A Seminar on

SYSTEM VERILOG -DATA TYPES

PRESENTED BY:

RAIKODE ABHISHEK

DEV'S VLSI TRANING INSTITUTE

CONTENTS

INTRODUCTION

- System Verilog is a **hardware description** and **verification language** used in digital design.
- It extends Verilog by adding many advanced features, including **rich data types**.
- Data types help to store, access, and **manipulate** (to modify or change) different kinds of data (numbers, logic, structures, etc.).
- These data types improve **code readability**, **simulation accuracy**, and **verification strength.**
- System Verilog supports both **2-state** (0,1) and **4-state** (0,1,x,z) types.

- **What is a data-type ?**
  - **A storage format** having a specific **range or type** is called data type.
  - A data type defines **what kind of value a variable can hold**.
  - Datatypes in Verilog are not sufficient to develop **efficient testbenches** and **testcases.**
  - Data types also divided into **static and dynamic**.
- **Static vs dynamic:**
  - **Static:** Fixed size at compile- time (eg.,reg [7:0])
  - **Dynamic:** Size can change at runtime (eg., dynamic arrays, queues)

- **Verilog Data- types are classified into two types**
  - **Net types** : (no connection and no storage).
  - Used in **continuous assignments (assign)** or as module ports .
  - Cannot be assigned **inside procedural blocks like (initial or always).**
    - **Wire ,tri**
    - **Wand , Wor**
    - **supply0, supply1**
    - **trio, tri1**
    - **tireg**
    - **triand,trior**
    - **uwire**

- **Variable data types:**
  - Variables are used to **store values that can change during simulation**.
  - They are used inside **procedural blocks like initial and always.**
  - Can be assigned using**= (blocking)** or <= **(non-blocking**.)
  - **Reg**
  - **Integer**
  - **Real**
  - **Time**
  - **Realtime**

- **System Verilog data types** :

Bulit in data types are :

- Bit , logic , reg , int , integer , byte , shortint , longint
- Real, shortreal, time, realtime
- String and Arrays

- **User- defined data types :**

- Typedef
- Enum
- Struct
- union

```
15
16 // UNSIGNED DATA-TYPES (SIZE-NOT FIXED)
17
18 // BIT-DATA TYPE // DEFAULT VAULE 0 // SIZE - NOT FIXED  // 2-STAGE [0&1] // BIT=UNSIGNED MEANS IT WILL GIVE 2'S COMPLEMENT ANSWER
19 module tb;
20   bit [1:0] a;
21   initial
22     begin
23       a=2;
24       $display("a=%b",a);
25   end
26 endmodule
27
28 // UNSIGNED - (SIZE NOT FIXED )
29 // REG - DATA TYPE //  DEFAULT VAULE X // SIZE - NOT FIXED  // 4-STAGE [01XZ] // REG=UNSIGNED MEANS IT WILL GIVE 2'S COMPLEMENT ANSWER
30 module tb;
31   reg [3:0] a;
32   initial
33     begin
34       a=2;
35       $display("a=%b",a);
36   end
37 endmodule
38
39
40 // LOGIC-DATA TYPE // DEFAULT VAULE X // SIZE - NOT FIXED  // 4-STAGE [01XZ] // LOGIC=UNSIGNED MEANS IT WILL GIVE 2'S COMPLEMENT ANSWER
41 module tb;
42   logic [3:0] a;
43   initial
44     begin
45       a=2;
46       $display("a=%b",a)
47   end
48 endmodule
49
50 // UNSIGNED 4-STAGE FIXED DATA-TYPE
51 // TIME-DATA TYPE // DEFAULT VAULE X // SIZE - 64  // 4-STAGE [01XZ] // TIME=UNSIGNED MEANS IT WILL GIVE 2'S COMPLEMENT ANSWER
52 module tb;
53   time  a;
54   initial
55     begin
56       a=2;
57       $display("a=%b",a);
58   end
59 endmodule
60
```

```
63 // SIGNED DATA TYPES - 2 STAGE DATA TYPES // to this data types we can"t assign any size becasue they are bulit in defalut
64 // INT-DATA TYPE // DEFAULT VAULE 0 // SIZE -32 // 2-STAGE [0&1] // INT=SIGNED EX: -2=-2 & 2=2
65 module tb;
66   int a;
67   initial
68     begin
69       a=2;
70       $display("a=%b",a);
71   end
72 endmodule
73
74
75 // SHORTINT-DATA TYPE // DEFAULT VAULE 0 // SIZE =16  // 2-STAGE [0&1] // SHORTINT=SIGNED EX: -2=-2 & 2=2
76 module tb;
77   shortint a;
78   initial
79     begin
80       a=2;
81       $display("a=%b",a);
82   end
83 endmodule
84
85
86 // LONGINT-DATA TYPE // DEFAULT VAULE 0 // SIZE =64  // 2-STAGE [0&1] // LONGINT=SIGNED EX: -2=-2 & 2=2
87 module tb;
88   longint a;
89   initial
90     begin
91       a=2;
92       $display("a=%b",a);
93   end
94 endmodule
95
96
97 // BYTE-DATA TYPE // DEFAULT VAULE 0 // SIZE =8 // 2-STAGE [0&1] // BYTE=SIGNED EX: -2=-2 & 2=2
98 module tb;
99   byte a;
100  initial
101    begin
102      a=2;
103      $display("a=%b",a);
104  end
105 endmodule
:
```

```
106
107 // REAL-DATA TYPE // DEFAULT VAULE 0 // SIZE =64 // 2-STAGE [0&1] // REAL=SIGNED EX: -2=-2 & 2=2
108 module tb;
109   real a;
110   initial
111     begin
112       a=2;
113       $display("a=%b",a);
114   end
115 endmodule
116
117 // SHORTREAL-DATA TYPE // DEFAULT VAULE 0 // SIZE =32 // 2-STAGE [0&1] // SHORTREAL=SIGNED EX: -2=-2 & 2=2
118 module tb;
119   shortreal a;
120   initial
121     begin
122       a=2;
123       $display("a=%b",a);
124   end
125 endmodule
126
127
128 // REALTIME-DATA TYPE // DEFAULT VAULE 0 // SIZE =64 // 2-STAGE [0&1] // REALTIME=SIGNED EX: -2=-2 & 2=2
129 module tb;
130   realtime a;
131   initial
132     begin
133       a=2;
134       $display("a=%b",a);
135   end
136 endmodule
137
138 // INTEGER-DATA TYPE // DEFAULT VAULE X // SIZE = 32 // 4-STAGE [01XZ] // INTEGER=SIGNED EX: -2=-2 & 2=2
139 module tb;
140   integer a;
141   initial
142     begin
143       a=2;
144       $display("a=%b",a);
145   end
146 endmodule
147
```

- **What is a "String" ?**
  - In System Verilog, a **string is a dynamic**, **variable-length** sequence of characters.
  - It is a built-in data type used to **store and manipulate(to change or modify)** the text.
  - Strings are **non-synthesizable** and used mainly in **testbenches.**
- **How to create a "string" in Verilog ?**

```
// To create the string in verilog we use 1:8*10
// ( 8 is nothing but it is fixed character size and 1:10 means the total character should print in given string  RHS to LHS )
 module tb;
   reg[1:8*10] a;
  initial begin
    a="verilog system ";
    $display("%s",a);
  end
 endmodule

// output = log system
```

- **How to create a string in system Verilog?**

```
// sv string
module tb;
  string a;
  initial begin
    a="verilog";
    $display("%s",a);
  end
endmodule
//output = verilog
```

- **Operators used with string :**
  - Concatenation
  - Replication
  - Logical equality (==)
  - Logical inequality (!=)
  - Comparison (<,>,=,>=,<=)
  - Index

```
// concantenation
module tb;
string a,b,c;
initial begin
    a="abhishek";
    b="chinna";      // combining the no of strings in single string
   // c={a,b};
    $display("%s",{a,b});
    end
    endmodule
   // output - abhishekchinna



// replecation
module tb;
string a,b,c;
initial begin
    a="abhishek";
    b="chinna";      // combining the given strings amd it will repeate 2 times
    c={2{a,b}};
    $display("%s",c);
    end
    endmodule

//output - abhishekchinnaabhishekchinna



// logical equality (==)
module tb;
string a,b;
initial begin
    a="abhishek";
 //a="chinna";
    b="chinna";
    if(a==b)
        $display ("a=%s,b=%s",a,b); //equal means it will give a=chinna,b=chinna
else
         $display ("notequal");    // notequal means it will give not equal
end
endmodule

// output1 -  a=chinna,b=chinna
// output2 - notequal
```

```
// logical inequality (!=)

module tb;
string a,b;
initial begin
    a="abhishek";
    b="chinna";
   //a="chinna"
   if(a!=b)
     $display ("a=%s,b=%s",a,b);// not equal means it will print a and b
else
         $display ("equal");  // equal means it will output equal
end
endmodule

// output1 -  a=abhishek,b=chinna
// output2 - equal



//  comparision (<,>,=,>=,<=)
module tb;
string a,b;
initial begin
    a="abhishek";   // it will compare  starting letter of 2 strings a and c if both are equal then it will compare next letter or character
    b="chinna";
    $display("a=%d",a);   // convert string into decimal
    $display("b=%d",b);  //  // convert string into decimal
    if (a>b)
$display ("a=%s,b=%S"a,b);
    else
$display ("a lessthan b");
end
endmodule

//output -   a=1633839209
//           b=109300096593505
//           a lessthan b


module tb;
string a,b;
initial begin
    a="abhi";
    b="chinna";
   if (a<b)
//if (a<=b)             lessthan or equal to
//if (a>=b)             greaterthan or equal to
//if (a==b)             equal
      $display ("a=%s,b=%s",a,b);
    else
      $display ("a greaterthan b");
end
endmodule
```

- **Commonly used string methods**
- Length
- Toupper
- Tolower
- Putc
- Getc
- Compare
- Icompare
- Substring

```
13
14 // length  - it will display the total length of the string
15 module tb;
16 string a,b;
17 initial begin
18 a="abhi";     //  0123   the length of the string is 4
19 b="chinna";   //  012345   the length of the string is 6
20 $display ("%d",a.len);    // we have to use  (string_varaible.len)
21 $display ("%d",b.len);
22 end
23 endmodule
24
25 // output - 4 and 6
26
27
28
29 // toupper and tolower
30 module tb;
31 string a,b;
32 initial begin
33 a="abhi";
34 b="ABHI";
35   $display ("%s",a.toupper);  // for CAPITAL letters we use toupper
36   $display ("%s",b.tolower);  // for small letters we use tolower
37 end
38 endmodule
39
40 // output a=ABHI and b=abhi
41
42
```

```
43
44 // getc
45 module tb;
46 string a,b;
47 initial begin
48 a="abhi"; // 0123
49   $display ("%s",a.getc(2)); // if we use getc the character value  will display in output
50 end
51 endmodule
52 //  output - h
53
54
55 // putc
56 module tb;
57 string a,b;
58 initial begin
59 a="abhi chinna ";  // space also one character here
60   a.putc(6,"c");  // if we use putc the character replacement will happen here
61                   // in 6th position i want c
62   $display ("%s",a);
63 end
64 endmodule
65
66 // output - abhi ccinna
67
68
69
70 // substring
71
72  module tb;
73 string a,b;
74 initial begin
75 a="abhi chinna ";  // here space also one character
76   $display ("%s",a.substr(0,7)); // substring is used to display how many characters we need
77 end
78 endmodule
79
80 // output - abhi chi
81
82
```

```
83
84
85 // COMPARE (case sensitive ) - it will compare two  strings and gives the output
86 //    0 - if both strings are equal ,
87 //    1 - first string is greaterthan second string
88 //   -1 - first string is lesserrthan second string
89 module tb;
90   string a="vlsi";              // output =0
91   string b="vlsi";
92
93   // string a="vlsi dv ";       // output =1
94   // string b="vlsi";
95
96  // string a="vlsi";
97  // string b="vlsi dv ";       // output =-1
98
99
100  // string a="VLSI";           // output = -1
101   // string b="vlsi";
102
103
104   initial
105     begin
106       $display("%d",a.compare(b));
107       //$display("%d",b.compare(a));
108     end
109 endmodule
110
111
112
113
114 // I_COMPARE (case insensitive) - it will compare two  strings and gives the output
115 //    0 - if both strings are equal ,
116 //    1 - first string is greaterthan second string
117 //   -1 - first string is lesserrthan second string
118 module tb;
119   string a="vlsi";               // output =0
120   string b="vlsi";
121
122   // string a="vlsi dv ";        // output =1
123   // string b="vlsi";
124
125   // string a="vlsi";
126   // string b="vlsi dv ";        // output =-1
127
128
129   // string a="VLSI";           // output = 0
130   // string b="vlsi";
131
132
133   initial
134     begin
135       $display("%d",a.icompare(b));
136       //$display("%d",b.icompare(a));
137     end
138 endmodule
```

- **What is an Enum ?**
  - **Enum - enumeration** is a user defined data type.
  - readability , reusability , helps in FSM coding.
- **methods in Enum:**

- previous
- next
- first
- last
- num
- previous(n)
- next(n)

```
4
5  module tb;
6    enum {a,b,c} sv;
7    initial begin
8      $display("a=%0d,b=%0d,c=%0d",a,b,c); // for index value (for decimal values we use %d)
9      $display("a=%0s,b=%0d,c=%0d",sv.name,b,c); // for index element (for character we use %s)
10   end
11 endmodule
12
13 // output1 : a=0,b=1,c=2
14 // output2 : a=a,b=1,c=2
15
16
17
18 // methods in enum:
19 //previous , next , first , last, num, previous(n), next(n)
20
21
22 // previous
23 module tb;
24   enum {a,b,c,d,e,f} sv;  // by defalut a variable become reference
25   initial begin
26     sv=sv.prev;
27     $display("%d",sv);
28     $display("%s",sv.name);
29   end
30 endmodule
31 // output1 = 5
32 // output1 = f
33
```

```
36 // next
37 module tb;
38   enum {a,b,c,d,e,f} sv;  // by defalut a variable become reference
39   initial begin
40     sv=sv.next;
41     $display("%d",sv);
42     $display("%s",sv.name);
43   end
44 endmodule
45 // output1 = 1
46 // output1 = b
47
48
49
50 // first
51 module tb;                    // by defalut a variable become reference
52   enum {a,b,c,d,e,f} sv;
53   initial begin
54     sv=sv.first;
55     $display("%d",sv);
56     $display("%s",sv.name);
57   end
58 endmodule
59 // output1 = 1
60 // output1 = b
61
62
63 // last
64 module tb;
65   enum {a,b,c,d,e,f} sv;
66   initial begin
67     sv=sv.last;
68     $display("%d",sv);
69     $display("%s",sv.name);
70   end
71 endmodule
72 // output1 = 5
73 // output1 = f
74
75
```

```
78
79 //prev(N)
80 module tb;
81   enum {a,b,c,d,e,f} sv;
82   initial begin
83     sv=sv.prev(4);
84     $display("%d",sv);
85     $display("%s",sv.name);
86   end
87 endmodule
88 // output1 = 2
89 // output1 = c
90
91
92
93 // next(N)
94 module tb;
95   enum {a,b,c,d,e,f} sv;
96   initial begin
97     sv=sv.next(4);
98     $display("%d",sv);
99     $display("%s",sv.name);
100  end
101 endmodule
102 // output1 = 4
103 // output1 = e
104
105
106
107 //num means the total elements values
108 module tb;
109   enum {a,b,c,d,e,f} sv;
110   initial begin
111
112     $display("%d",sv.num);
113
114  end
115 endmodule
116 // output = 6
~
```

- **Struct in System Verilog :**
  - Struct is a **user-defined data type** that groups different types under one name.
  - Each member has **its own memory**.
  - You can **access all members at the same time**.
  - Can be **packed or unpacked**.

  - Can be **packed** or **unpacked**:
  - packed struct → bit-level, fixed size, static types only.
  - unpacked struct → flexible, allows dynamic types (like string, queue).

- **Union in System Verilog** :
  - Union is **user-defined type** where all members share the **same memory.**
  - Only **one member** is valid at a time.
  - Used to save memory (like in hardware).
  - Can be **packed or unpacked.**
  - Cannot use **dynamic types** (string , queue , class, etc).

```
1 // STRUCTURE
2 // A struct is a user-defined data type that groups different data types under one name.
3 // All members get separate memory and can be used at the same time.
4
5
6 //unpacked struct(by default),works with string also
7 module tb;
8   struct {reg [1:0]a;integer b;string s;} sv; // struct is structure used to combine different data types in single formate and sv is a struct name
9   initial                                     // a,b,s are the data-type names
10    begin
11      sv.a=2'b0x;                           // assigned some values
12      sv.b=3;
13      sv.s="dv";
14      $display("a=%b,b=%0d,s=%s",sv.a,sv.b,sv.s);
15    end
16 endmodule
17
18 //output - a=0x,b=3,s=dv
19
20
21
22 //packed struct,NOT - works with string.
23 module tb;
24   struct packed {reg [1:0]a;integer b;string s;} sv; // struct is structure used to combine different data types in single formate and sv is a struct name
25   initial                                     // a,b,s are the data-type names
26     begin
27       sv.a=2'b0x;                           // assigned some values
28       sv.b=3;
29       sv.s="dv";
30       $display("a=%b,b=%0d,s=%s",sv.a,sv.b,sv.s);
31     end
32 endmodule // output - error
33
```

```
35
36 // union
37 // A union is a user-defined data type where all members share the same memory.
38 // union unpacked or packed  won't work with string
39 // Packed union  bit-level layout, all members must be the same size in bits.we can't use dynamic data types. (string,dynamic array,queue,class)
40 module tb;
41    union  {reg [31:0]a;integer b;} sv;
42   initial
43     begin
44       sv.a=2'b0x;
45       sv.b=3;
46       //sv.s="1234";
47
48       $display("a=%b,b=%0d",sv.a,sv.b);
49     end
50 endmodule
~
~
~
```

- **What is an Event ?**
  - An event in System Verilog is a way to **communicate or signal between two processes**.
  - An event is like **a signal or flag** used to inform one block of code that something has happened in another block.
  - **One block triggers the event** and **another block wait until triggering happens**.
  - when the same delays in both blocks **wait keyword or ->>** symbol used to get all blocks **output.**
  - we can also **manually clear** an event using **ev.reset();** if needed.

```
// event
// An event in SystemVerilog is a way to communicate or signal between two processes.
// An event is like a signal or flag used to inform one block of code that something has happened in another block.
// One block triggers the event and another block wait until triggering happens.
// when same delays in both blocks wait keyword or ->> symbol to get all blocks output

//case-1

module en;
  event ev;

  initial begin
    fork

      begin
        #10;
        $display($time,"Triggering The Event ev"); // triggering block
        ->ev;
      end

      begin
        $display($time,"Waiting for the Event ev to trigger"); // waiting block

        @(ev.triggered);
        $display($time,"Event ev triggered");
      end
    join
  end
endmodule


// output-  0Waiting for the Event ev to trigger
//          10Triggering The Event ev
//           10Event ev triggered
```

```
//case-2

module en;
  event ev;

  initial begin
    fork

      begin
        #10;
        $display($time,"Triggering The Event ev"); // triggering block
        ->ev;
      end

      begin
        $display($time,"Waiting for the Event ev to trigger"); // waiting block
         #20;
        @(ev.triggered);
        $display($time,"Event ev triggered");
      end
    join
  end
endmodule

// output -  0Waiting for the Event ev to trigger
//          10Triggering The Event ev
```

```
// case - 3
module en;
  event ev;

  initial begin
    fork

      begin
        #10;
        $display($time,"Triggering The Event ev"); // triggering block
        //->ev;
        ->>ev;
      end

      begin
        $display($time,"Waiting for the Event ev to trigger"); // waiting block
         #10;
         //wait(ev.triggered);
         //@(ev.triggered);
        $display($time,"Event ev triggered");
      end
    join
  end
endmodule

// output-  0Waiting for the Event ev to trigger
//          10Triggering The Event ev
//           10Event ev triggered

~
~
```

- **What is a typedef?**

Typedef is used **to create a new (alias) for an existing data type**.
It make code **shorter, easier to read, reusable**.

```
typedef int my_int;  // my_int is a new data type in the code . We can create our own datatype.
module tb;
 my_int a;     // same as 'int a;'
 initial begin
  a = 100;
  $display("a = %0d", a);
 end
endmodule
```

THANK YOU