SystemVerilog Reference Guide

Data Types

Basic Data Types

- `bit` - 2-state (0, 1) single bit
- `logic` - 4-state (0, 1, X, Z) single bit, recommended for most uses
- `reg` - Legacy 4-state register type
- `wire` - Net type for continuous assignments
- `byte` - 8-bit signed integer
- `shortint` - 16-bit signed integer
- `int` - 32-bit signed integer
- `longint` - 64-bit signed integer
- `integer` - 32-bit 4-state signed integer
- `time` - 64-bit unsigned time value
- `real` - Double precision floating point
- `shortreal` - Single precision floating point
- `string` - Dynamic string type

Packed Arrays

- `bit [7:0]` - 8-bit packed array
- `logic [31:0]` - 32-bit packed vector

Unpacked Arrays

- `int array[10]` - Fixed-size unpacked array
- `int queue[$]` - Dynamic queue
- `int assoc[string]` - Associative array

User-Defined Types

- `typedef` - Creates user-defined types
- `enum` - Enumerated types
- `struct` - Structure definition
- `union` - Union definition

- `class` - Object-oriented class definition

Operators

Arithmetic Operators

- `+` - Addition
- `-` - Subtraction
- `*` - Multiplication
- `/` - Division
- `%` - Modulus
- `**` - Exponentiation

Bitwise Operators

- `&` - Bitwise AND
- `|` - Bitwise OR
- `^` - Bitwise XOR
- `~` - Bitwise NOT
- `~&` - Bitwise NAND
- `~|` - Bitwise NOR
- `~^` **or** `^~` - Bitwise XNOR

Logical Operators

- `&&` - Logical AND
- `||` - Logical OR
- `!` - Logical NOT

Comparison Operators

- `==` - Equality (4-state)
- `!=` - Inequality (4-state)
- `===` - Case equality (includes X and Z)
- `!==` - Case inequality (includes X and Z)
- `<` - Less than
- `<=` - Less than or equal

- `>` - Greater than
- `>=` - Greater than or equal

Shift Operators

- `<<` - Logical left shift
- `>>` - Logical right shift
- `<<<` - Arithmetic left shift
- `>>>` - Arithmetic right shift

Reduction Operators

- `&` - Reduction AND
- `|` - Reduction OR
- `^` - Reduction XOR
- `~&` - Reduction NAND
- `~|` - Reduction NOR
- `~^` - Reduction XNOR

Assignment Operators

- `=` - Blocking assignment
- `<=` - Non-blocking assignment
- `+=`, `-=`, `*=`, `/=` - Compound assignments

Concatenation and Replication

- `{a, b}` - Concatenation
- `{n{pattern}}` - Replication

Control Flow Constructs

Conditional Statements

- `if-else` - Conditional execution
- `case` - Multi-way branch
- `casex` - Case with don't care (X)
- `casez` - Case with high impedance (Z)
- `unique case` - Case with synthesis optimization hints

- `priority case` - Case with priority encoding

Loops

- `for` - Traditional for loop
- `while` - While loop
- `do-while` - Do-while loop
- `foreach` - Iterate over arrays
- `repeat` - Repeat a fixed number of times
- `forever` - Infinite loop

Loop Control

- `break` - Exit loop
- `continue` - Skip to next iteration
- `return` - Return from function/task

Module and Interface Constructs

Module Definition

- `module` - Basic design unit
- `endmodule` - End module definition
- `parameter` - Compile-time constant
- `localparam` - Local parameter
- `generate` - Generate blocks for replication

Port Declarations

- `input` - Input port
- `output` - Output port
- `inout` - Bidirectional port
- `ref` - Reference port (for interfaces)

Interface Constructs

- `interface` - Interface definition
- `endinterface` - End interface definition
- `modport` - Module port within interface

- `clocking` - Clocking block definition

Procedural Blocks

Always Blocks

- `always` - General always block
- `always_ff` - Flip-flop (sequential) logic
- `always_comb` - Combinational logic
- `always_latch` - Latch inference

Initial and Final

- `initial` - Execute once at time zero
- `final` - Execute at end of simulation

Functions and Tasks

Function Definition

- `function` - Pure function definition
- `endfunction` - End function definition
- `automatic` - Automatic (recursive) function
- `static` - Static function

Task Definition

- `task` - Task definition (can have timing)
- `endtask` - End task definition

Subroutine Arguments

- `input` - Input argument
- `output` - Output argument
- `inout` - Bidirectional argument
- `ref` - Reference argument

Object-Oriented Programming

Class Definition

- `class` - Class definition

- **`endclass`** - End class definition
- **`extends`** - Class inheritance
- **`virtual`** - Virtual class/method
- **`pure virtual`** - Pure virtual method

Class Members

- **`static`** - Static class member
- **`local`** - Local (private) member
- **`protected`** - Protected member
- **`const`** - Constant member
- **`rand`** - Random variable
- **`randc`** - Random cyclic variable

Class Methods

- **`new()`** - Constructor method
- **`this`** - Reference to current object
- **`super`** - Reference to parent class

Assertions and Coverage

Immediate Assertions

- **`assert`** - Immediate assertion
- **`assume`** - Assumption for formal verification
- **`cover`** - Coverage point

Concurrent Assertions

- **`property`** - Property definition
- **`endproperty`** - End property definition
- **`sequence`** - Sequence definition
- **`endsequence`** - End sequence definition

Temporal Operators

- **`##`** - Delay operator
- **`|->` ** - Implication

- `|=>` - Overlapping implication
- `throughout` - Throughout operator
- `within` - Within operator

System Tasks and Functions

Display Functions

- `$display` - Print formatted text
- `$write` - Print without newline
- `$monitor` - Monitor signal changes
- `$strobe` - Display at end of time step

File I/O

- `$fopen` - Open file
- `$fclose` - Close file
- `$fwrite` - Write to file
- `$readmemh` - Read hex memory file
- `$readmemb` - Read binary memory file

Simulation Control

- `$finish` - End simulation
- `$stop` - Suspend simulation
- `$time` - Current simulation time
- `$realtime` - Real-valued simulation time

Random Functions

- `$random` - Pseudo-random number
- `$urandom` - Uniform random number
- `$urandom_range` - Random in range

String Functions

- `$sformatf` - Format string
- `$sscanf` - Parse string
- `$strlen` - String length

- `$substr` - Substring

Constraints (for Random Verification)

Constraint Blocks

- `constraint` - Constraint definition
- `solve...before` - Constraint ordering
- `if...else` - Conditional constraints
- `foreach` - Array constraints

Constraint Operators

- `inside` - Membership operator
- `dist` - Distribution constraint
- **`->` ** - Constraint implication

Compiler Directives

Preprocessing

- `define` - Macro definition
- `undef` - Undefine macro
- `ifdef` - Conditional compilation
- `ifndef` - Negative conditional compilation
- `else` - Else in conditional compilation
- `endif` - End conditional compilation
- `include` - Include file

Simulation Directives

- `timescale` - Time unit and precision
- `default_nettype` - Default net type
- `celldefine` - Cell definition start
- `endcelldefine` - Cell definition end

Packages and Imports

Package Definition

- **package** - Package definition
- **endpackage** - End package definition
- **export** - Export from package

Import Statements

- **import** - Import from package
- **::** - Scope resolution operator

Special Keywords

Simulation and Synthesis

- **$root** - Top-level scope
- **$unit** - Compilation unit scope
- **bind** - Bind assertion modules
- **alias** - Create signal alias

Miscellaneous

- **void** - Void data type
- **null** - Null pointer
- **default** - Default case/clocking
- **global** - Global clocking
- **disable** - Disable named blocks
- **fork...join** - Parallel execution
- **fork...join_any** - Join when any completes
- **fork...join_none** - Non-blocking fork

Array Methods

Dynamic Arrays

- **.size()** - Get array size
- **.delete()** - Delete array elements
- **.exists(index)** - Check if index exists

Queues

- **.size()** - Queue size

- `.insert(index, item)` - Insert at index
- `.delete(index)` - Delete at index
- `.push_front(item)` - Add to front
- `.push_back(item)` - Add to back
- `.pop_front()` - Remove from front
- `.pop_back()` - Remove from back

Array Reduction Methods

- `.sum()` - Sum all elements
- `.product()` - Product of elements
- `.and()` - AND reduction
- `.or()` - OR reduction
- `.xor()` - XOR reduction

Array Locator Methods

- `.find()` - Find elements matching condition
- `.find_index()` - Find indices of matching elements
- `.find_first()` - Find first matching element
- `.find_last()` - Find last matching element
- `.min()` - Find minimum element
- `.max()` - Find maximum element
- `.unique()` - Find unique elements

Array Ordering Methods

- `.reverse()` - Reverse array order
- `.sort()` - Sort array
- `.rsort()` - Reverse sort array
- `.shuffle()` - Randomly shuffle array

Coverage Constructs

Covergroup

- `covergroup` - Coverage group definition

- `endgroup` - End coverage group
- `coverpoint` - Coverage point
- `cross` - Cross coverage
- `bins` - Coverage bins
- `ignore_bins` - Bins to ignore
- `illegal_bins` - Illegal value bins

This reference covers the major SystemVerilog constructs used in design and verification. Each construct serves specific purposes in hardware description, simulation, and formal verification workflows.