Empty Book Template

Ryan Hou

Invalid Date

Table of contents

Preface **4**

Resources **5**

1 Introduction **6**
1.1 Perspective . . . 6
1.2 High Level Ideas . . . 6

2 Verilog - Data Types **7**
2.1 Data Types . . . 7
2.2 Scalar/Vector . . . 7
2.3 Arrays . . . 7
2.4 Net Types & Strength . . . 8
2.5 References . . . 8

3 Verilog - Building Blocks **9**
3.1 Modules . . . 9
3.2 Assign . . . 9
3.3 Operators . . . 9
3.4 Concatenation . . . 10
3.5 Always Block . . . 10
3.6 Initial Block . . . 10
3.7 Generate . . . 10

4 Verilog - Behavioral Modeling **11**
4.1 Blocking / Non-Blocking . . . 11
4.2 Functions & Tasks . . . 11
4.3 Delays . . . 11

7 SystemVerilog - Data Types **14**
7.0.1 Enumerations . . . 14
7.1 Arrays . . . 14
7.1.1 Static Array . . . 14
7.1.2 Dynamic Array . . . 14
7.1.3 Associative Array . . . 15
7.1.4 Queue . . . 15

7.1.5 Packed Array . . . 15
7.1.6 Unpacked Array . . . 16
7.2 Structs . . . 16
7.3 User-Defined Types . . . 17
7.3.1 Alias . . . 17

8 SystemVerilog - Control Flow **18**
8.1 Functions . . . 18
8.2 Tasks . . . 18

10 SystemVerilog - Interface **20**
10.1 Clocking Block . . . 21

14 Synthesis **26**
14.1 Verilog . . . 26

15 Summary **27**

References **28**

Preface

My notes on ???.

Resources

Some relevant resources:

- Resource Name

Textbooks:

- Book 1

1 Introduction

1.1 Perspective

> Note 1: Definition - Some definition
>
> **Term** is defined as blah blah blah...

This note does ...

1.2 High Level Ideas

2 Verilog - Data Types

2.1 Data Types

Almost all data types can only have one of four different values (0, 1, X, Z) except for `real` and `event` types.

(Note difference between logic and bit in SystemVerilog, where bit is a two-state variable)

`wire`

`reg`

Non-synthesizable types:

`integer` - general purpose 32bit int. Not synthesizable, good for loop counters, simulation tasks, etc.

`time` - unsigned 64b for storing time quantities

`realtime` - stores time as floating point

`real` - float

Strings are stored in `reg`, using 1 byte per char

2.2 Scalar/Vector

Note that part selection is inclusive from [high:low]

2.3 Arrays

```
reg        y1 [11:0];        // y is an scalar reg array of depth=12, each 1-bit wide
wire [0:7] y2 [3:0]          // y is an 8-bit vector net with a depth of 4
reg  [7:0] y3 [0:1][0:3];    // y is a 2D array rows=2,cols=4 each 8-bit wide
```

2.4 Net Types & Strength

2.5 References

- ChipVerify - Verilog

3 Verilog - Building Blocks

3.1 Modules

3.2 Assign

3.3 Operators

```
// Arithmetic Operators
a + b
a - b
a * b
a / b
a % b
a ** b

// Relational Operators

// Equality Operators
a === b // a equal to b, including x and z
a !== b // a not equal to b, including x and z
a == b  // a equal to b, result can be unknown
a != b  // a not equal to b, result can be unknown

// Logical Operators


// Bitwise


// Shift
```

3.4 Concatenation

Concatenation

Replication

Sign extension

3.5 Always Block

3.6 Initial Block

There are mainly two types of procedural blocks in Verilog - initial and always

initial block is not synthesizable

`$finish`

3.7 Generate

4 Verilog - Behavioral Modeling

4.1 Blocking / Non-Blocking

4.2 Functions & Tasks

4.3 Delays

Inter-assignment Delays:

```
// Delay is specified on the left side
#<delay> <LHS> = <RHS>
```

Intra-assignment Delays:

```
// Delay is specified on the right side
<LHS> = #<delay> <RHS>
```

5

6

7 SystemVerilog - Data Types

logic

bit

byte

int

string

7.0.1 Enumerations

```
enum          {RED, YELLOW, GREEN}         light_1;         // int type; RED = 0, YELLOW = 1
enum bit[1:0] {RED, YELLOW, GREEN}         light_2;         // bit type; RED = 0, YELLOW = 1

// A custom data-type can be created so that the same data-type may be used to declare other

typedef enum {TRUE, FALSE} e_true_false;
e_true_false  answer;
answer = TRUE;
```

7.1 Arrays

7.1.1 Static Array

7.1.2 Dynamic Array

```
[data_type] [identifier_name]  [];

bit [7:0]   stack [];       // A dynamic array of 8-bit vector
string      names [];       // A dynamic array that can contain strings
```

```
int     array [];

initial
array = new [3];
// This creates one more slot in the array, while keeping old contents
array = new [array.size() + 1] (array);
```

Methods:

size()

delete()

7.1.3 Associative Array

```
int         m_data [int];           // Key is of type int, and data is also of type int
int         m_name [string];        // Key is of type string, and data is of type int

m_name ["Rachel"] = 30;
m_name ["Orange"] = 2;

m_data [32'h123] = 3333;
```

7.1.4 Queue

```
int     m_queue [$];        // Unbound queue, no size


m_queue.push_back(23);      // Push into the queue

int data = m_queue.pop_front(); // Pop from the queue
```

7.1.5 Packed Array

There are two types of arrays: packed and unpacked

A packed array is guaranteed to be represented as a contiguous set of bits.

```
bit [3:0]   data;           // Packed array or vector
logic       queue [9:0];    // Unpacked array
```

Example of multi-dimensional packed array:

```
bit [2:0][3:0][7:0]     m_data;     // An MDA, 12 bytes

initial begin
    // 1. Assign a value to the MDA
    m_data[0] = 32'hface_cafe;
    m_data[1] = 32'h1234_5678;
    m_data[2] = 32'hc0de_fade;
end
```

7.1.6 Unpacked Array

7.2 Structs

```
// Structures -> a collection of variables of different data types
struct {
    byte    val1;
    int     val2;
    string  val3;
} struct_name;

// use typedef to actually make it a type

typedef struct {
    string fruit;
    int    count;
    byte     expiry;
} st_fruit;
```

Packed vs unpacked struct. Struct is unpacked by default

A packed structure is a mechanism for subdividing a vector into fields that can be accessed as members and are packed together in memory without gaps.

```
typedef struct packed {
  bit [3:0] mode;
  bit [2:0] cfg;
  bit       en;
} st_ctrl;
```

7.3 User-Defined Types

```
// Declare an alias for this long definition
typedef unsigned shortint           u_shorti;
typedef enum {RED, YELLOW, GREEN}   e_light;
typedef bit [7:0]                   ubyte;
```

7.3.1 Alias

alias keyword

8 SystemVerilog - Control Flow

8.1 Functions

8.2 Tasks

9

10 SystemVerilog - Interface

Example

```
interface apb_if (input pclk);
    logic [31:0]    paddr;
    logic [31:0]    pwdata;
    logic [31:0]    prdata;
    logic           penable;
    logic           pwrite;
    logic           psel;
endinterface
```

Port directions:

```
interface myBus (input clk);
  logic [7:0]  data;
  logic      enable;

  // From TestBench perspective, 'data' is input and 'write' is output
  modport TB  (input data, clk, output enable);

  // From DUT perspective, 'data' is output and 'enable' is input
  modport DUT (output data, input enable, clk);
endinterface
```

Example usage:

```
module dut (myBus busIf);
  always @ (posedge busIf.clk)
    if (busIf.enable)
      busIf.data <= busIf.data+1;
    else
      busIf.data <= 0;
endmodule
```

```
// Filename : tb_top.sv
module tb_top;
  bit clk;

  // Create a clock
  always #10 clk = ~clk;

  // Create an interface object
  myBus busIf (clk);

  // Instantiate the DUT; pass modport DUT of busIf
  dut dut0 (busIf.DUT);

  // Testbench code : let's wiggle enable
  initial begin
    busIf.enable  <= 0;
    #10 busIf.enable <= 1;
    #40 busIf.enable <= 0;
    #20 busIf.enable <= 1;
    #100 $finish;
  end
endmodule
```

10.1 Clocking Block

```
interface my_int (input bit clk);
    // Rest of interface code

    clocking cb_clk @(posedge clk);
        default input #3ns output #2ns;
        input enable;
        output data;
    endclocking
endinterface
```

Example Usage:

```
// To wait for posedge of clock
@busIf.cb_clk;

// To use clocking block signals
busIf.cb_clk.enable = 1;
```

11

12

13

14 Synthesis

14.1 Verilog

Verilog constructs that are not synthesizable:

- Initial blocks
- Delay constructs
- Real Data Types
- Forj/Join Constructs
- Random Functions
- X and Z States
- Primitives
- Force and Release

15 Summary

In summary...

References