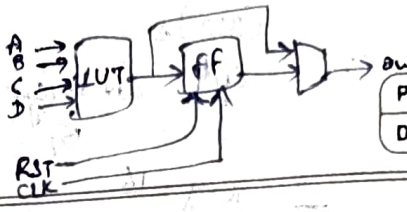
CMOS (22nm)
FinFET (14nm)
Quantum



PAGE NO :
DATE :

Verilog One shot:-

Size, fabrication, manual design, CAD, lower power design, standaring design flow, increase performance

Verilog, VHDL — CAD :- HDL based I/P → O/P (more detailed)

Behavioral → Register Transfer → Gate → Transistor level

Behavioral → Data Path → Logic → Physical → Manufacturing → Chip/FPGA

Behavioral → Flowgraph; Data Path → Bus/Reg; Logic → Gate/netlist; Physical → Transistor

Netlist: Directed graph

Design flow other steps
- Simulation
- Formal verification
- Testability analysis

Writing: easy Behavioral, bit tough structural → Physical

Description of digital system ≡ modules

Instantiation ⇒ create a copy/

Synthesis - ASIC, FPGA (Easy) → Programmable Hard

POR: alumina width = 1, origin [0,35]

(Process Chips) ASIC - High performance, optimised, expensive

FPGA - Fast Turnaround Time, less performance

Simulation: DUT - Design under Test; stimulus → response (monitor)

initial → executes only once; $monitor → prints only when vary

Test bench: begin $monitor ( ), $finish, end. $time system function gives current simulation time

>> iverilog -o mysim example.v example_test.v

>> vvp mysim

gtkwave example.vcd

Programmable logic cells, routing channels — Entirely by user

FPGA → IO cells, Look Up table blocks

Design flow

(Design entry → Implementation → Download)

Gate Array Chips: Generic masks (Independent) → Customization; much low level, Cheaper

supply > strong > pull > large > weak > medium > small > high z
driving (S) storage (S) (S)

PAGE NO :
DATE :

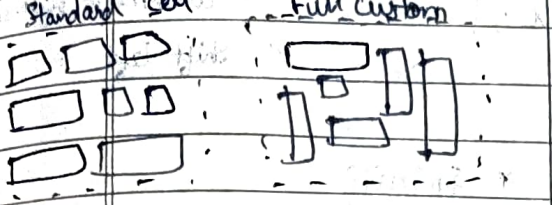
ASIC → Full Custom (memory cell/chip)

Widely used → Standard Cell "stick diagram" (rows, channels, through cells)

for logic chip design, compromise can be done : standard cell + Data path cell + PLAs (Programmable Logic Arrays)

Memory (layout 00)

Processor (Not much compact)



* assign statement ⇒ behavioural description

KFC

assign #1 c = a & b;

net, wire ↳ Register / net

Continuously driven

** Default value of net = z
: wire ≅ tri
wand, wor, supply0, supply1
unknown 'x' state

** Default value of reg = x
Synthesis tool try to determine the size of reg using data flow analysis

reg integer real time

* reg data-type Eg: real pi = 314.159e-2 = 314.159 × 10^-2

<size>'<base><number>

KFC

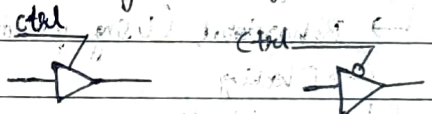
reg [31:0] register_bank[15:0];

Eg: 4'b0101

Eg: 25 //signed number, 32 bits

16 - 32 bit

register_bank[5];



if ctrl = 1, out = in
= 0, out = z

if ctrl = 0, out = in
= 1, out = z

bufif1 bufif0

* Parameter :-

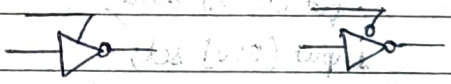
constant with given name

parameter H1 = 25, L0 = 5;

GATES used :-

NOT

buffer (signal value is stored)

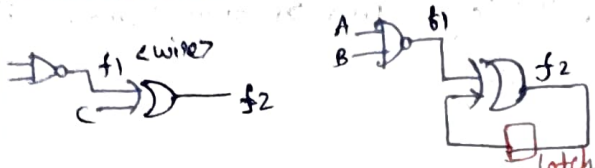


`timescale <reference_time_unit> / <time_precision>
(measure of time) (1, 10 or 100) s, ms, µs, ns, ps

Q) Specify connectivity?

i) Positional association (same order) Example DUT (X1, X2, X3, X4, X5, X6, OUT);

ii) Explicit association (arbitary order) example DUT (.OUT(Y), .X1(A), .X2(B), .X3(C), .X4(D));

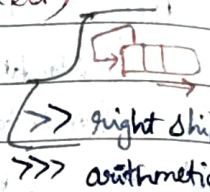


PAGE NO :
DATE :

* Reduction operators:- (unary)

multiple bit

Eg: wire [3:0] x;
wire y;
assign y = &x;



>> right shift ÷2
>>> arithmetic right shift
<< shift left *2

* Concatenation operator:

assign f = {a, b};
if a is 3 bit, b is 4 bit vector
f becomes 7 bits vector

* {n{m}}

Eg: assign f = {2'b10, 3{2'b01}, x};
f = 10 01 01 x

KFC

Q) a = 101x b = 1010

a == b ⇒ true

Exactly equal to

a === b ⇒ false

Modelling → Behavioral (Easy)
→ Structural
Hardware implementation

{ reg can be storage cell / wire }

assign, static

Description styles
→ Dataflow (assignment statements) (Continuous assignment) (Combinational ckt)
→ Behavioral (Using procedural statements similar to High level programs)
→ Blocking
→ Non-blocking

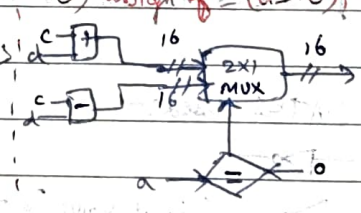
Eg: (A) MUX:-

```
module generate_MUX (data, sel, out);
input [15:0] data;
input [3:0] sel;
output out;
assign out = data[sel];
end module
// assign out = sel ? a : b;
```

Non constant index in RHS

Conditional operator creates MUX

(A) Q) assign f = (a==0) ? (c+d) : (c-d);



16 of 2x1 MUX

(A) assign out[sel] = in;

Not const index on LHS ⇒ Decoder

(A) assign Q = En ? D : Q;

Latch is generated

reg → seq
↳ comb.
net → comb

PAGE NO :
DATE :

initial begin reg = end

it ⇒ OS(all) blocks

Sequential statements

Behavioural / Procedural

blocks

Initial

Always

* output reg [7:0] data;
* reg clock = 0; instead of reg clock; inital clock = 0;

(a) always @ (event_expression) is required for both sequential & comb,.. ckt.

(b) one statement ⇒ begin...end not required.

casez : Treats all 'z' values in case alternatives as don't cares.

casex : Treates all 'x' & 'z' values in case item as don't cares

1.xxx
x1xx
xx1x
xxx1

for Fixed no of times

repeat < expression >
    Sequential_statement;

forever
    sequential_statement;
    delay needs to specified if not

- posedge : {0, x, z} → 1, 0 to {x, z} time will never advance ftime
  negedge : {1, x, z} → 0, 1 to {x, z} Eg: forever #5 clk = ~clk;
* @ (a, b, c) = @ (a or b or c)

combinational

sync seq.

Blocking V/S Non blocking procedural assignment statements

LHS = RHS

LHS <= RHS Swapping 2 numbers

comb → net → block (volleyball)

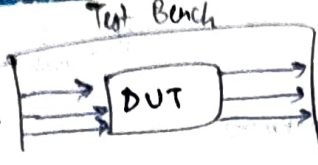
reg sum[15]
net, reg sum[20:15]

[stmts execute concurrently]

Eg:
```
integer a,b,c
initial
begin
a=10; b=20; c=15;
a = b+c;   // a = 35
b = a+5;   // b = 40
c = a-b;   // c = -5
end
```

a <= b+c 35
b <= a+5 15
c <= a-b -10

2° lecture
Watch



Traffic light code
5 Flip Flops will be ...

PAGE NO :
DATE : 2 (table)

Test bench :- (Only for simulation)

Initial procedural block, DUT, print values / see waveforms

$display → prints immediately

Design

Eg:

```
module example
(A,B,C,D,E,F,Y);
input A,B,C,D,E,F;
output Y;
wire t1,t2,t3,Y;
nand #1 G1 (t1,A,B);
and #2 G2 (t2,C,~B,D);
nor #1 G3 (t3,E,F);
nand #1 G4 (Y,t1,t2,t3);
endmodule
```

```
module testbench;
reg A,B,C,D,E,F; wire Y;
example DUT (A,B,C,D,E,F,Y);
initial
begin
$monitor ($time, "A = %b, B = %b,
C = %b, D = %b, Y = %b",
A,B,C,D,Y);
#5 A=1; B=0; C=0
#5 A=0; B=0; C=1; D=1; E=0;
#5 $finish;
end
endmodule
```

- Input → reg
- output → wire

$display, $monitor, $dumpfile, $dumpvars, #5 $finish,

$dumpfile (< .vcd >); [Value change dump]

dumpvars level = 0 (all variables in module)

dumpvars level = 1 (only listed variables), dumpall (all)

$dumplimit (filesize);

$random (<seed>) seed is optional and is used to ensure that same sequence of random numbers are generated each time the test is run

FSM :- (state table, state transition diagram)

- most of the practical circuits are sequential in nature.
- variation of FSM : ASM (Algorithmic State Machine Chart).

- Deterministic FSM can be mathematically defined as a 5-tuple

~~(Σ, T,~~ (Σ, T, S, S_0, δ, w)

Inputs ↵ ↳ outputs

S → set of states (finite)

S_0 → initial state

δ → state transition function

w → output function

Mealy $w : S \times \Sigma \rightarrow T$

Moore $w : S \rightarrow T$

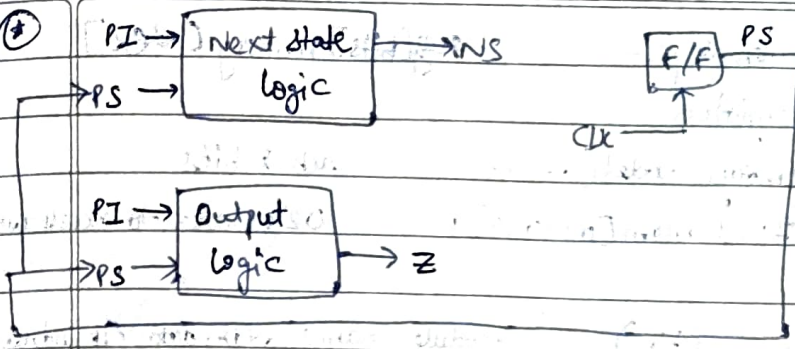
29 lecture watch

```
module fulladder (s, cout, a, b, cin);
  input a, b, cin;
  output s, cout;
  assign s = sum (a, b, cin);
endmodule
```

```
function sum;
  input x, y, z;
  begin
    sum = x^y^z;
  end
```

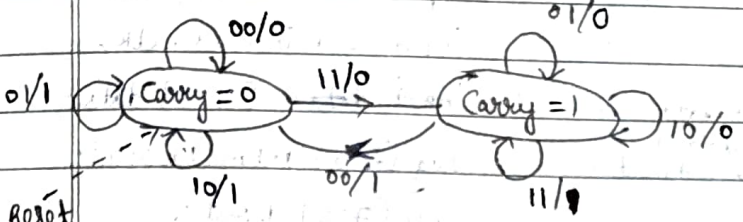
PAGE NO :
DATE :

(*)



Sequence Detector "0110" (Moore model)

(*) Serial Adder :-



Task can give more than 1 output

More general

inout variable → Bidirectional

DATA PATH and Controller Design

Functional Units/ Computations (blocking statements) : FSM, generates control signals to the data path in proper sequence (non-blocking)

- PIPO register

Synthesizable Verilog :-

- No use of delay
- No feedback in comb. ckt
  - ↳ may turn in seq. ckt
- output for all cases (if, switch, etc.)
- No use of fork...join, wait, disable, force...release
- variables in loop control not supported

* Netlist * UDP (User defined primitive)
* Continuous assignments * Functions
* Behavioural State
* Interconnected modules using one/more of above mentioned modules

Styles

[I] HALTED (WB)

[II] TAKEN_BRANCH (EX)

Set Less Than - SLT

MIPS → Example of RISC

4 X 32
R0, R1, R2, R3 each 32 bits
4 words

low level module ≡ leaf level module

PAGE NO :
DATE :

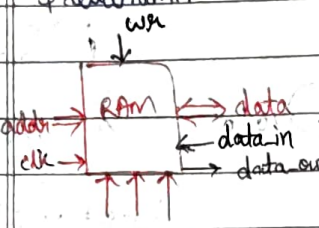
Modelling memory :- reg[bits] memory [index]

- Array of registers

module memory_model(.....)
  reg [7:0] mem[0:1023]; each 8 bits, 1024 index to access word
endmodule

* $readmemb (bin)
* $readmemh (hexadecimal)

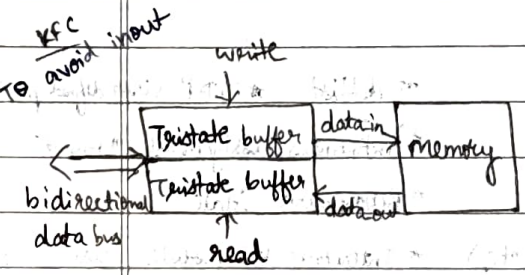
c)



module ram_1(addr, data, clk, rd, wr, cs);
input [9:0] addr;
input rd, wr, cs, clk;
wire (inout) [7:0] data;
reg [7:0] mem[1023:0];
reg [7:0] d_out;
assign data = (cs && rd)? d_out : 8'bz;
always@(posedge clk)
  if (cs && wr && !rd)
    mem[addr] = data;
always @ (posedge clk)
  if (cs && rd && !wr)
    dout = mem[addr];
endmodule

d)

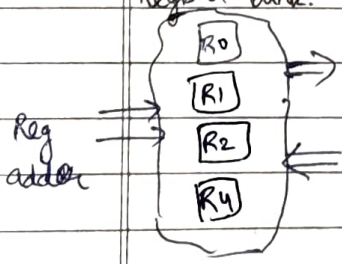
KFC / avoid inout



master slave 2phase clock / Latches

Linear
Non-linear

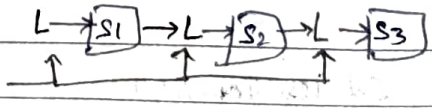
Register bank :-



- Supports concurrent access

Pipelining :-

Uses :-
* Instruction execution
* Arithmetic computation
* memory access



Latch (MS FF)

Q) How processing is carried out?

Data structure ← Reservation Table !!!

KFC

34 lecture

Watch

speedup & efficiency:-

$T_m = \max\{t_i\}$

$T_{m+d_L} = \tau$ $\quad f = \frac{1}{\tau}$

$\tau \Rightarrow t_{skew+jitter} + t_{logic+setup}$

PAGE NO :

DATE :

$d_L \rightarrow$ latch delay

$t_i \rightarrow$ time delay of stages

$\tau \Rightarrow$ Time period of pipeline (clock period)

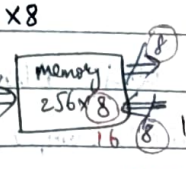
- $T_k = [(k-1)+N]\tau$
- $T_1 = NK\tau$
- $S_k = \frac{T_1}{T_k} = \frac{NK}{K+N-1}$

K stages

** $N \rightarrow \infty$, $S \rightarrow K$

- $E_k = \frac{S_k}{K} = \frac{N}{K+N-1}$ Pipeline η
- $H_k = \frac{N}{T_k} = \frac{N}{(K+N-1)\tau}$ Pipeline Throughput

Q) 256 words × 8

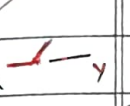


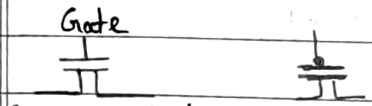
$2^8 = 256$

To make it Compactible

only simulation NO synthesis

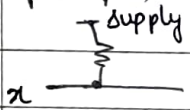
- switch level circuit comprises of netlist of MOS transistors
  - ↳ nMOS, pMOS
  - Switches
    - ↳ Ideal (zero Ω)
    - ↳ Resistive (finite low Ω)



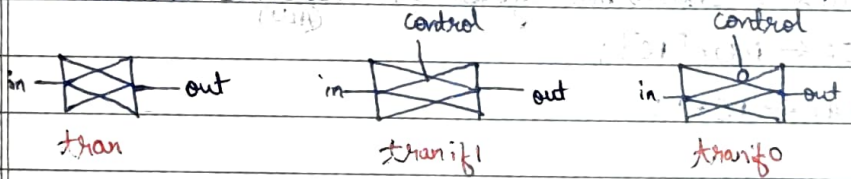


KFC

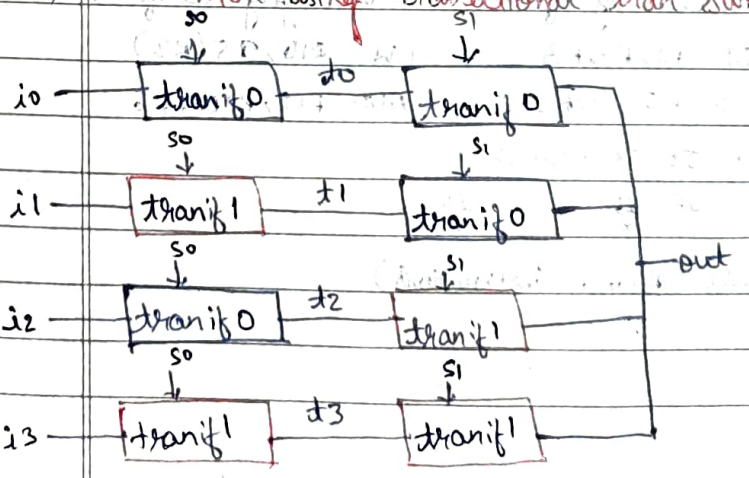
- cmos: voltage degradation is avoided
- * pullup(x) * pulldown(x) ⇒ have lower signal strength due to resistance

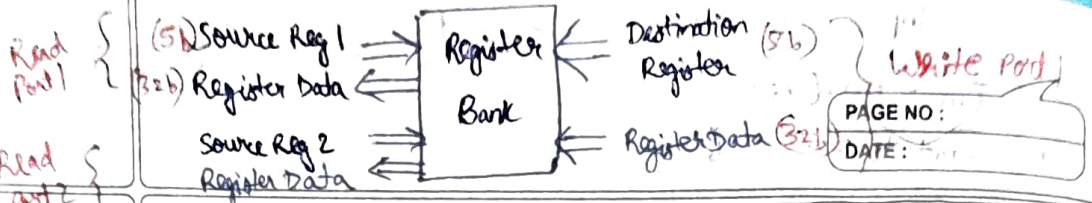


- Bidirectional switches:-



Q) 4×1 MUX using Bidirectional 'tran' switches:- MUX/DEMUX





PAGE NO :
DATE :

Pipeline Implementation: Look Up Table → SRAM

- RISC - MIPS32 - Large number of registers (32 bit processor)

(*) R0 to R31 (GPRs)

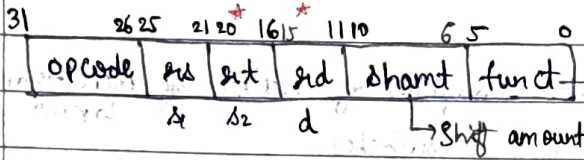
(*) R0 contains constant 0 (*) word addressable

ADD R1, R2, R3 // R1 = R2+R3

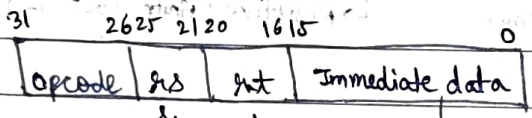
ADDI R1, R2, 25 : CLB - Configurable Logic Block

Instruction Coding (R, I, J) (Register, Immediate, Jump)

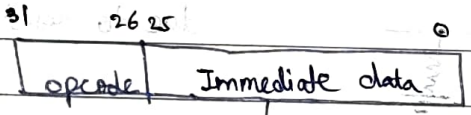
i) R-type:-



ii) I-type:-



iii) J-type:-



ALU Memory (load)

MIPS32 Instruction Cycle: IF, ID, EX (ALU), MEM, WB (Write Back)

IR ← MEM[PC];

NPC ← PC+1;

Hazards in Pipeline:-

a) Structural (one copy of data → 2 are trying to fetch)

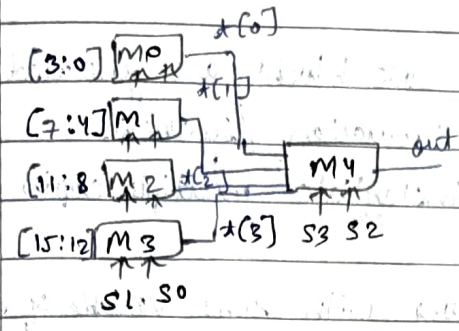
b) Data

```
ADD R1, R2, R3        LW R10, 25(R3)
SUB R5, R1, R3        ADD R12, R10, R6
IF ID EX MEM (WB)
   IF ID  ...  ID
```

c) Control Hazards (Branch instructions)

0011
^0110
0101

PAGE NO :
DATE :



16:1 MUX using 4:1 MUX

ASSIGNMENTS :-

- Table (FPGA, Full Custom, Semi Custom, Gated Array)
  Full custom > Semicustom > Gate Array > FPGA
  Decreasing order of speed

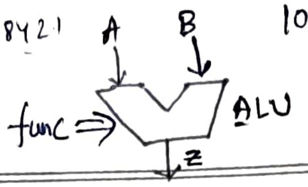
- Design, Behavioural, Data Path, Logic, Physical, Manufacturing, Chip/Board

- No of distinct functions $2^{(2^n)}$ for n variables
- If size of value is not explicitly mentioned, it will be 32 bits (Constant)
- $b \wedge x = x$
- y = y >> 1 (y = 2435) will be halved everytime
- Produral block, only register type variable can be used in LHS
- assign a = b + c ; a should be net (Continuous driven)
- always @(posedge clock)
  begin
  y = x;
  z = y;
  end
  2 D-Flipflops will be generated
  Non-blocking :) ⇒ 2 bit shift register
- Primitives :

→ can be used to specify comb. ckt with single output.

→ Can be used to specify FSM with only one state variable.

shamt ⇒ amt $ L6 8421   101 ⇒ M   Load 1000



PAGE NO :
DATE :

01XZ   LHXZ

- Generate block:-
  - Multiple copies of code blocks are generated dynamically before simulation / synthesis.
  - Can be used to instantiate multiple copies of some module.
  - Must be used along with some variable of type "genvar".

- State changes are carried out inside an "always" block triggered by clock.
- Control signals are generated in an "always" block using blocking assignments.

VVIP / MFC

OPCODES :)

	R(5)		I (14)		J
000000	ADD	001000	LW	010000	J
000001	SUB	001001	SW		
000010	AND	001010	ADDI		
000011	OR	001011	SUBI		
000100	SLT	001100	SLTI		
000101	MUL	001101	BNEQZ		
111111	HLT	001110	BEQZ		

Q) LW R20, 84(R9)

001000   10100   0000000001010100

01001

21340052 (hex)

Q.) BEQZ R25, Label

001110   11001   00000   YYYYY...YY

= 3620YYYY (hex)

✓✓ ALU codes (4 bits)

0000	ADD	Arithmetic
0001	SUB	Select
0010	MUL	Logical
0011	SELA	Shift
0100	SELB	
0101	AND	
0110	OR	
0111	XOR	
1000	NEGA	
1001	NEGB	
1010	SRA	
1011	SLA	