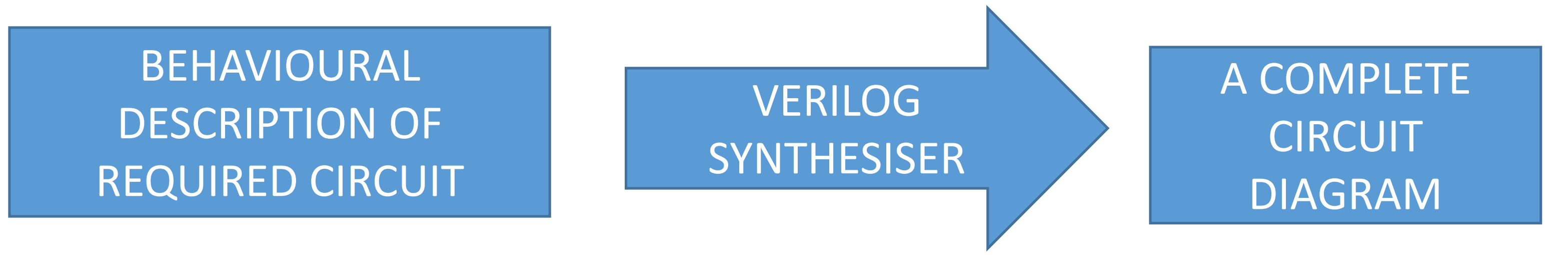
Verilog

Verilog

- Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL)
- Offer a way to design circuits using text-based descriptions.
- HDLs are used to describe a digital system
- Not a programming language despite the syntax being similar to C
- Synthesized (analogous to compiled for C) to give the circuit logic diagram


BEHAVIOURAL DESCRIPTION OF REQUIRED CIRCUIT
VERILOG SYNTHESISER
A COMPLETE CIRCUIT DIAGRAM

Verilog Module Structure

Module STARTING

```
module design
 (
  input  [2:0] data,
  output [2:0] result
  );
```

Module name portlist , port declaration parameter declarations

```
  reg reset_n, clk;
  reg done;
  reg FF_Q, FF_D;
  integer     i;
  wire [2:0]  data_in, data_out;
```

Declaration of wire and reg variables

```
  alg_module alg_instance (
  .clk     (clk ),
  .reset_n (reset_n),
  .d_in     (data_in),
  .d_out    (data_out)
  );
```

Instantiation of other module

```
  assign data_in = 3'b001 + data;
  assign result=data_out ^ data_in;
```

Continuous statements or Data flow statement

```
  initial begin
    forever #(5) clk=~clk;
  end
```

Initial block: blocking assignments

always block: Non-blocking assignments

```
always @(*) begin
  done=0;
  if ( start==1 ) begin
                done=1;
  end else if ( start==0 ) begin
                done=0;
  end else begin
                done=1;
  end

  case(FF_D)
              1'b0: begin
                FF_D=0;
              end
              1'b1: begin
                FF_D=1;
              end
  endcase
end
```

always block: Non-blocking assignments

```
always @ ( posedge clk  or negedge reset_n) begin
  if (reset_n ==0) begin
                FF_Q  <=  0 ;
  end  else begin
                FF_Q  <= FF_D;
  end
end
```

tasks and functions

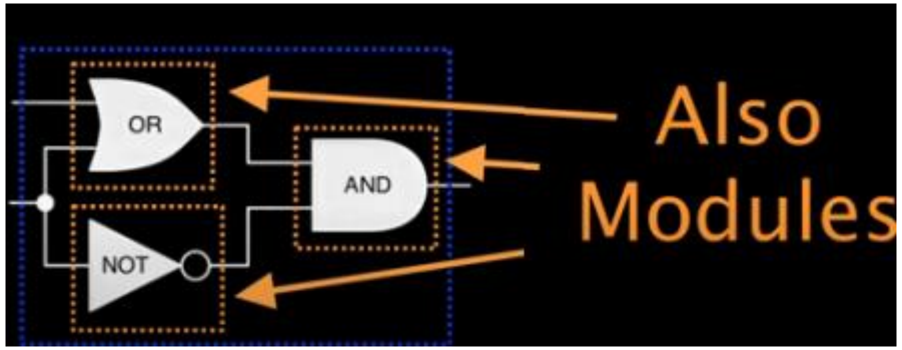
```
task decrypt_task ;
endtask
```

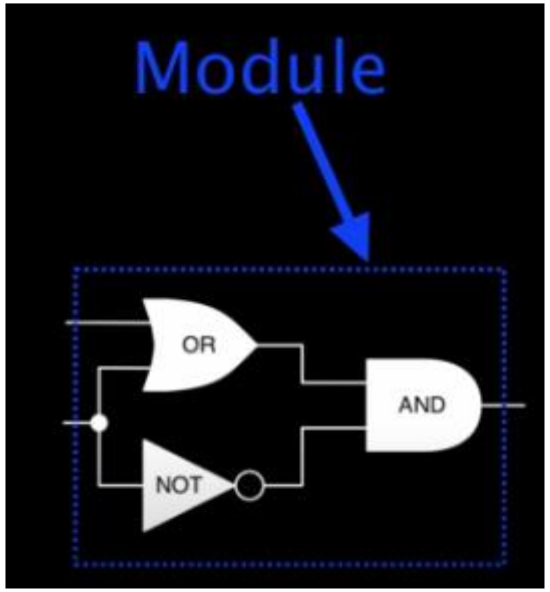
Module ENDING

```
endmodule
```

CODING IN VERILOG

- Breaking circuits into various building blocks called "**module**"
- Defining module
- Connecting various modules
- Communication between a module and its environment is achieved by using Ports
- Ports are of three types: input, output, inout

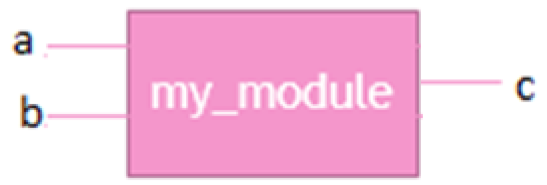




Verilog module

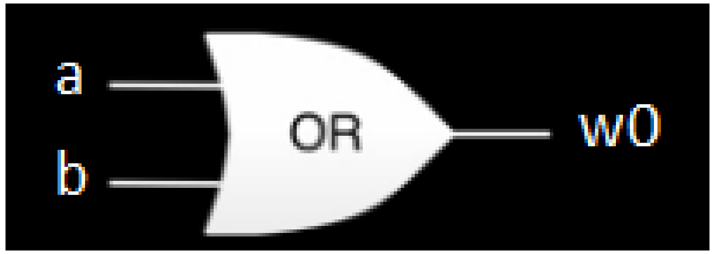
- A "Black Box" in Verilog with inputs, outputs and internal logic working.
- A Module in Verilog is declared within the pair of keywords **module** and **endmodule**. Following the keyword module are the module name and port interface list.

```
module my_module ( c, a, b );
 input a, b;
 output c;
 ...
Endmodule   //comment
```

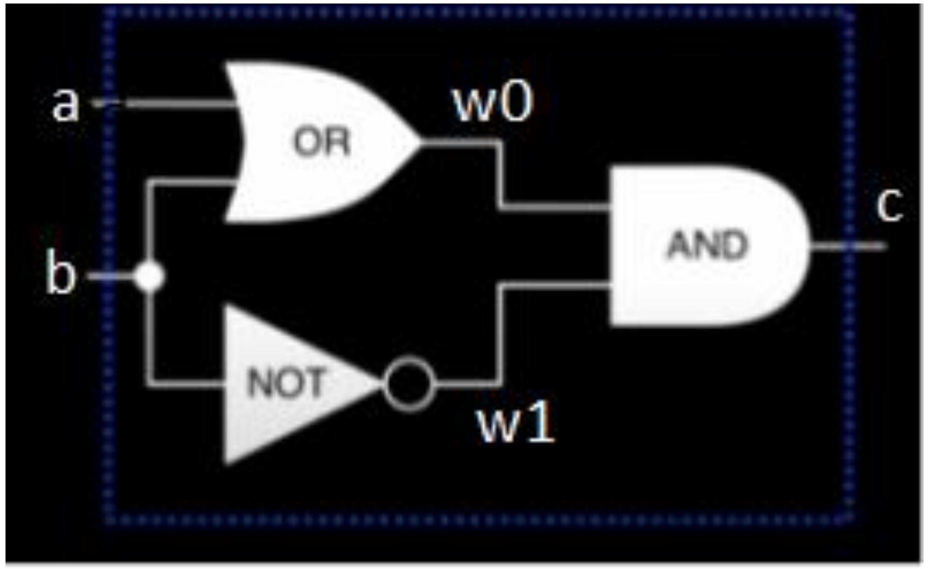


Instantiating module

- Module_name name(port1,port2,port3,…..);

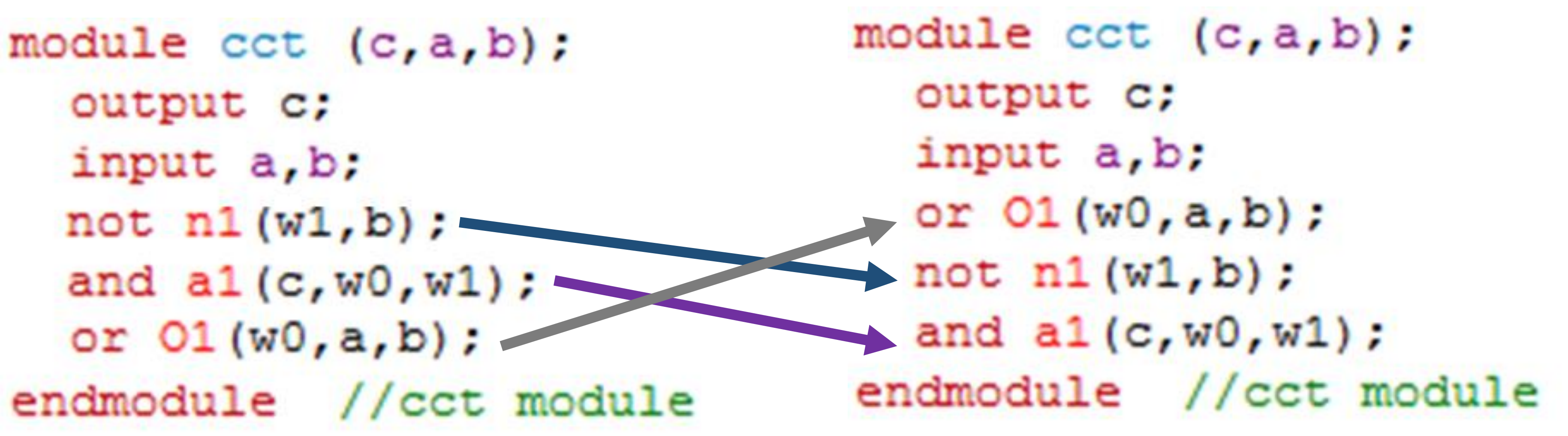


or O1(w0,a,b);



```
module cct (c,a,b);
   output c;
   input a,b;
   or O1(w0,a,b);
   not n1(w1,b);
   and a1(c,w0,w1);
endmodule  //cct module
```

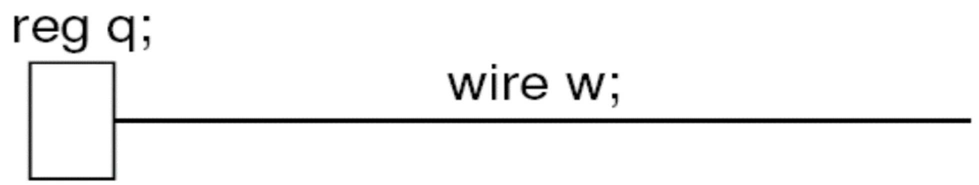
Order doesn't matter: both describe the same circuit



```
module cct (c,a,b);
  output c;
  input a,b;
  not n1(w1,b);
  and a1(c,w0,w1);
  or O1(w0,a,b);
endmodule  //cct module
```

```
module cct (c,a,b);
  output c;
  input a,b;
  or O1(w0,a,b);
  not n1(w1,b);
  and a1(c,w0,w1);
endmodule  //cct module
```

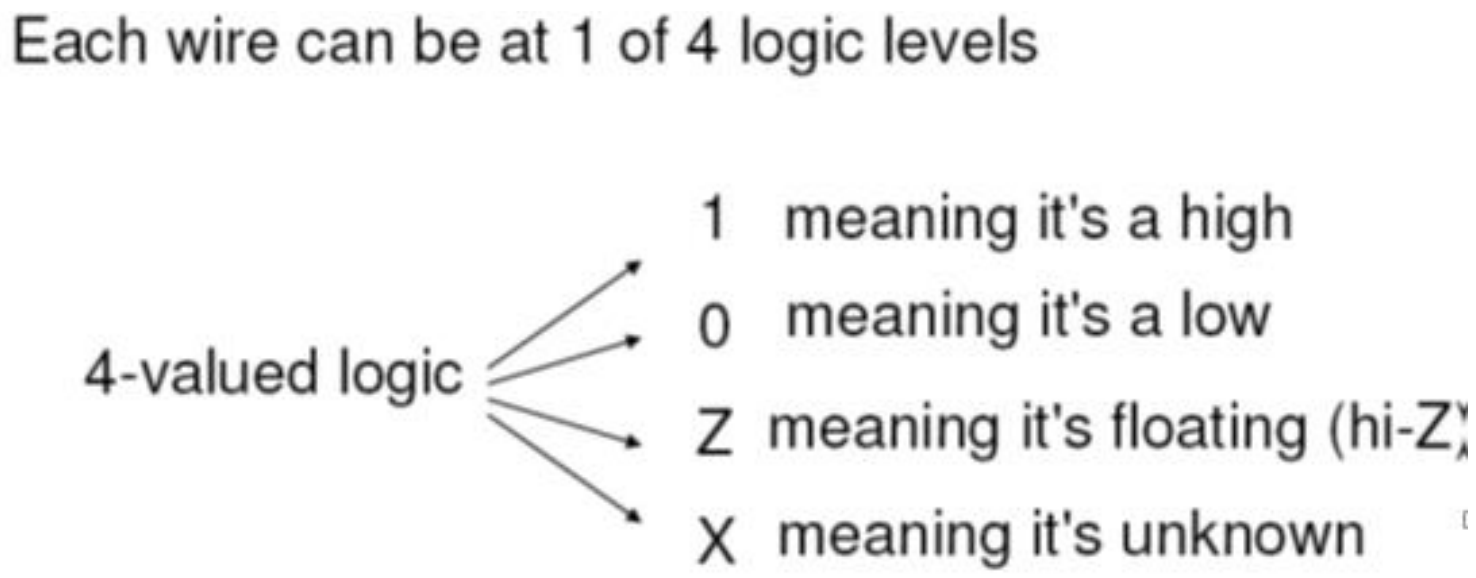
Data Types



- wire
  - Represents a physical connection, connects two points .
  - Does not have a storage so it is used for designing combinational logic, (this kind of logic can not store a value).
  - Default data type is wire (if you declare a variable without specifying reg or wire, it will be a 1-bit wide wire).

- Reg
  - is a variable with a storage. If the register is assigned a value, it remains until it is assigned another value.
  - it can be used for modeling both combinational and sequential logic.
  - Reg data type can be driven from initial and always block.

Wire and Reg value levels



Each wire can be at 1 of 4 logic levels

4-valued logic

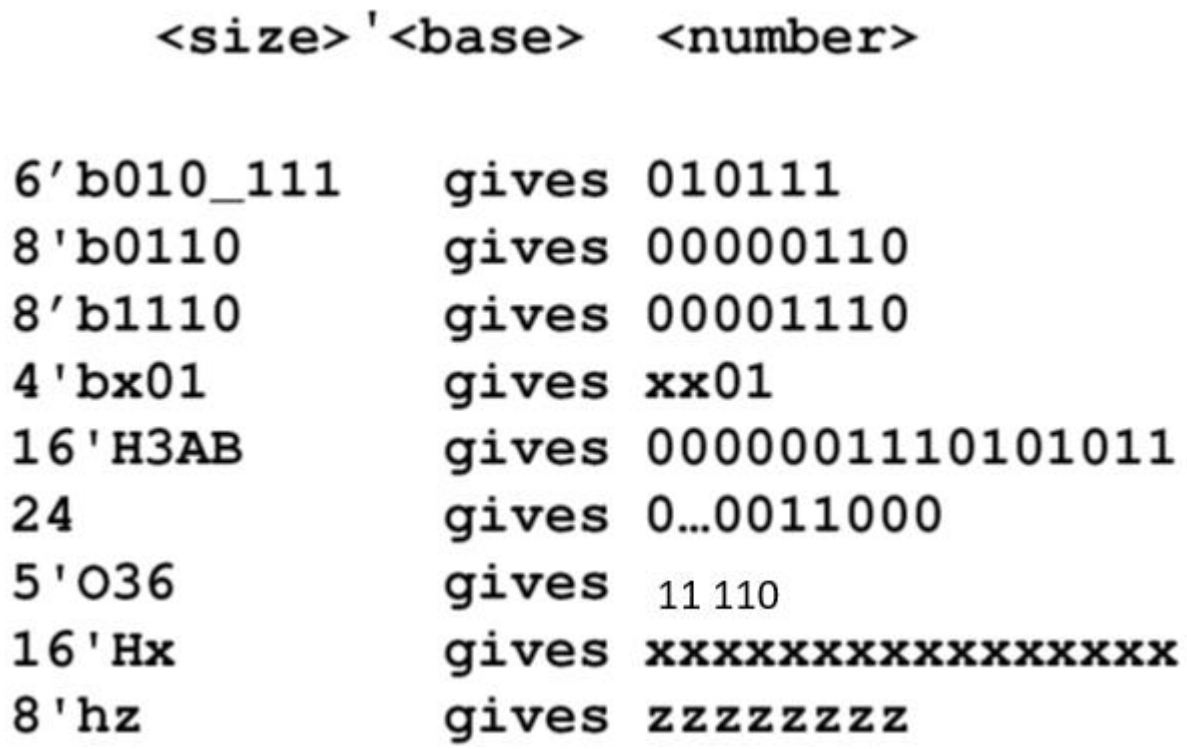
1	meaning it's a high
0	meaning it's a low
Z	meaning it's floating (hi-Z
X	meaning it's unknown

Constants

```
<size>'<base>  <number>
```

number of *bits*	b or B Binary	number in selected base
	o or O Octal	may use _ as a spacer
	h or H Hex	0-extended with 0 or 1 msb
	d or D Decimal	x-extended with x msb
		z-extended with z msb

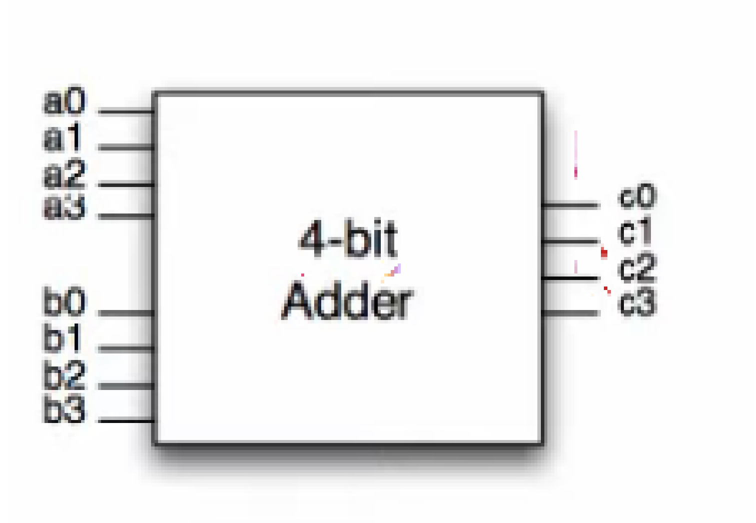
Constants

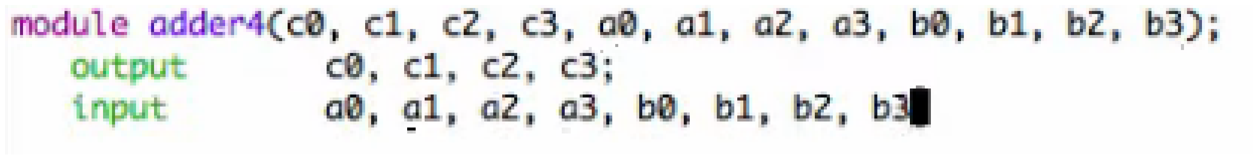


```
<size>'<base> <number>
```

6’b010_111	gives	010111
8'b0110	gives	00000110
8’b1110	gives	00001110
4'bx01	gives	xx01
16'H3AB	gives	0000001110101011
24	gives	0…0011000
5'O36	gives	11 110
16'Hx	gives	xxxxxxxxxxxxxxxx
8'hz	gives	zzzzzzzz

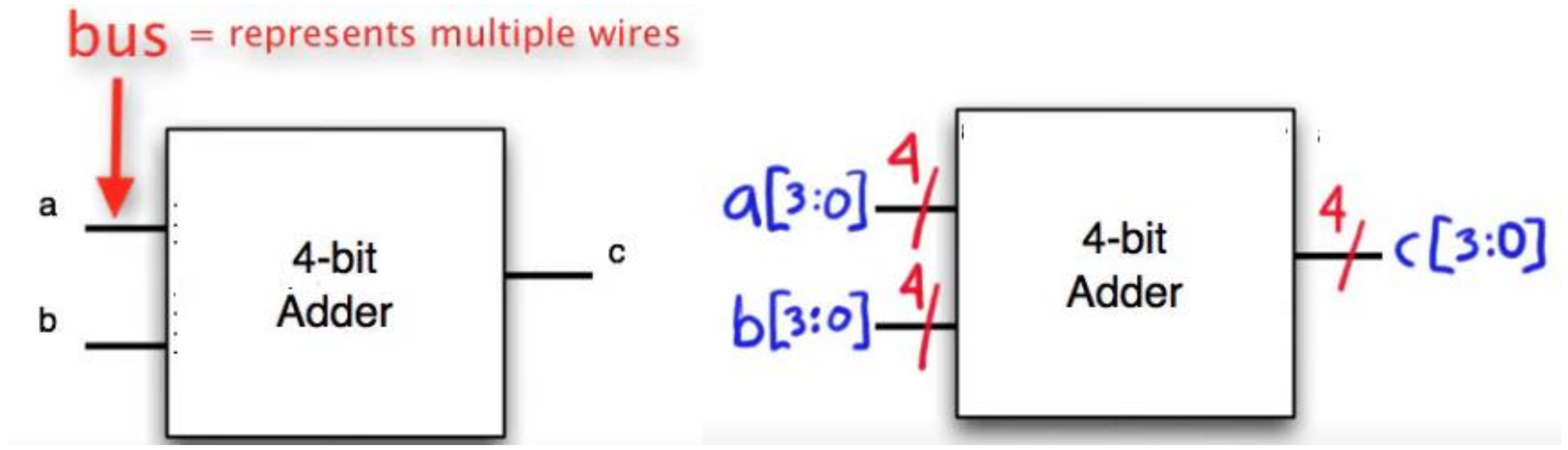
Bus notation

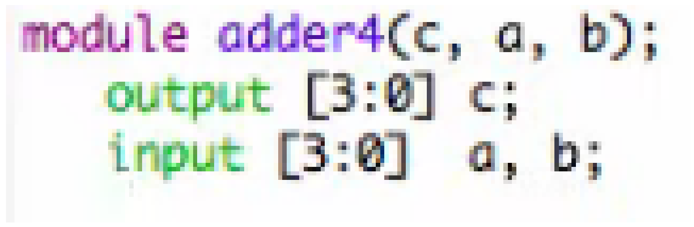




```
module adder4(c0, c1, c2, c3, a0, a1, a2, a3, b0, b1, b2, b3);
   output      c0, c1, c2, c3;
   input       a0, a1, a2, a3, b0, b1, b2, b3
```

Bus notation

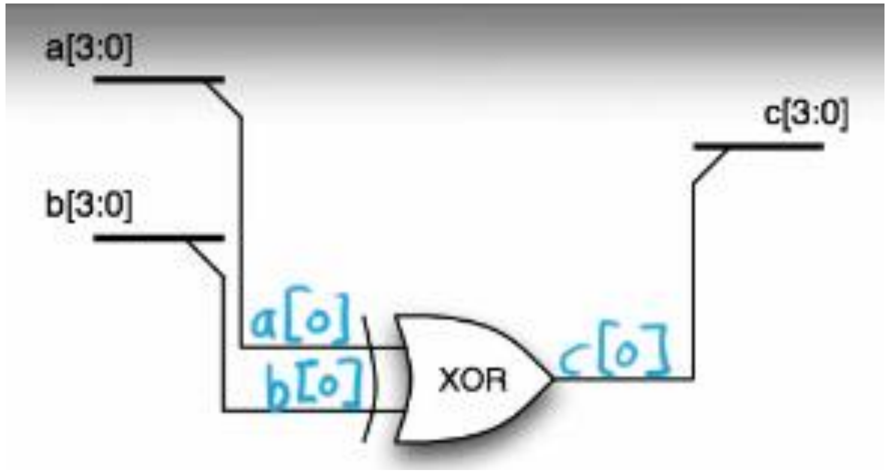




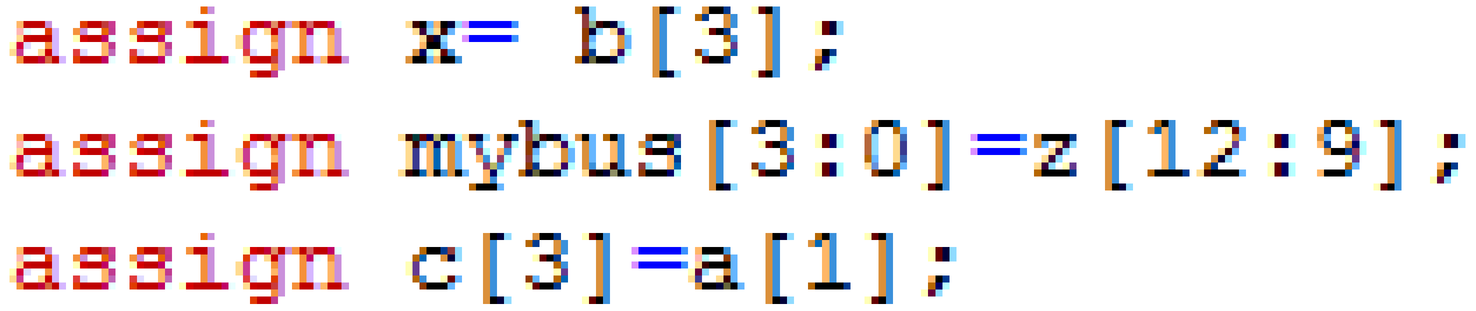
```
module adder4(c, a, b);
   output [3:0] c;
   input [3:0]  a, b;
```

```
module adder4(c, a, b);
   output [3:0] c;
   input  [3:0] a, b;

   xor  lsb_xor(c[0], a[0], b[0]);
```

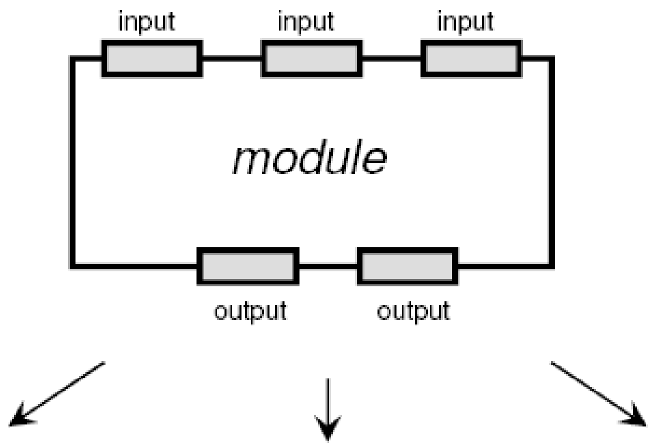


Direct assignment



```
assign x= b[3];
assign mybus[3:0]=z[12:9];
assign c[3]=a[1];
```

Modeling Types In Verilog



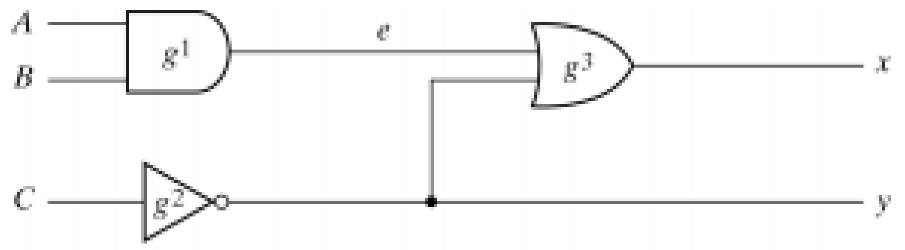
Model with submodules and gates
=
Structural

Model with always and initial blocks
=
Behavioral Procedural

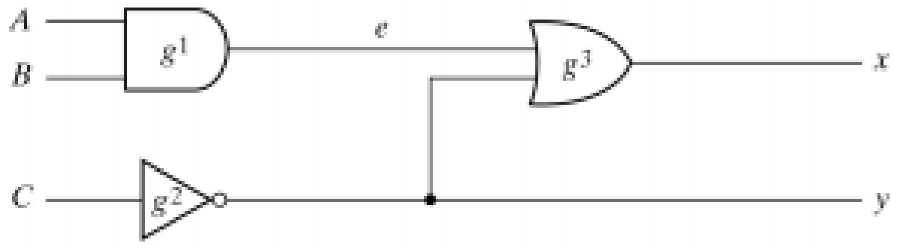
Model with assign statements
=
Behavioral Dataflow

Example of structural (gate instantiation)

```
module smpl_circuit(A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and g1(e,A,B);
  not g2(y,C);
  or g3(x,e,y);
endmodule
```



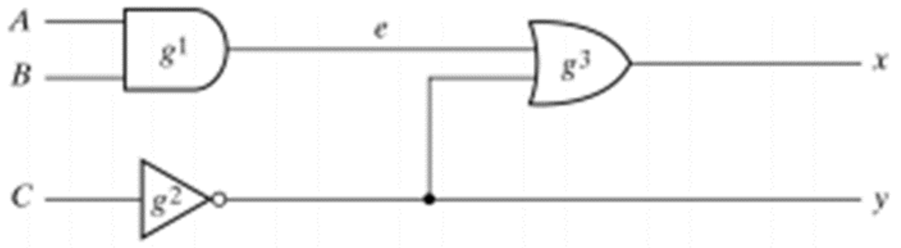
Example of continuous assignment (Dataflow modeling type)



```
module circuit_bln (x,y,A,B,C,D);
input A,B,C,D;
output x,y;
  assign x = A | (B & C) | (~B & C);
  assign y = (~B & C) | (B & ~C & ~D);
endmodule
```

Example of Procedural modeling type

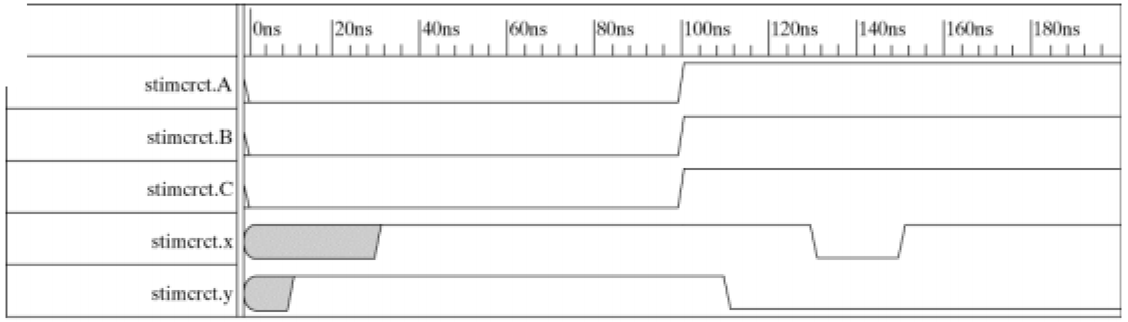
```
module circuit_bln (x,y,A,B,C,D);
input A,B,C,D;
output x,y;
reg x,y;
always@ (*)
begin
  x = A | (B & C) | (~B & C);
  y = (~B & C) | (B & ~C & ~D);
end
endmodule
```



Logic simulation: generates waveforms

```
module smpl_circuit(A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and g1(e,A,B);
  not g2(y,C);
  or g3(x,e,y);
endmodule
```

- Detect errors before fabrication


0ns
20ns
40ns
60ns
80ns
100ns
120ns
140ns
160ns
180ns
stimcrct.A
stimcrct.B
stimcrct.C
stimcrct.x
stimcrct.y