Verilog Basics
- Dataflow Modeling

Outline

- Dataflow Modeling
- Continuous Assignements
  - Arithmetic Operators
  - Logical Operators
  - Relational Operators
  - Equality Operators
  - Bitwise Operators
  - Reduction Operators
  - Shift Operators
  - Concatenation Operator
  - Conditional Operator
- Operator Precedence

Continuous Assignment

- Dataflow modeling allows a circuit to be designed in terms of the data flow between registers and design process data
- Syntax:

```
assign #delay <id> = <expr>;
assign out = in & in2;
```

optional

net type !!

implicit

- Where to write them:
  - inside a module or outside procedures
- Properties:
  - they all execute in parallel
  - are order independent
  - are continuously active

Operator

Operator Type	Operator Symbol	Operation Performed	Number of Operands
Arithmetic	*	multiply	two
	/	divide	two
	+	add	two
	-	subtract	two
	%	modulus	two
	**	power (exponent)	two
Logical	!	logical negation	one
	&&	logical and	two
	||	logical or	two
Relational	>	greater than	two
	<	less than	two
	>=	greater than or equal	two
	<=	less than or equal	two
Equality	==	equality	two
	!=	inequality	two
	===	case equality	two
	!==	case inequality	two
Bitwise	~	bitwise negation	one
	&	bitwise and	two
	|	bitwise or	two
	^	bitwise xor	two
	^~ or ~^	bitwise xnor	two
Reduction	&	reduction and	one
	~&	reduction nand	one
	|	reduction or	one
	~|	reduction nor	one
	^	reduction xor	one
	^~ or ~^	reduction xnor	one
Shift	>>	Right shift	Two
	<<	Left shift	Two
	>>>	Arithmetic right shift	Two
	<<<	Arithmetic left shift	Two
Concatenation	{ }	Concatenation	Any number
Replication	{ { } }	Replication	Any number
Conditional	? :	Conditional	Three

Arithmetic Operators

- +, -, *, /, %, **
  - `16 % 4 => 0`
- If any operand is *x (unknown)* the result is *x*
  - `4'b101x + 4'b1011 => 4'bx`
- Negative number:
  - wires can be assigned negative but are treated as unsigned

```
wire [15:0] wA;
wire [15:0] wB;
..
assign wA = -16'd12;  // stored as 2^16-12 = 65524
assign wB = wA/3;     // evaluates to 21861
```

Not what we expected

Logical Operators

- && → logical AND
- || → logical OR
- ! → logical NOT
- Operands evaluated to ONE bit value: *0*, *1* or *x*
- Result is ONE bit value: *0*, *1* or *x*

```
A = 6;          A && B  → 1 && 0 → 0
B = 0;          A || !B → 1 || 1 → 1
C = x;          C || B  → x || 0 → x
```

but `C&&B=0`

Relational Operators

- > → greater than
- < → less than
- >= → greater or equal than
- <= → less or equal than

- Result is one bit value: *0*, *1* or *x*

```
1 > 0          → 1
3'b1x1 <= 0    → x
10 < z         → x
```

Equality Operators

==	→ logical equality	Return *0*, *1* or *x*
!=	→ logical inequality	
===	→ case equality	Return *0* or *1*
	– *Including x and z*	
!==	→ case inequality	
	– *Including x and z*	

```
4'b 1z0x ==  4'b 1z0x  → x
4'b 1z0x !=  4'b 1z0x  → x
4'b 1z0x === 4'b 1z0x  → 1
4'b 1z0x !== 4'b 1z0x  → 0
```

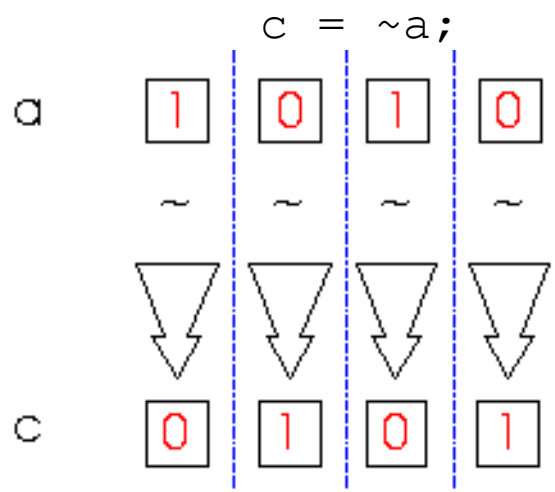
Bitwise Operators (1)

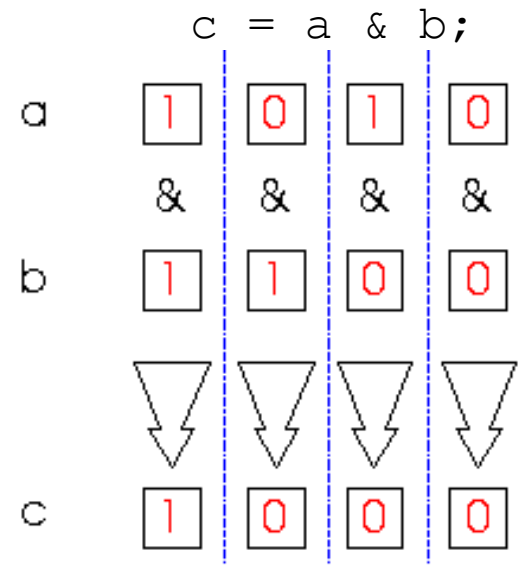
&	→ bitwise AND
|	→ bitwise OR
~	→ bitwise NOT
^	→ bitwise XOR
~^ or ^~	→ bitwise XNOR

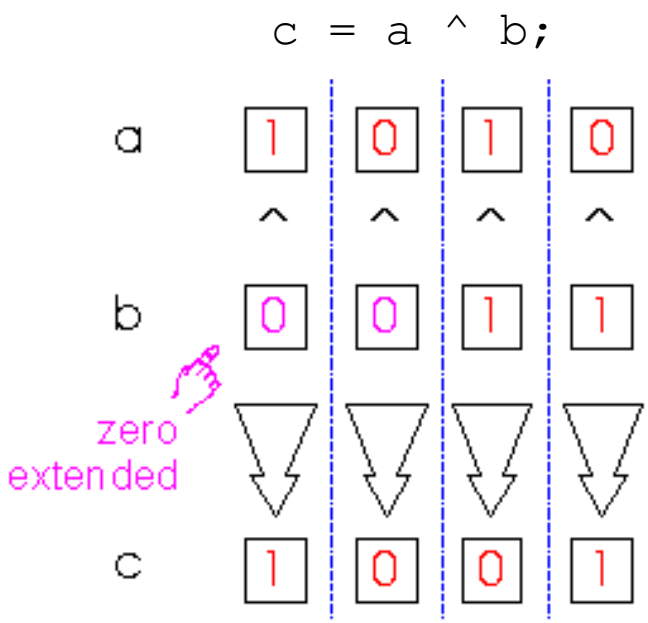
Operation on bit by bit basis

Bitwise Operators (2)

```
a = 4'b1010;
b = 4'b1100;
```








```
a = 4'b1010;
b = 2'b11;
```

Reduction Operators

`&`	→ AND
`|`	→ OR
`^`	→ XOR
`~&`	→ NAND
`~|`	→ NOR
`~^` or `^~`	→ XNOR

- Input is one multi-bit operand → Output is one single-bit result

```
a = 4'b1001;
..
c = |a; // c = 1|0|0|1 = 1
```

Shift Operators

- >> → shift right
- << → shift left

- Result is same size as first operand, **always zero filled**

```
a = 4'b1010;
...
d = a >> 2;   // d = 0010
c = a << 1;   // c = 0100
```

Concatenation Operator

- {op1, op2, ..} → concatenates op1, op2, to single number
- Operands must be sized !!

```
reg a;
reg [2:0] b, c;
..
a = 1'b 1;
b = 3'b 010;
c = 3'b 101;
catx = {a, b, c};            // catx = 1_010_101
caty = {b, 2'b11, a};        // caty = 010_11_1
catz = {b, 1};               // WRONG !!
```

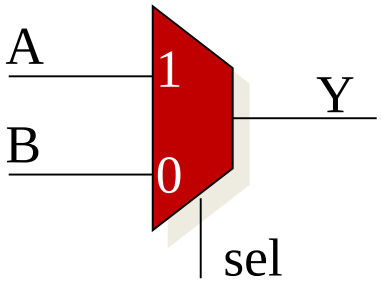
- Replication ..

```
catr = {4{a}, b, 2{c}};      // catr = 1111_010_101101
```

Conditional Operator

- `cond_expr ? true_expr : false_expr`

- Like a 2-to-1 mux ..




```
assign Y = (sel)? A : B;
```

Operator Precedence

+-!~unary	highest precedence
*/%	
+-(binary)	
<< >>	
< <= => >	
== != === !==	
& ~&	
^ ^~ ~^	
| ~|	
&&	
||	
?: conditional	lowest precedence

Use parentheses to enforce your priority

Example

```
//  4-to-1 multiplexer.
//  Port list is taken exactly from
//  the I/O diagram.
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);

//  Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;

assign out = (~s1 & ~s0 & i0)|
             (~s1 & s0 & i1) |
             (s1 & ~s0 & i2) |
             (s1 & s0 & i3) ;


endmodule
```