# SystemVerilog Cheatsheet

A concise reference for SystemVerilog syntax and constructs, covering data types, operators, procedural statements, and verification features.

## Data Types & Declarations

### Basic Data Types

| | |
|---|---|
| `logic` | Two-state type, can be 0 or 1. Preferred for synthesizable designs. |
| `reg` | Historically used for sequential logic outputs; now largely replaced by `logic`. |
| `bit` | Two-state, unsigned data type. |
| `int` | 32-bit signed integer. |
| `real` | 64-bit floating-point number. |
| `time` | 64-bit unsigned integer representing simulation time. |

### Arrays

| | |
|---|---|
| Fixed-size array | `logic [7:0] data [0:15];` // 16 elements, each 8 bits wide. |
| Dynamic array | `int dyn_array[]; dyn_array = new[array_size];` |
| Associative array | `bit [63:0] assoc_array [string];` // Index with string. |

### User-Defined Types

| | |
|---|---|
| `typedef` | `typedef logic [3:0] nibble_t; nibble_t my_nibble;` |
| `struct` | `typedef struct { logic valid; logic [7:0] data; } packet_t; packet_t my_packet;` |
| `enum` | `typedef enum {IDLE, READ, WRITE} state_t; state_t current_state;` |

## Operators & Expressions

### Arithmetic Operators

| | |
|---|---|
| `+`, `-`, `*`, `/`, `%` | Addition, subtraction, multiplication, division, modulo. |
| `**` | Exponentiation. |

### Logical Operators

| | |
|---|---|
| `&&`, `\|\|`, `!` | Logical AND, OR, NOT. Operates on boolean values (1 or 0). |

### Bitwise Operators

| | |
|---|---|
| `&`, `\|`, `^`, `~` | Bitwise AND, OR, XOR, NOT. Operates on individual bits. |
| `~&`, `~\|`, `~^` | Bitwise NAND, NOR, XNOR. |

### Reduction Operators

| | |
|---|---|
| `&`, `\|`, `^` | Reduction AND, OR, XOR. Operates on all bits of a vector to produce a single-bit result. |

### Shift Operators

| | |
|---|---|
| `<<`, `>>`, `<<<`, `>>>` | Logical left shift, logical right shift, arithmetic left shift, arithmetic right shift. |

### Comparison Operators

| | |
|---|---|
| `==`, `!=`, `===`, `!==` | Equality, inequality, case equality, case inequality. Case equality considers X and Z. |
| `>`, `<`, `>=`, `<=` | Greater than, less than, greater than or equal to, less than or equal to. |

## Procedural Statements

### Sequential Blocks

| | |
|---|---|
| `always_comb` | Combinational logic block. Re-evaluates whenever any of its inputs change. |
| `always_ff` | Sequential logic block. Used for describing flip-flops and registers. |
| `always_latch` | Latch inferrence. Avoid using latches in synchronous design. |

### Conditional Statements

| | |
|---|---|
| `if-else` | `if (condition) begin // statements end else begin // statements end` |
| `case` | `case (expression) value1: statement; value2: statement; default: statement; endcase` |

### Loop Statements

| | |
|---|---|
| `for` | `for (int i = 0; i < 10; i++) begin // statements end` |
| `while` | `while (condition) begin // statements end` |
| `repeat` | `repeat (8) begin // statements end` |

### Task and Function

| | |
|---|---|
| `task` | Can consume simulation time. Can have input, output, and inout arguments. |
| `function` | Cannot consume simulation time. Returns a single value. Can only have input arguments. |

## Verification Features

### Assertions

| | |
|---|---|
| `assert property` | Checks if a property holds true. Can be used for functional coverage. |
| `cover property` | Collects coverage information based on property evaluation. |

### Constrained Random Verification

| | |
|---|---|
| `rand` | Specifies that a variable should be randomized. |
| `constraint` | Defines constraints that the random values must satisfy. |

### Coverage

| | |
|---|---|
| Functional Coverage | Measure of how well the design's functionality has been exercised during verification. Check `covergroup` and `coverpoint` |