SystemVerilog Quick Reference

For Beginning Digital Logic Students

Data Types

Nets: `wire, tri` (connect modules)
Vars: `logic` (4-state: 0,1,X,Z), `reg` (4-state: 0,1,X,Z), `bit` (2-state: 0,1)
Ints: `int` (32b), `shortint` (16b), `longint` (64b), `byte` (8b)
Vectors: `logic [7:0] a;` (8-bit, MSB=7, LSB=0)
Arrays: `logic [3:0] mem [0:15];` (16 words × 4 bits)

4-State Values (0,1,X,Z)

0: Logic low (ground)
1: Logic high (VDD)
X: Unknown/undefined (conflict or uninitialized)
Z: High-impedance (tri-state, floating)
Examples:

```
logic a = 1'b0;  // 0
logic b = 1'b1;  // 1
logic c = 1'bx;  // X (unknown)
logic d = 1'bz;  // Z (high-Z)
// Z is used for tri-state buffers:
assign bus = enable ? data : 1'bz;
```

Literals

`4'b1010` (binary), `8'h3F` (hex), `12'd25` (decimal)
`1'bx` (unknown), `1'bz` (high-Z)

Keywords

`module, input, output, wire, logic, assign always_comb, always_ff, initial, if, else case, for, while, begin, end`

Module Template

```
module name (
  input  logic a, b,
  output logic y
);
  // Logic here
endmodule
```

Testbench Template

```
module name_tb;
  logic a, b, y;
  name dut(.a(a), .b(b), .y(y));

  initial begin
    a=0; b=0; #10;
    a=1; #10;
    $finish;
  end

  initial
    $monitor("a=%b␣b=%b␣y=%b", a,b,y);
endmodule
```

Operators with Examples

Bitwise: `a & b` (AND), `a | b` (OR), ã (NOT), `a ^b` (XOR), `a ^| b` (XNOR)
Logic: `a && b` (AND), `a || b` (OR), `!a` (NOT)
Arithmetic: `a + b, a - b, a * b, a / b, a % b, a ** b`
Comparison: `a == b, a != b, a < b, a <= b, a > b, a >= b`
Case: `a === b` (4-state), `a !== b` (4-state)
Shift: `a << 2` (left), `a >> 2` (right), `a <<< 2` (arithmetic left), `a >>> 2` (arithmetic right)
Rotate: `{a[6:0], a[7]}` (rotate right), `{a[0], a[7:1]}` (rotate left)
Concatenation: {a,b}, {4{a}} (replicate)
Conditional: `sel ? a : b`

Simulation Tasks

`$display("format", vars)` - print once
`$monitor("format", vars)` - print on change
`$writememb("file", array)` - write to file
`$readmemb("file", array)` - read from file
`$time, $finish`

Timing & Delays

Simulation delays: `#10;` (10 time units)

```
  clk = 0;
  forever #5 clk = ~clk;
end
```

Testbench timing:

```
initial begin
  a = 0; b = 0;
  #10 a = 1;  // Wait 10 units, then set a=1
  #5  b = 1;  // Wait 5 more units, set b=1
  #10 $finish;
end
```

iverilog Commands

`iverilog -o exe prog.sv` - compile
`iverilog -g2012 prog.sv` - SystemVerilog
`vvp exe` - run simulation
`gtkwave dump.vcd` - view waveforms

Always Blocks

`always_comb` - combinational
`always_ff @(posedge clk)` - sequential
`always @(*)` - legacy combinational
`always @(posedge clk or negedge rst)` - with reset

Control Flow Statements

if-else:

```
if (condition) begin
  // statements
end else if (condition2) begin
  // statements
end else begin
  // statements
end
```

case:

```
case (sel)
  2'b00: y = a;
  2'b01: y = b;
  2'b10: y = c;
  default: y = 0;
endcase
```

for loop:

```
for (int i = 0; i < 8; i++) begin
```

```
while (count > 0) begin
  count = count - 1;
  // other statements
end
```

repeat:

```
repeat (8) begin
  clk = ~clk;
  #5;
end
```

Always Block Examples

Combinational:

```
always_comb begin
  if (sel == 2'b00)
    y = a;
  else if (sel == 2'b01)
    y = b;
  else
    y = c;
end
```

Sequential:

```
always_ff @(posedge clk) begin
  if (rst)
    q <= 0;
  else
    q <= d;
end
```

With Reset:

```
always_ff @(posedge clk or negedge rst_n) begin
  if (!rst_n)
    q <= 0;
  else
    q <= d;
end
```

Common Errors & Debugging

Blocking vs Non-blocking:

```
// WRONG - blocking in sequential
always_ff @(posedge clk)
  q = d;  // Use <= instead

// CORRECT
always_ff @(posedge clk)
  q <= d;
```

Incomplete sensitivity lists:

```
// WRONG - missing signals
always @(a or b)  // missing c
  y = a & b & c;

// CORRECT - use always_comb
always_comb
  y = a & b & c;
```

Undriven signals:

```
// WRONG - output not driven
output logic y;  // y never assigned
```

Vivado vs iverilog

iverilog: `iverilog -g2012 -o exe file.sv`
Vivado: Use `*.sv` extension, add to project
iverilog: `vvp exe` to run
Vivado: Run simulation in GUI or `vivado -mode batch`
iverilog: `gtkwave dump.vcd`
Vivado: Built-in waveform viewer

File Extensions & Naming

Design files: `*.sv` (SystemVerilog)
Testbenches: `*_tb.sv`
Examples: `and_gate.sv`, `and_gate_tb.sv`
Module names: Match filename (case-sensitive)

Tips

- Use `logic` not `reg`
- Separate testbench from design
- Use meaningful signal names
- Comment your code
- Test incrementally
- Use `<=` for sequential, `=` for combinational
- Always use `begin/end` with control statements
- Check for undriven outputs
- Use `always_comb` instead of manual sensitivity lists
- Initialize variables to avoid X states