HARDWARE DESCRIPTION LANGUAGE FOR DIGITAL DESIGN

數位設計硬體描述語言

Structural Modeling

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

NCKU EE
LPHP Lab

OUTLINE

- Verilog Basics
- Structural Modeling
- Dataflow Modeling

DATAFLOW MODELING

BASIC STATEMENTS

- Continuous Assignments

```
assign #5 {c_out, sum[3:0]} = a[3:0] + b[3:0] + c_in;
```

- Variations – Net declaration assignments
  - Only one net declaration assignment per net

```
wire out;                    // regular continuous assignment
assign out = in1 & in2;
wire  out = in1 & in2;     // net declaration assignment
```

- Variations – Implicit net declarations

```
wire in1, in2;
assign out = in1 & in2; // out is declared automatically
```

EXPRESSIONS

- expression = operators + operands
- Operators

Arithmetic
+: add
- : subtract
* : multiply
/ : divide
% : modulus
**: exponent

Shift
<< : left shift
>> : right shift
<<< : arithmetic left shift
>>>: arithmetic right shift

Bitwise
~ : NOT
&: AND
| : OR
^: XOR
~^, ^~: XNOR

case equality
===: equality
!==: inequality

Equality
==: equality
!=: inequality

Reduction
&: AND
|: OR
~&: NAND
~|: NOR
^: XOR
~^, ^~: XNOR

Logical
&&: AND
|| : OR
! : NOT

Relational
>: greater than
<: less than
>= : greater than or equal
<=: less than or equal

Miscellaneous
{ , }: concatenation
{c{ }}: replication
? : conditional

EXPRESSIONS

- Operands
  - constants
  - parameters
  - nets
  - variables (reg, integer, time, real, realtime)
  - bit-select
  - part-sel
  - array element
  - function calls

7

OPERANDS

THREE WAYS OF SPECIFYING DELAYS

- Regular assignment delay: inertial delay (default)

```
wire in1, in2, out;
assign #10 out = in1 & in2;
```

- Net declaration assignment delay

```
wire #10 out = in1 & in2;
```

- Net declaration delay

```
wire #10 out;
assign out = in1 & in2;
```

OPERANDS: CONSTANTS

- integer
  - Simple decimal form: -123; 12345
  - Base format notation: 16'habcd; 2009; 4'sb1001 // -7
- real
  - Decimal notation: 1.5 ; .3 //illegal; 1295.872
  - Scientific notation: 15E12; 32E-6; 26.176_45_e-12
- string: a sequence of characters within double quotes("")
  - only be specified in a single line
  - a character in an 8-bit ASCII code

DATA TYPES

- Nets: any hardware connection points

- Variables: any data storage elements
  - reg
  - integer
  - time
  - real
  - realtime

DATA TYPES: THE reg VARIABLE

- Hold a value between assignments
- May be used to model hardware registers
- Examples

```
reg a, b;
reg [7:0] data_a;
reg [0:7] data_b;
reg signed [7:0] d;
```

DATA TYPES: THE integer VARIABLE

- Contains integer values
- Has at least 32 bits
- Is treated as a signed reg variable with the LSB being bit 0

```
integer i, j;
integer data[7:0];
```

DATA TYPES: THE time VARIABLE

- Used for storing and manipulating simulation time
- Used in conjunction with the $time system task
- Only unsigned value and at least 64 bits, with the LSB being bit 0

```
time events;
time current_time;
```

DATA TYPES: THE real and realtime VARIABLE

- Cannot use range declaration
- Defaulted to zero (0.0)

```
real events;
realtime current_time;
```

DATA TYPES: THE real and realtime VARIABLE

- Cannot use range declaration
- Defaulted to zero (0.0)

```
real events;
realtime current_time;
```

OPERANDS: VECTORS

- A vector = multiple bit width
  - [high:low] or [low:high]
  - The leftmost bit is the MSB
  - include nets and reg data types

- A scalar = 1-bit vector

OPERANDS: BIT-SELECT AND PART-SELECT

- integer and time are allowed
- real and realtime are not allowed
- Constant part select: data_bus[3:0], bus[3]
- Variable part select:

  [<starting_bit>+:width]: data_bus[8+:8]

  [<starting_bit>-:width]: data_bus[15-:8]

OPERANDS: ARRAY AND MEMORY ELEMENTS

- All net and variable data types are allowed
- An array element can be a scalar or a vector

```
wire a[3:0];
reg  d[7:0];
wire [7:0]  x[3:0];
reg  [31:0] y[15:0];
integer states [3:0];
time    current[5:0];
```

OPERANDS: MEMORY

- Memory
  - model ROM, RAM, or register files
- Reference
  - a whole word or
  - a portion of a word of memory

```
reg [7:0] mema [7:0];
reg [7:0] memb [3:0][3:0];
wire sum [7:0][3:0];

mema[4][3]
mema[5][7:4]
memb[3][1][1:0]
sum[5][0]
```

OPERATORS

BITWISE AND ARITHMETIC OPERATORS

- Bitwise:
  - A bit-by-bit operation
  - A z is treated as x
  - The shorter operand is zero-extended

Symbol	Operation
~	Bitwise negation
&	Bitwise and
|	Bitwise or
^	Bitwise exclusive or
~^, ^~	Bitwise exclusive nor

- Arithmetic:
  - The result is x if any operand bit has a value x
  - Negative numbers are represented as 2's complement

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponent (power)
%	Modulus

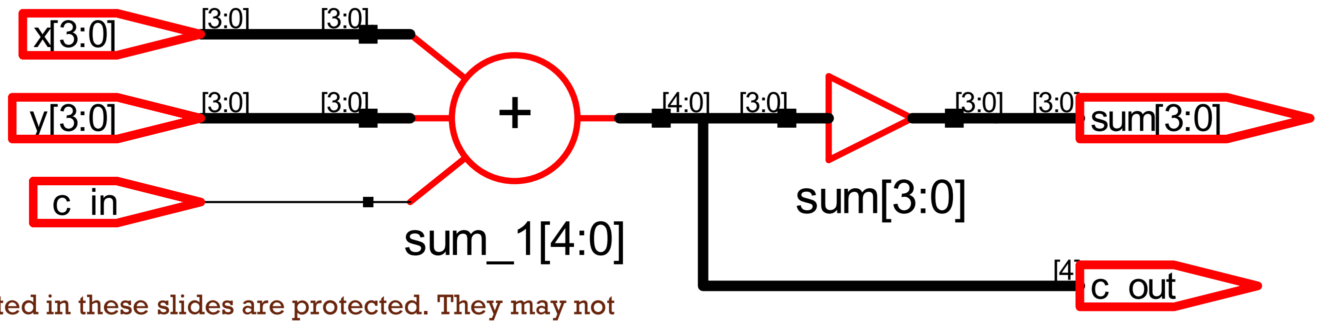
CONCATENATION/REPLICATION OPERATORS

- Concatenation operator
  y = {a, b[0], c[1]};
- Replication operator
  y = {a, {4{b[0]}}, c[1]};

Symbol	Operation
{ , }	Concatenation
{const_expr{ }}	Replication

- An Example: a 4-bit full adder

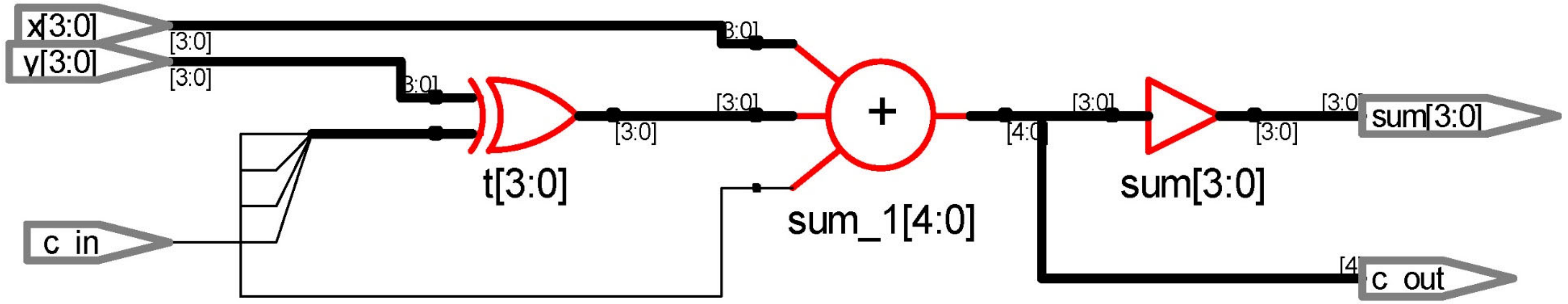
```
// specify the function of a 4-bit adder
   assign {c_out, sum} = x + y + c_in;
```



A 4-BIT TWO'S COMPLEMENT ADDER

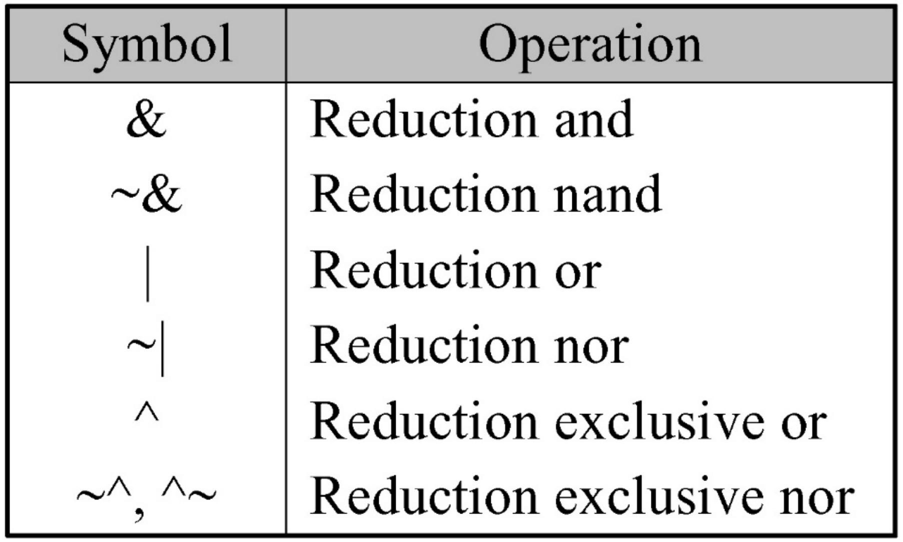
- How do we do 2's complement addition?

```
// specify the function of a two's complement adder
    assign t = y ^ {4{c_in}};
    assign {c_out, sum} = x + t + c_in;
```



REDUCTION OPERATORS

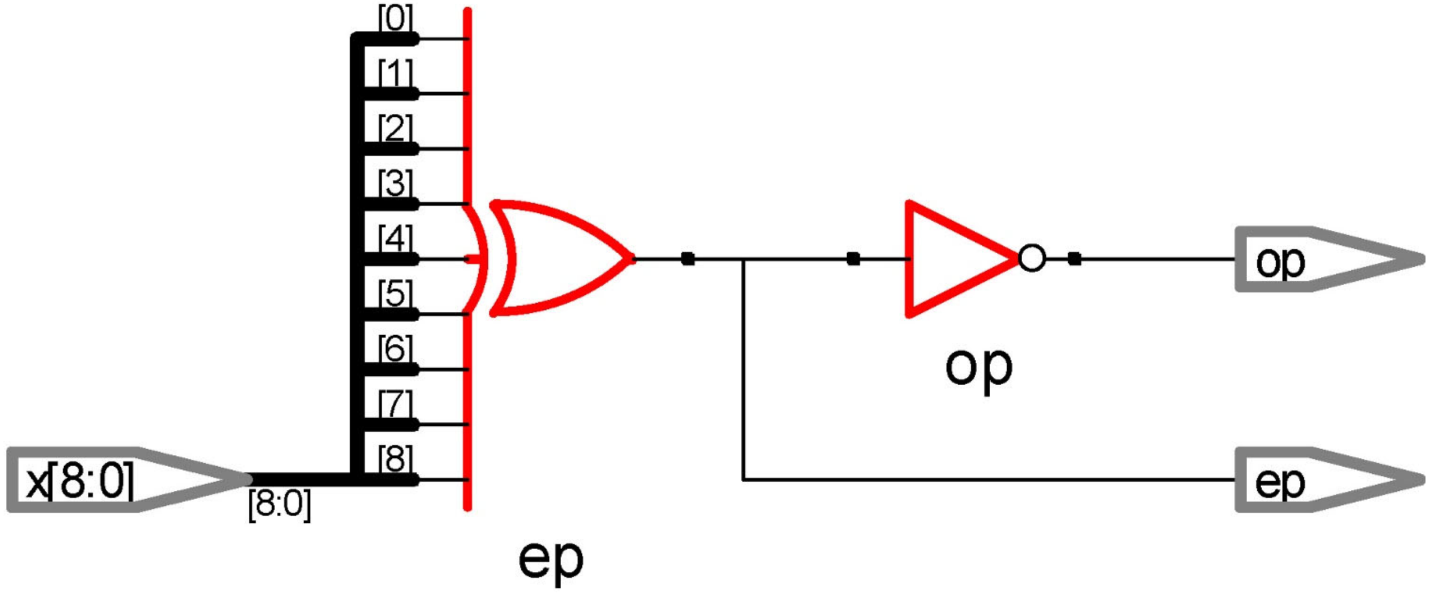
- Perform only on one vector operand
  - Carry out a bit-wise operation
  - Yield a 1-bit result
  - Work bit by bit from right to left



Symbol	Operation
&	Reduction and
~&	Reduction nand
|	Reduction or
~|	Reduction nor
^	Reduction exclusive or
~^, ^~	Reduction exclusive nor

A 9-bit Parity Generator

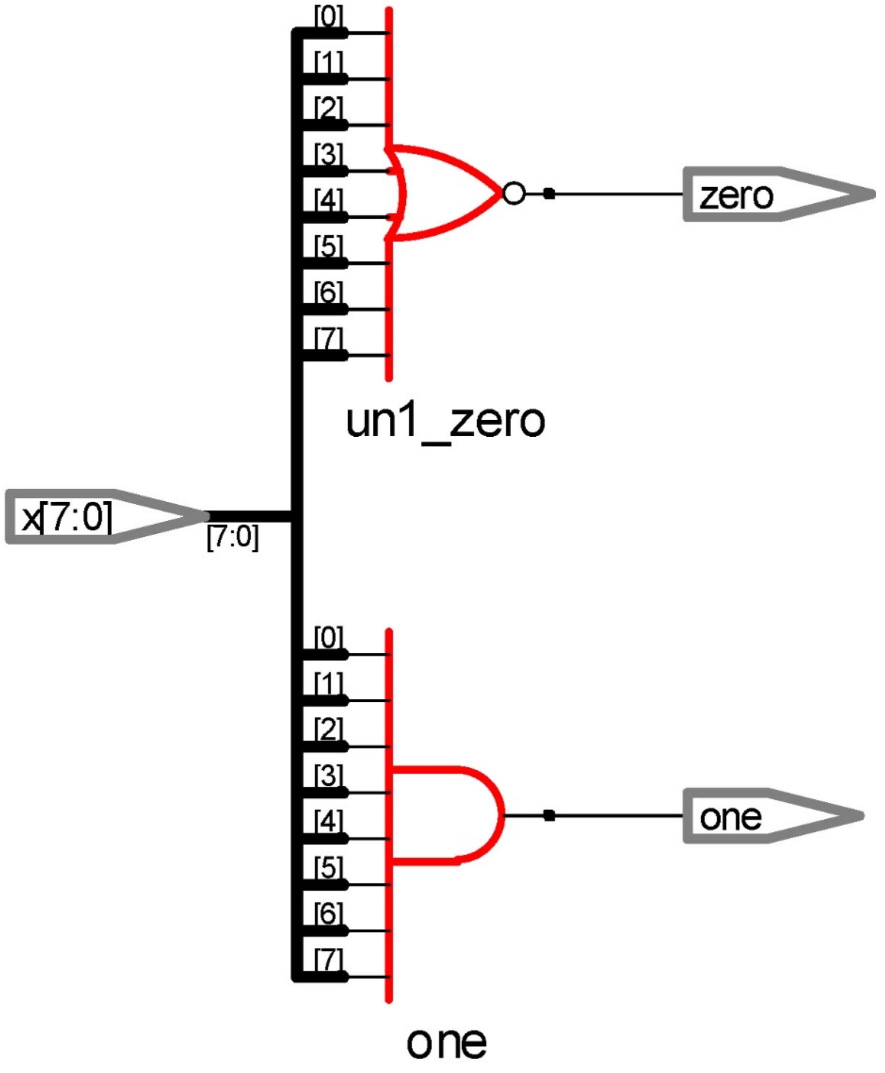
```
// dataflow modeling using reduction operator
assign ep = ^x;     // even parity generator
assign op = ~ep;  // odd parity generator
```



NCKU EE
LPHP Lab

AN ALL-BIT-ZERO/ONE DETECTOR

```
// dataflow modeling
   assign zero = ~(|x);  // all-bit zero
   assign one = &x;      // all-bit one
```



LOGICAL AND RELATIONAL OPERATORS

- Logical:
  - Always evaluate to a 1-bit value, 0, 1, or x
  - The result is x (a false condition) if any operand bit is x or z

Symbol	Operation
!	Logical negation
&&	Logical and
||	Logical or

- Relational:
  - The result is 1 if the expression is true and 0 if the expression is false
  - Return x if any operand bit is x or z

Symbol	Operation
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

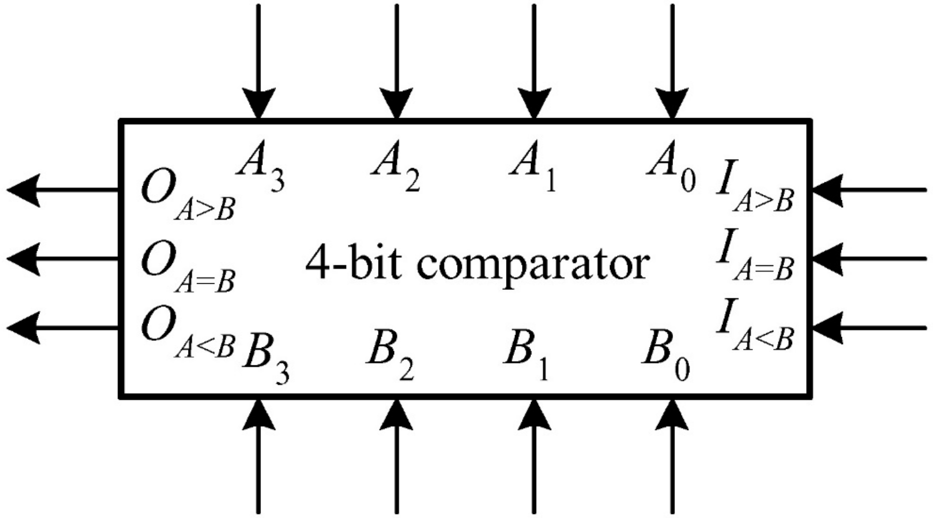
EQUALITY OPERATORS

- Compare the two operands bit by bit
  - The shorter operand is zero-extended
  - Return 1 if the expression is true and 0 if the expression is false
- The operators (==, !=)
  - yield an x if any operand bit is x or z
- The operators (===, !==)
  - yield a 1 if the two operands exactly match
  - 0 if the two operands not exactly match

Symbol	Operation
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality

NCKU EE
LPHP Lab

A 4-BIT MAGNITUDE COMPARATOR



```
// dataflow modeling using relation operators
   assign Oaeqb = (a == b) && (Iaeqb == 1);                 //  =
   assign Oagtb  = (a > b) || ((a == b)&& (Iagtb == 1));  // >
   assign Oaltb   = (a < b) || ((a == b)&& (Ialtb == 1));   // <
```

SHIFT OPERATORS

- Logical shift operators
- Arithmetic shift operators

Symbol	Operation
>>	Logical right shift
<<	Logical left shift
>>>	Arithmetic right shift
<<<	Arithmetic left shift

```
input  signed [3:0] x;
output [3:0] y;
output signed [3:0] z;

assign y = x >> 1;
assign z = x >>> 1;
```

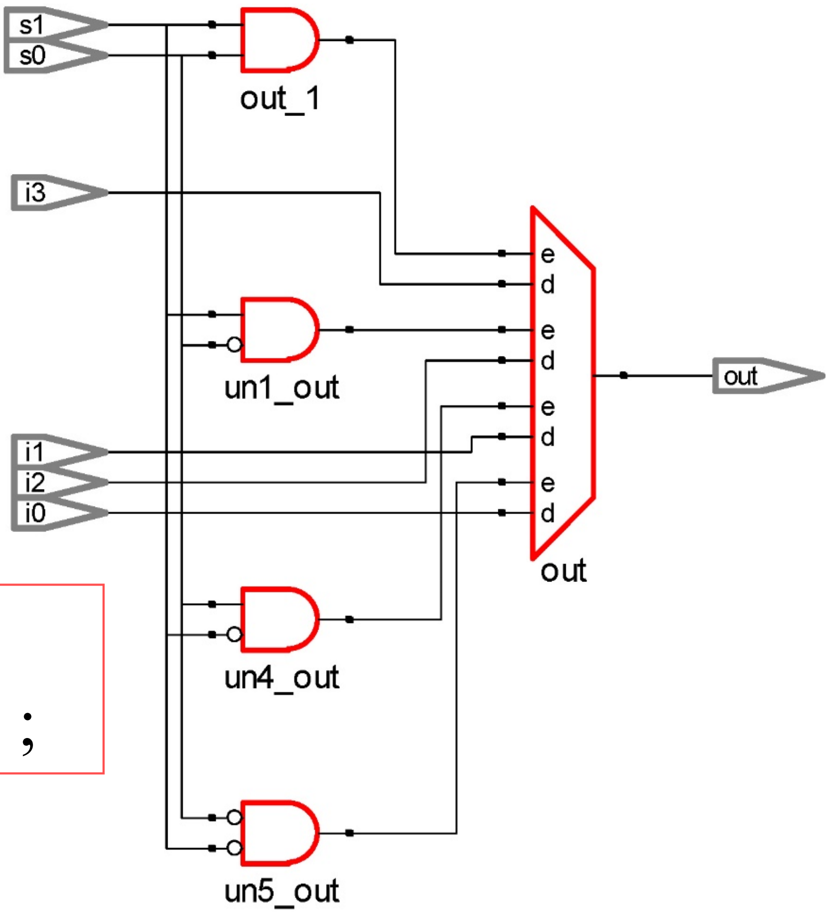
CONDITIONAL OPERATORS

- Usage: condition_expr ? true_expr: false_expr;
  - If condition_expr = x or z: the result = true_expr & false_expr (0 and 0 gets 0, 1 and 1 gets 1, others gets x)
- A simple example,

```
assign out = selection ? in_1: in_0;
```

- A 4-to-1 MUX

```
// using conditional operator (?:)
assign out = s1 ? ( s0 ? i3 : i2) : (s0 ? i1 : i0) ;
```



s1
s0
out_1
i3
un1_out
i1
i2
i0
e
d
e
d
e
d
e
d
out
out
un4_out
un5_out