COMP ARCH (LABS)

Date / /

* For graphical waveforms -
  testbench me pur code - initial begin
  $dumpfile("filename.vcd");
  $dumpvars;
  end
  & cmd line se : run gtkwave filename.vcd

- Clock with cycle time = 10
  reg clk; initial
  clk = 1'b0; //set to 0
  always
  #5 clk = ~clk; // toggle every 5 unit. time

- Values = 0, 1, x, z ⇒ high impedance ⇒ wrong connections
  ↓
  unknown value ⇒ is being evaluated.

- "wire" have values continuously driven on them by o/p of device, that they r connected to

- "reg" → a variable that can hold a value.
  doesn't needs a driver.

- Vector → multiple bit wire/reg wire [7:0] w; /* 8 bit wire */

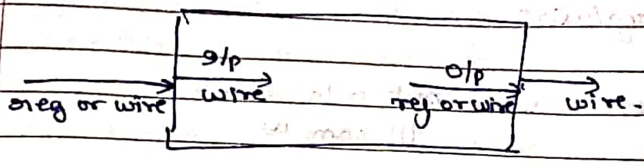
- Variable Vector Part Select → should be constant.
  [<starting_bit> +: width]
  byte = data [31 - : 8] ⇒ data [31:24]
  can be variable

Date / /

- Array →
  ~~integer count [0:7]~~ /* ~~array of~~
  reg bool [31:0] /* Array of 32 1-bit reg variables */
- Memory (RAM/ROM) → model using 1-D array of registers
  (bi word size = 1 bit)
  word size = 8 bit = reg [7:0] membyte [0:1023]
- Parameters ⇒ constants.
  parameter port_id = 5;
- $display ⇒ printf ki tarah
  eg $display("At time %d, virtual address is %h", $time, virtual_adr);
- $monitor (parameters)
  ↳ variables or signals. or strings
  displays all parameters if var/signal change.
- $stop to pause simulation (/* debugging ke lie */)
  $finish to finish & exit.
- `define /* Macros */
  `define WORD_SIZE 32. use ⇒ `WORD_SIZE.
- `include /* to include directive/files.
- `ifdef
- Ports - i/p, o/p.

Date / /

Port connection rules



- an illegal port declaration fadd fa (sum, c_out, a, b, c_in);
  - reg (sum), wire (c_out); reg on a, b, c_in crossed out
  - reg → port op should be a wire outside module.

⊙ Connecting ports to external signals

ⓐ by ordered list

ⓑ by name. fadd fa (.c_out (C_OUT), .sum (SUM), .b (B), .c_in (C_IN), .a (A));

↳ here I can reorder.

⊙ Gate-Level Modeling.

only use basic gates → and/or/not/xor/nand/nor/xnar

i/p pe no limit to these gates

⊙ Data-flow Modeling → Abstraction - consider data flow instead of gate level now.

⊙ Continuous assignment → to drive value onto a wire

assign out = in1 & in2;

Date / /

- LHS should be a wire
  RHS can be reg/wire

- Implicit assignment → wire out = in1 & in2
  ||| same as
  wire out;
  assign out = in1 & in2;
  |||
- Implicit wire declarn ⇒ assign out = in1 & in2

◉ Operators.

Arithmetic → +, -, *, /, %, **
mod, pow.

Logical → &&, ||, !

Relational >, <, >=, <=

Equality ==, !=, ===, !==
even check x==x & z==z

Bitwise ~, &, |, ^ (xor)

Shift >>, <<, >>>, <<<
logical, arithmetic

Concatenation { }
Replication { { } }

Date / /

• Behavioral → Algorithm / behavior of circuit done by actually implementing at a lower level of abstraction.

(a) Structured Procedures → always & initial
(i) initial
↳ starts at time 0 (simul^n time).
begin end me stmts likhde
used for initialization, monitoring, waveforms & all.
executes only ONCE.
(ii) always
Kind of an ∞ loop chalta rehta.
start at time 0.
to stop activity, off power ($finish) or interrupt ($stop).

(b) Procedural Assignments → to update value of reg.
(a) Blocking.
sequential execution hi hota hai
If RHS has more # bits from LHS, extra bits from MSP r discarded
If RHS has less, zeros r padded to MSB.

(b) Non blocking → using operator <=

simulator reads values of RHS variable.
Evaluate RHS & stores internally in simulator.
Write to LHS r scheduled to be executed at time specified in delays
Storing values ⇒ ~~order~~ of exec^n of writes isn't Important

Date / /

Otherwise race condition occurs.
- So used for concurrent data transfers

- Event based timing control.
  @ is used to specify event control.
  @(clock) q=d; //execute when clock changes value
  :
  always @(posedge clock)

- if else
  if ( )
  begin
  :
  end
  else
  :

switch case

- case (expression)
  val1 : expr1;
  val2 : expr2;
  :
  default : exprn;
  endcase

- Loop

  while ( )
  begin
  end

  for (; ;)
  begin
  end

- Parallel Blocks
  fork...join
  execute concurrently
  order of exec' controlled by timing delays (such as relative to time blk was entered)

  Sequential blocks
  begin .. end.
  stmt execute sequentially

Date / /

⊙ generate loop

```
genvar j;
generate for (j=0; j<5; j=j+1) begin: generate_loop
    xor g1 (...);
end
endgenerate.
```

Give any name (Give 1 MUST)

generate_loop [0] - g1
[1] - g1
: