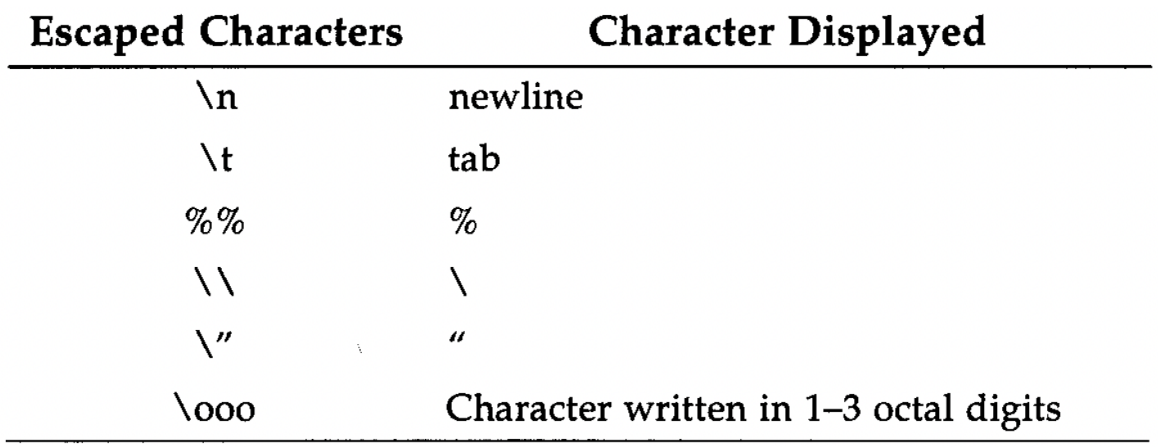
Verilog Operators

Smita Khole

Strings

- Strings can be stored in **reg**
- Width of register variable should be large to hold the string
- Each character takes 1 byte
- If reg width is smaller than string, then Verilog will truncate the leftmost bits of the string
- If reg width is greater it will fill it with zero

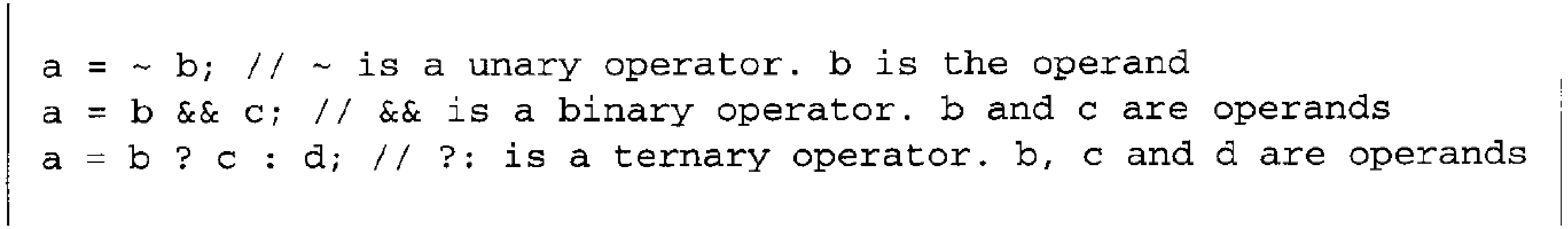
Special characters



Escaped Characters	Character Displayed
\n	newline
\t	tab
%%	%
\\	\
\”	“
\ooo	Character written in 1–3 octal digits

Operators

- Three types:
  - Unary
  - Binary
  - Ternary



```
a = ~ b; // ~ is a unary operator. b is the operand
a = b && c; // && is a binary operator. b and c are operands
a = b ? c : d; // ?: is a ternary operator. b, c and d are operands
```

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

Operator Type	Operator Symbol	Operation Performed	Number of Operands
Shift	`>>`	Right shift	two
	`<<`	Left shift	two
Concatenation	`{ }`	Concatenation	any number
Replication	`{ { } }`	Replication	any number
Conditional	`?:`	Conditional	three

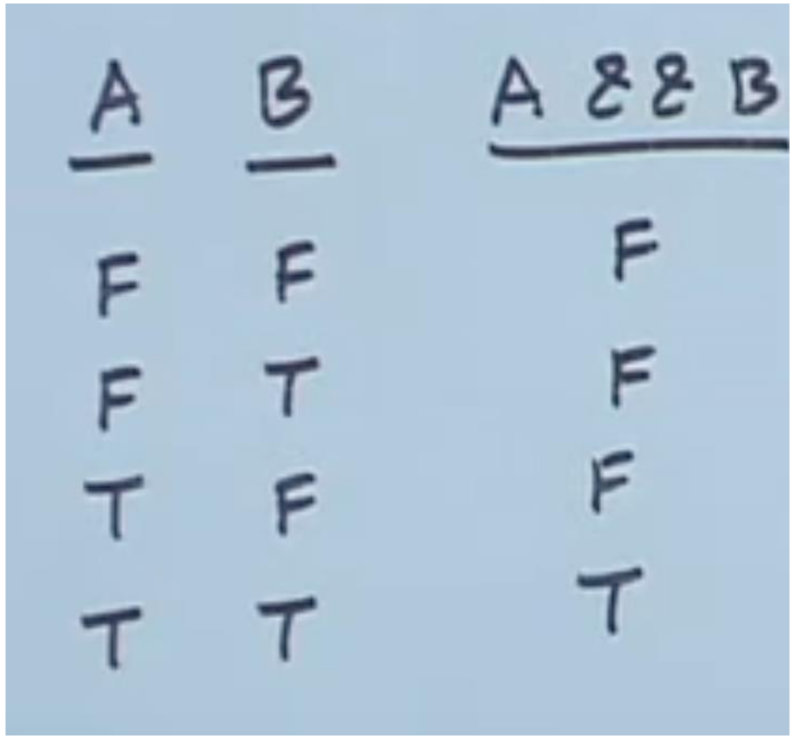
```
A = 4'b0011; B = 4'b0100; // A and B are register vectors
D = 6; E = 4; // D and E are integers

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
```

If any operand value is **x** then result of entire expression is **x**

```
in1 = 4'b101x;
in2 = 4'b1010;
sum = in1 + in2; // sum will be evaluated to the value 4'bx
```

Logical operators

- Evaluate 1 bit value , 0-false ; 1- true; x ambiguous
- If operand=0 then equivalent logic 0
- If operand ≠ 0 then equivalent logic 1
- If operand x or z then equivalent to x



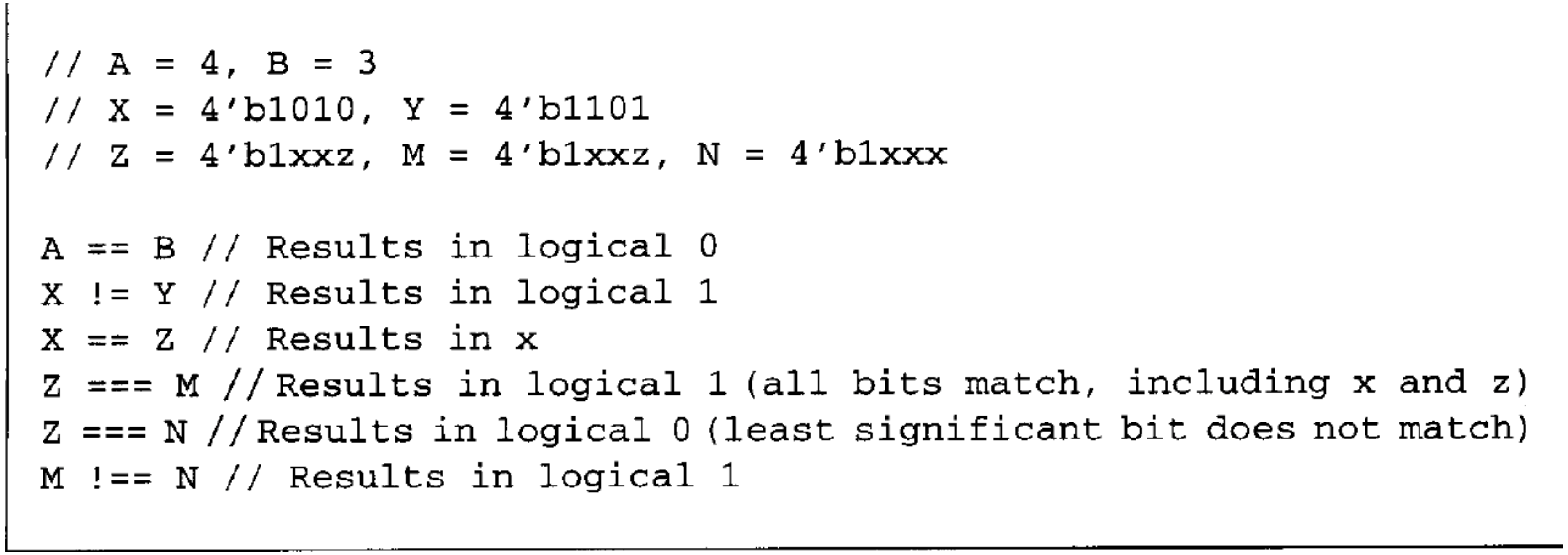
A	B	A && B
F	F	F
F	T	F
T	F	F
T	T	T

Relational Operators

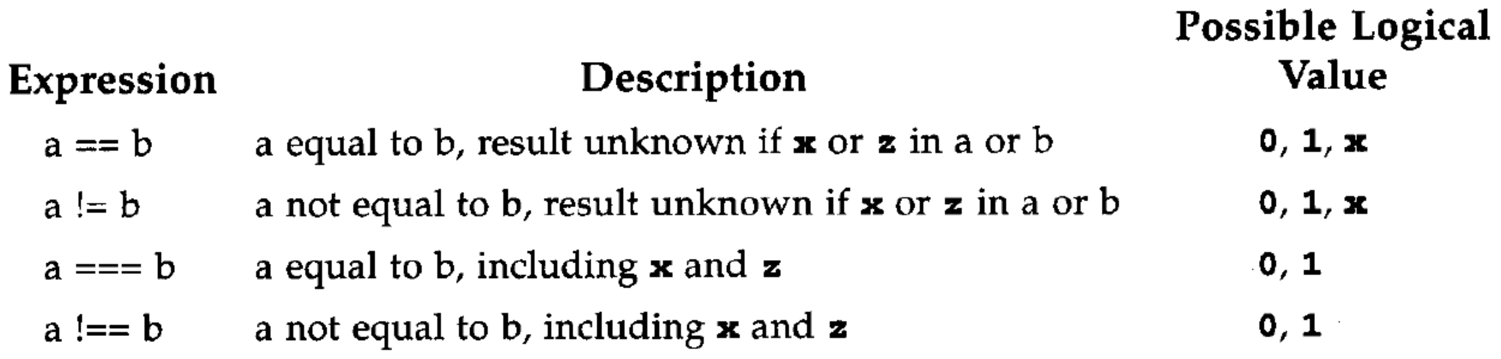
```
// A = 4, B = 3
// X = 4'b1010, Y = 4'b1101, Z = 4'b1xxx

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



```
// 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 (least significant bit does not match)
M !== N // Results in logical 1
```



Expression	Description	Possible Logical Value
a == b	a equal to b, result unknown if **x** or **z** in a or b	**0**, **1**, **x**
a != b	a not equal to b, result unknown if **x** or **z** in a or b	**0**, **1**, **x**
a === b	a equal to b, including **x** and **z**	**0**, **1**
a !== b	a not equal to b, including **x** and **z**	**0**, **1**

Bitwise Operators

bitwise and	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

bitwise or	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

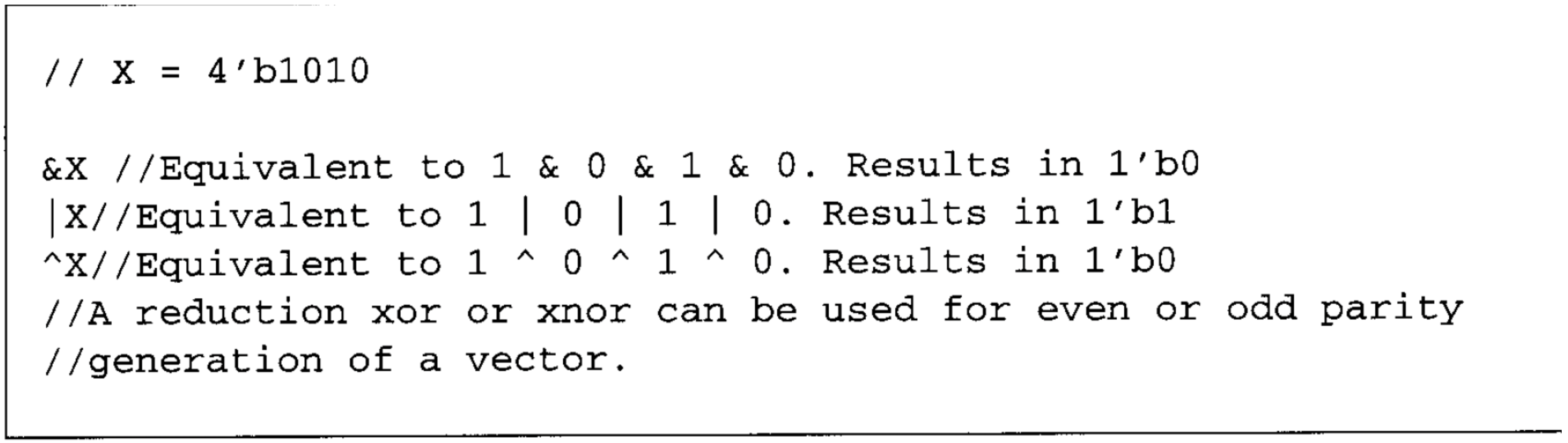
bitwise xor	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

bitwise negation	Result
0	1
1	0
x	x

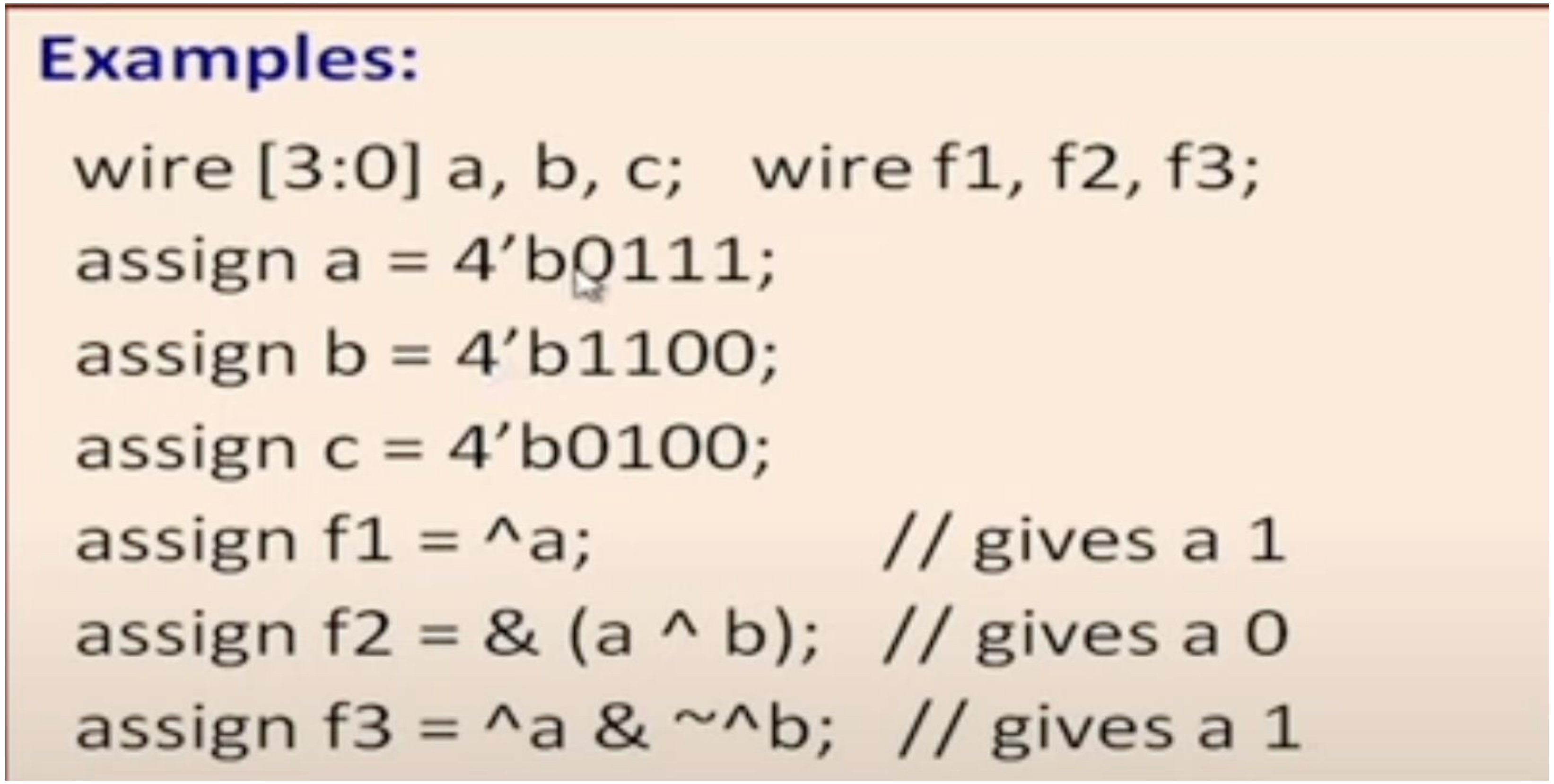
bitwise xnor	0	1	x
0	1	0	x
1	0	1	x
x	x	x	x

Reduction Operators

- Take only one operand
- Perform bitwise operation on single vector operand
- Yield 1 bit result



```
// 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
//A reduction xor or xnor can be used for even or odd parity
//generation of a vector.
```

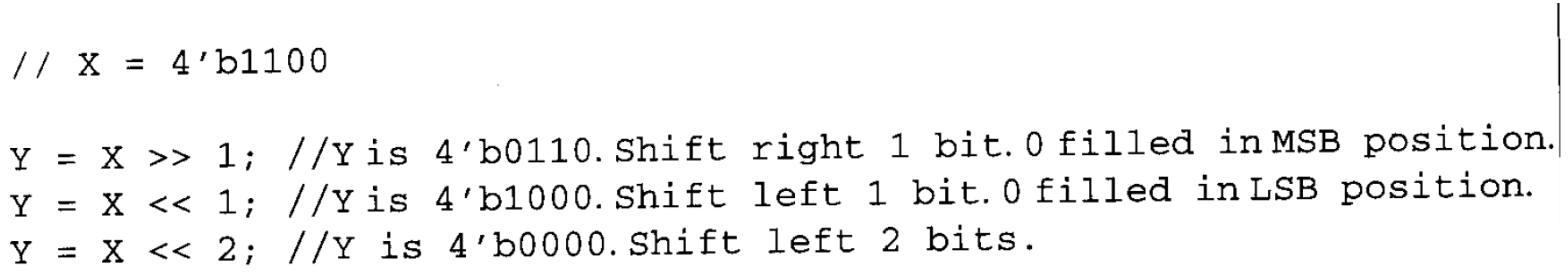


Examples:

```
wire [3:0] a, b, c;   wire f1, f2, f3;
assign a = 4'b0111;
assign b = 4'b1100;
assign c = 4'b0100;
assign f1 = ^a;              // gives a 1
assign f2 = & (a ^ b);   // gives a 0
assign f3 = ^a & ~^b;   // gives a 1
```

Shift Operators

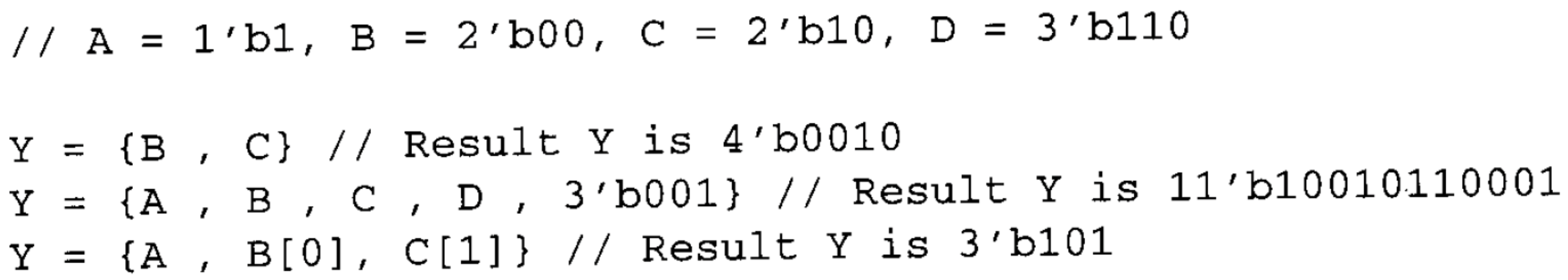
- Shifts the vector right or left by specified number of bits



```
// 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.
```

Concatenation Operator

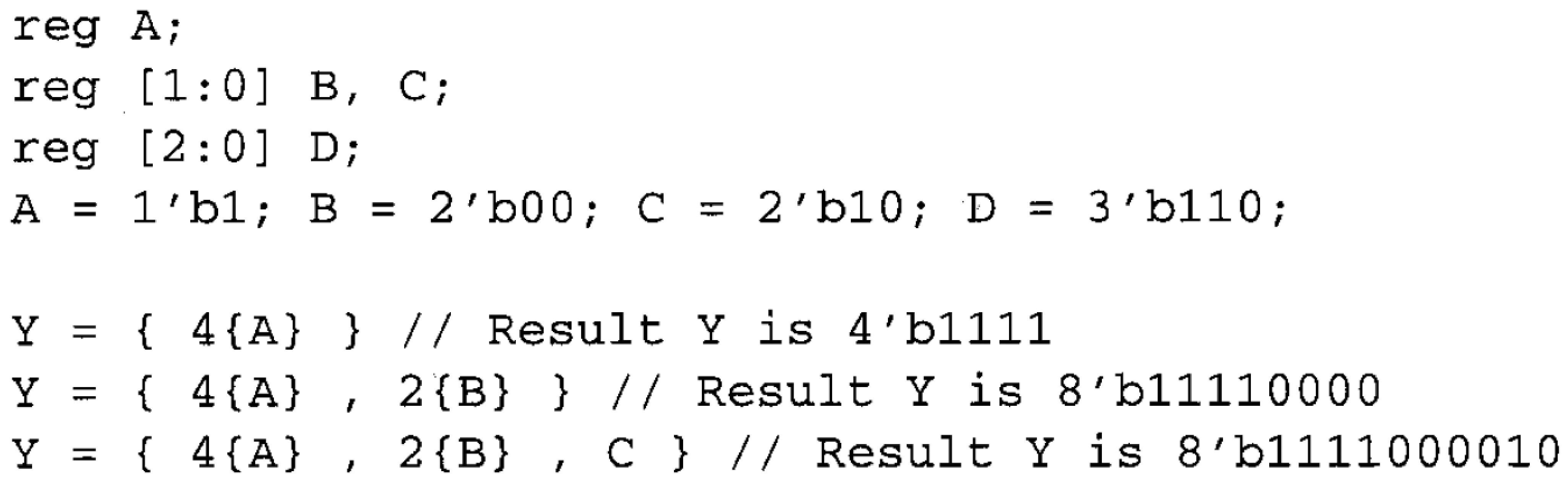
- Appends multiple operands ({ })
- Operands must be sized



```
// 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
```

Replication Operator

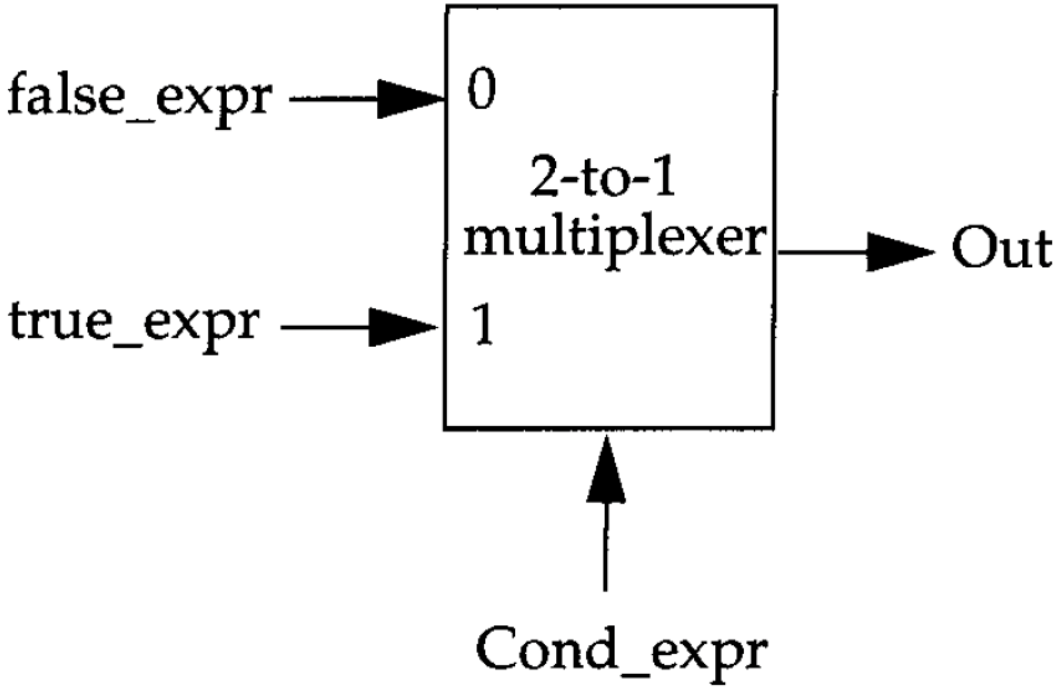
- Repetitive concatenation of same number



```
reg A;
reg [1:0] B, C;
reg [2:0] D;
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’b1111000010
```

Conditional Operator

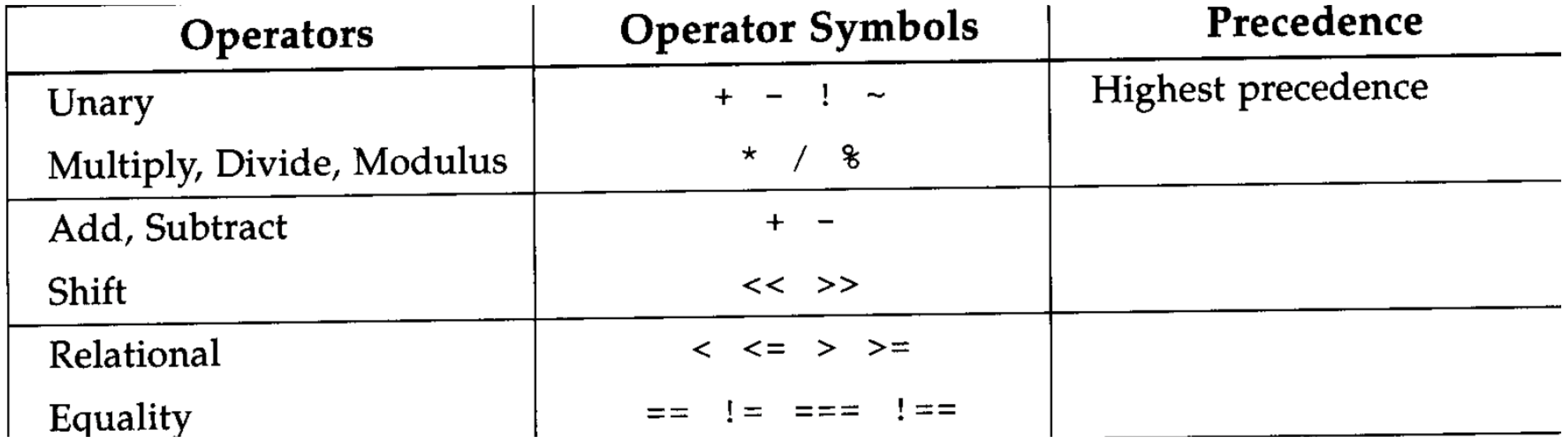
- Takes three operands (?:)
- condition_expr? True_expr: false_expr;



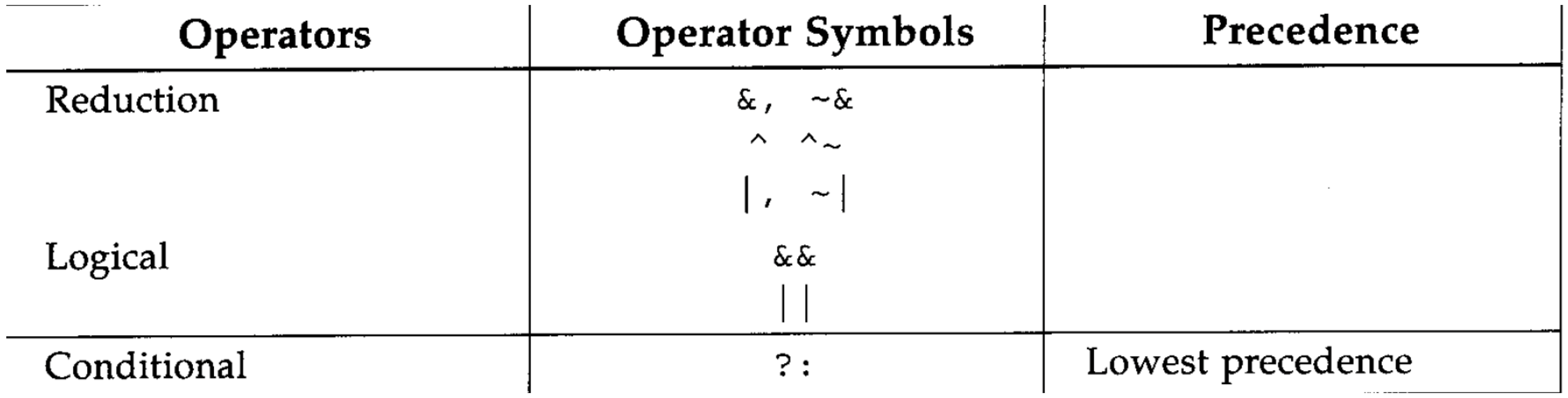
```
//model functionality of a tristate buffer
assign addr_bus = drive_enable ? addr_out : 36'bz;

//model functionality of a 2-to-1 mux
assign out = control ? in1 : in0;
```

```
assign out = (A == 3) ? ( control ? x : y ): ( control ? m : n) ;
```

Operator Precedence



Operators	Operator Symbols	Precedence
Unary	+ - ! ~	Highest precedence
Multiply, Divide, Modulus	* / %	
Add, Subtract	+ -	
Shift	<< >>	
Relational	< <= > >=	
Equality	== != === !==	



Operators	Operator Symbols	Precedence
Reduction	&, ~&	
	^ ^~	
	|, ~|	
Logical	&&	
	||	
Conditional	? :	Lowest precedence