Structure of a Design or Test bench code

Centurion University
Shaping Lives...
Empowering Communities...

Packages

Parameters/defines, type defines, structs etc useful across files

Module/Program

- Contains declarations
  - functions/tasks, data types, objects etc
- Contains concurrent processes/blocks
  - module instance, continuous assign, always blocks etc

always blocks/initial blocks/processes

- Contains sequential code
  - If then else, case etc -> control constructs
  - Repeat/for loops/do..while -> loops
  - Blocking/non-blocking assignments etc

Data Types

2 valued data type	
bit:	1 bit (packed array)
byte:	8 bit (character)
shortint:	16 bit
int:	32 bit

4 valued data type	
logic:	1 bit (packed array)
integer:	32 bit
time:	64 bit unsigned

```
bit [14:0] bus; // unsigned
bit signed [11:0] i,q;

shortint unsigned addrs;
logic [15:0] addrs;
```

```
string myName = "John Smith";
byte c = "A"; // assign to c "A"
bit [1:4][7:0] s = "hello" ; // assigns to s "ello"
```

Integer Data Type

shortint	2-state (1, 0), 16 bit signed
int	2-state, 32 bit signed
longint	2-state, 64 bit signed
byte	2-state, 8 bit signed
bit	2-state, user-defined vector size
logic	4-state (1,0,x,z) user-def
reg	4-state user-defined size
integer	4-state, 32 bit signed
time	4-state, 64 bit unsigned

Signed/Unsigned

byte, shortint, int, integer and longint defaults to signed

Use unsigned to represent unsigned integer value

Example: **int unsigned ui;**

bit, reg and logic defaults to unsigned

To create vectors, use the following syntax:

logic [1:0] L; // Creates 2 bit logic vector

Strings

- String – dynamic allocated array of bytes
- SV provides methods for working with strings

Str1 == Str2	Equality
Str1 != Str2	Inequality
<, <=, >, >=	Comparison
{Str1, Str2, … Strn}	Concatenation
Str1[index]	indexing – return 0 if out of range

String Methods

- len
- putc
- getc
- toupper
- tolower
- compare
- icompare
- substr

- atoi, atohex, atoct, atobin
- atoreal
- itoa
- hextoa
- octtoa
- bintoa
- realtoa

Operators

Bitwise Operators		
~	~m	Invert each bit of m
&	m&n	Bitwise AND of m and n
|	m|n	Bitwise OR of m and n
^	m^n	Bitwise EXOR of m and n
~^	m~^n	Bitwise EX NOR of m and n

Unary Reduction Operators		
&	&m	AND all bits of m together (1-bit result)
~&	~&m	NAND all bits of m together (1-bit result)
|	|m	OR all bits of m together (1-bit result)
~|	~|m	NOR all bits of m together (1-bit result)
^	^m	EXOR all bits of m together (1-bit result)
~^	~^m	EXNOR all bits of m together (1-bit result)

Source/Acknowledgement: On-line Verilog HDL Quick Reference Guide

Centurion UNIVERSITY
Shaping Lives
Empowering

Operators

Arithmetic Operators		
+	m+n	Add n to m
-	m-n	Subtract n from m
-	-m	Negate m (2's complement)
*	m*n	Multiply m by n
/	m/n	Divide m by n
%	m%n	Modulus of m/n

Logical Operators		
!	!n	Is m not true?
&&	m&&n	Are both m AND n true?
||	m||n	Are both m OR n true?

Operators

Equality Operators (Compares logic values of 0 and 1)		
==	m==n	Is m equal to n? (1-bit True/False result)
!=	m!=n	Is m not equal to n? (1-bit True/False result)
Identity Operators (Compares logic values of 0,1,X and Z)		
===	m^n	Is m identical to n? (1-bit True/False results)
!==	m~^n	Is m not equal to n? (1-bit True/False result)
Wild Eqality Operators		
=?=	m=?=n	A equals b, X and Z values are wild cards
!?=	m!?=n	A not equals b, X and Z values are wild cards

- The three types of equality (and inequality) operators in SystemVerilog behave differently when their operands contain unknown values (X or Z).
- The == and != operators result in X if any of their operands contains an X or Z.
- The === and !== check the 4-state explicitly, therefore, X and Z values shall either match or mismatch, never resulting in X.
- The =?= and !?= operators treat X or Z as wild cards that match any value, thus, they too never result in X.

Centurion UNIVERSITY

Shaping Lives...
Empowering Communities...

Summary

- Supports a wide range of data types just like any programming language
- Need to be careful on where to use 2- valued versus 4 valued data types