

# Verilog basics

# Simple things

# Coding examples

```

Comments
// Single line comment
/* */ Multiple line comment

Definitions
input clock
output something
output reg something_reg
inout bidir
input [1:0] data_bus

Signal and wire declaration
wire wire_name USE IN ASSIGN STATEMENT
wire [31:0] bus_wire

reg signal USE IN ALWAYS BLOCK
reg [31:0] bus_signal
reg [1:0] bus = 2'b01 Initialized

parameter WIDTH = 8;
reg [WIDTH-1:0] my_bus;

Arithmetic operators
+ Addition
- Subtraction
* Multiplication
/ Divide
% Modulus
** Power Operator

Bitwise operators
~ Invert single bit or each bit in a bus
& AND single bits or two buses
| OR two single bits or two buses
^ XOR two single bits or two buses

Logical operators
== Inputs Equal
!= Inputs Not Equal
< Less-than
<= Less-than or Equal
> Greater-than
>= Greater-than or Equal
! Not True
&& Both Inputs True
|| Either Input True

Shift operators
<< Left shift
>> Right shift

Bit select, replicate, concatenate
and bit-reduction
q[3:1] = b[2:0]; Select some bits

q = {a, b, c}; Concatenate a, b and c
q = {3(a)}; Replicate a, 3 times
q = {5(a), b}; Replicate a, 5 times and
concatenate to b
q = &a [2:0]; a[0] & a[1] & a[2]
q = |a [2:0]; a[0] | a[1] | a[2]
q = ^a [2:0]; a[0] ^ a[1] ^ a[2]
(this is parity)

Shifting with bit select & concatenate
reg [7:0] shift_l, shift_r;
always @ (posedge clock)
    shift_l <= {shift_l [6:0] , new_value_l};

always @ (posedge clock)
    shift_r <= {new_value_r, shift [7:1]};

Variable value assignment
wire a;
wire [31:0] b;

assign a = 1'b0 Decimal 0
assign b = 32'b1 Decimal 0
assign b = 32'b00001 - Decimal 0
assign b = 52 Decimal 52
assign b = 32'd52 Decimal 52
assign b = 32'hFF Decimal 255
assign b = b'hFF Decimal 255

Multiple statements (begin - end)
Only one statement:
...
if (reset)
    signal1 <=value;
else
...

Multiple statements:
...
if (reset)
    begin
        signal1 <=value1;
        signal2 <=value1; //Executed parallel
    end
else
...

```

```

Rule of thumb 1:
Assignment Variable type Operator
assign wire =
always reg <=

Rule of thumb 2:
assign Combinatorial (logic)
always @ ( * ) Combinatorial (logic)
always @ ( some wire ) Combinatorial (logic)
always @ (posedge clock) Sequential (register)

```

```

Combinatorial assignment
wire wire_name;
assign wire_name = signal_or_value;

OR
reg signal;
always @ ( * )
    signal <= signal_or_value;

Sequential assignment
reg signal;
always @ (posedge clock)
    if (reset)
        signal <= 0;
    else
        signal <= signal_or_value;

```

```

Conditional assignment
if (condition0)
    statement0;
else if (condition1)
    statement1;
else
    statement2;

OR
assign wire_name =
    (Condition) ? input1 : input0;

Tri state output:
assign port =
    (enable_signal) ? signal : 1'bz;

OR
case (two_bit select)
    2'b00 : statement0;
    2'b01 : statement1;
    2'b10 : statement2;
    2'b11 : statement3;
    default: statement_def;
endcase

```

```

Example module declaration
module something(
    input clock,
    input reset,

    input [7:0] bus_in,
    output [7:0] bus_out,
);
always @ (posedge clock)
    if (reset)
        bus_out <=0;
    else
        bus_out <= bus_in;
endmodule

```

```

Example module instantiation
...
wire clock, reset;
wire local_bus_in, local_bus_out;

something inst_name (
    .clock (clock),
    //module_port_name (local_name)
    .reset (reset),
    .bus_in (local_bus_in),
    .bus_out (local_bus_out)
);
...

```

```

D Flip-flop
reg ff;
always @(posedge clk)
    if (reset)
        ff <= 1'b0;
    else
        ff <= new_value;
end

Counter
reg [3:0] count;
always @(posedge clk)
    if (reset)
        count <= 0;
    else if (count_enable)
        if (load_enable)
            count <= default_value;
        else
            count <= count + 1;

```

```

Serial in, serial out shifter
reg [3:0] shift;
wire out, in;
always @(posedge clk)
    if (reset)
        shift <= 4'b0001;
    else if (clk_enable)
        shift <= (shift[2:0], in);
assign out = shift[3];

```

```

Parallel in, serial out shifter
reg [2:0] shift;
reg out;

always @(posedge clock)
    if (reset) begin
        shift <= 0;
        out <= 1'b0;
    end
    else if (load) begin
        shift <= load_input[2:0];
        out <= load_input[3];
    end
    else if (shift_enable) begin
        shift <= { shift[1:0] , 1'b0};
        out <= shift[2];
    end

```

```

Multiplexer
reg [3:0] output;

always @ ( * )
    case ( select )
        2'b00: output = input1;
        2'b01: output = input2;
        2'b10: output = input3;
        2'b11: output = input4;
        default: output = input1; //Security
    endcase

Decoder
reg [3:0] output;
wire [1:0] select;
always @ ( * )
    case (select)
        2'b00 : output <= 4'b0001;
        2'b01 : output <= 4'b0010;
        2'b10 : output <= 4'b0100;
        2'b11 : output <= 4'b1000;
        default: output <= 4'b0000;
    endcase

```

```

OR
wire [3:0] output;
wire [1:0] select;
assign output = 1 << (select);

State Machine
parameter state1 = 2'b00;
parameter state2 = 2'b01;
reg state = state1;

always@(posedge clock) begin
    if (reset)
        state <= state1;
    else
        case (state)
            state1 : if (condition)
                state <= next_state0;
                else
                    state <= next_state1;

            state2 : if (condition)
                state <= next_state2;
                else
                    state <= next_state3;

            default : state <= state1;
        endcase

```