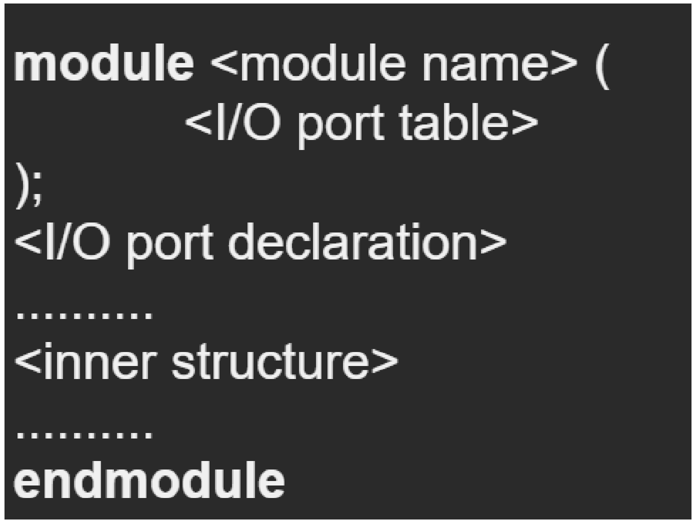
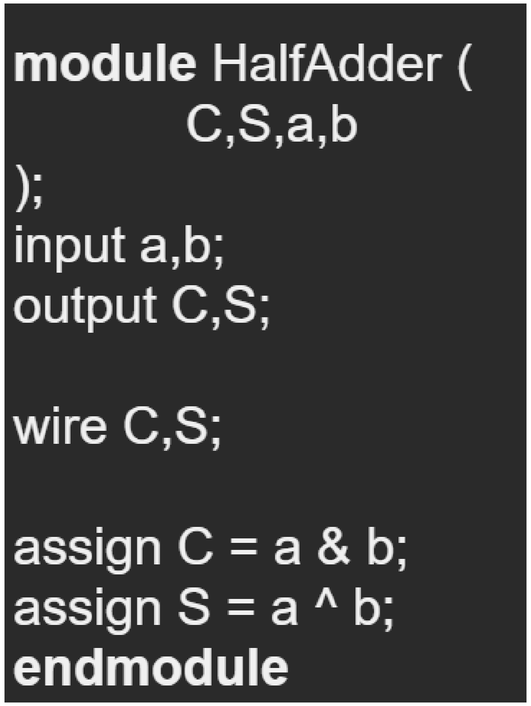
Fundamental of Verilog

Module



```
module <module name> (
            <I/O port table>
);
<I/O port declaration>
..........
<inner structure>
..........
endmodule
```

Structure



```
module HalfAdder (
            C,S,a,b
);
input a,b;
output C,S;

wire C,S;

assign C = a & b;
assign S = a ^ b;
endmodule
```

Example

Module Instantiations

```
<module name> <instance name> (port1, port2, ...);
```

Structure

```
module FullAdder(
        C,S,a,b,cin
);
input a,b,cin;
output C,S;

wire C,S,w1,w2,w3;

HalfAdder HA1 (.C(w1), .S(w2), .a(a), b(b));
HalfAdder HA2 (.C(w3), .S(S), .a(w2), b(cin));
assign C = w1 | w3;

endmodule
```

Example

Data Types (Nets and Registers)

Nets (wire)

- Represent connections between things
- Used in combination logic (assign statement or module instantiations)

Registers (reg)

- Represent data storage
- Used in sequential logic (initial block or always block)

Data Types (Vectors)

Represent as Bus or Multi-bit Memory

Bus

- wire [31:0] busName;

Multi-bit Memory

- reg [31:0] memName;

Data Types (Array and Memory)

<type> <length> <name> <array length>

Example: an array with 16 x 1-bit

- reg array1 [15:0];

Example: a memory with 1024 x 8-bits

- reg [7:0] mem1KB [1023:0];

Data Types (Numbers)

<bit>’<base><value>

Example: ‘12’ in 4 bits

Binary:	4’b1100
Octal:	4’o14
Decimal:	4’d12
Hex:	4’h0C

Gate-level Modeling

<gate> [name] (out, in1, in2, …);

- and, or, nand, nor, xor, xnor

<gate> [name] (out1, out2, …, input);

- buf, not

<gate> [name] (out, in, control);

- bufif1, bufif0, notif1, notif0

Example:

- and and1 (out, a1, a2);
- not not1 (out, a);

Dataflow Modeling

assign <out> = [in1] [operation] [in2];

- continuous assignment
- output must be ‘wire’ type
- combination logic

Example:

- assign AorB = A | B;
- assign {Carry, Sum} = A + B;

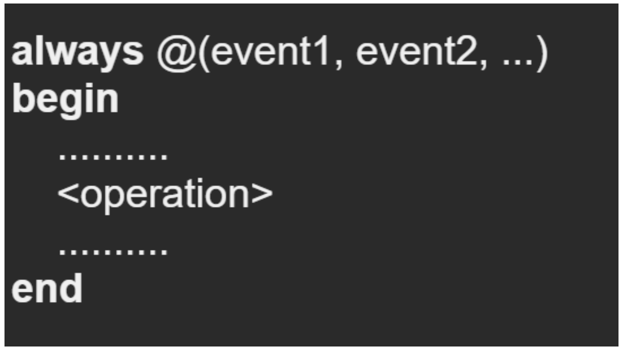
Behavioral Modeling (Procedural Constructs)

'initial' or 'always' procedural statements

- output must be 'reg' type

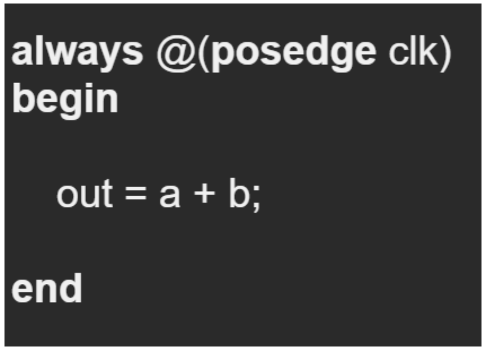
'initial' : only execute once

'always' : execute every event triggered



```
always @(event1, event2, ...)
begin
    ..........
    <operation>
    ..........
end
```

Structure: always

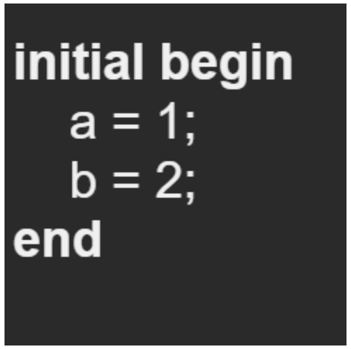


```
always @(posedge clk)
begin

    out = a + b;

end
```

Example: always



```
initial begin
    a = 1;
    b = 2;
end
```

Example: initial

Behavioral Modeling (Procedural Assignment)

‘Blocking’ procedural assignment

- Operations activated in order

‘Nonblocking’ procedural assignment

- All operations activated at same time

```
A = 1;
B = A;
C = B;

Result
A = 1;
B = 1;
C = 1;
```

Example: blocking

```
A <= 1;
B <= A;
C <= B;

Result
A = 1;
B = 0;
C = 0;
```

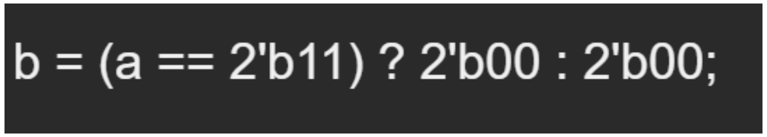
Example: nonblocking

Behavioral Modeling (Condition Statement)

Ternary operation

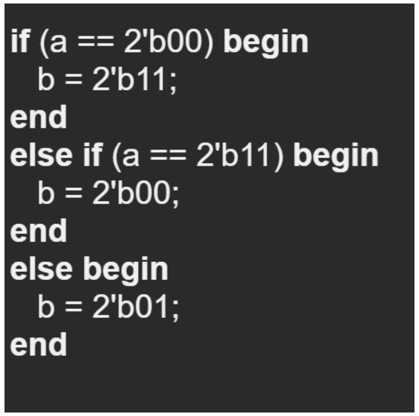
if…else

case…endcase



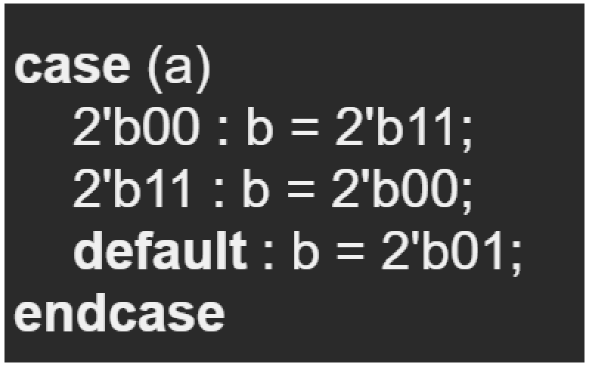
```
b = (a == 2'b11) ? 2'b00 : 2'b00;
```

Example: Ternary operation



```
if (a == 2'b00) begin
  b = 2'b11;
end
else if (a == 2'b11) begin
  b = 2'b00;
end
else begin
  b = 2'b01;
end
```

Example: if…else



```
case (a)
   2'b00 : b = 2'b11;
   2'b11 : b = 2'b00;
   default : b = 2'b01;
endcase
```

Example: case…endcase

Disclaimer