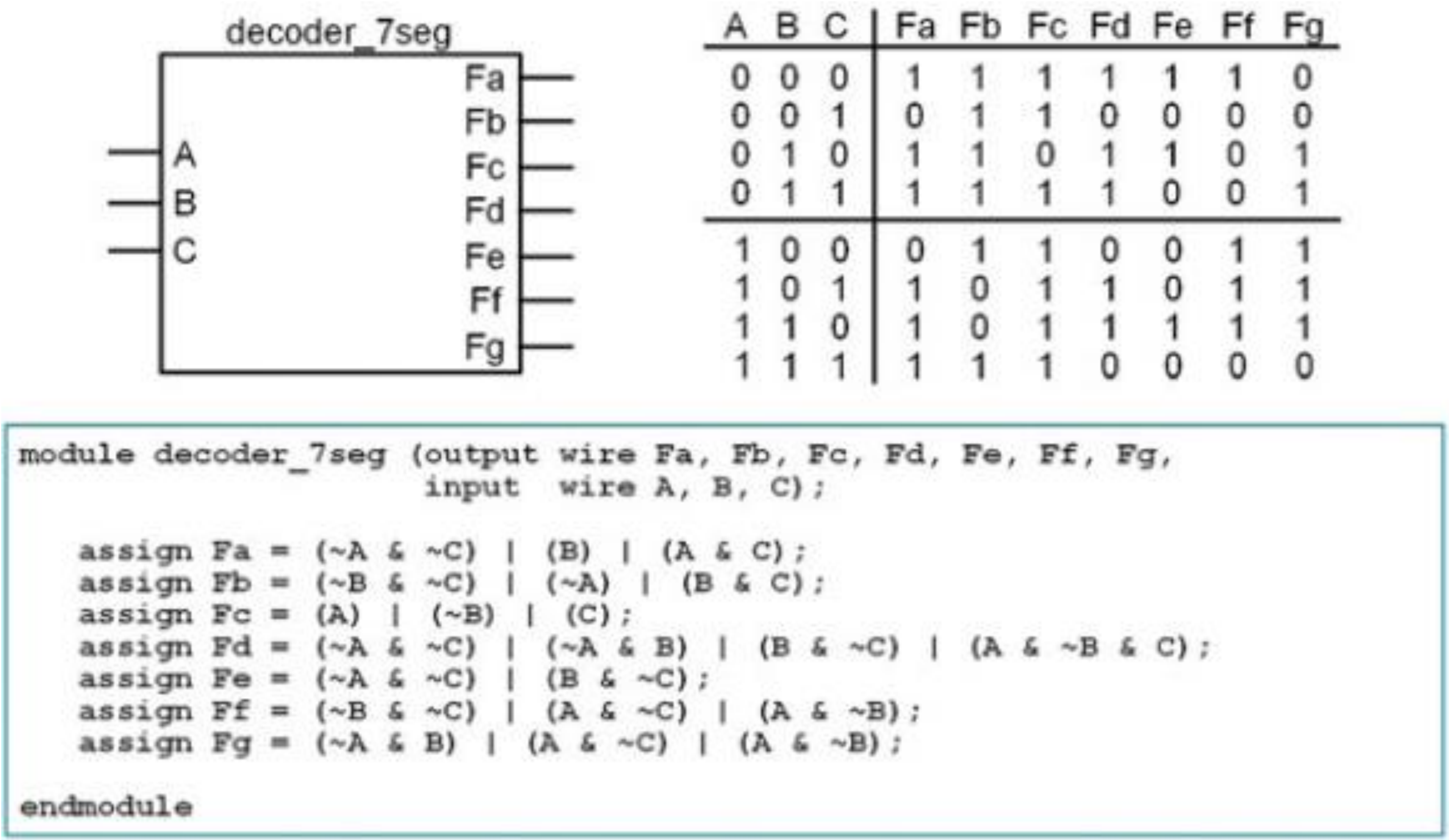
Fg →



C \ AB	00	01	11	10
0	0	1	1	1
1	0	1	0	1

→ Fg = A'·B + A·C' + A·B'

Exercise : 7 Segment Decoder

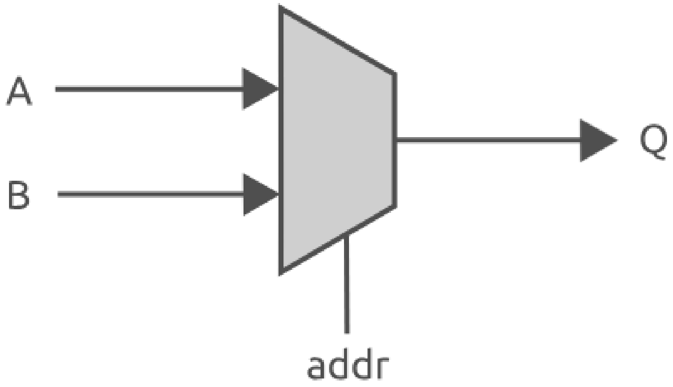


A	B	C	Fa	Fb	Fc	Fd	Fe	Ff	Fg
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

```
module decoder_7seg (output wire Fa, Fb, Fc, Fd, Fe, Ff, Fg,
                     input  wire A, B, C);

   assign Fa = (~A & ~C) | (B) | (A & C);
   assign Fb = (~B & ~C) | (~A) | (B & C);
   assign Fc = (A) | (~B) | (C);
   assign Fd = (~A & ~C) | (~A & B) | (B & ~C) | (A & ~B & C);
   assign Fe = (~A & ~C) | (B & ~C);
   assign Ff = (~B & ~C) | (A & ~C) | (A & ~B);
   assign Fg = (~A & B) | (A & ~C) | (A & ~B);

endmodule
```

Continuous Assignment with Conditional Operators

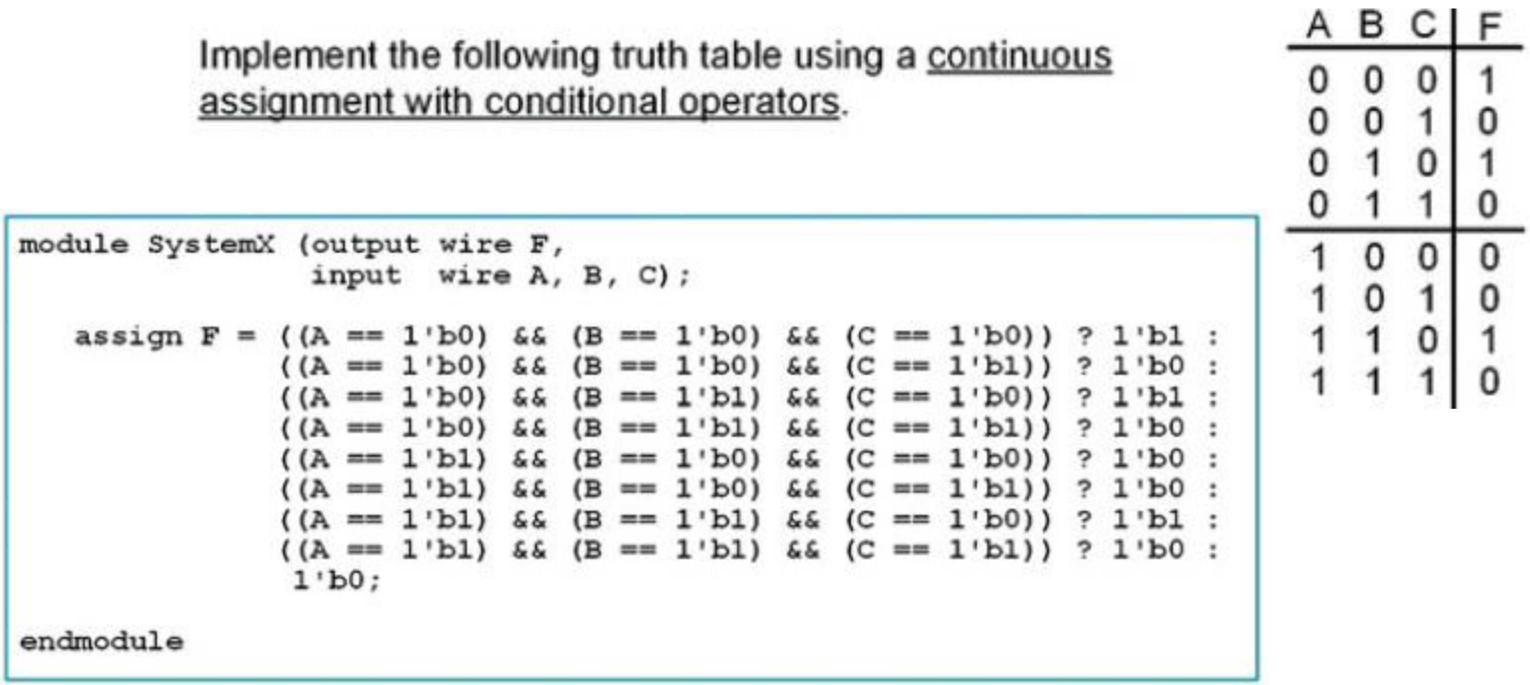
2:1 Multiplexer



```
assign q = addr ? b : a;
```

```
module mux (
    input wire addr,   // Address or control
signal
    input wire a,      // Input a
    input wire b,      // Input b
    output wire q      // Output q
);

    // Conditional assignment
    assign q = addr ? b : a;

endmodule
```

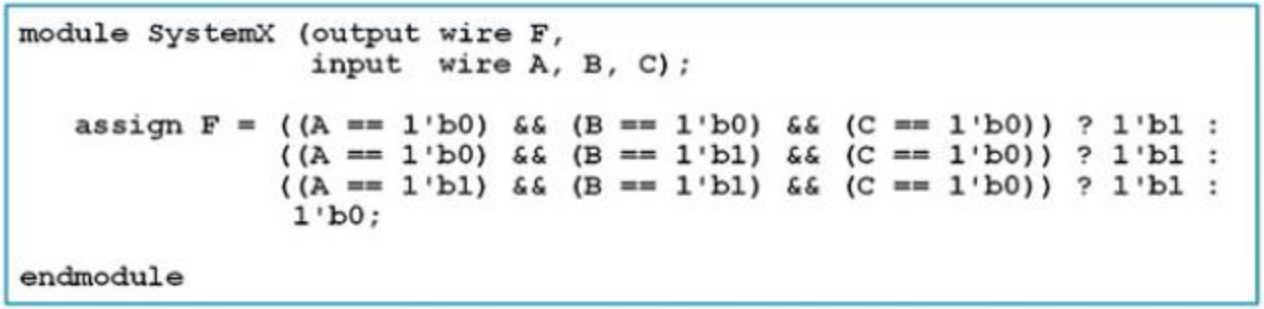
Continuous Assignment with Conditional Operators



Implement the following truth table using a continuous assignment with conditional operators.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

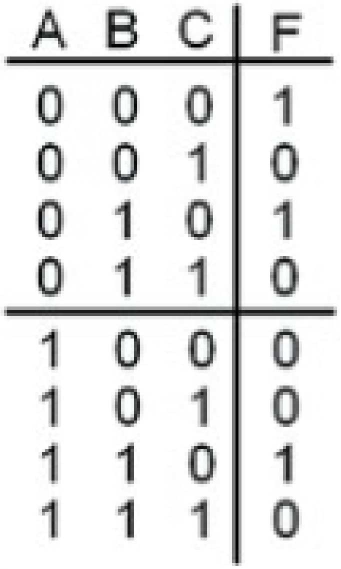
```
module SystemX (output wire F,
                input  wire A, B, C);

   assign F = ((A == 1'b0) && (B == 1'b0) && (C == 1'b0)) ? 1'b1 :
              ((A == 1'b0) && (B == 1'b0) && (C == 1'b1)) ? 1'b0 :
              ((A == 1'b0) && (B == 1'b1) && (C == 1'b0)) ? 1'b1 :
              ((A == 1'b0) && (B == 1'b1) && (C == 1'b1)) ? 1'b0 :
              ((A == 1'b1) && (B == 1'b0) && (C == 1'b0)) ? 1'b0 :
              ((A == 1'b1) && (B == 1'b0) && (C == 1'b1)) ? 1'b0 :
              ((A == 1'b1) && (B == 1'b1) && (C == 1'b0)) ? 1'b1 :
              ((A == 1'b1) && (B == 1'b1) && (C == 1'b1)) ? 1'b0 :
               1'b0;

endmodule
```



```
module SystemX (output wire F,
                input  wire A, B, C);

   assign F = ((A == 1'b0) && (B == 1'b0) && (C == 1'b0)) ? 1'b1 :
              ((A == 1'b0) && (B == 1'b1) && (C == 1'b0)) ? 1'b1 :
              ((A == 1'b1) && (B == 1'b1) && (C == 1'b0)) ? 1'b1 :
               1'b0;

endmodule
```

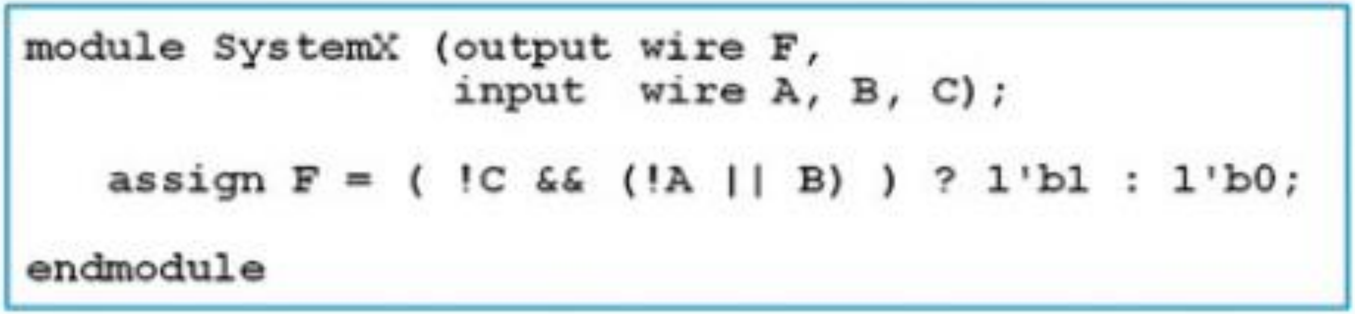
Exercise : Conditional Operators

Implement the truth table using continuous assignment with conditional operator



A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

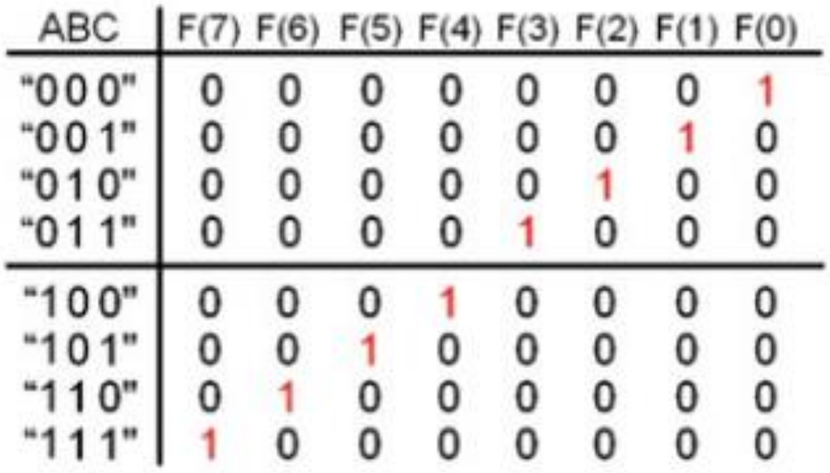
$$F = C' \cdot (A' + B)$$



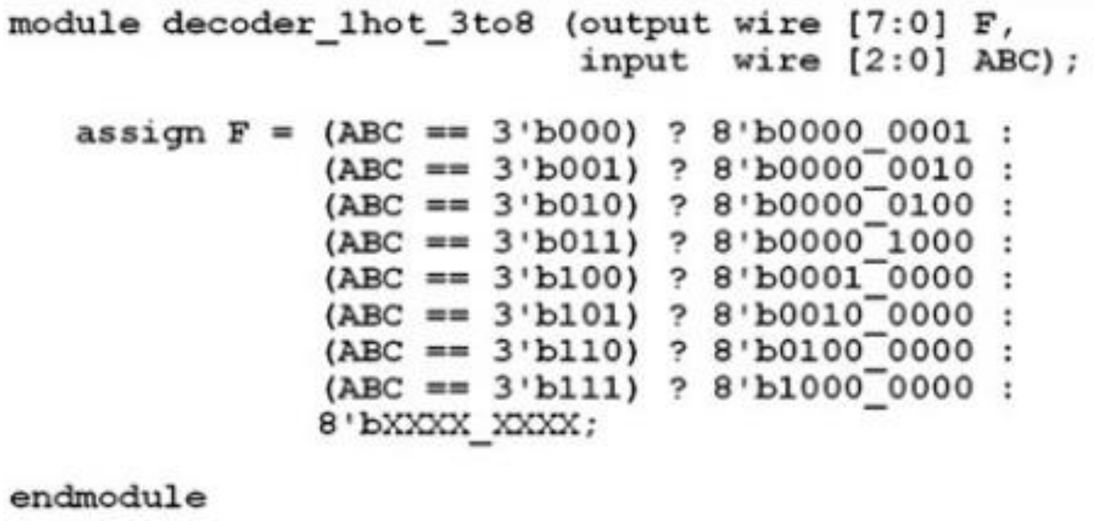
```
module SystemX (output wire F,
                input  wire A, B, C);

  assign F = ( !C && (!A || B) ) ? 1'b1 : 1'b0;

endmodule
```

Exercise : Conditional Operators

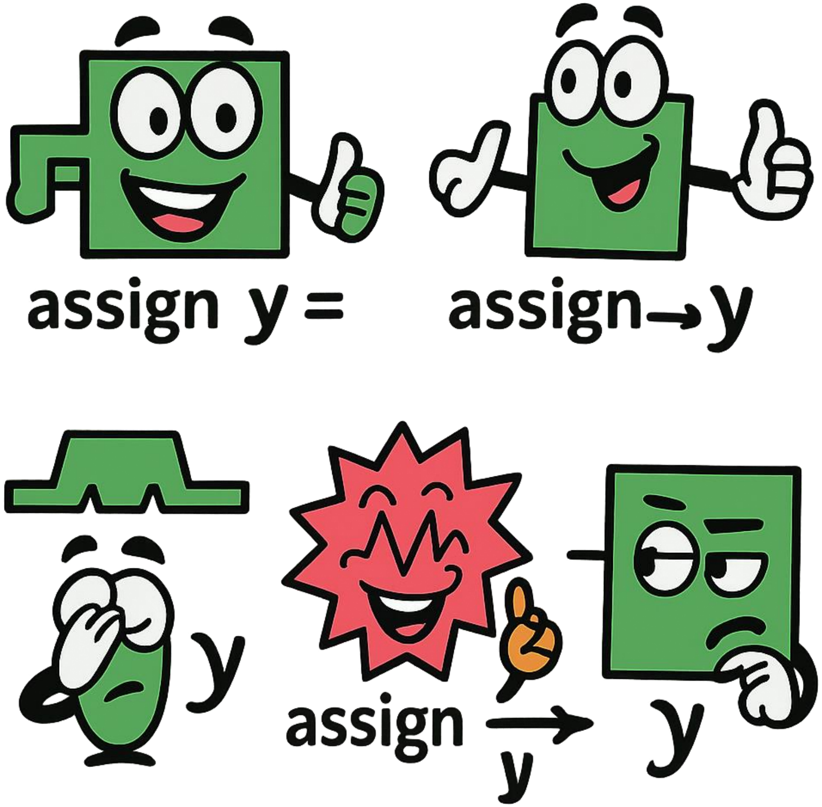
3-to-8 One-Hot Decoder



ABC	F(7)	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
“000”	0	0	0	0	0	0	0	1
“001”	0	0	0	0	0	0	1	0
“010”	0	0	0	0	0	1	0	0
“011”	0	0	0	0	1	0	0	0
“100”	0	0	0	1	0	0	0	0
“101”	0	0	1	0	0	0	0	0
“110”	0	1	0	0	0	0	0	0
“111”	1	0	0	0	0	0	0	0



```
module decoder_1hot_3to8 (output wire [7:0] F,
                          input  wire [2:0] ABC);

  assign F = (ABC == 3'b000) ? 8'b0000_0001 :
             (ABC == 3'b001) ? 8'b0000_0010 :
             (ABC == 3'b010) ? 8'b0000_0100 :
             (ABC == 3'b011) ? 8'b0000_1000 :
             (ABC == 3'b100) ? 8'b0001_0000 :
             (ABC == 3'b101) ? 8'b0010_0000 :
             (ABC == 3'b110) ? 8'b0100_0000 :
             (ABC == 3'b111) ? 8'b1000_0000 :
             8'bXXXX_XXXX;

endmodule
```



Thank you !

Happy Learning