Verilog Basics
- Behavioral Model

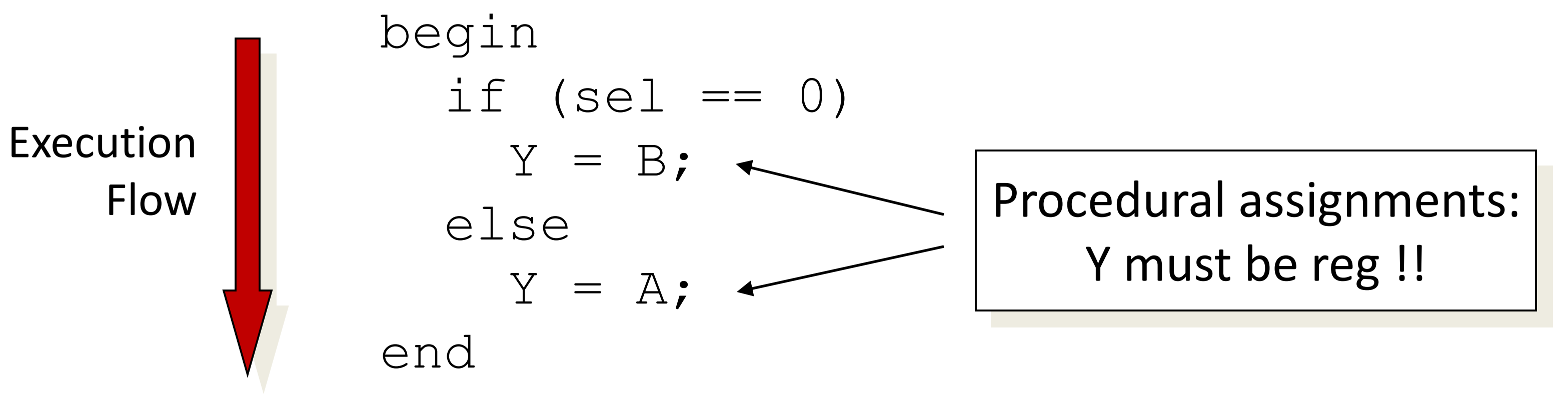
Original by Thanasis Oikonomou
Modified by Ing-Chao Lin

Outline

- Procedures
  - Initial
  - Always
- Procedural Statements
  - if-else
  - case
  - for
- Mixed Model

Behavioral Model - Procedures (i)

- always and initial
- Procedures = sections of code that we know they execute sequentially
- Procedural statements = statements inside a procedure (they execute sequentially)
- e.g. another 2-to-1 mux implem:



```
begin
  if (sel == 0)
    Y = B;
  else
    Y = A;
end
```

Procedural assignments:
Y must be reg !!

Behavioral Model - Procedures (ii)

- Modules can contain any number of procedures
- Procedures execute in parallel (in respect to each other) and ..
- .. can be expressed in two types of blocks:
  - initial $\rightarrow$ they execute only once
  - always $\rightarrow$ they execute for ever (until simulation finishes)

“Initial” Blocks

- Start execution at sim time zero and finish when their last statement executes

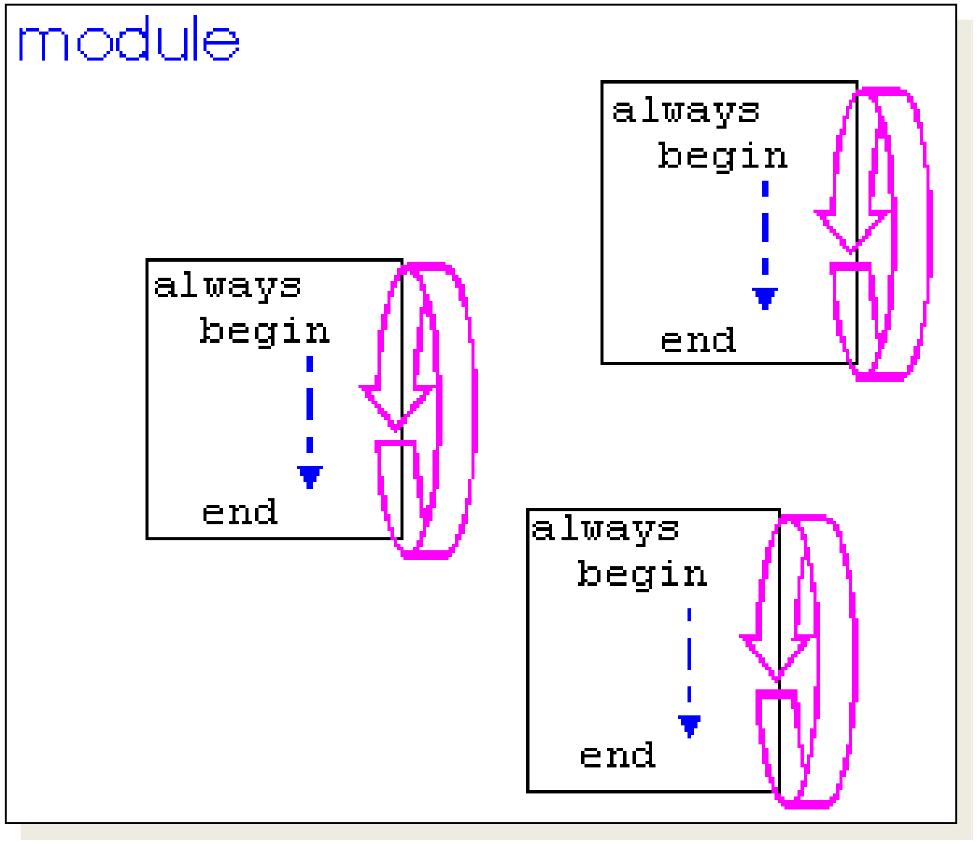
```
module first;

initial
   $display(“I’m first”);

initial begin
   #50;
   $display(“Really?”);
end

endmodule
```

Will be displayed at sim time 0

Will be displayed at sim time 50

“Always” Blocks

- Start execution at sim time zero and continue until sim finishes



Events

- @

```
always @(signal1 or signal2 or ..) begin
    ..
    end
```

execution triggers every time any signal changes

```
always @(posedge clk) begin
    ..
    end
```

execution triggers every time clk changes from *0* to *1*

```
always @(negedge clk) begin
    ..
    end
```

execution triggers every time clk changes from *1* to *0*

Examples

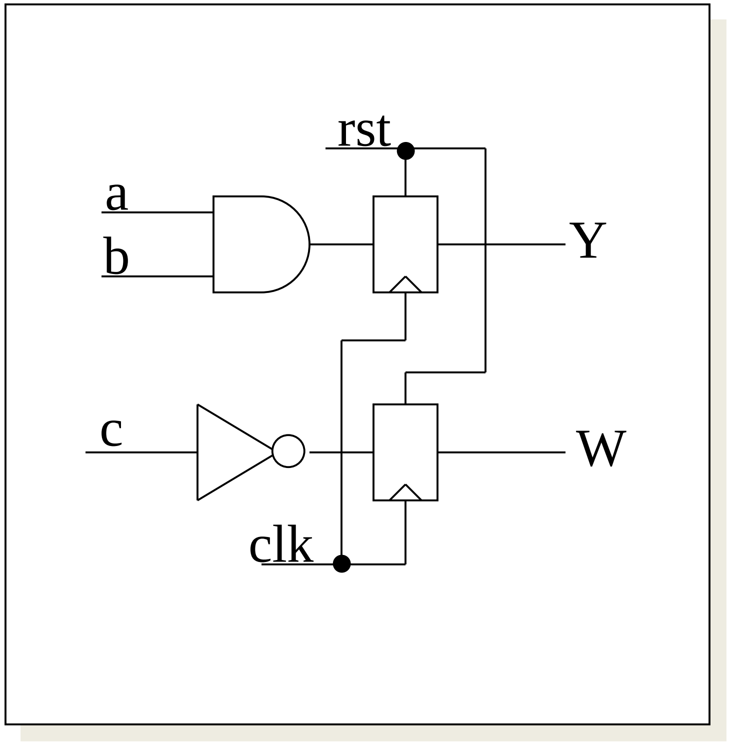
- Half adder implem

```
module half_adder(S, C, A, B);
output S, C;
input A, B;

reg S,C;
wire A, B;

always @(A or B) begin
  S = A ^ B;
  C = A && B;
  end

endmodule
```

- Behavioral edge-triggered DFF implem

```
module dff(Q, D, Clk);
output Q;
input D, Clk;

reg Q;
wire D, Clk;

always @(posedge Clk)
  Q = D;

endmodule
```

Example



```
always @(posedge rst or posedge clk)
begin
   if (rst) begin
      Y = 0;
      W = 0;
   end
   else begin
      Y = a & b;
      W = ~c;
   end
end
```

Procedural Assignments

- Blocking Assignment (use =)
- Nonblocking Assignment (use <=)

```
//  Blocking assignments
initial begin
a = #10 1'b1;//  The simulator assigns 1 to a at time 10
b = #20 1'b0;//  The simulator assigns 0 to b at time 30
c = #40 1'b1;//  The simulator assigns 1 to c at time 70
end

//  Nonblocking assignments
initial begin
d <= #10 1'b1;//  The simulator assigns 1 to d at time 10
e <= #20 1'b0;//  The simulator assigns 0 to e at time 20
f <= #40 1'b1;//  The simulator assigns 1 to f at time 40
end
```

Procedural Statements: if

```
if (expr1) begin
  true_stmt1;
end

else if (expr2) begin
  true_stmt2;
  ..
end

else begin
  def_stmt;
end
```

E.g. 4-to-1 mux:

```
module mux4_1(out, in, sel);
output out;
input [3:0] in;
input [1:0] sel;

reg out;
wire [3:0] in;
wire [1:0] sel;

always @(in or sel)
  if (sel == 0)
    out = in[0];
  else if (sel == 1)
    out = in[1];
  else if (sel == 2)
    out = in[2];
  else
    out = in[3];

endmodule
```

Procedural Statements: case

```
case (expr)

item_1: stmt1;
item_2: stmt2;
..
default:
   def_stmt;

endcase
```

E.g. 4-to-1 mux:

```
module mux4_1(out, in, sel);
output out;
input [3:0] in;
input [1:0] sel;

reg out;
wire [3:0] in;
wire [1:0] sel;

always @(in or sel)
   case (sel)
   0: out = in[0];
   1: out = in[1];
   2: out = in[2];
   3: out = in[3];
   default: out = in[0];
   endcase
endmodule
```

Procedural Statements: for

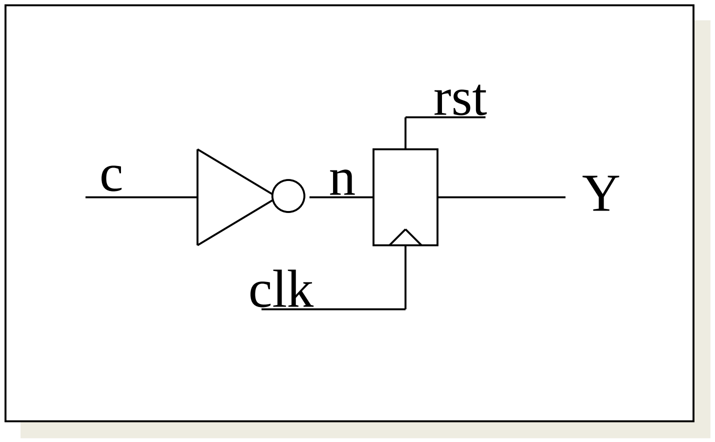
for (init_assignment; cond;step_assignment)
    stmt;

E.g.

```
module count(sum, Y, clk);
output [3:0] sum;
input clk, Y;

reg [3:0] Y [0:2];
integer i;

assign sum = Y[0] + Y[1] + Y[2];

always @(posedge clk)
  for (i = 0; i < 3; i = i + 1)
    Y[i] = Y[i] + 1;

endmodule
```

Mixed Model

Code that contains various both structure and behavioral styles



```
Module simple(Y, c, clk, rst);
Output Y;
Input c, clk, rst;

Reg Y;
Wire c, clk, rst;
Wire n;

Not(n, c); // gate-level

Always @(res or posedge clk)
  if (rst)
    Y = 0;
  else
    Y = n;
endmodule
```

Backup slides