12. Commonly used Compiler Directives

```
deassign statement
force statement
release statement
defparam statement
```

Operators

- Division and modulus operators for variables
- Case equality and inequality operators (= and !=)

Gate-Level Constructs

- `pullup, pulldown, tranifi0, tranifi1, rtranifi0, rtranifi1`

Miscellaneous Constructs

- Compiler directives like `ifdef, `endif and `else
- Hierarchical names within a module

Verilog is a registered trademark of Cadence Design Systems, Inc.
Verilog Quick Reference Card is intended for quick reference. Please use Verilog LRM for details.

13. Observing Outputs

```
$display("Value of variable is ", var);
integer flag;
initial flag = $open("out_file");
always @(.*); // dump data in text file
	$display(flag, ", ", data[7:0]);
	……
	$close("out_file");
end
$monitor($time, ", a = ", a, ", b = ", b, ", clock, ", reset);$monitor(flag, ", value = ", flag, ", add[15:0]);$monitoron;
$monitoroff;
```

14. Simulation Control

```
initial begin
	dumpfile ("my.dump"); // dump in this file
dumpvars;
	dumpvars (1, top); // dump all signals
dumpvars (2, top, m1); // dump 2 levels below top, m1
#1000 dumpoff; // stop dump
#1000 dumpon; // start or restart dump
$stop; // stop for interaction
#1000 $finish; // come out of simulation
end
```

15. Language Constructs not supported by most Synthesis tools

- Declarations and Definitions
  - `time declaration`
  - `event declaration`
  - `triand, trior, tri1, tri0, and trireg` net types
- Ranges and arrays for integers
- Primitive definition

- Statements
  - `initial statement`
  - `delay control`
  - `event control`
  - `wait statement`
  - `repeat statement`
  - `fork statement`

---

**Better ROI**

Download White Paper to find out how.

**Comit**

The Contract Engineering Company

Verilog® Quick Reference Card

1. **Module**

```
module module_name (list of ports);
input / output / inout declarations
net / reg declarations
integer declarations
parameter declarations
```

- gate/switch instances
- hierarchical instances
- parallel statements

```
endmodule
```

2. **Parallel Statements**

```
Following statements start executing simultaneously inside module
initial begin
(sequential statements)
end
always begin
(sequential statements)
end
assign wire_name = [expression];
```

3. **Basic Data Types**

- **a. Nets**
  - e.g. `wire, wand, tri, wor`
  - Continuously driven
  - Gets new value when driver changes
  - LHS of continuous assignment
    ```
    tri [15:0] data;
    // unconditional
    assign data[15:0] = data_in;
    // conditional
    assign data[15:0] = enable ? data_in : 16'b0;
    ```

- **b. Registers**
  - e.g. `reg`
    - Represents storage
    - Always stores last assigned value
    - LHS of an assignment in procedural block.
      ```
      reg signal;
      @ (posedge clock) signal = 1'b1; // positive edge
      @ (reset) signal = 1'b0; // event (both edges)
      ```

4. **Sequential Statements**

- Given below are simple examples instead of BNF type of definitions.
  - if (reset == 0) begin
    ```
    data = 8'b00;
    end
    ```

© 2000 Comit Systems, Inc.
10. Blocking and Non-blocking Statements

// These blocking statements exhibit race condition.
always @(posedge clock)
    a = b;
always @(posedge clock)
    b = a;

// This Non-blocking statement removes above race condition
// and gives true swapping operation
always @(posedge clock)
    a <= b;
always @(posedge clock)
    b <= a;

11. Functions and Tasks

Function
- A function can enable another function but not another task.
- Function always executes in 0 simulation time.
- Functions must not contain any delay, event, or timing control statement.
- Functions must have at least one input argument. They can have more than one input.
- Functions always return a single value. They cannot have output or inout arguments.

E.g.:
parity = calc_parity(a, b, d);

function calc_parity;
    input [31:0] address;
    begin
        calc_parity = a'address;
    end
endfunction

Task
- A task can enable other tasks and functions.
- Tasks may execute in non-zero simulation time.
- Tasks may contain delay, event or timing control statements.
- Tasks may have zero or more arguments of type input, output or inout.
- Tasks do not return a value but can pass multiple values through output and inout arguments.

Cycle_read (read_in, oe_in, data, addr);

endtask

task Cycle_read;
    input read, oe, data; // notice the sequence output [7:0] data;
    input [15:0] address;
    begin
        #10 read_pin = read;
        #05 oe_pin = oe;
        data = some_function(address);
    end
endtask

8. Specify Blocks

specify
    // specparam declarations (min:typ:max)
    specparam t_setup = 8:10, t_hold = 11:12:13;
    // timing constraints checks
    $setup (data, posedge clock, t_setup);
    $hold (posedge clear, data, t_hold);
    // simple pin to pin path delay
    (a => out) = 9; // => means parallel connection
    // edge sensitive pin to pin path delay
    (posedge clock => (out_a, in)) = (10, 6);
    // state dependent pin to pin path delay
    if (state_a = "0b01") (a, b => out) = 15;
    // => means full connection
endspecify

9. Memory Instantiation

module mem_test;
    reg [7:0] memory [0:10]; // memory declaration
    integer i;
    initial begin
        // reading the memory content file
        $readmemh("contents.dat", memory);
        // display contents of initialized memory
        for (i = 0; i < 5; i = i + 1)
            $display("Memory [%d] = %h", i, memory[i]);
    end
endmodule

"contents.dat" contains
    @02    ab    da
    @06    00    01

- This simple memory model can be used for feeding input data values to simulation environment.
- $readmemh can be used for reading binary values from contents file.