HARDWARE DESCRIPTION LANGUAGE FOR DIGITAL DESIGN

數位設計硬體描述語言

Behavioral Modeling

Materials partly adapted from “Digital System Designs and Practices Using Verilog HDL and FPGAs,” M.B. Lin.

NCKU EE
LPHP Lab

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

OUTLINE

- Verilog Basics
- Structural Modeling
- Dataflow Modeling
- Behavioral Modeling

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

BEHAVIORAL MODELING

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

PROCEDURAL CONSTRUCTS

- initial block
  - Executes precisely once during the simulation
  - Starts at simulation time 0

```
reg  x, y, z;
initial  begin       // complex statement
         x = 1`b0;  y = 1`b1;  z = 1`b0;
#10      x = 1`b1;  y = 1`b1;  z = 1`b1;
end
initial x = 1`b0;  // single statement
```

- always block
  - Starts at simulation time 0
  - Executes continuously during simulation

```
reg  clock;
initial  clock = 1`b0;

always  #5 clock = ~clock;
```

NCKU EE
LPHP Lab

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

PROCEDURAL ASSIGNMENTS

- Syntax

```
variable_lvalue  = [timing_control] expression
variable_lvalue <= [timing_control] expression
[timing_control] variable_lvalue  = expression
[timing_control] variable_lvalue <= expression
```

- Two types
  - blocking
  - nonblocking

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

BLOCKING ASSIGNMENTS

- Are executed in the specified order
- Use the “=“ operator

```
// blocking assignments
initial begin
   x = #5 1'b0;    // at time 5
   y = #3 1'b1;    // at time 8
   z = #6 1'b0;    // at time 14
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

BLOCKING ASSIGNMENTS

```
output reg  [3:0] sum;
output reg  c_out;
reg    [3:0]  t;

always @(x, y, c_in) begin
    t = y ^ {4{c_in}};
    {c_out, sum} = x + t + c_in;
end
```

Q: What is wrong with: t = y ^ c_in ?
Q: Does a reg variable correspond to a memory element to be synthesized?

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

NONBLOCKING ASSIGNMENTS

- Use the <= operator
- Used to model several concurrent data transfers

```
reg  x, y, z;
// nonblocking assignments
initial begin
   x <= #5 1'b0;  // at time 5
   y <= #3 1'b1;  // at time 3
   z <= #6 1'b0;  // at time 6
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

NONBLOCKING ASSIGNMENTS

```
input  clk;
input  din;
output reg [3:0] qout;
// a 4-bit shift register
always @(posedge clk)
    qout <= {din, qout[3:1]};  // Right shift
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

RACE CONDITIONS

```
// using blocking assignment statements
always @(posedge clock)   // has race condition
        x = y;
always @(posedge clock)
        y = x;

// using nonblocking assignment statements
always @(posedge clock)   // has no race condition
        x <= y;
always @(posedge clock)
        y <= x;
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

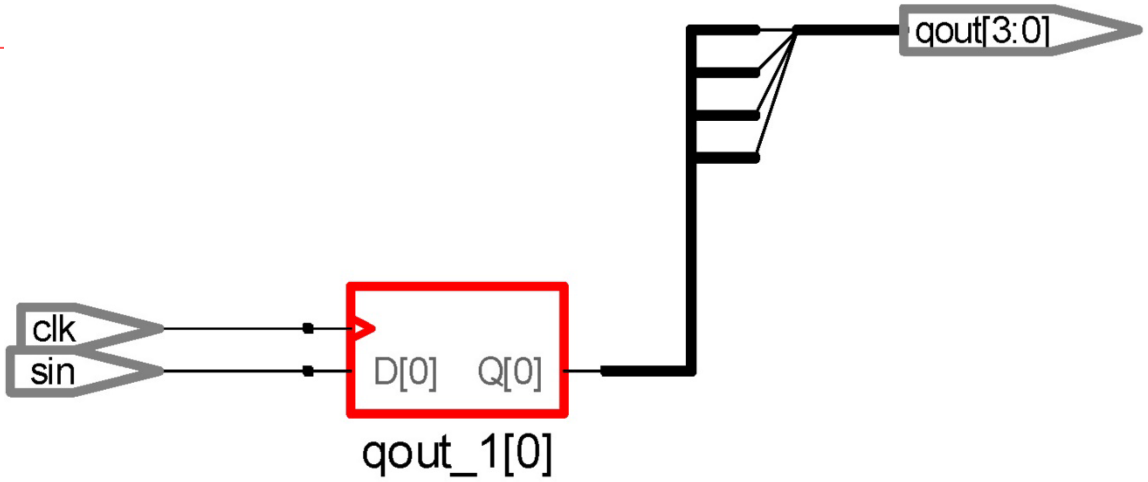
EXECUTION OF NONBLOCKING STATEMENTS

- Read
- Evaluate and schedule
- Assign

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

BLOCKING VS. NONBLOCKING ASSIGNMENTS

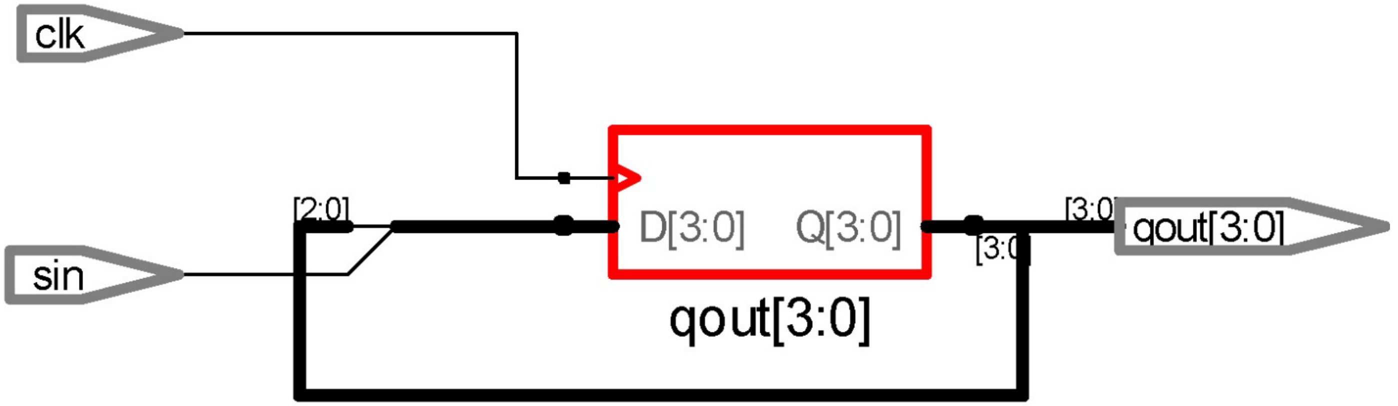
```
// A 4-bit shift register
always @(posedge clk) begin
        qout[0] = sin;
        qout[1] = qout[0];
        qout[2] = qout[1];
        qout[3] = qout[2];
end
```



The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

Blocking vs. Nonblocking Assignments

```
// A 4-bit shift register
always @(posedge clk) begin
    qout[0] <= sin;
    qout[1] <= qout[0];
    qout[2] <= qout[1];
    qout[3] <= qout[2];
end
```



The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

BLOCKING VS. NONBLOCKING ASSIGNMENTS

- Suppose that count is 1 and finish is 0 before entering the always block

```
always @(posedge clk) begin: block_a
    count = count – 1;
    if (count == 0) finish = 1;
end
```

Result: finish = 1.
(Different from that of gate-level.)

```
always @(posedge clk) begin: block_b
    count <= count – 1;
    if (count == 0) finish <= 1;
end
```

Result: finish = 0.
(Same as that of gate-level.)

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

CODING STYLES

- blocking operators (=)
  - combinational logic
- nonblocking operators (<=)
  - sequential logic

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

DELAY TIMING CONTROL

- Regular delay control
  - Defers the execution of the entire statement

```
reg  x, y;
integer count;
   #25         y <= ~x;                  // at time 25
   #15  count <= count  + 1;       // at time 40
```

- Intra assignment delay control
  - Defers the assignment to the left-hand-side variable

```
       y = #25  ~x;               // assign to y at time 25
count = #15 count  + 1;    // assign to count at time 40
```

```
       y <= #25  ~x;              // assign to y at time 25
count <= #15 count  + 1;  // assign to count at time 15
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

EVENT TIMING CONTROL

- Edge-triggered event control
  - Named event control
  - Event or control
- Level-sensitive event control

Q: What is an event?

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

EDGE-TRIGGERED EVENT CONTROL

- Edge-triggered event control
  - @(posedge clock)
  - @(negedge clock)

```
always @(posedge clock)  begin
      reg1 <= #25  in_1;
      reg2 <= @(negedge clock) in_2 ^ in_3;
      reg3 <= in_1;
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

NAMED EVENT CONTROL

- A named event

```
event  received_data;                          // declare
always @(posedge clock)
     if (last_byte) -> received_data;  // trigger
always @(received_data)                     // recognize
     begin ... end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

EVENT OR CONTROL

- Event or control
  - or
  - ,
  - * or (*) means any signals

```
always @(posedge clock or negedge reset_n)
    if (!reset_n) q <= 1`b0;
    else          q <= d;
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

LEVEL-SENSITIVE EVENT CONTROL

- Level-sensitive event control

```
always
     wait  (count_enable) count = count –1 ;

always
     wait  (count_enable) #10 count = count –1 ;
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

IF…ELSE STATEMENT

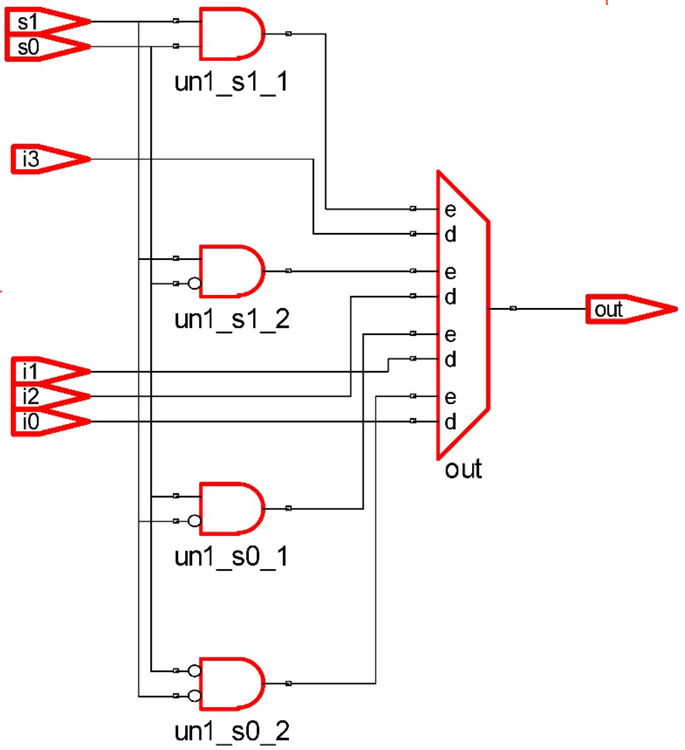
- Syntax

```
if (<expression>)  true_statement ;

if (<expression>) true_statement;
else false_statement;

if (<expression1>) true_statement1;
else if (<expression2>) true_statement2;
else false_statement;
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

IF…ELSE STATEMENT

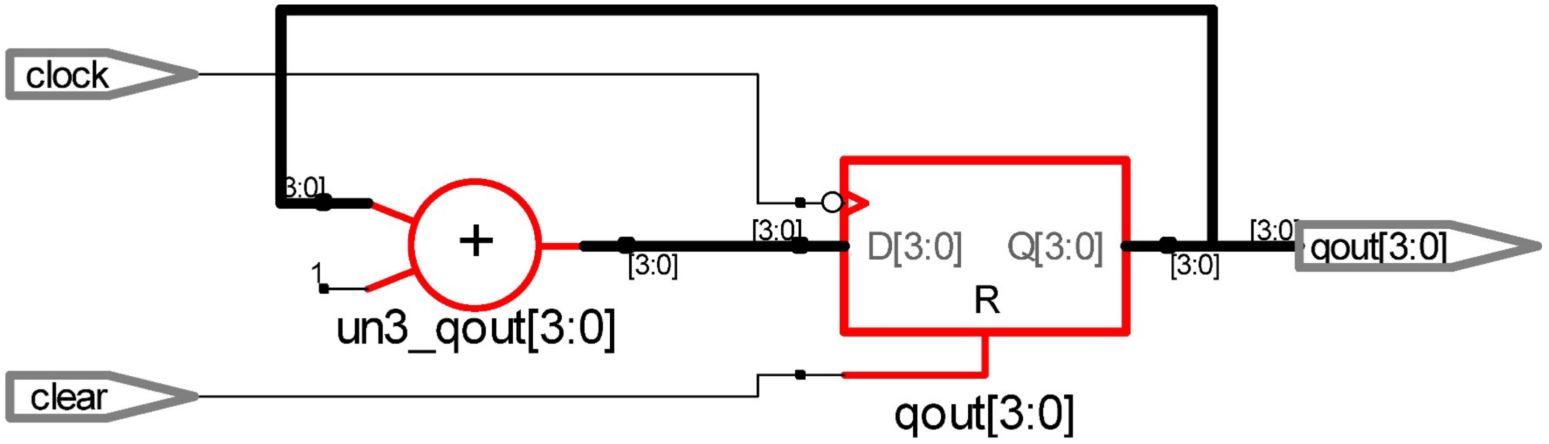
```
// using if…else statement
always @(*) begin
    if (s1) begin
        if (s0)  out = i3; else out = i2; end
    else begin
        if (s0)  out = i1; else out = i0; end
end
```



The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

A SIMPLE 4-BIT COUNTER

```
// the body of the 4-bit counter.
always @(negedge clock or posedge clear)
    if (clear)
        qout <= 4'd0;
    else
        qout <= (qout + 1) ;   // qout = (qout + 1) % 16;
```



NCKU EE
LPHP Lab

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

CASE (CASEX/CASEZ) STATEMENT

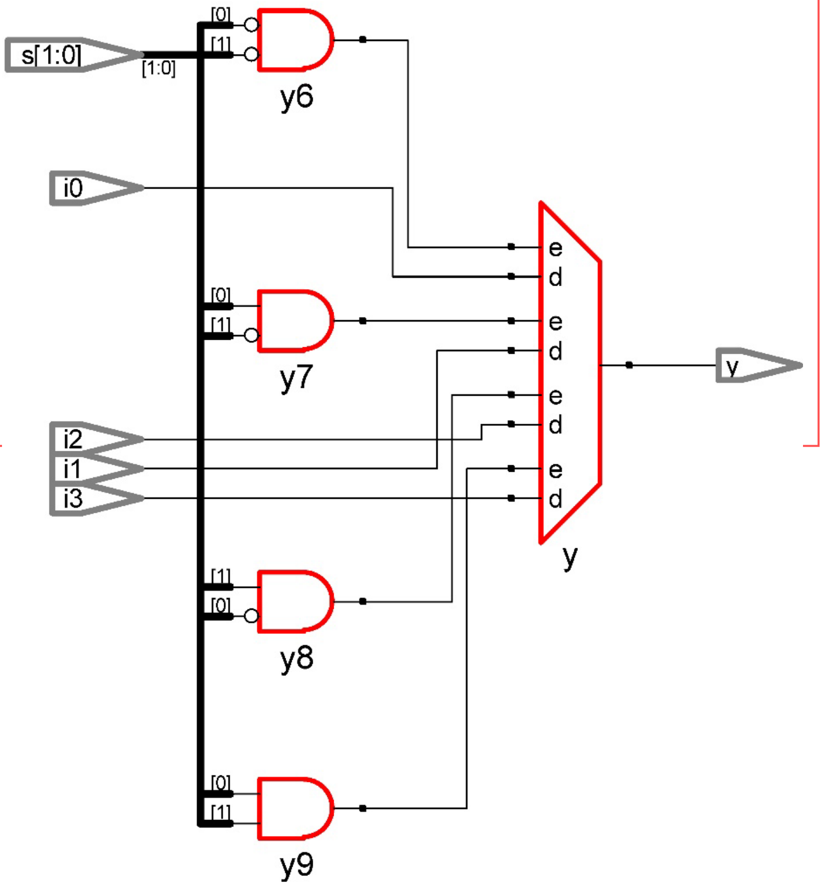
- Syntax

```
case (case_expression)
   case_item1 {,case_item1}:
 procedural_statement1
   case_item2 {,case_item2}:
 procedural_statement2
   ...
   case_itemn {,case_itemn}:
 procedural_statementn
    [default: procedural_statement]
endcase
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

A 4-TO-1 MUX EXAMPLE

```
always @(I0 or I1 or I2 or I3 or S)
  case (S)
    2'b00: Y = I0;
    2'b01: Y = I1;
    2'b10: Y = I2;
    2'b11: Y = I3;
  endcase
```



Q: Model a 3-to-1 multiplexer.

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

CASEX AND CASEZ STATEMENTS

- casex and casez statements
  - Compare only non-x or z positions
- casez treats all z values as don't cares
- casex treats all x and z values as don't cares

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

CASEX AND CASEZ STATEMENTS

```
// count the trailing zeros in a nibble
always @(data)
   casex (data)
       4'bxxx1: out = 0;
       4'bxx10: out = 1;
       4'bx100: out = 2;
       4'b1000: out = 3;
       4'b0000: out = 4;
       default:  out = 3'b111;
   endcase
```

Q: Is the default statement necessary?

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

FOR LOOP

- Syntax

```
for (init_expr; condition_expr; update_expr) statement;
```

```
init_expr;
while (condition_expr) begin
    statement;
    update_expr;
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

FOR LOOP

```
// count the zeros in a byte
integer i;
always @(data) begin
   out = 0;
   for (i = 0; i <= 7; i = i + 1) // simple condition
        if (data[i] == 0) out = out + 1;
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

FOR LOOP

```
// count the trailing zeros in a byte
integer i;
always @(data) begin
   out = 0;
   for (i = 0; data[i] == 0 && i <= 7; i = i + 1)
         out = out + 1;
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

REPEAT LOOP

- Syntax
  repeat (counter_expr) statement;

```
i = 0;
repeat (32) begin
    state[i] = 0;
    i = i + 1;
end
repeat (cycles) begin
     @(posedge clock) buffer[i] <= data;
     i <= i + 1;
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

FOREVER LOOP

- Syntax

  forever statement;

```
initial  begin
    clock <= 0;
    forever begin
        #10 clock <= 1;
        #5   clock <= 0;
    end
end
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

THE FOREVER LOOP STRUCTURE

```
reg clock, x, y;

initial
    forever @(posedge clock) x <= y;
```

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.