Dataflow Modeling

Dataflow modeling

- gate-level modeling approach works very well only for small circuits
- implements the function at a level of abstraction higher than gate level
- data flow between registers and processes data rather than instantiation of individual gates

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);

output out;
input i0, i1, i2, i3, s1, s0;

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

endmodule
```

Continuous Assignments

- most basic statement in dataflow modeling, used to drive a value onto a net.
- Continuous assignments are always active.
- assignment expression is evaluated as soon as one of the right-hand-side operands changes and the value is assigned to the left-hand-side net.
- The operands on the right-hand side can be registers or nets or function calls.

Expressions, Operators and Operands

- Expressions are constructs that combine operators and operands to produce a result.

- Operands can be constants, integers, real numbers, nets, registers, and memories or function calls.

```
c = a - b; //a and b are real operands
```

- Operators act on the operands to produce desired results

```
d1 && d2 // && is an operator on operands d1 and d2
```

Number Specification

- Sized numbers are represented as

  <size> '<base format> <number>

<size> is written in decimal and specifies the number of bits in the number.

base formats are decimal ('d or 'D), hexadecimal ('h or 'H), binary ('b or 'B) and octal ('o or 'O)

```
4'b1111 // This is a 4-bit binary number
12'habc // This is a 12-bit hexadecimal number
16'd255 // This is a 16-bit decimal number
```

Unsized numbers

```
23456 // This is a 32-bit decimal number by default
'hc3 // This is a 32-bit hexadecimal number
'o21 // This is a 32-bit octal number
```

Number Specification

x and z Values

```
12'h13x // 12-bit hex number; 4 LSB bits unknown
6'hx // This is a 6-bit hex number
32'bz // This is a 32-bit high impedance number
```

Negative numbers

```
-8'd3 //8-bit negative number stored as 2's complement of 3
4'd-2 // Illegal specification
```

Underscore characters

```
12'b1111_0000_1010 // underline characters for readability
```

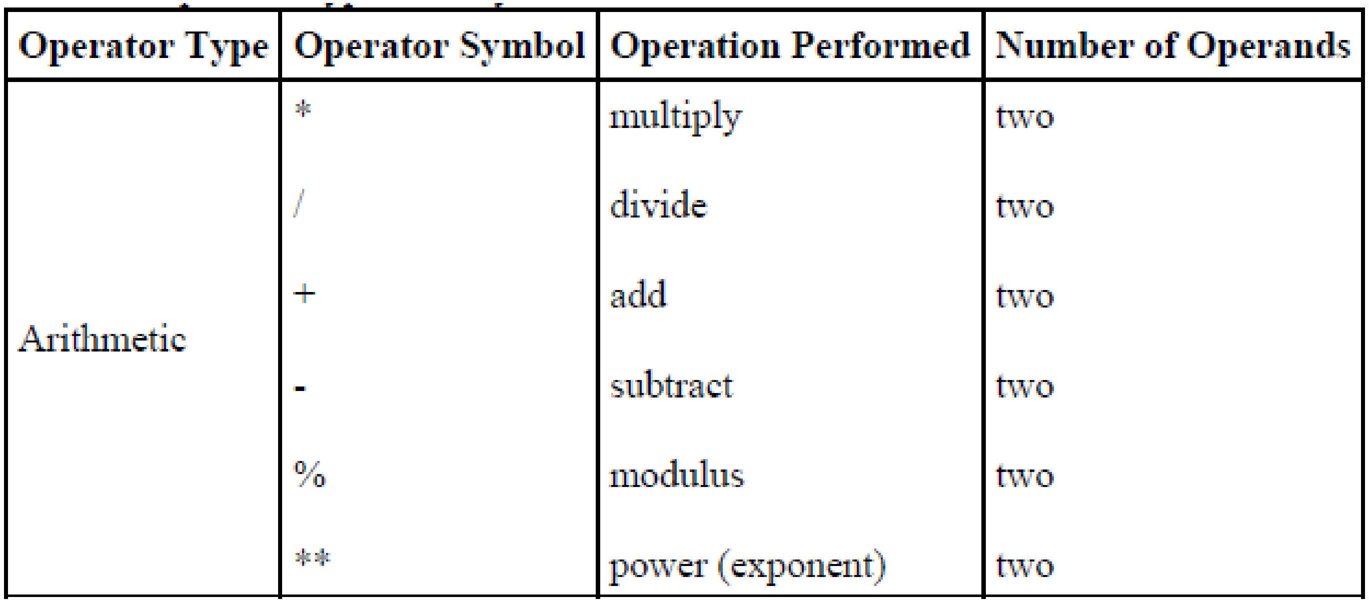
Strings

```
"Verilog HDL " // is a string
"a / b"        // is a string
```

Operator Types

- Arithmetic operators
- Logical operators
- Relational operators
- Equality operators
- Bitwise operators
- Reduction operators
- Shift operators
- Concatenation operators
- Conditional operators

Arithmetic operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Arithmetic	*	multiply	two
	/	divide	two
	+	add	two
	-	subtract	two
	%	modulus	two
	**	power (exponent)	two

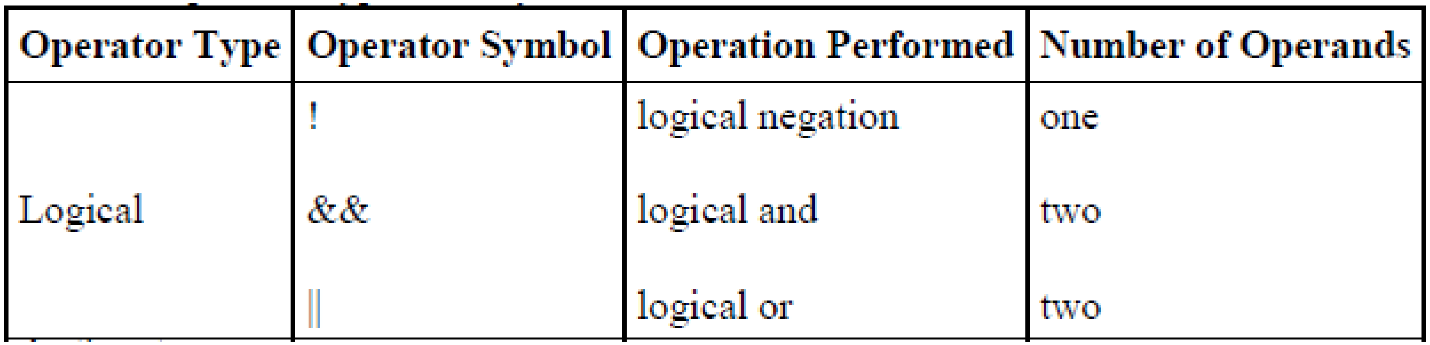
```
A = 4'b0011; B = 4'b0100; // A and B are vectors
D = 6; E = 4; F=2         // D and E are integers
A * B // Multiply A and B. Evaluates to 4'b1100
D / E // Divide D by E. Evaluates to 1. Truncates any
         fractional part.
A + B // Add A and B. Evaluates to 4'b0111
B - A // Subtract A from B. Evaluates to 4'b0001
F = E ** F; //E to the power F, yields 16
```

Arithmetic operators

- Unary operators
- The operators + and - can also work as unary operators.
- They are used to specify the positive or negative sign of the operand
- Have higher precedence than binary operators

```
-4 // Negative 4
+5 // Positive 5
```

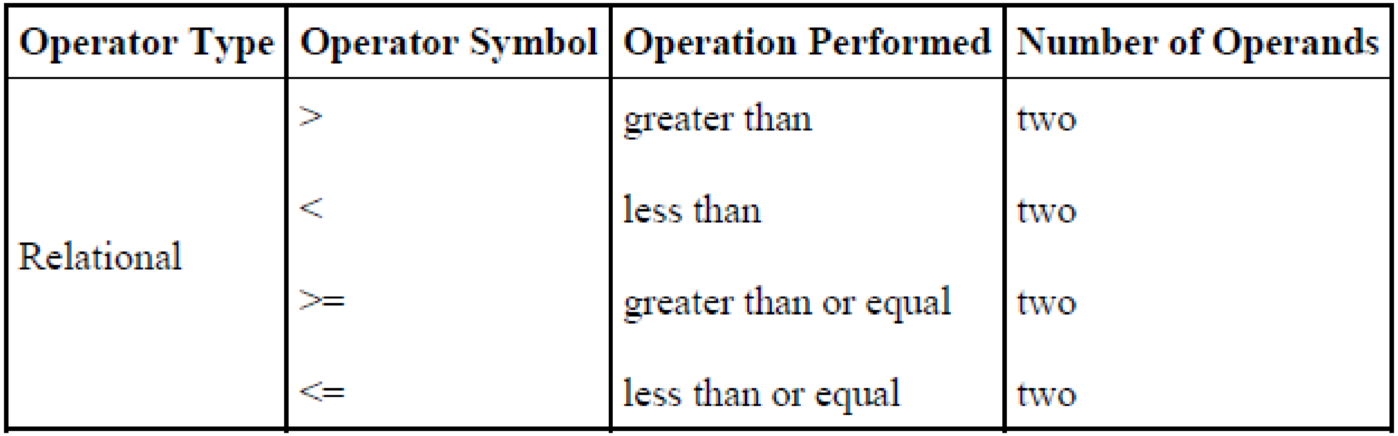
Logical operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Logical	!	logical negation	one
	&&	logical and	two
	||	logical or	two

```
A = 3; B = 0;
A && B // Evaluates to 0. Equivalent to (logical-1 && logical-0)
A || B // Evaluates to 1. Equivalent to (logical-1 || logical-0)
!A     // Evaluates to 0. Equivalent to not(logical-1)
!B     // Evaluates to 1. Equivalent to not(logical-0)

A = 2'b0x; B = 2'b10;
A && B // Evaluates to x. Equivalent to (x && logical 1)

 (a == 2) && (b == 3) // Evaluates to 0
```

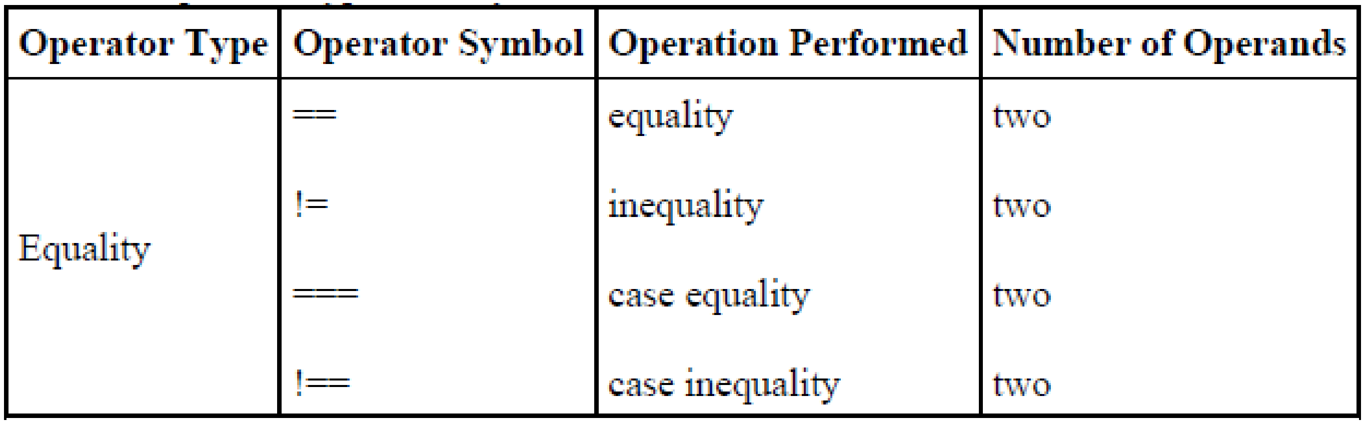
Relational operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Relational	>	greater than	two
	<	less than	two
	>=	greater than or equal	two
	<=	less than or equal	two

```
// A = 4, B = 3
// X = 4'b1010, Y = 4'b1101, Z = 4'b1xxx
A <= B // Evaluates to a logical 0

A > B // Evaluates to a logical 1

Y >= X // Evaluates to a logical 1

Y < Z // Evaluates to an x
```

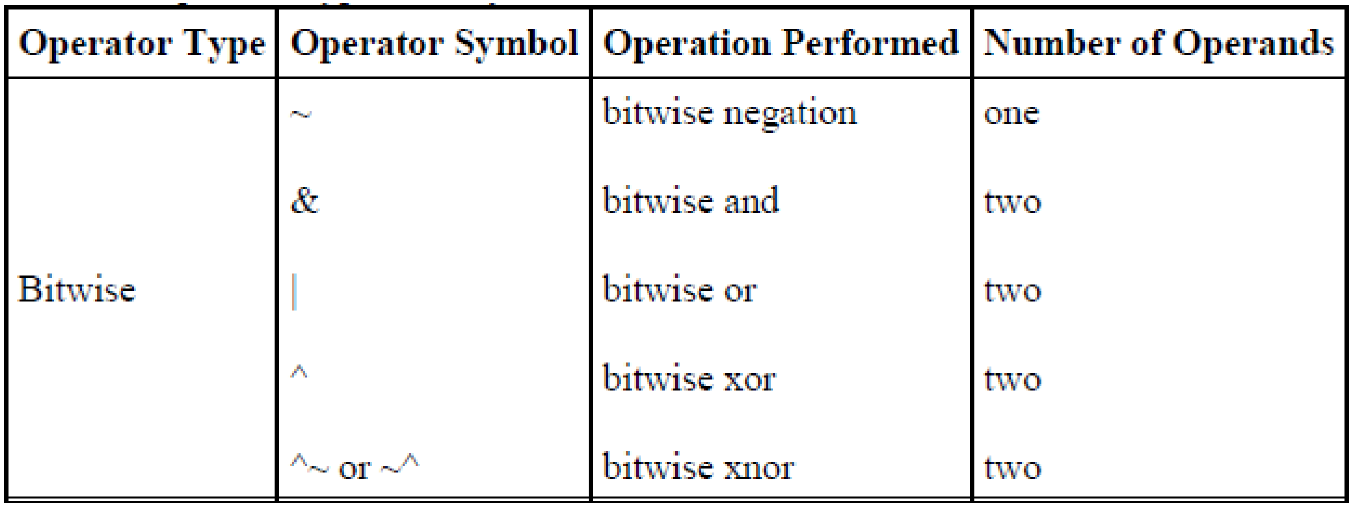
Equality operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Equality	==	equality	two
	!=	inequality	two
	===	case equality	two
	!==	case inequality	two

```
// A = 4, B = 3
// X = 4'b1010, Y = 4'b1101
// Z = 4'b1xxz, M = 4'b1xxz, N = 4'b1xxx

A == B // Results in logical 0
X != Y // Results in logical 1

X == Z // Results in x
Z === M //Results in logical 1(all bits match,including x and z)
Z === N //Results in logical 0 (LSB does not match)
M !== N // Results in logical 1
```

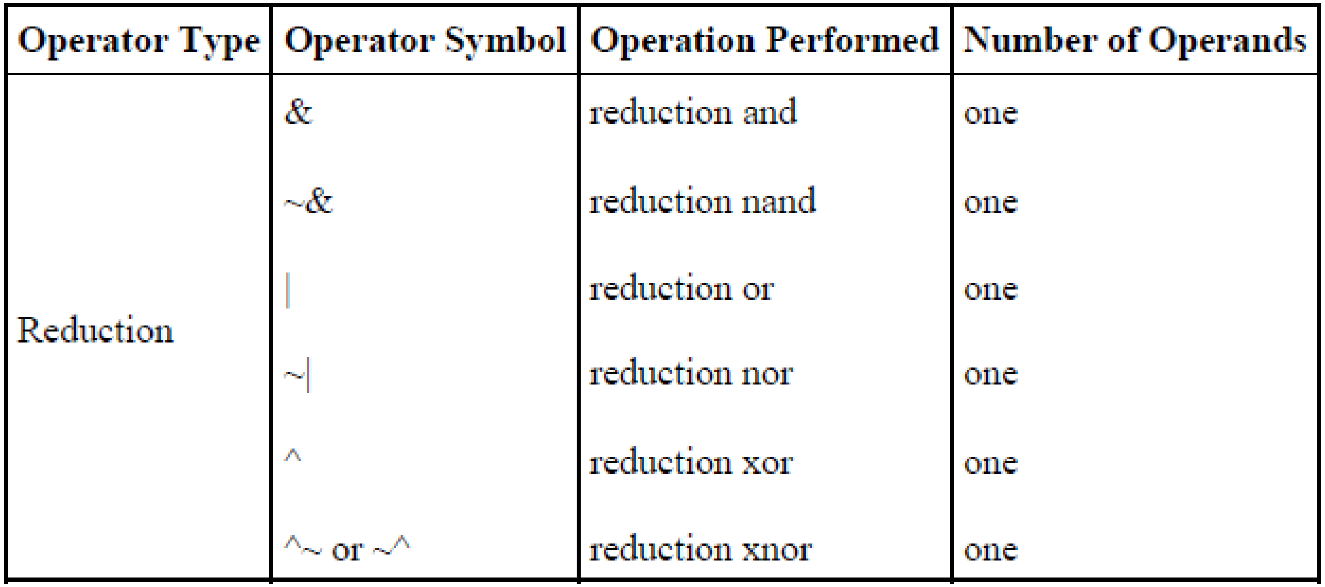
Bitwise operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Bitwise	~	bitwise negation	one
	&	bitwise and	two
	|	bitwise or	two
	^	bitwise xor	two
	^~ or ~^	bitwise xnor	two

```
// X = 4'b1010, Y = 4'b1101
// Z = 4'b10x1

~X // Negation. Result is 4'b0101
X & Y // Bitwise and. Result is 4'b1000
X | Y // Bitwise or. Result is 4'b1111
X ^ Y // Bitwise xor. Result is 4'b0111
X ^~ Y // Bitwise xnor. Result is 4'b1000
X & Z // Result is 4'b10x0
```

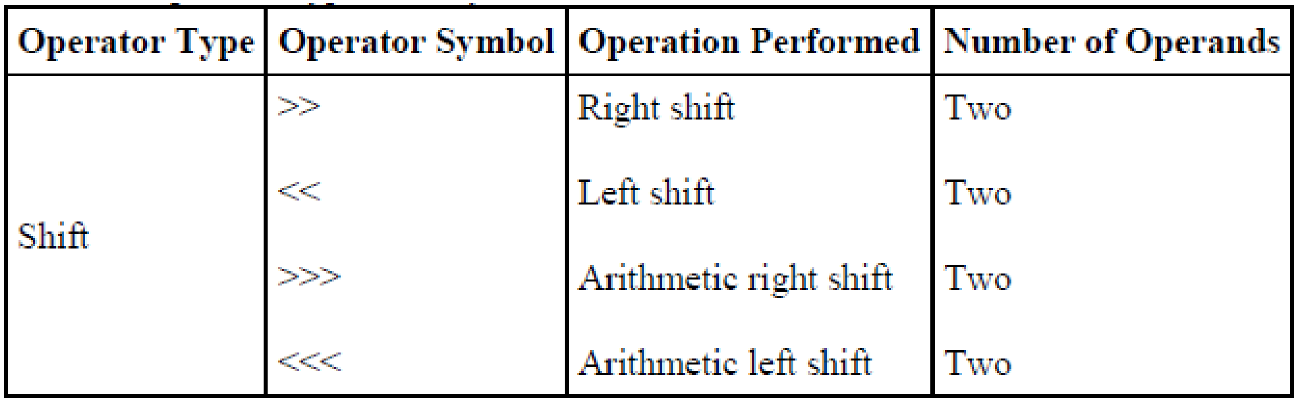
Reduction operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Reduction	&	reduction and	one
	~&	reduction nand	one
	|	reduction or	one
	~|	reduction nor	one
	^	reduction xor	one
	^~ or ~^	reduction xnor	one

```
// X = 4'b1010
&X  //Equivalent to 1 & 0 & 1 & 0. Results in 1'b0
|X  //Equivalent to 1 | 0 | 1 | 0. Results in 1'b1
^X  //Equivalent to 1 ^ 0 ^ 1 ^ 0. Results in 1'b0
```

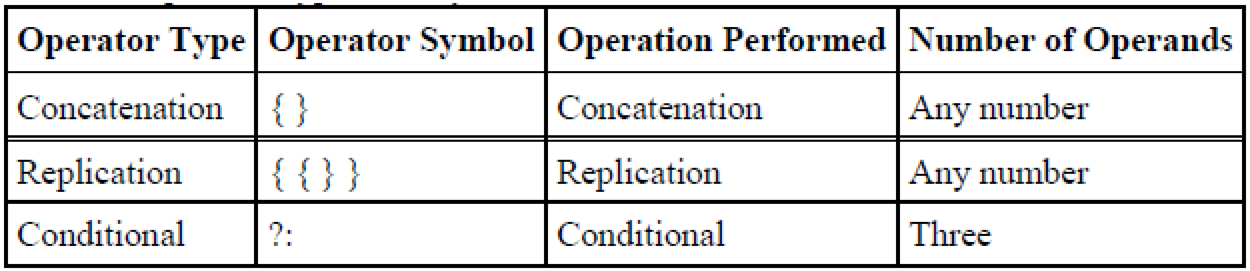
Shift operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Shift	>>	Right shift	Two
	<<	Left shift	Two
	>>>	Arithmetic right shift	Two
	<<<	Arithmetic left shift	Two

```
// X = 4'b1100
Y = X >> 1; //Y is 4'b0110. Shift right 1 bit. 0 filled in MSB
              position.
Y = X << 1; //Y is 4'b1000. Shift left 1 bit. 0 filled in LSB
            position.
Y = X << 2; //Y is 4'b0000. Shift left 2 bits.

integer a, b, c; //Signed data types
a = 0;
b = -10; // 00111...10110 binary
c = a + (b >>> 3); //Results in -2 decimal, due to arithmetic shift
```

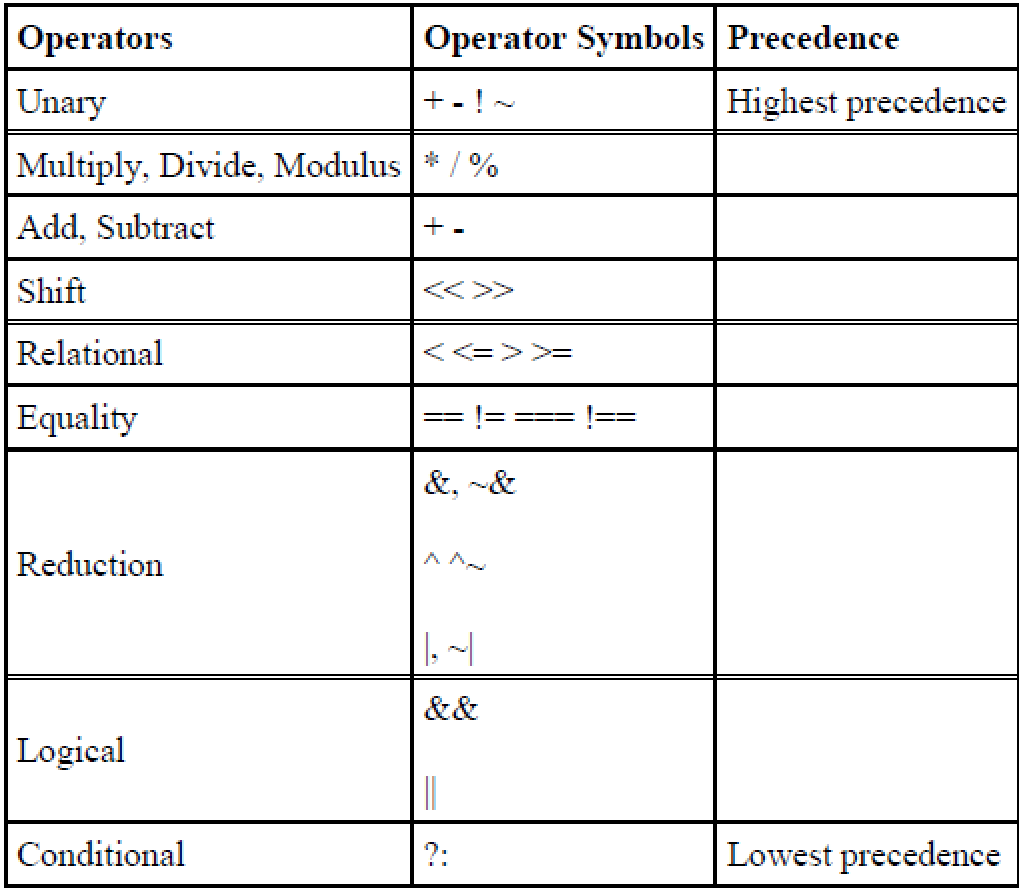
Concatenation, reduction & conditional operators



Operator Type	Operator Symbol	Operation Performed	Number of Operands
Concatenation	{ }	Concatenation	Any number
Replication	{ { } }	Replication	Any number
Conditional	?:	Conditional	Three

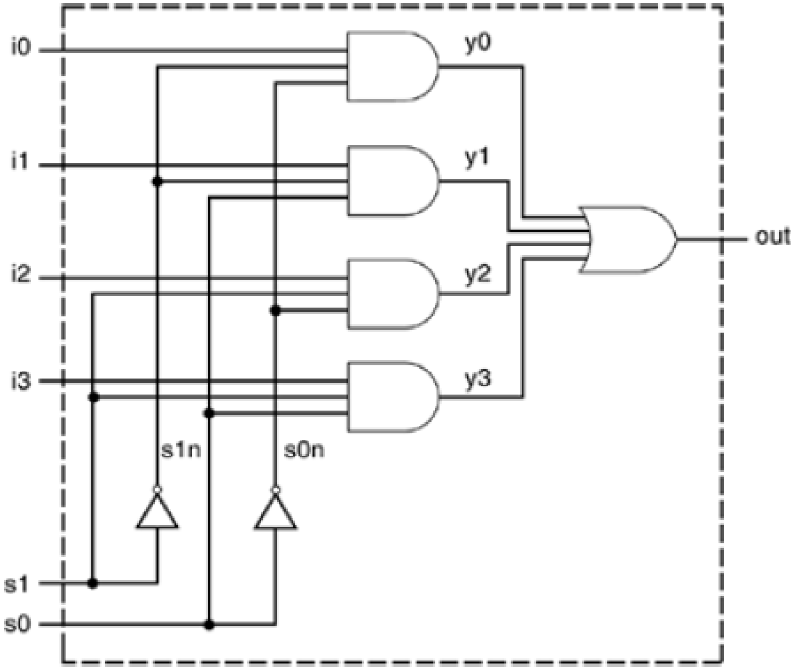
```
// A = 1'b1, B = 2'b00, C = 2'b10, D = 3'b110
Y = {B , C} // Result Y is 4'b0010
Y = {A , B , C , D , 3'b001} //Result Y is 11'b10010110001
Y = {A , B[0], C[1]} // Result Y is 3'b101

A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;
Y = { 4{A} } // Result Y is 4'b1111
Y = { 4{A} , 2{B} } // Result Y is 8'b11110000
Y = { 4{A} , 2{B} , C } // Result Y is 8'b111000010

condition_expr ? true_expr : false_expr ;

assign out = control ? in1 : in0;
```

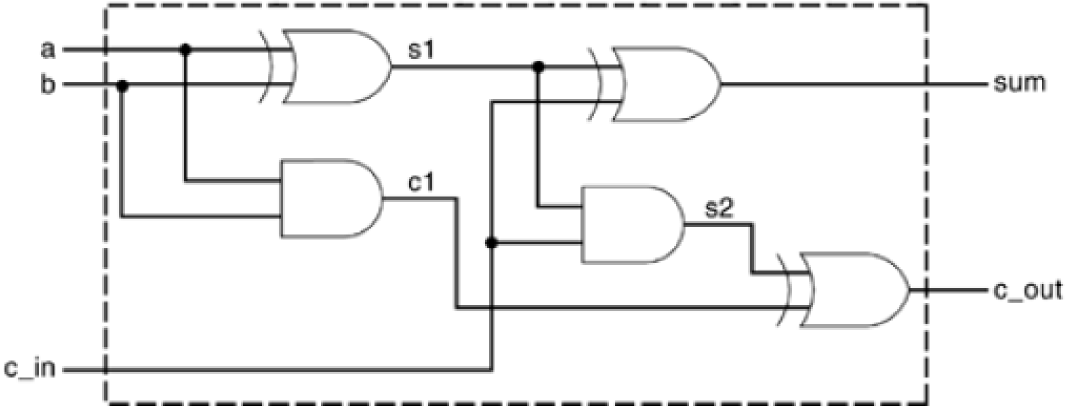
Operators precedence



Operators	Operator Symbols	Precedence
Unary	+ - ! ~	Highest precedence
Multiply, Divide, Modulus	* / %	
Add, Subtract	+ -	
Shift	<< >>	
Relational	< <= > >=	
Equality	== != === !==	
Reduction	&, ~& ^ ^~ |, ~|	
Logical	&& ||	
Conditional	?:	Lowest precedence

Examples


i0
i1
i2
i3
y0
y1
y2
y3
out
s1n
s0n
s1
s0


a
b
s1
sum
c1
s2
c_out
c_in