

Verilog Syntax Cheat Sheet

- Data Formats:

- Standard Syntax: `[WIDTH]'[BASE][VALUE]`
 - Example: `16'ha`; -> 16 bit field, value of `0x000a`
 - Bases: d = decimal, b = binary, h = hex
 - Strings treated as 1 byte-per-character

- Operations

- Bitwise

- Example: `3'b010 | 3'b110 -> 3'b111` [OR]
`3'b101 & 3'b100 -> 3'b100` [AND]
`3'b101 ^ 3'b100 -> 3'b001` [XOR]
`~3'b010 -> 3'b101` [NOT]

- Shifts

- Example `4'b0111 << 1 -> 4'b1110` [Left Shift]
`4'b1111 >> 1 -> 4'b0111` [Right Shift]

- Arithmetic

- Example: `3'b010 + 3'b100 -> 3'b110` [Addition]
`3'b100 - 3'b001 -> 3'b011` [Subtraction]

- Comparators

- Example: `3'b100 > 3'b011 -> 1'b1` [Greater Than]
`3'b100 == 3'b010 -> 1'b0` [Equal To]
`3'b100 != 3'b000 -> 1'b1` [Not Equal To]

- Wires and Regs

- Wires

- Continuous assignments; do not "hold value"
- Typically used for combinatorial logic assignment
- If unassigned, dflt value is 'z' (high impedance)
- Use w/ 'assign' Statements
- Example: `assign z = x & y;`
`assign w = z | v;`
 - z updates whenever x and y update
 - w updates whenever z or v updates

- Regs

- Hold prior value until sensitivity list fires
- A register does not always imply a flip-flop
- Often used for clocked logic (i.e. registers)
- Has a default value of 'x' (unknown) if unassigned
- Example: `always@(posedge clk) begin`
`if (rst)`
`q <= '0;`
`else`
`q <= y;`
`end`
 - Triggered on positive edge of each clock
 - 'q' synthesizes as a register
- For best results you should only assign a reg in a single process
- Use '<=' in processes to assign to regs.

- Variable declaration (Scalars, 2d Arrays)

- Types

- Single Bits:
 - Ex: `wire x; reg y;` // 1 Bit Each
- Bit Vectors
 - Ex: `wire [2:0] x; reg [2:0] y;` // 3 Bits Each
- 2d Vectors
 - Ex: `wire [7:0] x [3:0]` // 4x8 Entries

- Bit Manipulation

- Bit Slicing

- Example: `reg [15:0] data; wire [7:0] field;`
`// Take Lower 8 Bits of Data`
`assign field = data[7:0];`

- Bit Concatenation

- Example: `reg [15:0] data; wire [7:0] field;`
`// Flip the Nibbles`
`assign field = {data[3:0], data[7:4]};`

- Array Indexing

- Example: `wire [7:0] x [3:0]`
`x[0] // 8 Bit Entry at Array Idx 0`
`x[1][3:0] // Lower 4 Bits of Idx 1`

- Modules

- Encapsulate blocks of logic into a reusable container with a well-defined interface

```
Example: module adder
#(
    parameter WIDTH = 16
)
(
    input [WIDTH-1:0] a; // Signals In
    input [WIDTH-1:0] b;
    output [WIDTH:0] sum; // Signals Out
    output overflow;
);

// Logic Goes Here
endmodule;
```

- Instantiating Modules

- Example: `adder` // Module Name
`#(.WIDTH (16))`
`adderInst` // Instance Name
`(.a (w_a),`
`// Port b is attached to wire w_b`
`.b (w_b),`
`.sum (w_sum),`
`.overflow (w_overflow));`

- Constructs

- If/Else

- Example: `always@(posedge clk) begin`
`if (y)`
`x <= v;`
`else if (w)`
`x <= w;`
`else`
`x <= x;`
`end`
 - If...Else cases imply priority and are evaluated sequentially

- Case Statements

- Example: `// Combinatorial Logic`
`always@(*) begin`
`case (x)`
`1'b0:`
`y <= v;`
`1'b1:`
`y <= w;`
`default:`
`y <= '0;`
`endcase`
`end`

- This should resolve to a 2-to-1 Mux. 'x' is the select line. v/w are Inputs
- 'default' case is not strictly necessary here since all combinations are covered. It is best practice to include it though.