EE 201 SystemVerilog reference sheet

r.2024.2.9

www.ece.tufts.edu/ee/201

GRAY_ITALICS represent user-defined names or operations
Purple constructs are only for simulation (at least in this course)

keywords
literals (constants)

```
// This is a comment
/* Multi-line
   comment */
module MODULE_NAME (
          input logic PORT_NAME, // Single bit input
          output logic[3:0] ANOTHER // 4-bit output
       );
  // Body of your module goes here
endmodule
```

Separated by commas

← No semicolon!

Continuous assignments

```
assign RESULT_SIGNAL = SIGNAL1 & SIGNAL2;
```
Also works for or, xor, not, nand, nor, xnor (`|`, `^`, `~`, `~&`, `~|`, `~^`)

```
assign RESULT_SIGNAL = (SIGNAL1 == 2'b11) ? SIGNAL2 : 0;
```
Ternary operator: `CONDITION ? TRUE : FALSE`

```
assign HIGHEST_BIT = EIGHT_BIT_VEC[7];
```
Extract a single bit (7 is MSB, 0 is LSB)

```
assign TWO_BIT_VEC = EIGHT_BIT_VEC[3:2];
```
Extract multiple bits

```
assign SIX_BIT_VEC = {3'b000, EIGHT_BIT_VEC[3:2], SINGLE_BIT};
```
Concatenate bits and vectors

```
assign REDUCTION = &VECTOR;
```
AND all bits of a vector together; also works for `|`, `^`, `~&`, `~|`, `~^`)

Types

`logic ONE_BIT` Basic logic type, can take values 0, 1, X, Z

`logic[N:0] MY_VECTOR` Vector of bits, can hold numeric values

`logic SOME_BITS[N:0]` Array of single bits (not a vector!)

`reg` and `wire` should not be used in your code.

Literals

`0, 1, 1'bx, 1'bz`

`9'h101`	`3'b101`	`7'd101`
9-bit hex	3-bit binary	7-bit decimal

`5, 38, 10000000` decimal by default

Always blocks

```
always_comb
begin
    // Combinational logic with if/case/print goes here
    // Use blocking assignments (=) only
end;
```

Other patterns with always blocks are generally problematic!

Clock signal for flip-flops

```
always_ff @(posedge CLK)
begin
    // Sequential logic goes here
    // Use non-blocking assignments (<=) only
end;
```

If/else

Only inside an `always` or `initial` block, use tenary operator otherwise.

```
if CONDITION begin
  SIGNAL <= VALUE1;
end else if OTHER_CONDITION begin
  SIGNAL <= VALUE2;
end else begin
  SIGNAL <= VALUE3;
end;
```

Choose blocking/non-blocking assignments as appropriate!

Case

```
case (INPUT_SIGNAL)
  VALUE1 : OPERATION1;
  VALUE2 : OPERATION2;
  default: DEFAULT_OPERATION;
endcase
```