→ Introduce notion of RAM

2.4.18

Random Access Memory

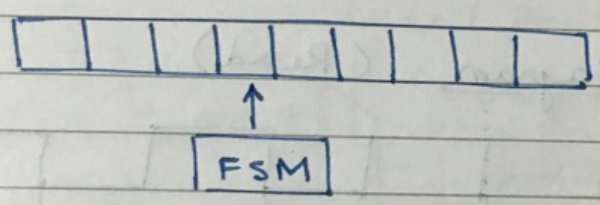
Combi → FSM → RTL → RTL + RAM ≈ Turing Machine

↕

Parallelism (forks)

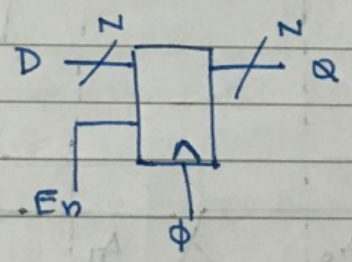
↳ In VHDL → behav

Turing Machines

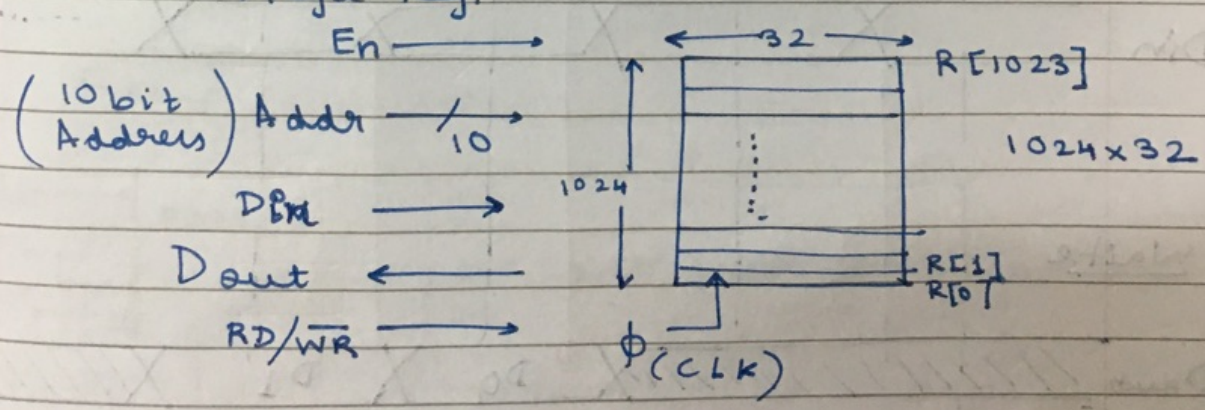


→ Registers (RTL) + RAM → Turing Machine

- Recap: Reg.



RAM → Mega Reg.



$Q(k+1) \leftarrow [En(k) \,?\, D(k) : Q(k)]$

↳ Action of a Register

RD & $\overline{WR}$ are the same

→ **Program:**

if ($\overline{WR}(k)$ == 0) && (En(k) == 1

R[Addr(k)](k+1) ← Din(k)

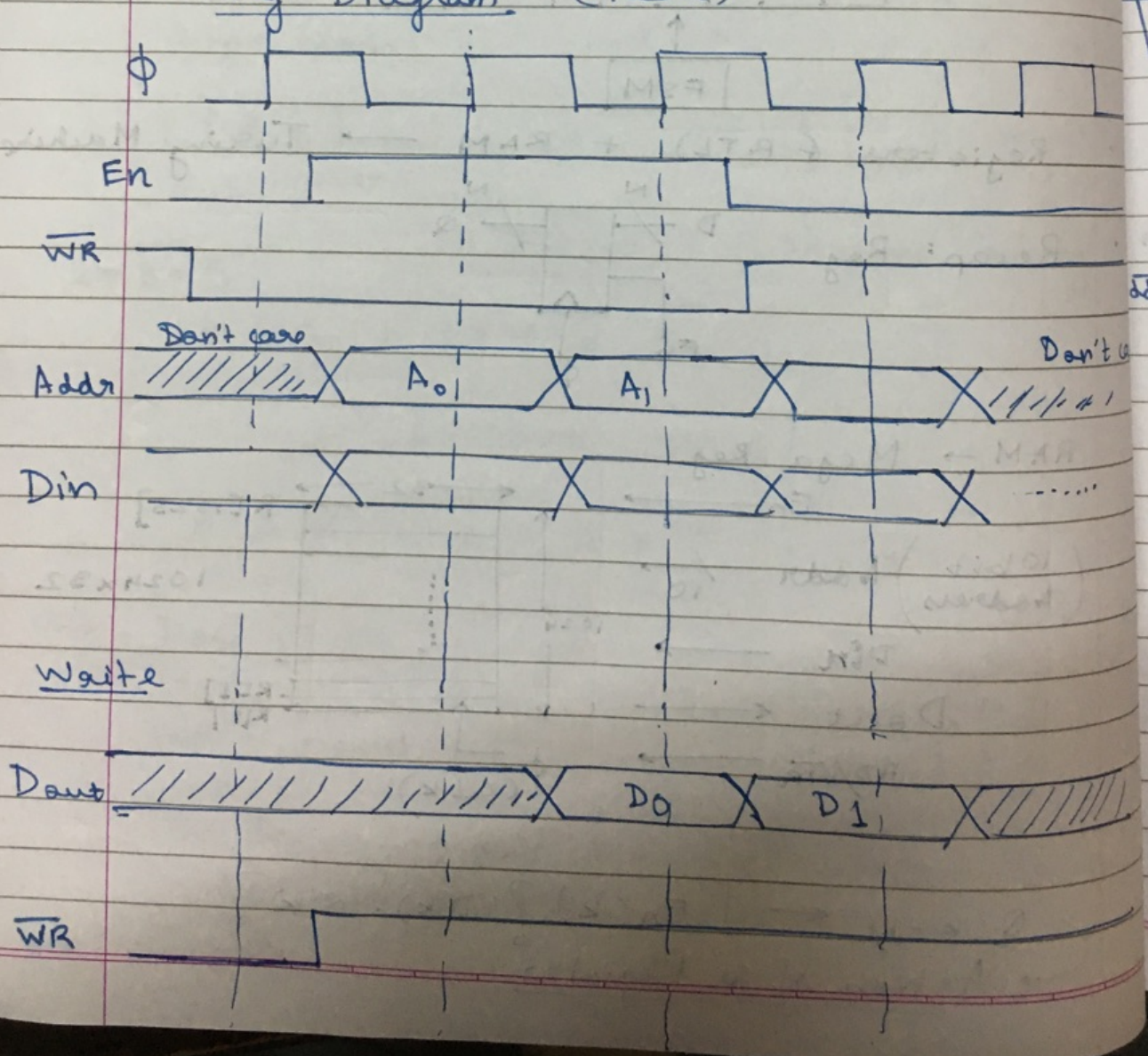
↳ Takes 1 clock cycle to load

→ Single Cycle Access synchronous memory

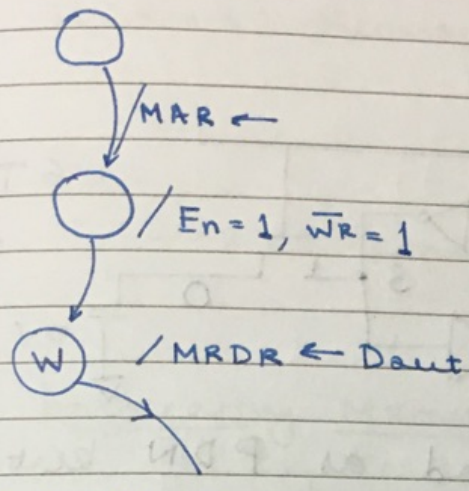
if (RD == 1) && (En(k) == 1)

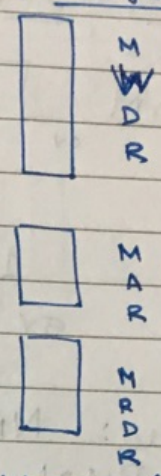
Dout(k+1) ⇐ R[Addr(k)]

→ Timing Diagram (Read)



Interface Reg. to Memory

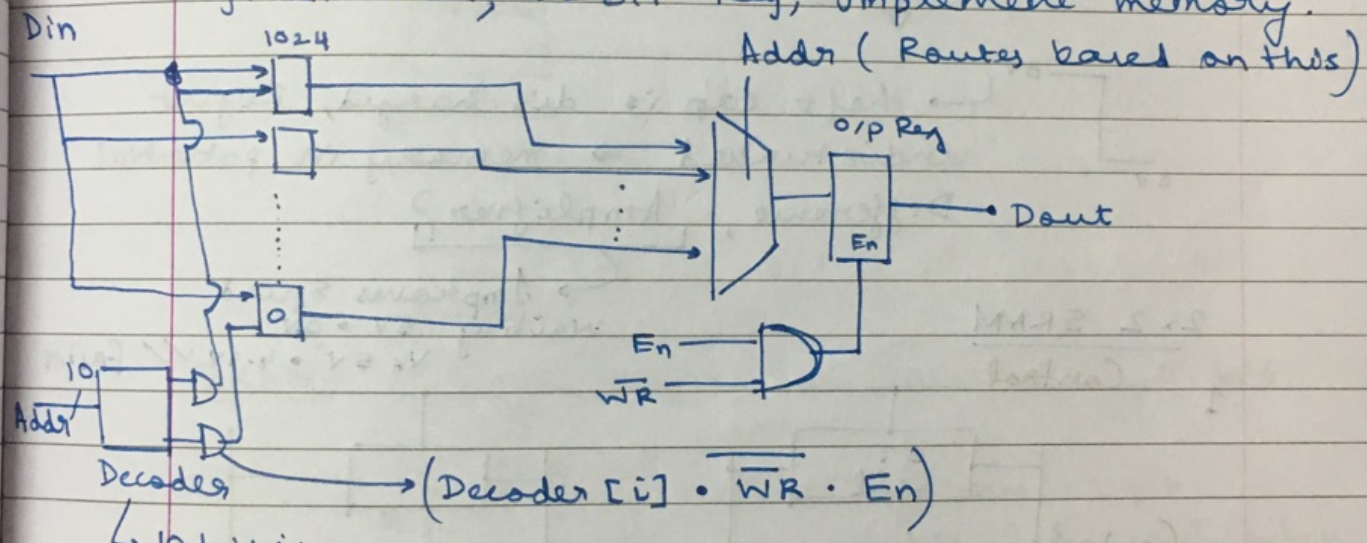




Mem. Add. Reg.

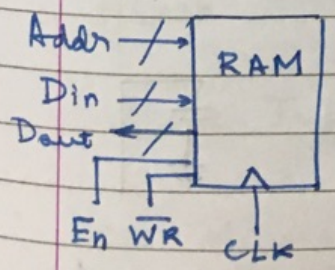
→ Not efficient : No. of Reg ↑, 3 CLK cycles

- Given 1024, 32 bit Reg, Implement memory.



↳ 10 bit i/p
1024 outputs

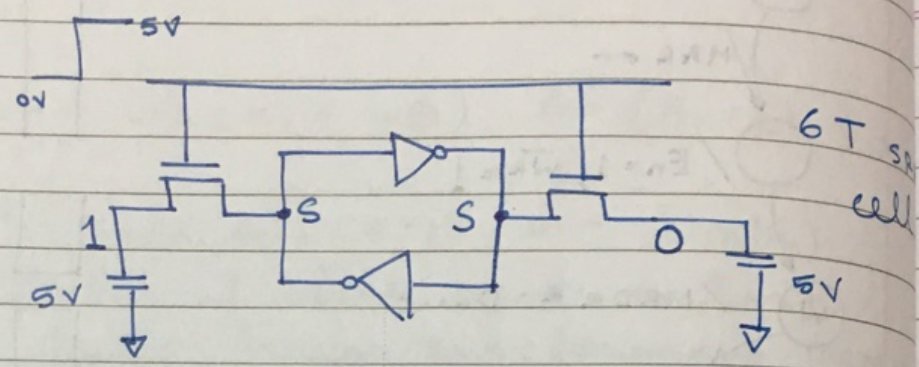
3.4.18 Recap



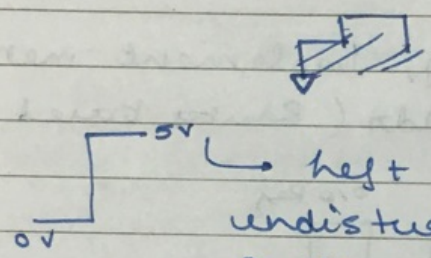
Last time

→ RAM - SRAM, DRAM

SRAM



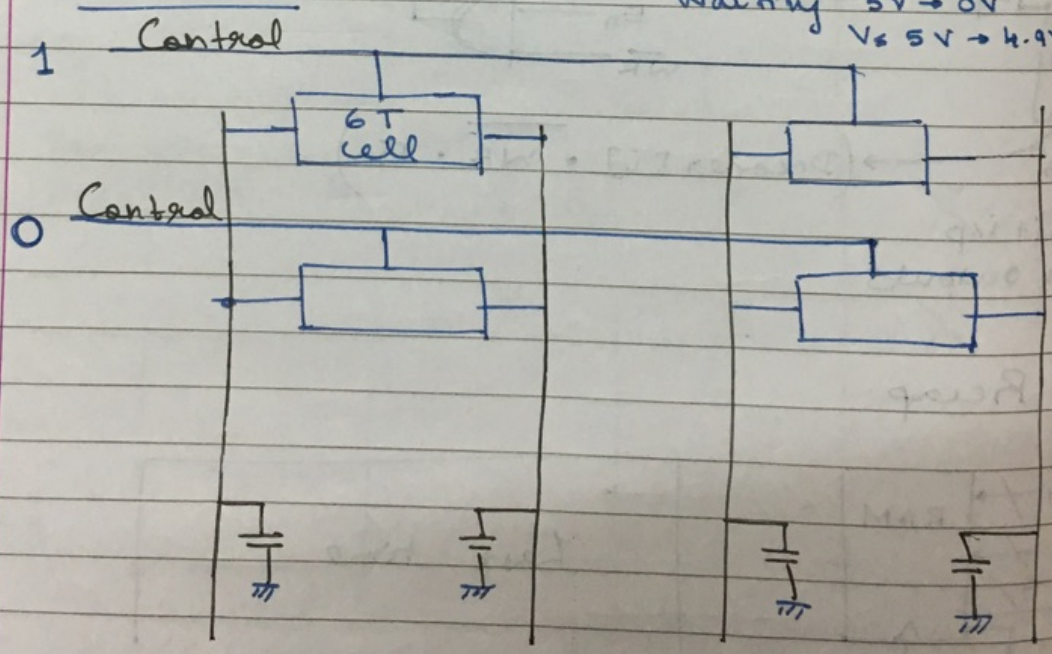
→ Idea: NMOS is Good as PDN but not PUN.



→ left cap is discharged, Right undisturbed ⇒ memory in potential Difference, Amplifier?

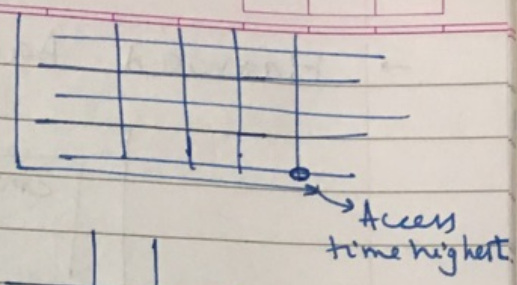
→ Improves speed waiting 5V → 0V
Vs 5V → 4.9V

2×2 SRAM

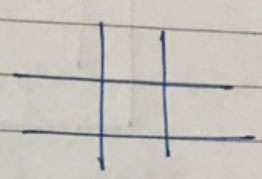


→ The Word (32 bits) which has control = 1 will be Active.

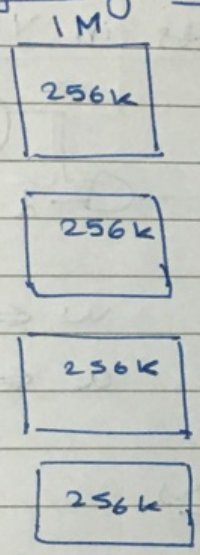
1024 × 32 Memory



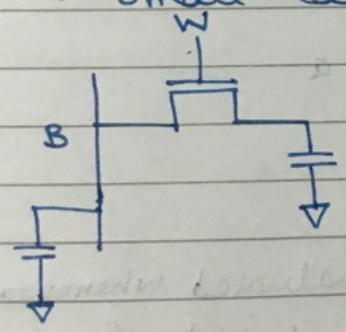
2×2 Memory



→ Aggregating Memory

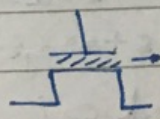


DRAM : Small Capacitors store a single bit

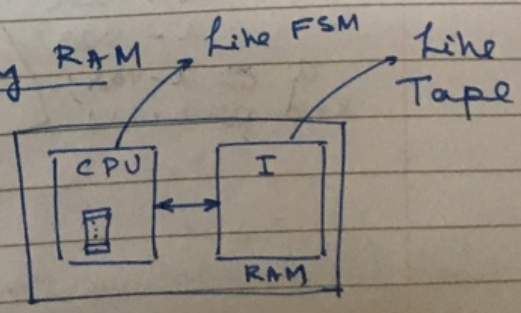


→ Stores 1 bit
→ We loose info when we read → rewrite
→ Slow Access → 100 × of SRAM

Flash :

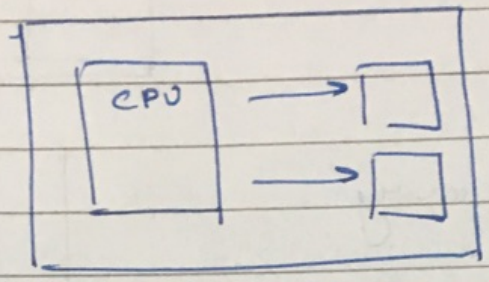


Mimic Turing Machines using RAM



• Von Neumann Computer (As powerful as a finite Turing Machine)

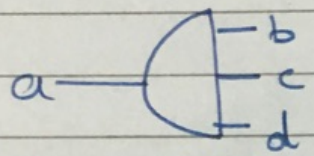
→ Harward Architechture

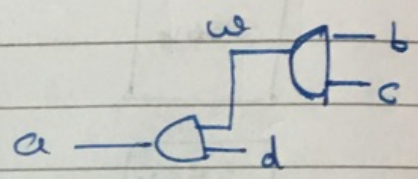


Heap & Stack
Seperate as
Something like th

5.4.18

Process Statements in VHDL

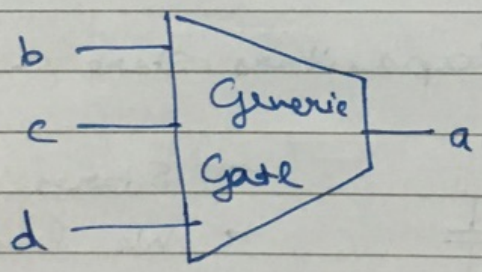




a ⇐ b and c and d

w ⇐ b and c;
a ⇐ w and d;

Now way to design drivers



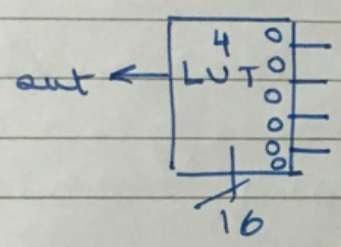

```
process (b, c, d) // Evaluated whenever event on eith
    variable a_var : std_logic;
    ↳ used for temp. storage
begin
    a_var := b and c;
    a_var := a_var and d;
    a ⇐ a_var;
end process
```

↳ Immediate assignment used for variables

Signal Vs. variable

• Transaction happens in the future, • ~~Variable~~ • There is a delay b/w execution & evaluation	• Immediate assignment (In software)

FPGA has a LUT: Implement? (Any function on 4 i/p's)



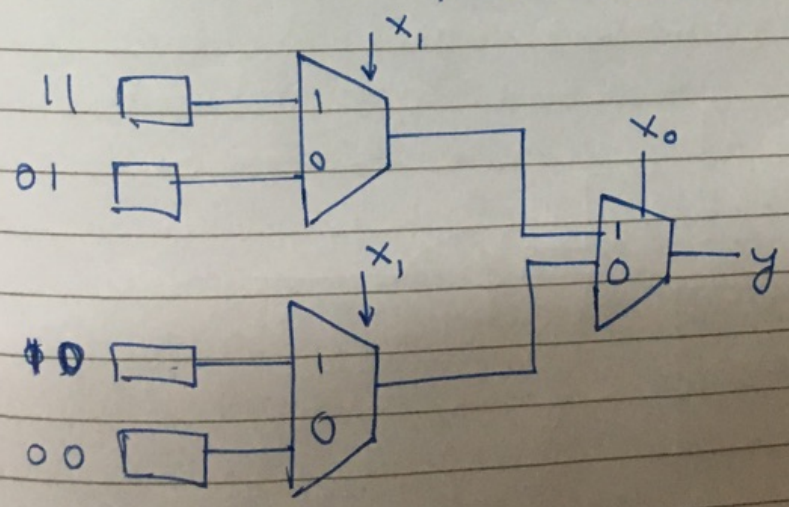
2^{2^4} functions

⇒ 16 bits of Memory required to implement

→ An LUT can implement Any function

We can use a MUX

→ like for 2 inputs (Covers all cases)

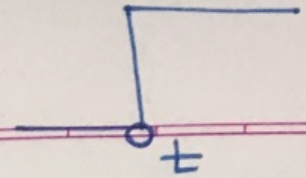


i/p x_0, x_1

LHS ⇐ RHS, LHS must be signal / out type.

a_var := b, LHS must be variable.

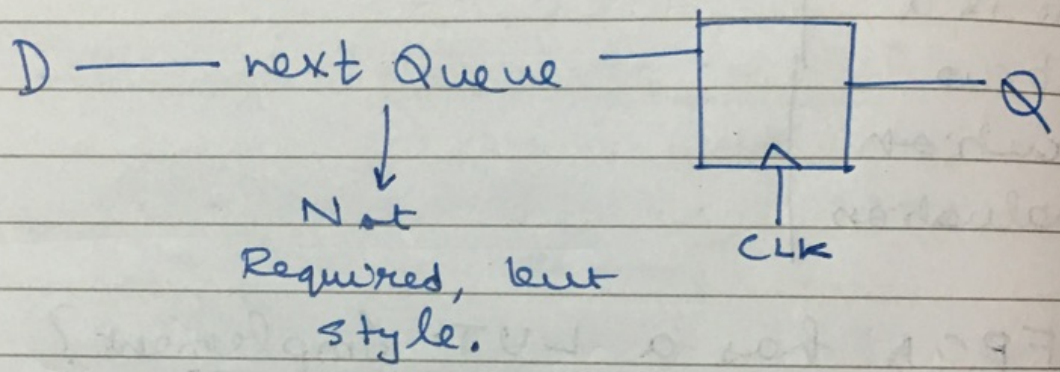
('event) →



CLK'event → 1 at time t if clock cha
at time t
→ This is on the PC

CLK = '1' → value just after is used.

→ What does syn. tool do to make H/W

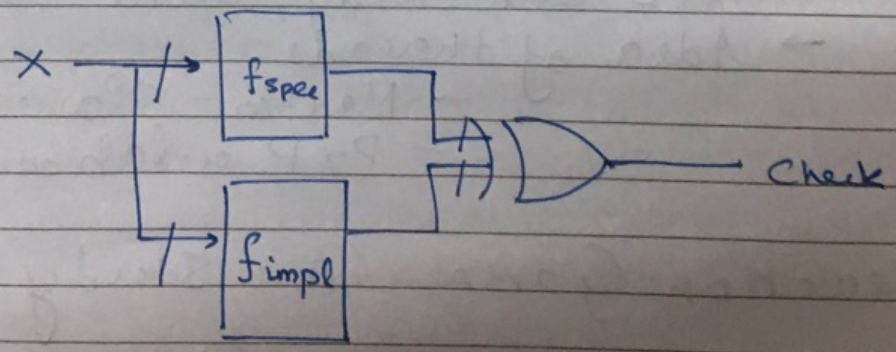


9.4.18

Testing, Verification & Validation

→ Are spec ≡ implementation

→ Can't go through all cases



→ Make sure / show that Check (X) ≡ 0

→ The satisfiability Problem

CLA Adders / Serial Adder

$x_3 \quad x_2 \quad x_1 \quad x_0$

$y_3 \quad y_2 \quad y_1 \quad y_0$

$s_3 \quad s_2 \quad s_1 \quad s_0$

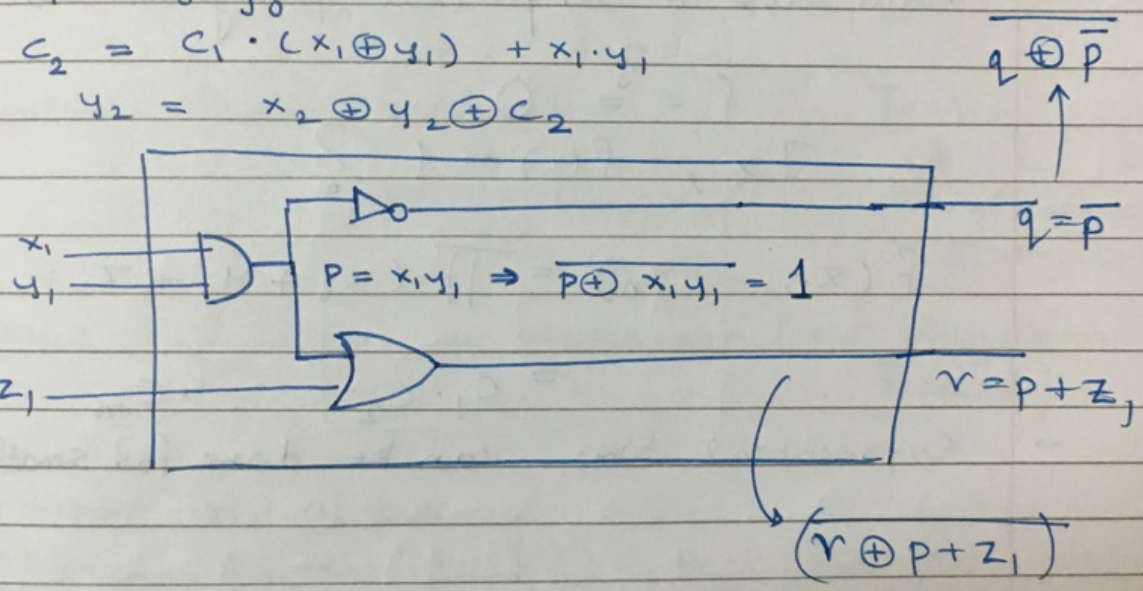
$$s_0 = x_0 \oplus y_0$$

$$y_1 = x_1 \oplus y_1 \oplus c_1$$

$$c_1 = x_0 \cdot y_0$$

$$c_2 = c_1 \cdot (x_1 \oplus y_1) + x_1 \cdot y_1$$

$$y_2 = x_2 \oplus y_2 \oplus c_2$$



→ Combination of all formulas

$$f(\ ) = \left(\overline{q \oplus \bar{p}}\right) \cdot \left(\overline{p \oplus x_1 y_1}\right) \cdot \left(\overline{r \oplus (p + z_1)}\right)$$

→ But we don't have access to individual output pins to Design check.

ins outs intermediates

$f_{spec}(x, y, w)$

$f_{impl}(x, y', w')$

formula

A	B	
1	1	1
1	0	0
0	1	1
0	0	1

→ $A \Rightarrow B$ is always true $(\bar{A} + B) \equiv A \Rightarrow$
$\equiv \overline{AB}$

We want

$$\underbrace{(f_{spec} \cdot f_{impl})}_{\text{Both meet their specifications}} \Rightarrow \underbrace{(y == y')}_{\text{then the outputs will be the same}}$$

→ How will a computer program do it?

$$f \overset{?}{=} 0$$

$$\exists x, \ f(x) = 1 \ ?$$

$$f(x_1, \ldots, x_n) = \prod (x_i + y_i + z_i + \ldots)$$

$$= c_1 \cdot c_2 \cdot \ldots \cdot c_m$$

→ Exponential time, can be done for small samples

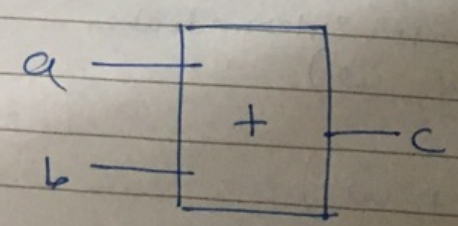
Specification ⇔ Implementation

properties ↔ invariants

properties ↔ invariants.

→ How do we know if Specification is correct?
We need

properties ↔ invariants



Properties of Adder: (P(a, b, c))

Exhaustive list of properties may/may not exist

- $a + b = b + a$
- $a + a = 2a$ (a Shifted by Right)
- ⋮
- .....

↳ Solution: It should never be the case that spec → true but property not.

Testing

→ Different from verification . Testing

Verification ↓ Design Process

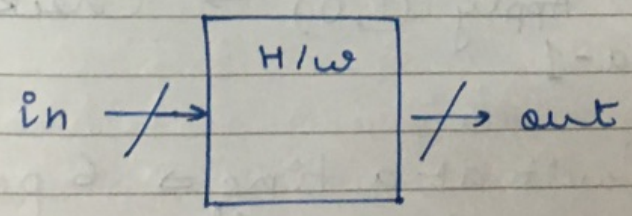
Testing ↓ Manufacturing process

→ Related to working of Hardware (ex. Krypton Board)

Questions

→ What can go wrong.

→ How do we identify a test for it.



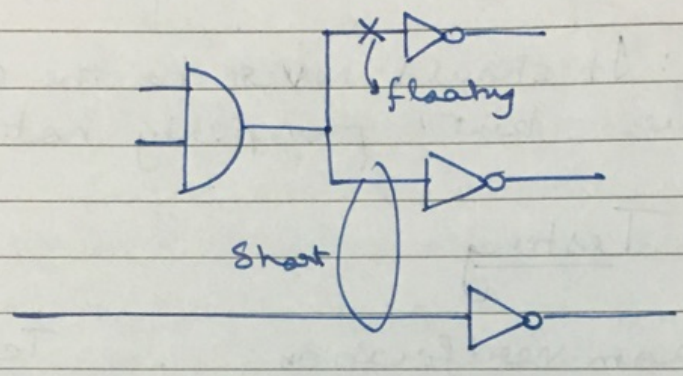
1) Wiring problems
   - → Broken Connection
   - → Short Circuit

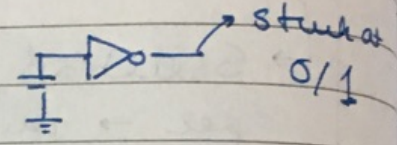
2) Defective Gates / transistors
   - → Always off / on

→ ~~Sol~~ Too many defects → ~~Sol~~ Simple Model (Stuck-at mod

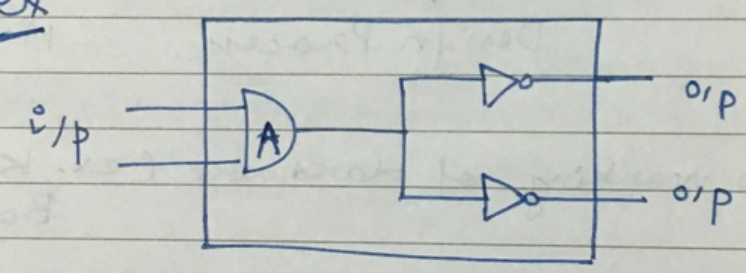
Gate → OK, Stuck at 0, Stuck a 1

ex





ex



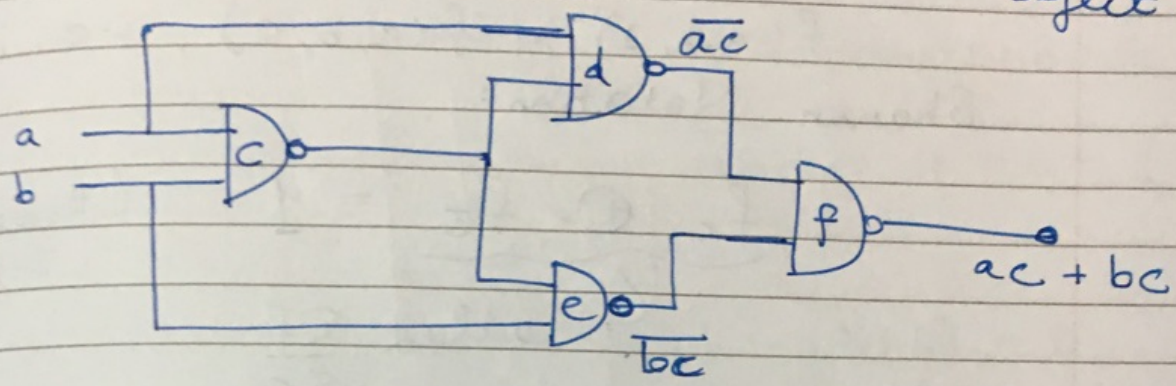
if A is ~~sa~~ s-a-0
→ Apply (1,1) → Check

if A is s-a-1 → Apply (1,0) → Check

→ If one fault at a time ⇒ 6 possible faults above can detect all.

10.4.15

Testing for faults (Detect the fault)
↳ Defect



→ We assume that only one Gate is faulty at a time

c	s-a-0	
	s-a-1 ✓	
d	s-a-0 ✓	
	s-a-1	
e	s-a-0 ✓	
	s-a-1	
f	s-a-0	
	s-a-1 ✓	

1) C : s-a-0

Then, we contraints.

$$\overline{a \cdot b} = 1$$

But for $a=0$, $b=0$ the error is not observable at f,

$$f = ac + be = (a+b) \cdot c$$

Hence if we want a change in c to reflect in f, then $(a+b) = 1$

$$c = ab$$

Constraints ⇒ $\overline{ab} = 1$, $a+b = 1$ (?)

$$\boxed{\bar{a} \cdot b + a \cdot \bar{b} = 1}$$

↳ $a \oplus b = 1$ to detect fault

→ Requirement to Detect fault at c using:

$f(a,b,1) \neq f(a,b,0)$, i.e in Shanon Notation:

$$\underbrace{f_c \oplus f_{\bar{c}}}_{\text{called } \frac{\delta f}{\delta c}} = 1$$

(Boolean Difference)

→ Hence our test ~~is~~ for c : s-a-0 is

$$c \cdot \frac{\delta f}{\delta c} = 1$$

$c = 1, \quad \overline{ab} = 1$

$$f(a,b,0) \neq f(a,b,1)$$

$$\frac{\delta f}{\delta c} = 1 = f_c \oplus f_{\bar{c}} = 1$$

$\Downarrow$

$\underbrace{a = 1, b = 0}$, $\Rightarrow c = 1, d = 0, e = 1, f = 1$

↳ Detects c: s-a-0 (f exp ≠ f obs), Also detects d: s-a-0 progress to o/p is blocked ⇒ fault is not propagated

Why d: s-a-0 is detected

→ Detect c: s-a-1 (Green cases detected)

$c = 0$ $\quad \overline{ab} = 0$ → constraints

$f = (a+b) \cdot c$ $\quad \overline{ab} = 0 \Rightarrow ab = 1$

→ $f = (a+b) \cdot c$

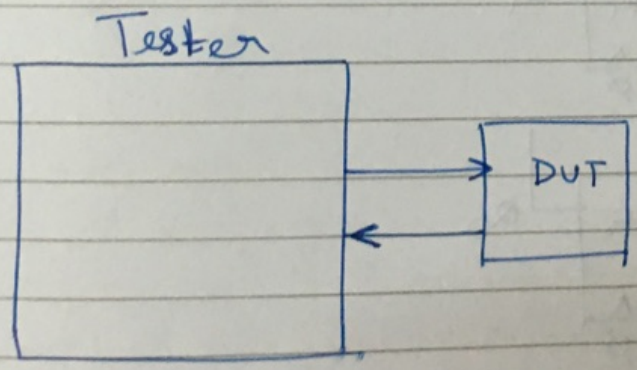
for c to reflect in f, $(a+b) = 1$ and $ab = 1$

→ $ab = 1$

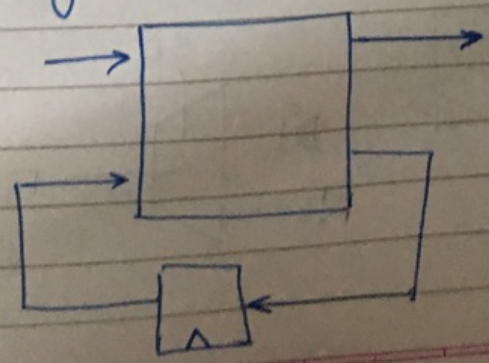
$$\boxed{a = 1, \ b = 1}$$

Ref: Kohani

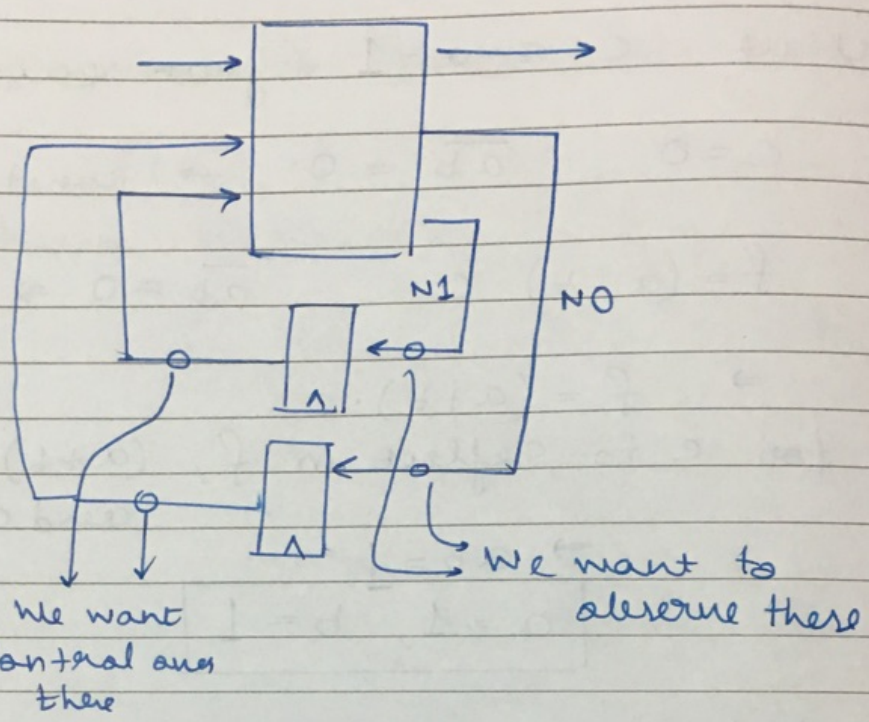
→ We must find input patterns that excite faults.



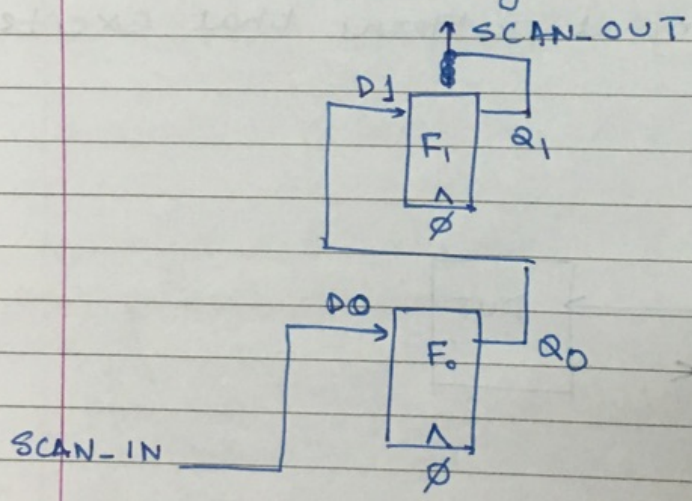
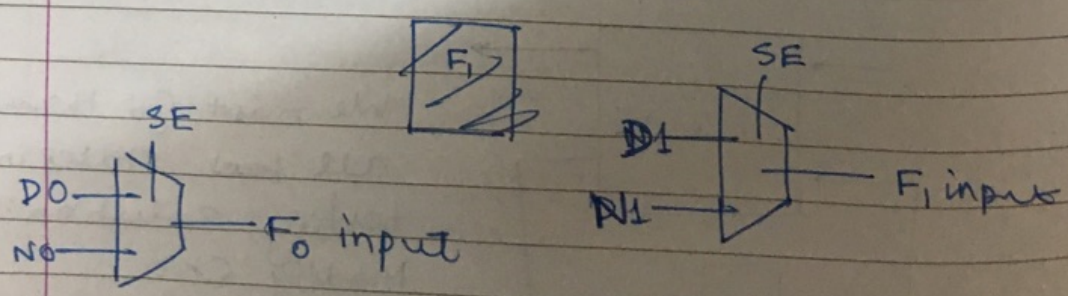
→ Testing Sequential Circuits



We must go through all states in testing, solution: Modify Ckt.



→ Solution: Modify DFF



Replace Inputs of DFF