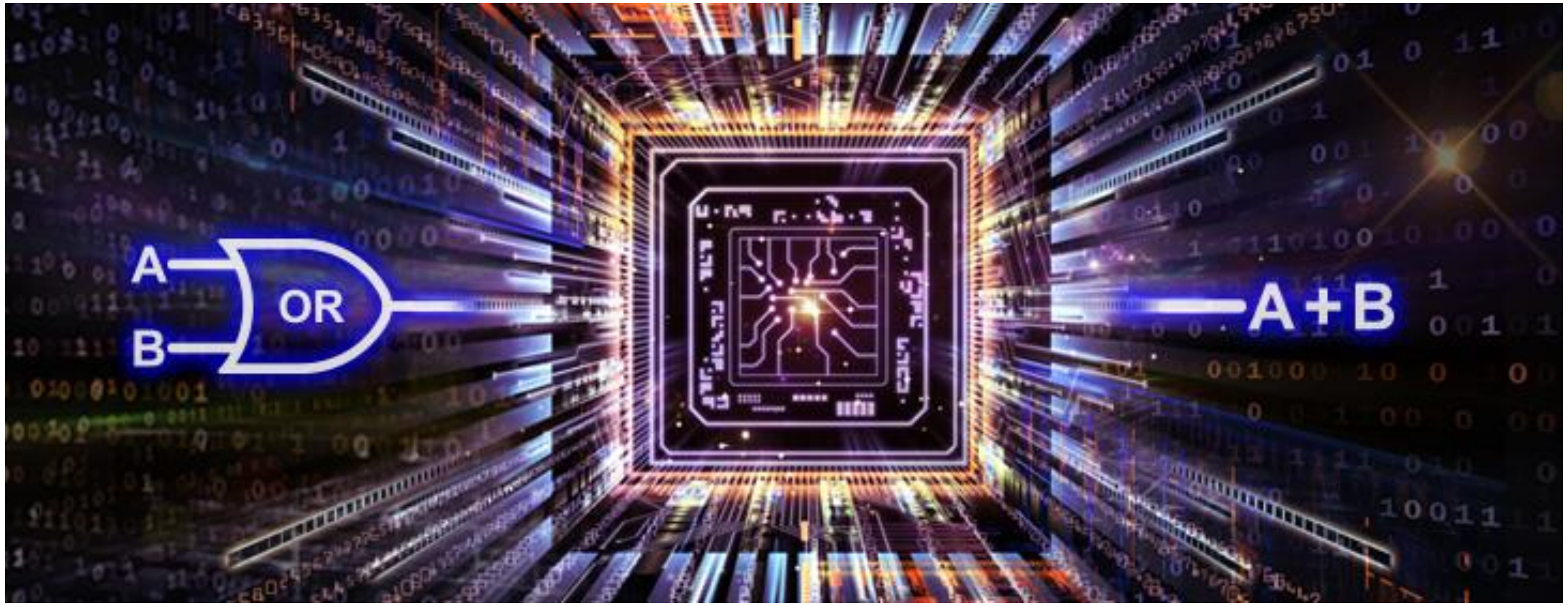


ECS1005 – Digital Systems

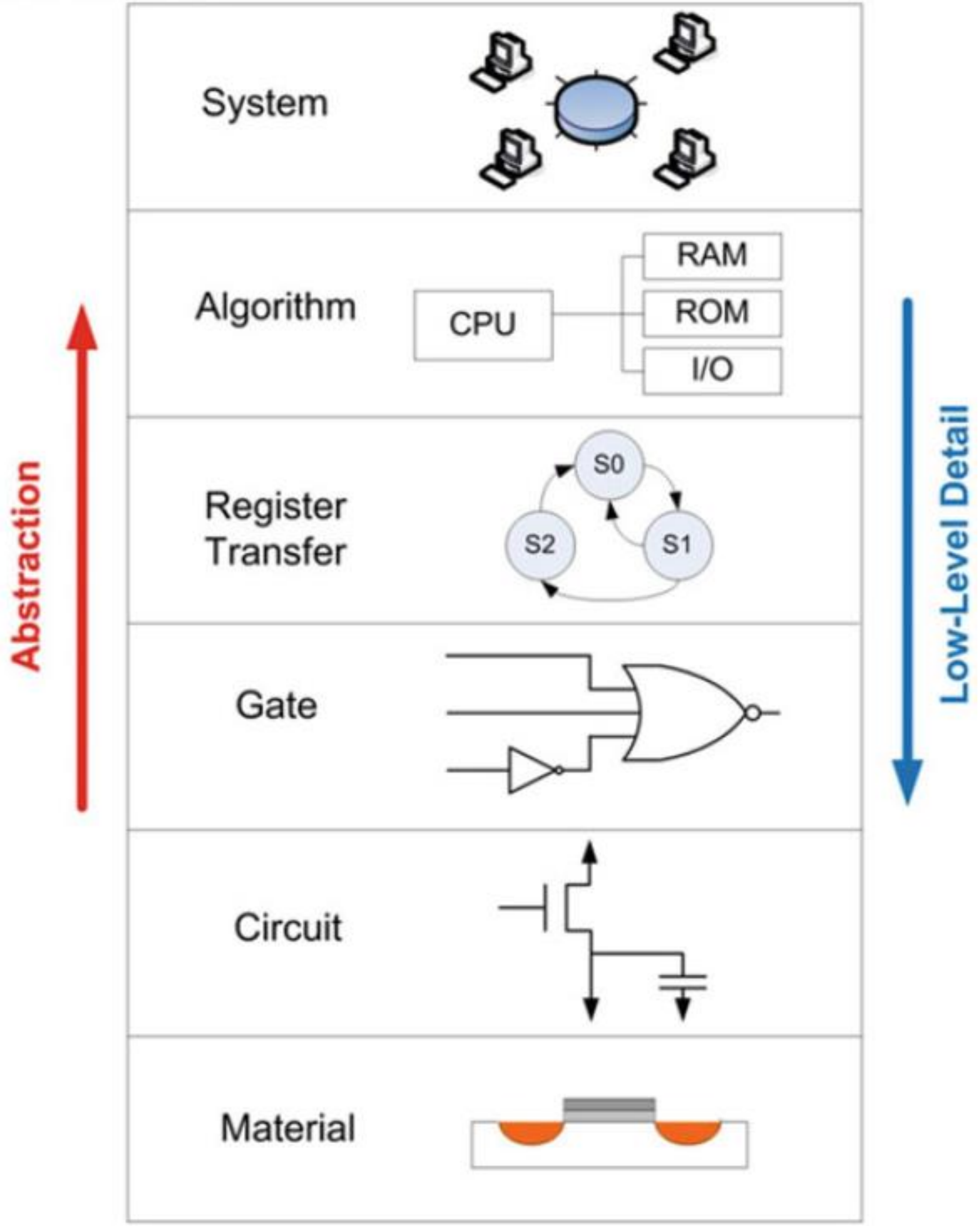
Dr. Ayesha Khalid

Lab 3 on the 30th of November

(Week 11, Semester 1)

HDL Abstraction

Abstraction is an important concept in engineering design because it allows us to specify how systems will operate without getting consumed prematurely with implementation details.



Verilog levels of abstraction

- **Behavioral or algorithmic level**
  This is the highest level of abstraction provided by Verilog HDL. A module can be implemented in terms of the desired design algorithm without concern for the hardware implementation details. Designing at this level is very similar to C programming.
- **Dataflow level**
  At this level the module is designed by specifying the data flow. The designer is aware of how data flows between hardware registers and how the data is processed in the design.
- **Gate level**
  The module is implemented in terms of logic gates and interconnections between these gates. Design at this level is similar to describing a design in terms of a gate-level logic diagram.
- **Switch level**
  This is the lowest level of abstraction provided by Verilog. A module can be implemented in terms of switches, storage nodes, and the interconnections between them. Design at this level requires knowledge of switch-level implementation details.

Verilog allows the designer to mix and match all four levels of abstractions in a design. In the digital design community, the term **register transfer level (RTL)** is frequently used for a Verilog description that uses a combination of behavioural and dataflow constructs

Continuous assignment

A continuous assignment drives a value into a wire.
Syntax: assign wire_name = expression;

- *Continuous assignments* model combinational logic. They are evaluated concurrently.
- Each time the expression changes on the right-hand side, the right-hand side is re-evaluated, and the result is assigned to the wire on the left-hand side.

A wire must not be assigned more than once!

For tristated buffers, a single wire is assigned more than once, but we are not going to cover that.

Verilog Operators

5.4.3.1 Assignment Operator

Verilog uses the equal sign (=) to denote an assignment. The left-hand side (LHS) of the assignment is the target signal. The right-hand side (RHS) contains the input arguments and can contain both signals, constants, and operators.

Example:

```
F1 = A;       // F1 is assigned the signal A
F2 = 8'hAA;   // F2 is an 8-bit vector and is assigned the value 10101010₂
```

Verilog Operators

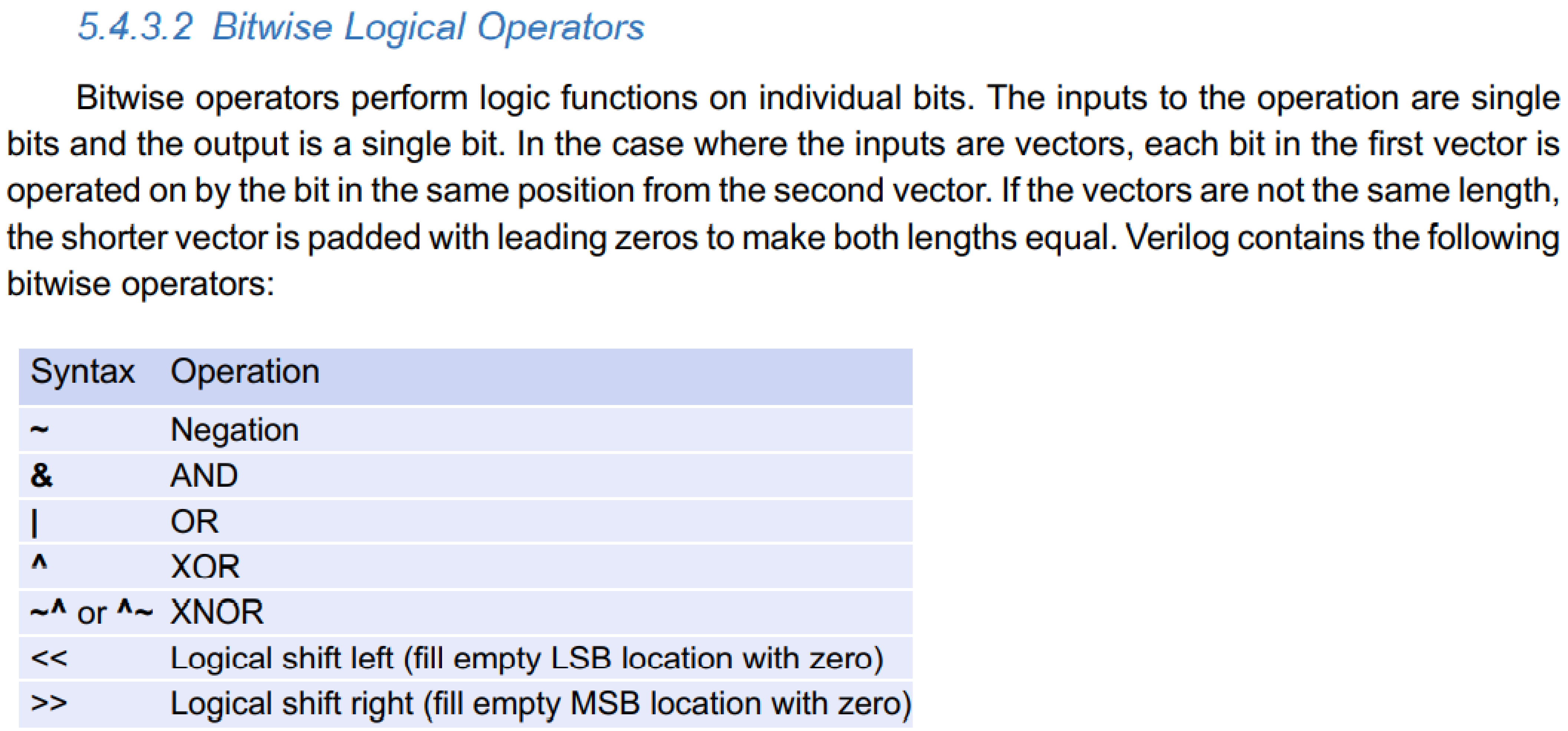
5.4.3.1 Assignment Operator

Verilog uses the equal sign (=) to denote an assignment. The left-hand side (LHS) of the assignment is the target signal. The right-hand side (RHS) contains the input arguments and can contain both signals, constants, and operators.

Example:

```
F1 = A;       // F1 is assigned the signal A
F2 = 8'hAA;   // F2 is an 8-bit vector and is assigned the value 10101010_2
```

Verilog Operators



5.4.3.2 Bitwise Logical Operators

Bitwise operators perform logic functions on individual bits. The inputs to the operation are single bits and the output is a single bit. In the case where the inputs are vectors, each bit in the first vector is operated on by the bit in the same position from the second vector. If the vectors are not the same length, the shorter vector is padded with leading zeros to make both lengths equal. Verilog contains the following bitwise operators:

Syntax	Operation
~	Negation
&	AND
|	OR
^	XOR
~^ or ^~	XNOR
<<	Logical shift left (fill empty LSB location with zero)
>>	Logical shift right (fill empty MSB location with zero)

Verilog Operators

Syntax	Operation
~	Negation
&	AND
|	OR
^	XOR
~^ or ^~	XNOR
<<	Logical shift left (fill empty LSB location with zero)
>>	Logical shift right (fill empty MSB location with zero)

Example:

```
~X          // invert each bit in X
X & Y       // AND each bit of X with each bit of Y
X | Y       // OR each bit of X with each bit of Y
X ^ Y       // XOR each bit of X with each bit of Y
X ~^ Y      // XNOR each bit of X with each bit of Y
X << 3      // Shift X left 3 times and fill with zeros
Y >> 2      // Shift Y right 2 times and fill with zeros
```

Verilog Operators

5.4.3.3 Reduction Logic Operators

A *reduction* operator is one that uses each bit of a vector as individual inputs into a logic operation and produces a single bit output. Verilog contains the following reduction logic operators.

Syntax	Operation
&	AND all bits in the vector together (1-bit result)
~&	NAND all bits in the vector together (1-bit result)
|	OR all bits in the vector together (1-bit result)
~|	NOR all bits in the vector together (1-bit result)
^	XOR all bits in the vector together (1-bit result)
~^ or **^~**	XNOR all bits in the vector together (1-bit result)

Example:

```
&X          // AND all bits in vector X together
~&X         // NAND all bits in vector X together
|X          // OR all bits in vector X together
~|X         // NOR all bits in vector X together
^X          // XOR all bits in vector X together
~^X         // XNOR all bits in vector X together
```

Verilog Operators (section 5.4.3)

Example:

```
&X          // AND all bits in vector X together
~&X         // NAND all bits in vector X together
|X          // OR all bits in vector X together
~|X         // NOR all bits in vector X together
^X          // XOR all bits in vector X together
~^X         // XNOR all bits in vector X together
```

Verilog Operators

5.4.3.4 Boolean Logic Operators

A Boolean logic operator is one that returns a value of TRUE (1) or FALSE (0) based on a logic operation of the input operations. These operations are used in decision statements.

Syntax	Operation
!	Negation
&&	AND
||	OR

Example:

```
!X          //A logical negation (!) operator will convert a non-zero or true operand into 0 and a zero or false opera
X && Y      //The result of a logical and (&&) is 1 or true when both of its operands are true or non-zero
X || Y      //The result of a logical or (||) is 1 or true when either of its operands are true or non-zero
```

Please note that the explanations in the examples given in text book is incorrect

Verilog Operators

5.4.3.5 Relational Operators

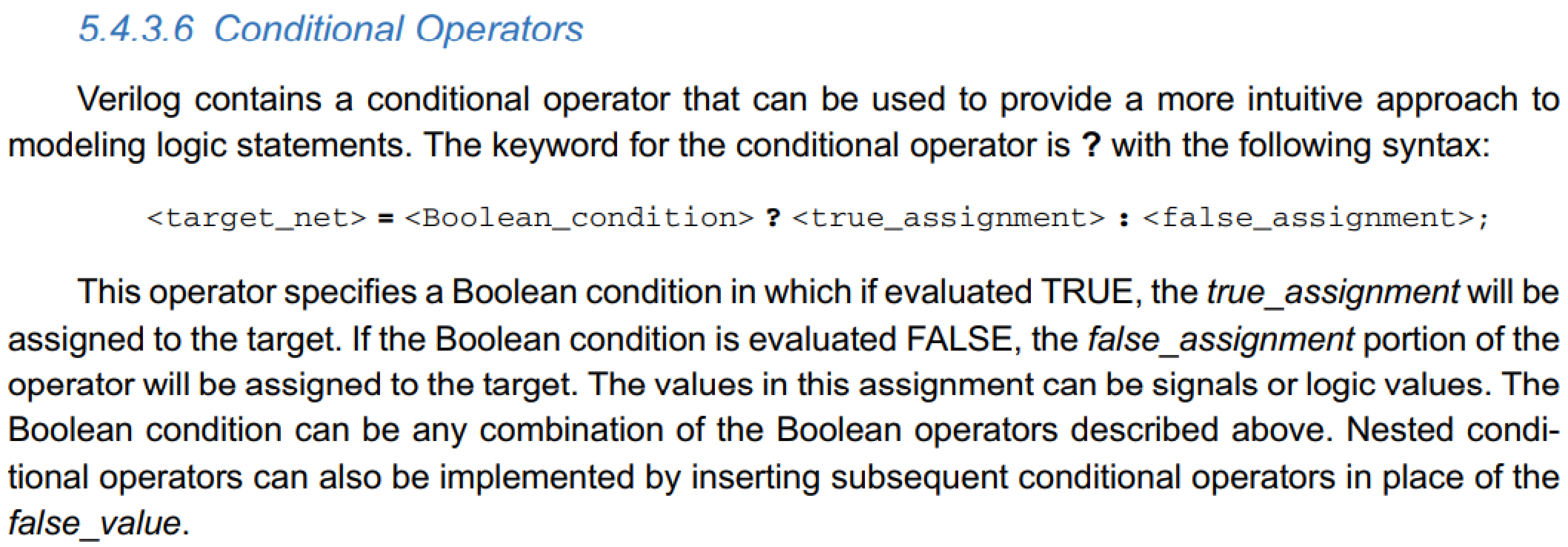
A relational operator is one that returns a value of TRUE (1) or FALSE (0) based on a comparison of two inputs.

Syntax	Description
==	Equality
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

Example:

```
X == Y      // TRUE if X is equal to Y, FALSE otherwise
X != Y      // TRUE if X is not equal to Y, FALSE otherwise
X < Y       // TRUE if X is less than Y, FALSE otherwise
X > Y       // TRUE if X is greater than Y, FALSE otherwise
X <= Y      // TRUE if X is less than or equal to Y, FALSE otherwise
X >= Y      // TRUE if X is greater than or equal to Y, FALSE otherwise
```

Verilog Operators



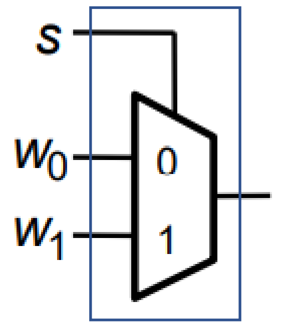
5.4.3.6 Conditional Operators

Verilog contains a conditional operator that can be used to provide a more intuitive approach to modeling logic statements. The keyword for the conditional operator is **?** with the following syntax:

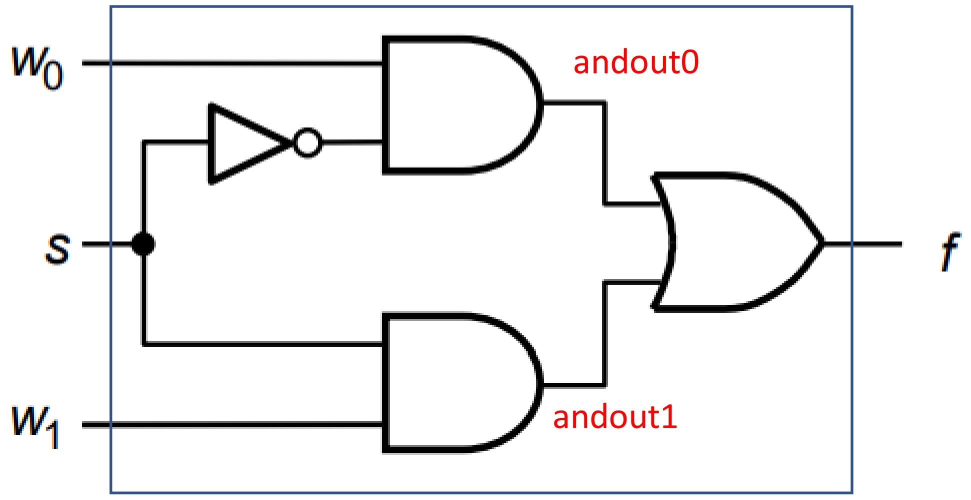
```
<target_net> = <Boolean_condition> ? <true_assignment> : <false_assignment>;
```

This operator specifies a Boolean condition in which if evaluated TRUE, the *true_assignment* will be assigned to the target. If the Boolean condition is evaluated FALSE, the *false_assignment* portion of the operator will be assigned to the target. The values in this assignment can be signals or logic values. The Boolean condition can be any combination of the Boolean operators described above. Nested conditional operators can also be implemented by inserting subsequent conditional operators in place of the *false_value*.

Conditional Operator



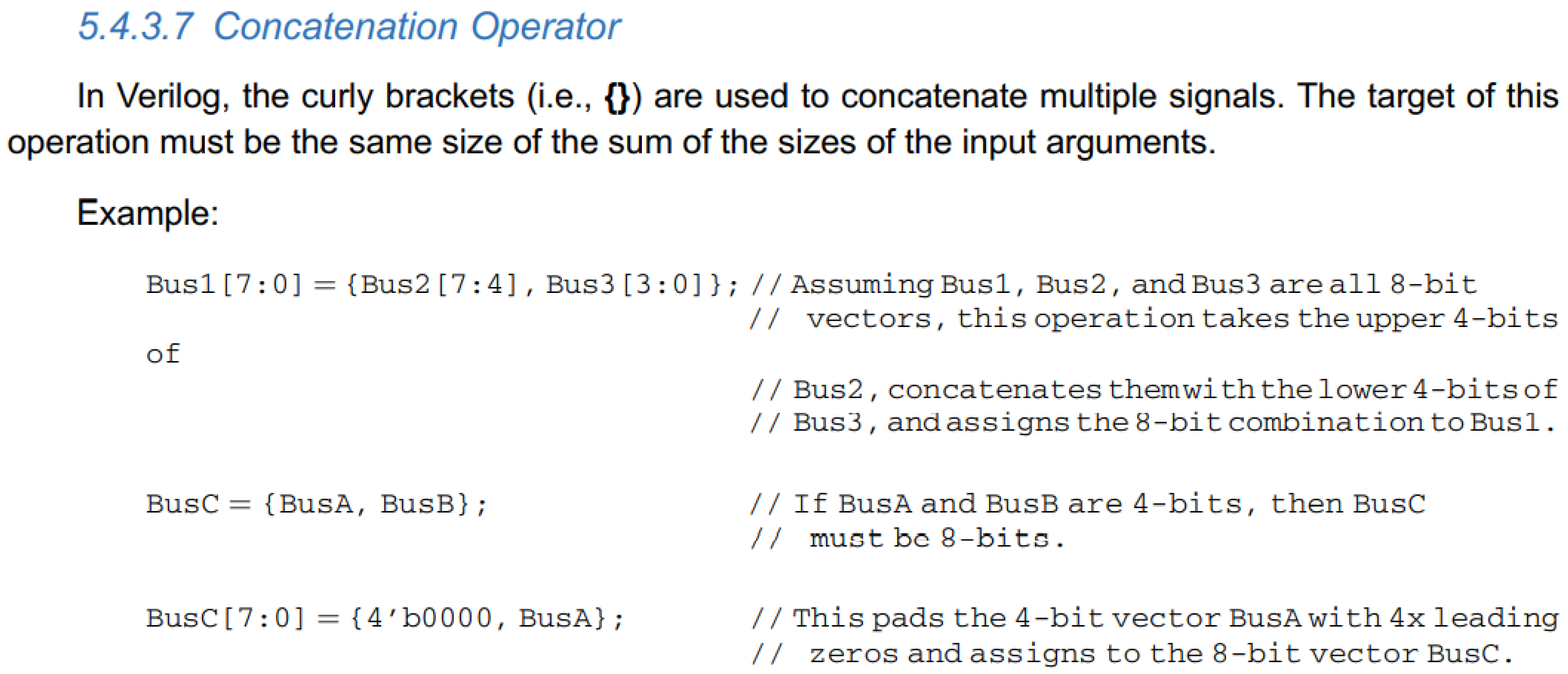
s	f
0	w_0
1	w_1



```
module MUX(f,s,w0,w1);
input s,w0,w1;
output f;
     assign  f= s?w1:w0;
endmodule
```

```
module MUX2(f,s,w0,w1);
input s,w0,w1;
output f;
 // Using gate level assignement
  wire sn, andout0, andout1;
  not (sn,s);
  and (andout0, sn,w0);
  and (andout1, s ,w1);
  or(f,andout0,andout1);
endmodule
```

Verilog Operators



5.4.3.7 Concatenation Operator

In Verilog, the curly brackets (i.e., **{}**) are used to concatenate multiple signals. The target of this operation must be the same size of the sum of the sizes of the input arguments.

Example:

```
Bus1[7:0] = {Bus2[7:4], Bus3[3:0]}; // Assuming Bus1, Bus2, and Bus3 are all 8-bit
                                    //  vectors, this operation takes the upper 4-bits
of
                                    // Bus2, concatenates them with the lower 4-bits of
                                    // Bus3, and assigns the 8-bit combination to Bus1.

BusC = {BusA, BusB};                // If BusA and BusB are 4-bits, then BusC
                                    //  must be 8-bits.

BusC[7:0] = {4'b0000, BusA};        // This pads the 4-bit vector BusA with 4x leading
                                    //  zeros and assigns to the 8-bit vector BusC.
```

Verilog Operators

5.4.3.7 Concatenation Operator

In Verilog, the curly brackets (i.e., **{}**) are used to concatenate multiple signals. The target of this operation must be the same size of the sum of the sizes of the input arguments.

Example:

```
// Stimulate the inputs, set input lines of adder
initial
begin

  for(int i=0; i<8; i=i+1) // checking all possible values of S1,S0
    begin
      #5 {tb_a,tb_b,tb_cin}=i;// 000,001, ...
    end

   #5 $finish;
end
```

Verilog Operators

5.4.3.8 Replication Operator

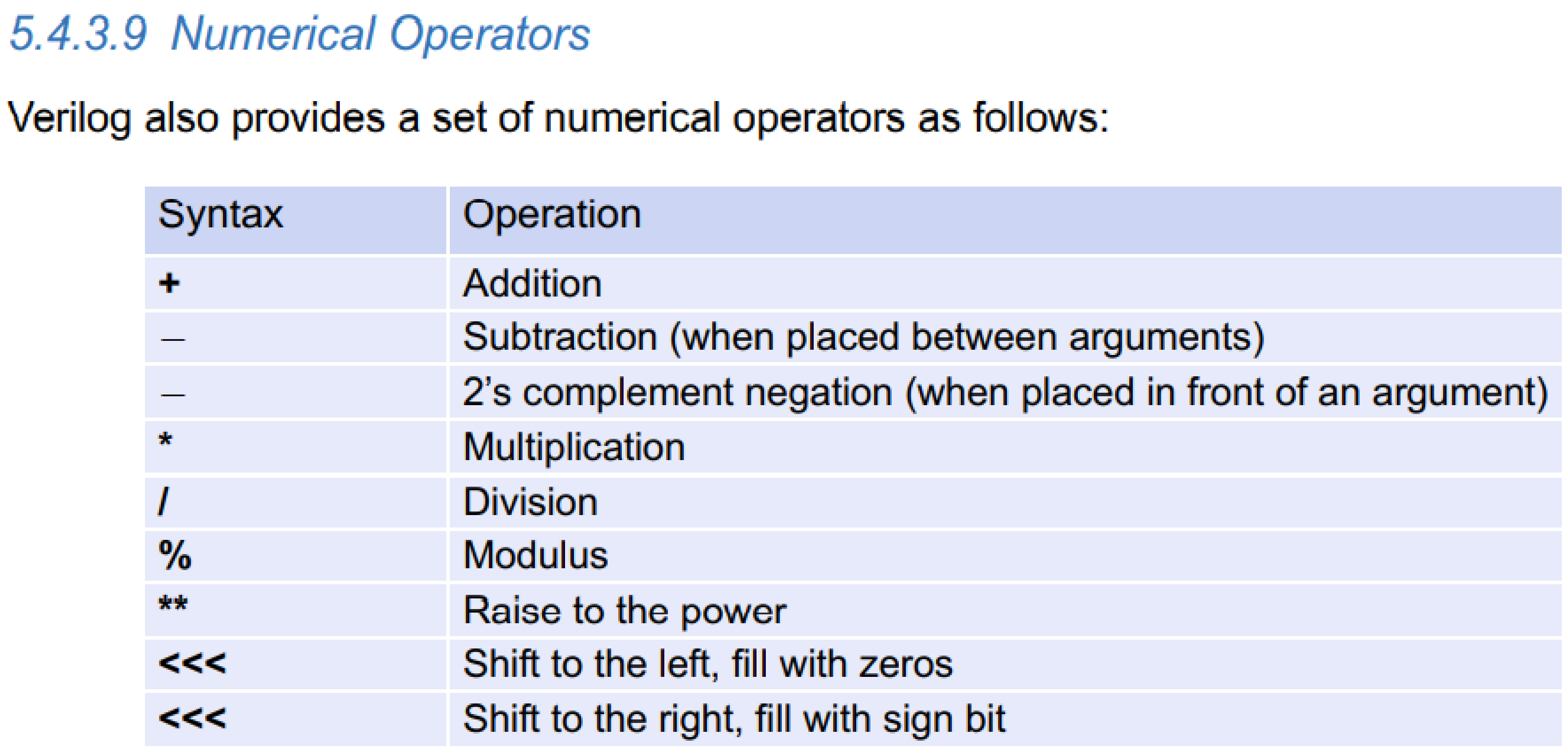
Verilog provides the ability to concatenate a vector with itself through the *replication operator*. This operator uses double curly brackets (i.e., **{{}}**) and an integer indicating the number of replications to be performed. The replication syntax is as follows:

```
{<number_of_replications>{<vector_name_to_be_replicated>}}
```

Example:

```
BusX = {4{Bus1}};          // This is equivalent to: BusX = {Bus1, Bus1, Bus1, Bus1};
BusY = {2{A,B}};           // This is equivalent to: BusY = {A, B, A, B};
BusZ = {Bus1, {2{Bus2}}};  // This is equivalent to: BusZ = {Bus1, Bus2, Bus2};
```

Verilog Operators



5.4.3.9 Numerical Operators

Verilog also provides a set of numerical operators as follows:

Syntax	Operation
+	Addition
–	Subtraction (when placed between arguments)
–	2’s complement negation (when placed in front of an argument)
*****	Multiplication
/	Division
%	Modulus
******	Raise to the power
<<<	Shift to the left, fill with zeros
<<<	Shift to the right, fill with sign bit

Verilog Operators

5.4.3.9 Numerical Operators

Verilog also provides a set of numerical operators as follows:

Example:

```
X + Y        // Add X to Y
X - Y        // Subtract Y from X
-X           // Take the two's complement negation of X
X * Y        // Multiply X by Y
X / Y        // Divide X by Y
X % Y        // Modulus X/Y
X ** Y       // Raise X to the power of Y
X <<< 3      // Shift X left 3 times, fill with zeros
X >>> 2      // Shift X right 2 times, fill with sign bit
```