INTRODUCTION TO VERILOG (Contd..)

EC39004 : VLSI LABORATORY

14th January 2019

Register data type

In Verilog, a "register" is a variable that can hold a value; unlike a "net" that is continuously driven and cannot hold a value

Does not necessarily mean that a reg variable will map to a hardware register during synthesis

Combinational circuit specifications also use register type variables; depends upon the Verilog construct used

Register data types supported by Verilog:

i) **reg** : Most widely used

ii) **integer** : Used for loop counting

iii) **real** : Used to store floating point numbers

iv) **time** : Keeps track of simulation time (not used in synthesis)

Reg Data Type

Default value of a "reg" data type is x (unknown/ don't-care)

It can be assigned a value in synchronism with a clock or event otherwise

The declaration explicitly specifies the size (default is 1-bit) eg.,

reg x, y; reg [15:0] bus; / / a 16-bit bus

Treated as an unsigned number in arithmetic expression

Must be used when we model actual sequential hardware elements like counters, shift registers etc.

Integer Data Type

It is a general purpose data type used for manipulating quantities

More convenient to use in situations like loop counting than "reg" data type

It is treated as a 2's complement signed integer in arithmetic expressions

Default size is 32 bits; however the synthesis tool tries to determine the size using data flow analysis

Example:

```
wire [15:0] x,y;
integer c;
c=x+y;
```

Size of "c" can be deduced by the synthesis tool to be 17 (16 bits + 1 carry bit)

Real data type

Used to store floating point numbers

When a real value is assigned to an integer, the real number is rounded off to the nearest integer

Example:

```
real e, pi;
initial
  begin
  e=2.718;
  pi=314.159e-2
  end
```

```
integer x;  initial
x = pi; / /  Gets value 3
```

Time Data Type

In Verilog, the simulation is carried out with respect to a logical clock called simulation time

The "time" data type can be used to store simulation time

The system function "$time" gives the current simulation time

Example:

```
time curr_time;  initial
----
curr_time = $time;
```

Vectors

Net or reg type variable can be declared as vectors of multiple bit-widths
→ If bit width is not specified, default size is 1-bit

Vectors are declared by specifying a range [range1: range2] where "range1" is the positional index of the MSB and "range2" is the positional index of the LSB eg., wire [31:0] a,b; / / 32-bit bus

Parts of a vector can be addressed and used in an expression, eg.,

```
assign sum=a[10:7]+b[3:0];
```

Specifying Constant Values

A constant value may be specified in either the sized or unsized form

Syntax of the sized form:

<size>'<base><number>

Variables of the type "integer" and "real" are typically expressed in unsized form

<u>Examples</u>:

4'b0101 / / 4-bit binary no: 0101

1'b0 / / Logic 0 (1 bit)

12'hB3C / / 12- bit binary number 1011 0011 1100 expressed in hexadecimal

12'h8xF / / 12- bit number 1000 xxxx 1111

25 / / Signed number in 32-bits (size not specified)

Parameters

A parameter is a constant with a given name

We cannot specify the size of a parameter

The size gets decided from the constant value itself; if size is not specified, it is taken to be 32 bits

Makes Verilog program more readable

Examples:

parameter HI= 25, LO=5; parameter up=2'b00, down=2'b11;

Predefined Logic Gates in Verilog

Verilog provides a set of predefined logic gates

→ Can be instantiated within a module to create a structured design

→ The gates respond to logic values (0,1, x or z) in a logical way

2 i/p AND	2 i/P OR	2 i/p XOR
0 & 0 = 0	0 | 0 = 0	0 ^ 0 = 0
0 & 1 = 0	0 | 1 = 1	0 ^ 1 = 1
1 & 1 = 1	1 | 1 = 1	1 ^ 1 = 0
1 & x = x	1 | x = 1	1 ^ x = x
0 & x = 0	0 | x = x	0 ^ x = x
1 & z = x	1 | z = x	1 ^ z = x
z & x = x	z | x = x	z ^ x = x

List of Primitive Gates

and (out, in1,in2,in3,in4,…);
nand (out, in1,in2,in3,in4,…);
or (out, in1,in2,in3,in4,…);
nor (out, in1,in2,in3,in4,…);
xor (out, in1,in2,in3,in4,…);
xnor (out, in1,in2,in3,in4,…);
not (out, in);
buf (out, in);

→ **These are logical primitives; not physical primitives**

→ **Instantiated in a design through positional association**

→ **When instantiating a gate an optional delay may be specified; eg.,**

and #5 (out, in1,in2); → they make meaning only during simulation; logic synthesis tools ignore time delays

Gates with Tristate Control

bufif1 (out, in,ctrl);

bufif0 (out, in,ctrl);

notif1 (out,in, ctrl);

notif0 (out,in, ctrl);

<u>Rules for instantiating gates</u>:

•The output port must be connected to a net,

eg., a wire; an output signal is a "wire" by default unless explicitly declared a s a register

•The input ports may be connected to nets or register type variables

•They have a single o/p, but can have any no: of i/ps (except NOT and BUF)

Verilog Operators

ARITHMETIC OPERATORS

--- may be unary or binary

+ Unary (sign) plus

- Unary (sign) minus

+ Binary plus (add)

- Binary minus (subtract)

*** Multiply**

/ Divide

% Modulus

**** Exponentiation**

Examples:

-(b+c)

(a-b)+(c*d)

(a+b)/(a-b) a%b

a3**

Verilog Operators (contd..)

LOGICAL OPERATORS

!	Logical Negation
&&	Logical AND
||	Logical OR

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

Examples:

```
(done && ack)
(a||b)
!(a&&b)
((a>b)||( c==0))
((a>b)&& !(b>c))
```

- The value 0 is treated as logical FALSE while any non- zero value is treated as TRUE
- Logical operators return either 0 (FALSE) or 1 (TRUE)

Verilog Operators (contd...)

RELATIONAL OPERATORS

Relational operators operate on numbers and return a Boolean value (true or false)

!=	Not equal
==	Equal
>=	Greater or equal
<=	Less or equal
>	Greater
<	Less

Examples:

(a!=b) ((a+b)==(c+d))
((a>b)&&(c<d)) (count<=0)
→ assign outp=(p==4'b1111);

Verilog Operators (contd…)

- **BITWISE OPERATORS**

  Bitwise operators operate on bits, and returns a value that is also a bit

~ bitwise NOT

& bitwise AND

| bitwise OR

^ bitwise XOR

~^ bitwise XNOR

Examples:

```
assign f1=~a|b;
assign f2=(a&b)|(b&c)|(c&a);
```

Verilog Operators (contd…)

- **REDUCTION OPERATORS**

  → Reduction operators accept a single word operand and produce a single bit as output

  → They are unary operators

& bitwise AND

| bitwise OR

~& bitwise NAND

~| bitwise NOR

^ bitwise XOR

~^ bitwise XNOR

```
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
```

Verilog Operators (contd…)

- **SHIFT OPERATORS**

>> shift right

<< shift left

>>> arithmetic shift right

```
module shifting(
 input signed [7:0] a, output [7:0] b, c, d, e);
 assign b=a<<2;
 assign c=a>>2;
 assign d= a>>>2;
 assign e= a>>>3;
endmodule
```

Result:

a=11000111; b=00011100; c=00110001; d=11110001; e=11111000

Verilog Operators (contd…)

- **CONDITIONAL OPERATOR**
- **Syntax:**

<condition_expression> ? <true_expression> :
<false_expression>

Example:

```
wire a,b,c;  wire [7:0]  x,y,z;
assign a=(b>c) ? b : c;
assign z=(x==y) ? x+2 : x-2;
```

Verilog Operators (contd…)

- **CONCATENATION OPERATOR**

  Joins together bits from two or more comma separated expressions

  {<expr1>, <expr2>,…, <exprn>}
- **REPLICATION OPERATOR**

  Joins together *n* copies of an expression *m*, where *n* is a constant →

  {n{m}}

Example:

```
assign f = {a[2:0],1'b1,c,d[7:4]};
assign g= {a[3:2],{2{a[3:1]}},a[7:6]};
assign h= {a[5:4],{3{a[2]}},a[6]};
```

Assignment (Implement barrel shifter)