MODULE-4 INTRODUCTION TO VERILOG

Introduction to VERILOG: Structure of Verilog Module, Operators, Data Types, Styles of description
VERILOG Data Flow Description: Highlights of Data Flow description, Structure of Data Flow Description

Why HDL?

What is Hardware description language (HDL):

HDL is a computer aided design (CAD) tool for the modern digital design and synthesis of digital systems.

Need for HDL

The advancement in the semiconductor technology, the power and complexity of digital systems has increased. Due to this, such digital systems cannot be realized using discrete integrated circuits (IC's).

Complex digital systems can be realized using high-density programmable chips such as application specific integrated circuits (ASIC's) and field programmable gate arrays (FPGA's). To design such systems, we require sophisticated CAD tool such as HDL.

HDL is used by designer to describe the system in a computer language that is similar to other software Language like C. Debugging the design is easy, since HDL package implement simulators and test benches. The two widely used Hardware description languages are VHDL and Verilog

A Brief History of Verilog

Evolution of Verilog

- In 1983, a company called **Gateway design Automation** developed a hardware-description language for its newly introduced logic simulator verilog_XL
- **Gateway** was bought by **cadence** in 1989 & **cadence** made Verilog available as public domain.
- In December 1995, Verilog HDL became IEEE standard 1364-1995.
- The language presently is maintained by the Open Verilog international (OVI) organization.
- Verilog code structure is based on C software language.

Structure of Verilog Module

The Verilog module has a declaration and a body. In the declaration, name, input and outputs of the modules are listed. The body shows the relationship between the input and the outputs with help of signal assignment statements.

The syntax of the Verilog module is shown below

```
module name of module(port_list);
  // declaration:
 input , output, reg, wire, parameter,
inout;
functions, tasks;
// statements or body
Initial statement
always statement
module instantiation
continuous assignment
endmodule
```

The example program is halfadder

```
module halfadder (a,b,sum,carry);
input a;
input b;
output sum;
output carry;
 assign sum=a ^b; // statement 1
assign carry=a &b; // statement2
end module
```

- Verilog is case sensitive. Halfadder and halfadder are two different modules in verilog. The declaration starts with predefined word **module**.
- The name of the module should start with alphabetical letter and can include special character underscore (_). It is user selected.
- Semicolon (;) is a line separator. The order in which the inputs, &outputs and their declarations are written is irrelevant.
- "=" is assignment operator, and symbols ^ and & are used for: "xor" and "and" respectively.
- The doubles slashes (//) signal a comment command or /*… ....*/ the pair is used to write a comment of any length.
- The program ends with predefined word **endmodule**

Verilog ports

input: the port is only an input port. In any assignment statement, the port should appear only on the right hand side of the assignment statement.(i.e., port is read.)

output: the port is an output port. In contrast to VHDL, the Verilog output port can appear on either side of the assignment statement.

inout: this port can be used as both an input and output. The inout port represents a bidirectional bus.

1.4 Operators

HDL has a extensive list of operators. Operator performs a wide variety of functions. Functions classified

1. Logical operators such as and, or, nand, nor, xor, xnor and not
2. Relational operators: to express the relation between objects. The operators include =, /=, <, <=, >and >=.
3. Arithmetic operators: such as +, -, * and division.

4. Shifts operators: To move the bits of an objects in a certain direction such as right or left sll, srl, sla, sra, rol and ror .

Logical operators

These operator performs Logical operations, such as and, or, nand, nor, xor, xnor, and not. The operation can be on two operands or on a single operand. The operand can be single bit or multiple bits.

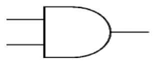
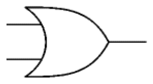
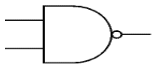
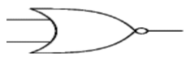
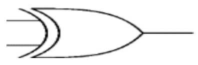
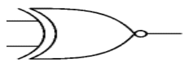
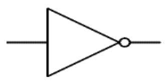
Verilog operator (bitwise)	**Equivalent logic**	**Operand type**	**Result type**
&		Bit	Bit
|		Bit	Bit
~(&)		Bit	Bit
~(|)		Bit	Bit
^		Bit	Bit
~^		Bit	Bit
~		Bit	Bit

Table 1.1 logical operators.

Verilog logical operators

Verilog logical operator can be classified as Bitwise, Boolean logical and reduction logical operators.

The bitwise operators are similar to VHDL logical operators. They operate on the corresponding bits of two operands. These are shown in table1.1

Example Z= x & y, if x=1011 and y=1010 are 4-bit signals then z=1010 is logical **and operation** of x and y.

Boolean operators operate on the two operands. The result is Boolean true (1) or false (0). These are shown in table 1.2

Example for z= x && y, if x=1011 and y=0001 then Z=1, 2nd case if x=1010 and y=0101 then z=0;

For z! =x if x=1111 then z=0;

Operators	**Operation**	**Number of operands**
&&	AND	two
||	OR	two

Table 1.2 Boolean operators

Reduction operators: These operators operate on a single operand. The result is Boolean.

Example y=&x, if x=1010 then y= (1&0&1&0) =0

Operators	**Operation**	**Number of operands**
&	Reduction AND	One
|	Reduction OR	One
~(&)	Reduction NAND	One
~(|)	Reduction NOR	One
^	Reduction XOR	One
~(^)	Reduction XNOR	One
!	Negation	One

Table 1.3 Verilog Reduction logical operators

1.4.2 Relational operators

These are implemented to compare the values of two objects. The result is false (0) or true (1).

Verilog Relational operators

Verilog has a set of Relational operator similar to VHDL. Returns Boolean values false (0) or true (1).

The result can also be of type unknown (X) when any of the operand include don't care or unknown (X) or high impedance. Table 1.4 shows the list of Verilog Relational operators

Example: if (A==B), if the values of A or B contains one or more don't care or Z bits. The value of the expression is unknown.

If A is equal to B, then result of the expression (A==B) is true (1).

If A is not equal to B, then result of the expression (A==B) is false (0).

Operators	**Description**	**Result type**
==	Equality	0,1,X
!=	Inequality	0,1,X
===	Equality Inclusive	0,1
!==	Inequality Inclusive	0,1
<	Less than	0,1,X
<=	Less than equal to	0,1,X
>	Greater than	0,1,X
>=	Greater than equal to	0,1,X

Table 1.5 List of Verilog Relational operators

1.4.3 Arithmetic operators

Arithmetic operators can perform a wide variety of operation, such as addition, subtraction, multiplication and division.

Verilog Arithmetic operators

It is not an extensive type-oriented language. Example y :=(A *B) calculates the values of y as the product of A times B. Table 1.7 shows the Verilog arithmetic operator

Operators	**Description**	**A or B type**	**Y type**
+	Addition A+B	A numeric B numeric	Numeric
-	Subtraction A-B	A numeric B numeric	Numeric
*	Multiplication A?B	A numeric B numeric	Numeric
/	division A/B	A numeric B numeric	Numeric

%	Modulus A%B	A numeric, not real B numeric, not real	Numeric, not real
**	Exponent A**B	A numeric B numeric	Numeric
{,}	Concatenation {A,B}	A numeric, or array B numeric, or array	Same as A

Table1.7 Verilog arithmetic operator

1.4.4 Shift and Rotate operators

A shift left represents multiplication by two, and a shift left represents division by two.

b). Verilog Shift and Rotate operators

It has basic shift operators. These are unary operators i.e., operate on single operand. Example if A=1110, is a 4 bit vectors table 1.8 shows the Verilog shift and rotate operators.

Operation	**Description**	**Operand A Before shift**	**Operand A After shift**
A <<1	Shift A one position left logical	1110	110X
A <<2	Shift A two position left logical	1110	10XX
A >>1	Shift A one position right logical	1110	X111
A >> 2	Shift A two position right logical	1110	XX11

Table 1.8 the Verilog shift operators

Data types

The data or operands used in the language must have several types to match the need for describing the hardware.

2Verilog Data types

There are different types of Verilog data types. Namely

1. Nets
2. Registers
3. Vectors
4. Integer
5. Real
6. Parameters
7. Array

Nets:

These are declared by the predefined word **“wire”.** Nets values are change continuously by the circuits that are driving them. A wire represents a physical wire in a circuit and is used to connect gates or modules. The value of a wire can be read, but not assigned to, in a function or block. Verilog supports 4 values for nets.

Value	Net Definition	Reg
0	Logic 0(false)	Logic 0
1	Logic 1(true)	Logic 1
X	Unknown	Unknown
Z	High impedance	High impedance

Eg. Wire sum; // statement declares a net by name sum.

Wire s1=1’b0; // this statement declares a net by the name of s1; it is initial value 1 bit with value 0.

Registers: Registers store values until they are updated. They are data storage elements. Declared by the predefined word **“reg”** Verilog supports 4 values for registers. As shown in above table.

Eg reg sum_total; // declares a register by the name sum_total.

Vectors:

These are multiple bits. A reg or net can be declared as a vector. Vectors are declared by brackets [].

Eg. Wire [3:0] a=4’b1010;

Reg [7:0] total =8’d12;

Integer: declared by the predefined word **“integer”.** Integers are general-purpose variables. For synthesis they are used mainly loops-indices, parameters, and constants.

Eg. Integer no_bits;//The above statement declares no_bits as an integer.

Real:

Real (floating point) numbers are declared with the predefined word **“real”.** Examples of real values are 2.4, 56.3 5e12.

Eg. Real weight; // the statement declares the register weight as real.

Parameters:

It represents global constants. Declared by the predefined word **“parameter”**

```
Eg. Module comp_genr (x, y, xgty, xlty, xeqy);
Parameter N=3;
Input [n:0] x,y;
Output xgty, xlty, xeqy;
Wire [N:0] sum, xb;
```

Array: there is no predefined word **“array”.** Registers and integers can be used as arrays.

```
Parameter N=4;
Parameter M=3;
Reg signed [M: 0] carry [0:N]
Reg [M: 0] b [0: N];
Integer sum [0: N];
```

The above statement declares an array by the name sum. It has 5 elements, and each element is an integer type.

array carry has 5 elements, and each elements is 4bits. They are in 2’S complement form

The array b has 5 elements, each element is 4 bits. The value of each bit can be 0, 1, X or Z;

1.6 Style (Types) of Descriptions

1.6.1 Behavioral Descriptions

This models the system as to how the outputs behave with inputs.

The definition of Behavioral Description is one where architecture (VHDL) or module (Verilog) includes the predefined word process (VHDL) or always or initial (Verilog).

This description is considered pure behavioral if it does not contain any other features from other styles.

Listings refer class notes.

VHDL Behavioral Description

Verilog Behavioral Description

1.6.2 Structural Descriptions

This model the system as components or gates, this description is defined by the presence of the Keyword component in the architecture (VHDL) or gates construct such as “and”, “or”, or “not” in the module (Verilog).

If the VHDL architecture or the Verilog module consists of only components or gates; this style is coined as pure structural.

Listings refer class notes.

VHDL structural Description

Verilog structural Description

1.6.3 Dataflow Descriptions

It describes how the systems signals flow from the input to the output. The dataflow statements are concurrent; their execution is controlled by events.

Usually, the description is done by writing the Boolean function of the outputs. It should not include any of keywords that identify behavioral, structural, or switch level descriptions.

Listings refer class notes.

VHDL dataflow Description

Verilog dataflow Description

1.6.4 Switch level Descriptions

It is the lowest level of description. The system is described using switches or transistors. The Verilog keywords nmos, pmos, cmos, tran, or tranifo describe the system. VHDL does not have built in switch level primitives, we are constructing packages to include such primitives and attach them to the VHDL module.

Listings refer class notes.

VHDL switch level Description

Verilog switch level Description

1.6.5 Mixed-type Descriptions

It uses more than type. Here we may describe some parts of the system using one type and other parts using another type. Example of Mixed-type Description using both dataflow and behavioral style is explained in the listing.

Listings refer class notes.

VHDL mixed-type Description

Verilog mixed type Description

1.6.6 Mixed-language Descriptions

It is newly added tool for HDL descriptions. The user can write a module in one language (VHDL or Verilog) and invoke or import a construct (entity or module) written in the other language.

Listings refer class notes.

VHDL mixed language Description

Verilog mixed language Description

VERILOG DATA FLOW DESCRIPTION:

Highlights of Data Flow description

Dataflow is a type of hardware description which shows how the signal flows from system inputs to outputs. It uses signal assignment statements which are executed concurrently when an event occurs on the signals on the right side of the statement.

In HDL language, programming is carried out two standard methods.

1. Concurrent program execution: In this method of program execution, all the statements within the program are executed simultaneously.

   The above gate network has two inputs A and B, two outputs Y1 and Y2. The outputs will get evaluated simultaneously whenever an event occurs on either of the inputs A or B or both, assuming the propagation delay of both the gates to be same.

   In order to describe the above hardware we need concurrent program execution where the outputs are updated whenever an event occurs on its inputs, irrespective of the order of statements. All combinational circuits need this style of execution for accurate description of the hardware.

2. Sequential program execution: In this method all the statements are executed sequentially in the order of their appearance.

   An example of hardware that requires this method of program execution is a flip-flop. The data given at D input will be transferred to the output only after the rising or falling edge of the clock. All sequential circuits like flip-flops, counters, registers require this method of program execution.

Structure of Data-Flow Description

A dataflow model specifies the functionality of the system without explicitly specifying its structure. It specifies how the system's signal flow from inputs to the outputs. The description is usually done by writing the Boolean functions of the outputs.

The dataflow statements are concurrent and their execution is controlled by events.
EVENT: An event is a change in the value of a signal, such as a change from 0 to 1 or 1 to 0.

Dataflow description is modeled using **concurrent signal assignment statements** (VHDL) and **continuous signal assignment statements** (Verilog).

Example program1:

VHDL dataflow description	**Verilog dataflow description**
entity system is	module system (I1, I2, O1,O2);
Port (I1, I2 : in bit ; O1, O2 : out bit) ;	input I1, I2;
end;	output O1, O2;
architecture dtf of system is	

begin
    O1 <= I1 and I2 ; --st1
    O2 <= I1 xor I2 ; --st2
end dtf;
	assign O1 = I1 & I2; //st1
	    assign O2 = I1 ^ I2; //st2
	end module

Above example shows HDL code, describing a system using dataflow description. The entity (module) name is system. I1 and I2 are the two inputs and O1 and O2 are the two outputs. St1 and st2 are signal assignment statements which assigns value to the outputs O1 and O2.

SIGNAL DECLARATION:

Input and output signals are declared in the entity (module) as ports. Intermediate Signals (other than input and output signals) are declared using the predefined word **signal** in VHDL and **wire** in Verilog as shown in the below example. In Verilog signals are declared using **reg** when the value of the signal needs to be stored.

```
signal  s1,  s2  :  bit;  --VHDL
wire s1, s2;             // Verilog
```

SIGNAL ASSIGNMENT STATEMENTS:

A signal assignment statement is used to assign a value to a signal. The left hand side of the statement should be declared as a signal. The right hand side can be a signal, a variable, or a constant. **'<='** is a signal assignment operator in VHDL and in verilog predefined word **assign** is used.

Execution of signal assignment statement has two phases. In the above example of system, assume that an event at T_0 occurs on either signal I1 or I2. This event changes the value of I1 from 0 to 1 and also the value of I2 from 0 to 1.

1. **Calculation**: The value of O1 is calculated using the current values of I1 and I2 at time T0. The value 1 and 1=1 is calculated. This is not yet assigned to O1.
2. **Assignment:** The calculated value 1 is assigned to O1 after a delay time. The delay time can be implicitly or explicitly specified. If no delay time is specified, the HDL uses a default, small delay of Δ (delta) seconds.

Continuous signal assignment statements:

A continuous assignment statement is the most basic statement in Verilog dataflow modeling. It is used to drive a value onto a net or assigns a value to a net. It uses the keyword **assign**. It has the following form,

assign LHS_target = RHS_Expression;

```
Example:  wire[3:0] Z, preset, clr;
          assign Z = preset & clr;
```

The target of the continuous assignment is Z and the right hand expression is (preset & clr). The continuous assignment statement executes whenever an event occurs on an operand on the right hand side of the expression, it is evaluated and assigned to the target.

Concurrent signal assignment statements:

One of the primary mechanisms for dataflow modeling in VHDL is the **concurrent signal assignment statement**. It has the following form,

LHS_target <= RHS_expression;

The value computed by the RHS_expression is assigned to the LHS_target. '<=' is called the signal assignment oprator.
Example: C <= A and B;
Right hand side expression A and B is computed and the value is assigned to the LHS_target C.

In HDL dataflow descriptions, concurrent signal assignment statements constitute the major part. In exampleprogram1 both st1 and st2 are concurrent statements. For the execution of concurrent statement to start, an event on the right hand side of the statement has to occur. An event is a change in the value of the signal or a variable. If an event occurs on more than one statement, then all those statements regardless of their order in the architecture (module) are executed concurrently (simultaneously).

CONSTANT DECLARATION AND ASSIGNMENT STATEMENTS:

The value of a constant is constant within the segment of the program where it is visible. A constant in VHDL can be declared using the predefined word ***constant*** and in Verilog it is declared by its type, such as ***time*** or ***integer.***

Ex: *constant* **period**: *time* ; --VHDL
time period ; //verilog

To assign a value to the constant we use assignment operator := in VHDL or = in verilog.

Ex: period := 100ns; --VHDL
Period = 100; //verilog

The above example assigns a value of 100 nanoseconds to the constant period which was declared above. In verilog there are no explicit units of time. 100 means 100 simulation screen time units. The declaration and assignment can be combined in one statement as:

constant **period**: *time* := 100ns; --VHDL
time period = 100 ; //verilog

<u>Conditional signal assignment statement</u>:

The conditional signal assignment statement selects one out of different values for the target signal based on the specified condition.

The typical syntax for this statement is as follows.

target_signal <= expression1 **when** *boolean_condition1* **else**
expression2 **when** *boolean_condition2* **else**
.
.
expression n;

When there is an event on any of the operands present in the boolean_condition or the expression, the execution of when-else starts. The boolean condition1 is evaluated first. If the result is true then expression1 is assigned to the target signal. If the result is false, next condition2 is checked. If the result is true then expression2 is assigned to the target or else condition3 is cheked and so on. If no conditions are true then expression n is assigned to the target_signal.

The target_signal will receive the value of the first expression whose boolean condition is TRUE. If more than one condition is true, the value of the first condition that is TRUE will be assigned.

Though the when-else statement is a concurrent statement, within the body the execution is sequential. Hence the first condition gets the highest priority. This special feature of when-else can be used to describe “priority encoders”.

Example:
```
Z <= A when s0=’0’ and s1=’0’ else
     B when s0=’0’ and s1=’1’ else
     C when s0=’1’ and s1=’0’ else
     D when s0=’1’ and s1=’1’;
```

In this example, the statement is executed any time an event occurs on A, B, C, D, s0 or s1. The first condition(s0=0 and s1=0) is checked, if false, the second condition is checked and so on and when the condition is true the corresponding value is assigned to Z.

The conditional signal assignment will be executed if any of the signals in the conditions or expression change.

```
entity MUX is
        port    (A, B, C, D: in std_logic;
                 SEL: in std_logic_vector (1 down to 0);
                 Z : out std_logic );
        end MUX;
architecture MUX41 of MUX is
begin
        Z <= A when SEL = “00” else
             B when SEL = “01” else
             C when SEL = “10” else
             D;
end MUX41;
```

<u>Selected signal assignment statement</u>:

The selected signal assignment is another concurrent statement in VHDL. It provides selective signal assignment. The syntax is as follows,

```
with choice_expression select
        target_name <= expression1 when choices1,
                       expression2 when choices2,
                       .
                       .
                       expression n when choices n;
```

Whenever an event occurs on a signal in the choice_expression or any expression1, expression2…, the statement gets executed. The choice_expression gets evaluated first and then this value is compared with all the choices simultaneously. When the value matches with any of the choices, the corresponding expression is assigned to the target_signal.

For example if the value of the choice_expression matches with choice1, then expression1 is assigned to the target_signal.

* The choice expression must contain atleast one signal because an EVENT can occur only on a signal. For example if the choice_expression is Y then Y must be a signal. If choice_expression is X+Y, then either of X or Y must be a signal.

The following rules must be followed for choices:

1. All possible values of the choice expression must be covered by the choices **exactly once**. Values not covered explicitly may be covered by an "**others**" choice.
2. The choices must be static expressions. Ex: 5, 5+6 or w+u where w and u are constants which are already declared.
3. The choices must be mutually exclusive.

Example 2.1: Data-flow description of a half adder

Truth table

Logic diagram

Logic symbol

VHDL and Verilog code

Simulation waveform

ASSIGNING A DELAY TIME TO THE SIGNAL-ASSIGNMENT STATEMENT

To assign a delay time to a signal-assignment statement, we use the predefined word **after** in VHDL or # (delay time) in Verilog.

```
s1 <= sel and b after 10ns;      --VHDL for 10ns delay
assign #10 s1 = sel and b;       //Verilog for 10 screen units delay
```

Note: In Verilog, we cannot specify the units of delay time. The delay is in simulation screen unit time.

Example 2.2: 2×1 Multiplexer with Active Low Enable

Truth table

Logic diagram

Logic symbol

VHDL and Verilog code

Simulation waveform