



Sharif University of Technology
Department of Computer Engineering

Digital System Design

Basic Concepts

Siavash Bayat-Sarmadi

Outline

2

- Lexical Conventions
- Data Types
- System Tasks
- Compiler Directives

Lexical Conventions

3

- Similar to C
- Case-sensitive language
- Number Specification
- Strings
- Operators
- Comments

Number Specification

4

- Sized numbers specification
 - 3 items should be specified
 1. Bit-length (only decimal allowed)
 2. Base format('b, 'o, 'h, 'd)
 3. Value
 - ' _ ' can be used to separate digits
 - Eg. 8'b0111_1101

Number Specification (Cont'd)

5

- Unsized numbers
 - Default base format decimal
 - Default size simulator/machine specific (at least 32 bits)
- Minus sign before the size specifier
 - ▣ -6'd3
 - ▣ 4'd-2 //Illegal specification

X and Z Values

6

- Unknown value denoted by an x
- High impedance value denoted by z
 - ▣ Question marks alternative for z
- Eg. 4'bxz?1
- Extension
 - ▣ Filled with x if the specified MSB is x
 - ▣ Filled with z if the specified MSB is z
 - ▣ Zero-extended otherwise

Strings

7

- Must be contained on a single line
- Example
 - ▣ “Hello Verilog World”
- Comments
 - ▣ Single-line
 - //
 - ▣ Multi-line
 - /* comments */

Data Types

8

□ Nets

- Represent physical connections between Hardware elements (wire, wand, wor, tri, triand, trior, trireg, etc)
- Nets may have zero or multiple drivers
- If no driver is connected to a net, its value will be z (high-impedance)
- Default value is z

Data Types (Cont'd)

9

□ Registers

- ▣ Represent data storage elements (reg)
- ▣ Registers retain value until another value is placed onto them
- ▣ A register does not need a driver or a clock
- ▣ Default value is **x**
- ▣ Can be declared as signed variables


Data Types (Cont'd)

10

Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
z	High impedance, floating state

Data Types (Cont'd)

11

Strength Level	Type	Degree
supply	Driving	strongest
strong	Driving	
pull	Driving	
large	Storage	
weak	Driving	
medium	Storage	
small	Storage	
highz	High Impedance	
		weakest

Data Types (Cont'd)

12

- Driving
 - ▣ For nets (wire, wor, wand, tri, ...)
- Storage
 - ▣ Trireg
 - trireg (large) out;
- Strength levels conflicts examples
 - ▣ **strong1** and **weak0** contended result is **strong1**
 - ▣ **strong0** and **strong1** contended result is **x**
- E.g.
 - ▣ **strong1 vdd; strong0 gnd;**

Data Types (Cont'd)

13

- integer (reg signed)
 - ▣ Keyword : integer
 - ▣ Default width is implementation-specific (at least 32 bits)
 - ▣ Stores values as signed quantities
 - ▣ Example:
 - integer counter;

Data Types (Cont'd)

14

□ real

- Keyword : real
- Default value is zero
- Can not have range declaration
- Can be specified in:
 - Decimal(e.g. , 3.14)
 - Scientific(e.g. , 3e6)

Data Types (Cont'd)

15

□ time

- ▣ Keyword : time
- ▣ Width is implementation-specific(at least 64 bits)
- ▣ The system function \$time is invoked to get the current simulation time

▣ example:

```
time save_sim_time;
```

```
initial
```

```
    save_sim_time = $time;
```

Data Types (Cont'd)

16

□ Vectors

- Nets or reg data types can be declared as vectors

- Example:

 - `wire [7:0] bus`

 - `reg [0:40] virtual_addr`

- Access to a part of vector

 - `bus[2:0] //bus[0:2]` is illegal

 - `virtual_addr[0:1]`

Data Types (Cont'd)

17

- Variable Vector Part Select
 - ▣ Starting bit can be varied
 - ▣ Width cannot
 - ▣ [starting_bit+ : width]
 - part-select increments from starting bit
 - ▣ [starting_bit- : width]
 - part-select decrements from starting bit

Data Types (Cont'd)

18

- Variable part select example
 - ▣ reg [255:0] data1;
 - data1[31- : 8];
 - data[31:24]
 - data1[24+:8];
 - data[31:24]
 - ▣ reg [0:255] data2;
 - data2[31- : 8];
 - data[24:31]
 - data2[24+:8];
 - data[24:31]

Data Types (Cont'd)

19

□ Arrays

▣ Syntax:

`<data_type> <var_name> [start_idx : end_idx]`

▣ Example:

- `integer count [0:7]`

- `integer matrix[4:0][0:255]`

▣ Assignment to elements of arrays

- `count[5] = 0;`

- `Matrix[1][0] = 0;`

Vectors Vs. Arrays

20

□ `reg [3:0] x, y;`

□ `reg x[3:0], y[3:0];`

□ `x [3:2] = y [3:2]`

▣ OK

□ `x [3:2] = y [3:2]`

▣ Illegal

□ `x = x + y;`

▣ OK

□ `x = x + y;`

▣ Illegal

Data Types (Cont'd)

21

□ Memories

- ▣ In digital simulation, one often needs to model Register Files, RAMs and ROMs
- ▣ Memory is array of registers in Verilog
- ▣ Each word is an element of the array
- ▣ Examples
 - `reg membit[0:1023]`
 - `reg [7:0] membyte[0:1023]`
- ▣ Usage
 - `membyte[500]`

Data Types (Cont'd)

22

□ Parameters

- Constants defined in a module
 - parameter port_id = 5;
- Can be overridden for an instance at compile time
 - defparam
- Local parameters
 - Localparam
- Cannot be modified by defparam

Parameter Example

23

```
module param;
parameter port_id = 4;
initial
$display("Displaying port_id : %d" ,
port_id)
endmodule

module top;
defparam t1.port_id = 10;
param t1 ();
param #(9) t2 ();
endmodule
```

Data Types (Cont'd)

24

□ Strings

- ▣ Can be stored in reg
- ▣ Each character requires 8 bits

□ Example:

- ▣ `reg [8*19:1] string_value;`
- ▣ `string_value = "Hello Verilog World";`

System Tasks

25

- Standard routines for
 - ▣ Displaying values
 - ▣ Monitoring values
 - ▣ Stopping and finishing simulation
- All system tasks appear in the form
 - ▣ \$<keyword>

Display Tasks(Cont'd)

26

□ \$display

- ▣ The format is very similar to printf in C
- ▣ Example
 - \$display("output is %d",wire_output);

Display Tasks(Cont'd)

27

□ \$monitor

- Mechanism used to monitor a signal when its value changes
- Example
 - \$monitor ("at time %t \t var1: %x \t var1: %x\n", \$time, var1, var2);
- Only one monitoring can be active at a time
- To switch monitoring on or off
 - \$monitoron
 - \$monitoroff

Simulation Control Tasks

28

- **\$stop**

- ▣ Stops simulation

- **\$finish**

- ▣ Terminates the simulation

Simulation Time Functions

29

\$time	Returns an integer that is a 64-bit time, scaled to the timescale unit of the module that invoked it.
\$stime	Returns an unsigned integer that is a 32-bit time, scaled to the time scale unit of the module that invoked it. If the actual simulation time does not fit in 32 bits, the low-order 32 bits of the current simulation time are returned.
\$realtime	Returns a real number time that, like \$time , is scaled to the time unit of the module that invoked it

Probablistic Functions

30

- **\$random**

- Returns a random signed integer that is 32-bits

String Format Specifications

31

Format	Display
%d or %D	Display variable in decimal
%b or %B	Display variable in binary
%s or %S	Display string
%h or %H	Display variable in hex
%c or %C	Display ASCII character
%m or %M	Display hierarchical name (no argument required)
%v or %V	Display strength
%o or %O	Display variable in octal
%t or %T	Display in current time format
%e or %E	Display real number in scientific format (e.g., 3e10)
%f or %F	Display real number in decimal format (e.g., 2.13)
%g or %G	Display real number in scientific or decimal, whichever is shorter

Compiler directives

32

- Defined by using ``<keyword>`
- **``define`**
 - ▣ Similar to `#define` in C
 - ▣ Defines text macros in Verilog
 - ▣ Example
 - ``define WORD_SIZE 32`
 - ``define s $stop`

Compiler directives (Cont'd)

33

□ ``include`

- Similar to `#include` in C
- Insert the contents of one source file into another file during compilation
- Example:

If file `sub.v` contains

```
wire a, b, c;
```

Example

34

And file main.v contains

```
module main;  
  `include "sub.v"  
  and a1 (a, b, c);  
endmodule
```

It is equivalent to having one file:

```
module main;  
  wire a, b, c;  
  and a1 (a, b, c);  
endmodule
```

Compiler directives (Cont'd)

35

- **`ifdef, `else, `endif**
- Allow one to optionally include lines of a Verilog source description during compilation
- The **`ifdef** compiler directive is followed by a `text_macro_name` in the code

Example

36

```
module selective_and (f, a, b);  
    output f;  
    input a, b;  
    `ifdef behavioral  
    wire f = a & b;  
    `else  
    and a1 (f, a, b);  
    `endif  
endmodule
```

Misc

37

- Escaped character
 - \t : tab
 - \n : newline
 - \\ : \
 - %% : %
 - \" : ”

38

File I/O Function

File I/O Function

39

- **\$fopen**
- \$fopen can have multiple modes: – ‘r’ for reading, ‘w’ for writing, and ‘a’ for appending
- Example

```
integer file_r, file_w;  
file_r = $fopen("filename", "r"); //Reading  
a file  
file_w = $fopen("filename", "w"); //Writing  
a file
```

File I/O Function

40

□ `$fclose`

□ Example:

```
$fclose(file_r); // Closing only one  
file
```

```
$fclose(); // Closing all opened  
files
```


File I/O Function

41

- **\$feof**

- ▣ Tests for end of file

- **\$ferror**

- ▣ Returns the error status of a file

- **\$fscanf**

- ▣ Parses formatted text from a file

- **\$fwrite**

- ▣ writes binary data from a register to the file

Example

42

```
`define NULL 0
module read_from_file;
integer file, ret;
reg [31:0] r_w, addr, data;
initial
    begin : file_read

        file = $fopen("data", "r");
        if (file == `NULL)
            disable file_read;
```

Example

43

```
while (!$feof (file))  
begin  
    ret = $fscanf(file, "%s %h %h\n",  
        r_w, addr,data);  
    $display ("commands: %s", r_w) ;  
end  
ret = $fclose (file);  
end  
endmodule
```

File I/O Function

44

- ❑ **\$readmemb, \$readmemh**
- ❑ To read data from a file and store it in memory
- ❑ \$readmemb reads binary data
- ❑ \$readmemh reads hexadecimal data

Example

45

```
module example_readmemh;
reg [31:0] data [0:3];
initial $readmemh("data.txt", data);
integer i;
initial begin
$display("read hexa_data:");
for (i=0; i < 4; i=i+1)
$display("%d:%h", i, data[i]);
end
endmodule
```