Cheat sheet

Verilog

Modules, combinational and sequential logic, state machines and testbenches: the patterns used in almost every design.

1 Modules and ports

Module with ports

```
module adder (
  input  wire [7:0] a, b,
  output wire [8:0] sum
);
  assign sum = a + b;
endmodule
```

Parameter

```
module counter #(parameter W = 8) (
  input clk, output reg [W-1:0] q);
```

Override with counter #(.W(16)) u0 (...);

Instantiate by name

```
adder u_add (.a(x), .b(y), .sum(s));
```

Named ports avoid order mistakes

2 Values and operators

Sized literals

```
8'hFF   4'b1010   12'd100   1'b0
```

Width ' base value

Concatenate, replicate

```
{a, b}   {4{1'b0}}   {8{sign}}
```

Bit and part select

```
x[3]   x[7:4]   x[base +: 4]
```

+: picks 4 bits starting at base

Reduction

```
&a   |a   ^a
```

AND, OR, XOR of all bits (^a = parity)

Signed

```
wire signed [7:0] s;
$signed(x)
```

Mixing signed and unsigned makes all unsigned

3 Combinational logic

Continuous assign

```
assign y = sel ? b : a;
```

always block

```
always @(*) begin
  y = 1'b0;          // default first
  if (en) y = a & b;
end
```

A default value avoids an unwanted latch

case

```
always @(*) begin
  case (op)
    2'd0: y = a + b;
    2'd1: y = a - b;
    default: y = '0;
  endcase
end
```

Always add default; '0 fills all bits (SystemVerilog)

4 Sequential logic

Flip-flop, sync reset

```
always @(posedge clk)
  if (rst) q <= 0;
  else     q <= d;
```

Use <= (non-blocking) in clocked blocks

Async active-low reset

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

Counter with enable

```
always @(posedge clk)
  if (rst) cnt <= 0;
  else if (en) cnt <= cnt + 1;
```

Two-flop synchroniser

```
always @(posedge clk) begin
  s1 <= async_in;
  s2 <= s1;      // use s2
end
```

For single bits crossing clock domains

5 State machine

States

```
localparam IDLE = 2'd0, RUN = 2'd1, DONE = 2'd2;
reg [1:0] state, next;
```

State register

```
always @(posedge clk)
  if (rst) state <= IDLE;
  else     state <= next;
```

Next-state logic

```
always @(*) begin
  next = state;
  case (state)
    IDLE: if (start) next = RUN;
    RUN:  if (fin)   next = DONE;
    DONE:            next = IDLE;
  endcase
end
```

6 Testbench

Time unit

```
`timescale 1ns/1ps
```

Clock

```
reg clk = 0;
always #5 clk = ~clk;   // 100 MHz
```

Stimulus

```
initial begin
  rst = 1; repeat (2) @(posedge clk);
  rst = 0; a = 8'd3; b = 8'd4;
  #10 if (sum !== 9'd7) $display("FAIL");
  $finish;
end
```

!== also catches x and z

Waveforms

```
initial begin
  $dumpfile("tb.vcd");
$dumpvars(0, tb);
end
```

Open tb.vcd in GTKWave

Print

```
$display("t=%0t a=%h", $time, a);
$monitor("q=%b", q);
```

$monitor prints whenever q changes

7 Common mistakes

Blocking in clocked blocks

```
q <= d;   not   q = d;
```

= in a clocked block causes race conditions

Latch by accident

```
Missing else or default in always @(*)
```

Give every output a value on every path

Two drivers

```
Same signal assigned in two always blocks
```

Assign each signal in exactly one block