07/01/2025

EMBEDDED SYSTEMS

V/S PCs: many diff. tasks

Dedicated functions, limited resources

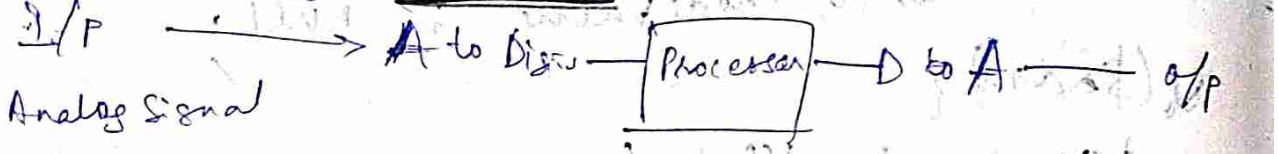
tasks system should respond to i/p within specified time frame

3 types of Real Time systems (deadline miss)

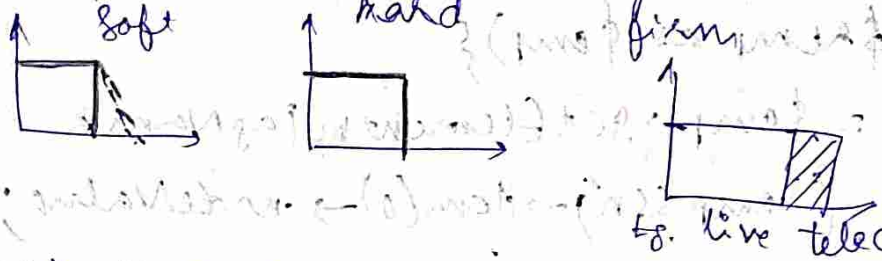
(1) Soft Real Time Systems: ~~Deadline~~ is not catastrophic

(2) Hard: deadline very important; gaming

eg. submarine



(3) Firm: If deadline is missed we will just ~~the o/p next~~ ignore that and move on;



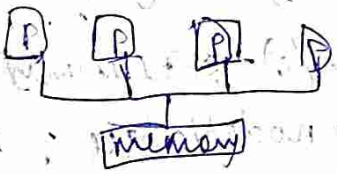
At SC when we miss deadline we are not going to take it hard we ignore it)

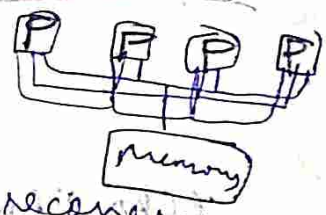
SISD: Uniprocessor

SIMD, MISD, MIMD: Multiprocessor

Robotic 100% parallelism

Interconnection Networks: Processor to memory

linear bus: 

Multi 

Adv: recovery

Disadv: all buses do the same tasks again and again

08/01/2025

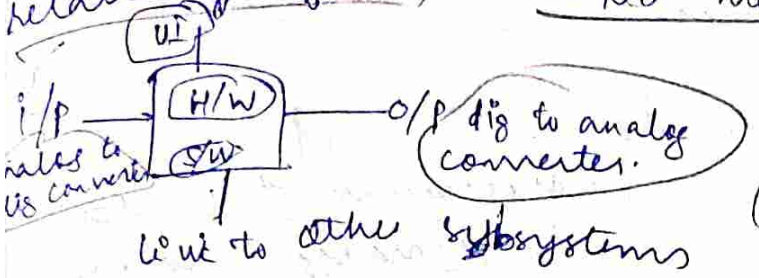
Features of Embedded System:

1. Special purpose/specific, single programming, cannot be changed once chip is made
2. Tight constraint of energy, cost, space, performance
3. Receives input → computation → and produces result

Microprocessor which control other subsystems is an embedded system

Microcontroller based system to do specific functions, cannot be modified by end user.

design constraint: less power, low cost, less space, relatively fast, limited memory, real time response

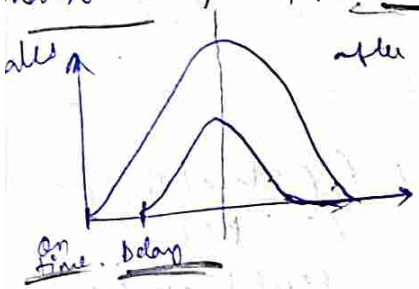


mcus also have timers, counters, PWM (pulse w/ modulation)

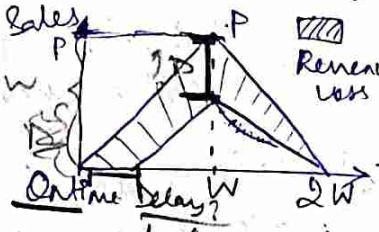
Microcontrollers: all components in a single chip
Microprocessor: separate components like memory.

16/01/2025 Goal of embedded system: to perform its own functionality, realise and

while designing:
Non Recurring Cost, Unit cost, size, performance, flexibility, maintain, time to prototype, time to market, safety



after 10 yrs product becomes obsolete



Area $= \frac{1}{2} \times b \times h$

$= \frac{1}{2} \cdot W \cdot 2W = W^2$ for ontime

for delay, $\frac{1}{2}(2W-D)(W-D)$

Revenue loss $= W^2 - \frac{1}{2}(2W-D)(W-D) = W^2 - \frac{1}{2}(2W^2 - 3WD + D^2)$

$\hookrightarrow RL = \frac{-D^2 + 3WD}{2} = \frac{D(3W-D)}{2}$

Revenue val in %: $\frac{D(3W-D)}{2W^2} \times 100\%$

C_{NRE} = Non recurring cost

C_{unit} = Cost of each unit

$$\text{Per unit cost} = \frac{C_{NRE} + N \times C_{unit}}{N}$$

21/01/2025 → in PC's RAM, ROM, I/O external

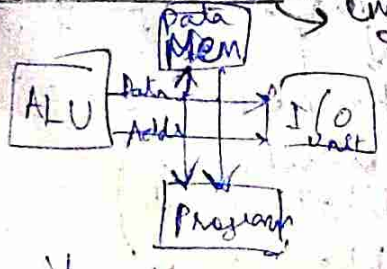
Microprocessor: components ALU, Registers, CU (control unit)

wide range of apps: flexible

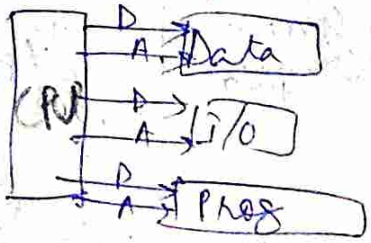
Microcomputer: used in PC's

→ mem, I/O, CPU on single chip, dedicated

Microcontroller: everything compressed on one chip → embedded sys



Von Neuman



Harvard

Harvard: separate buses

Von Neuman: common bus.

L1 Cache : Separate memory for data and ins
L2, L3 : Place anywhere.

Embedded vs PC

Program in memory, no separate secondary

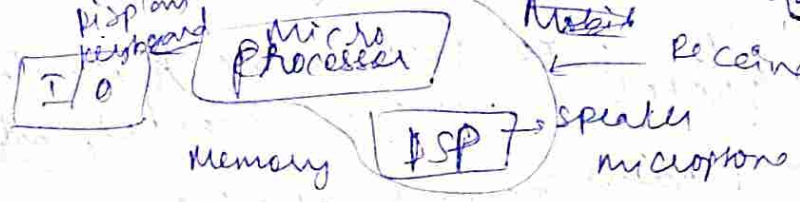
Programs in disk, loaded to primary memory

only some embedded have real time OS. (Not all E have OS) OS executes prog

Specification → System → Subsystem → Processor level design

Analog part

Gate level design



Processor → General, System specific (like embedded), Application specific
(e.g. DSP, multimedia)

Microcontroller → Counter, Timers, DIL, AIL

Models to develop design the embedded system.

1. Big bang : we have to design everything, no specification is given.

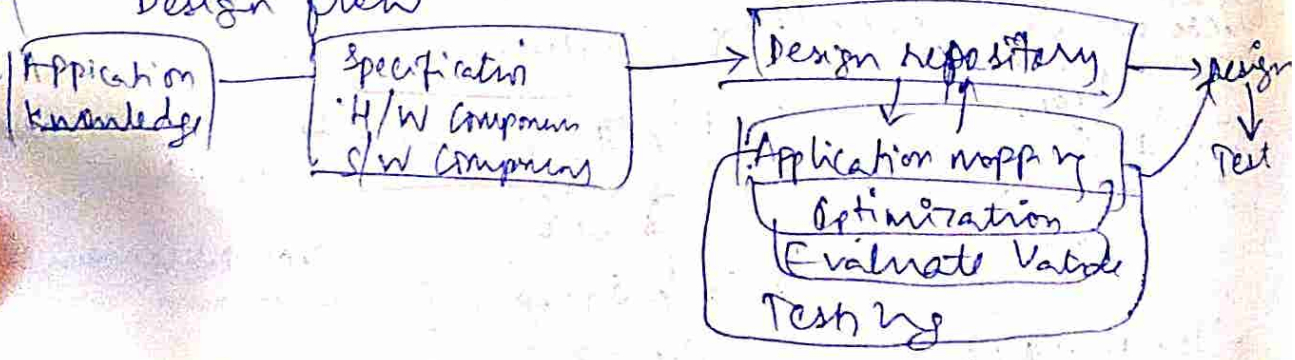
2. Code & fix : Some requirements are given

3. Waterfall : Output of one stage is given as I/p to next stage, these are diff stages.

4. Spiral : Each stage has feedback loop.

28/01/2025

Design flow



Specification model → Simulation / Architecture Exploration / Eval Validate

Model → Simulation / Environment system / Eval validate

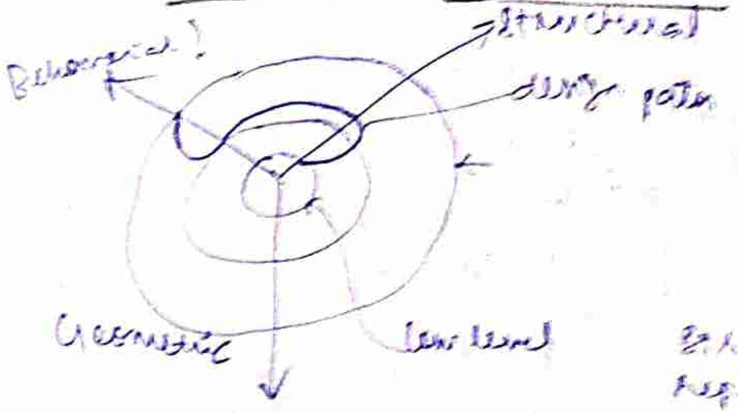
Communication models → Simulation / H/w Simulation / S/w Sim / Eval Validate

Performance → Implementation

Overall Loop

V-Model:

Requirement Analysis
↳ Sys Arch
Sys design
S/w Arch → S/w design → Unit test
Integration test
Sys Integration
Acceptance test



Gajski's Y chart

Behavioural : Combinational arrangement

Structural : actual process representation of behavioural part.

Processors

GPP (general purpose), ASIP (application specific), SPP (single purpose)

RISC High end — Programmable for specific domain, DSP, more options than GPP, some level of specialization and programmability

Low end — hardware system only, low level

Types of PU → Hardware

Control Path:

Central Logic, Instruction Register, Program Counter, Status registers, On chip Memory

Data Path:

Registers, ALU

I/O

Factors for Microprocessor

* Speed / power * Cost * Size

* Reduce time of a CC
* Spec benchmark
* Power

* Reduce no. of cycles for an ins.

to evaluate performance of system.)

Dynamic Power: (when system operating) { Differential voltage, dynamic power, frequency scaling } = power $\propto \alpha C V^2 f$ = energy $\propto C V^2$

Static Power: leakage power, when system is idle (not operating)

ASIP: uses microcontrollers (also used in embedded systems)
* Some add ons like timers, clks, converted A/D (makes initial)
* Has to read data from sensors
* Event-driven.
* Has program mem, data mem.
* Specialised instruction set.
* DSP Processors: multiply & add, vector instructions

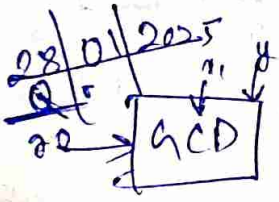
why not general purpose processors: $\sum_{i=1} x_i * y_i$ will take a CC each for

But DSP does vector add^n parallelly.

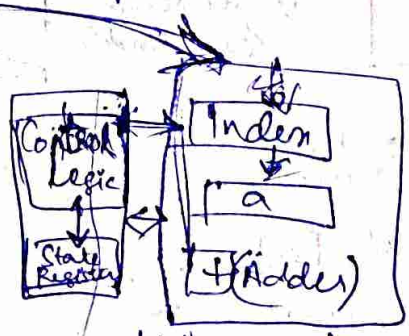
Design single purpose processor to execute this

```
a = 0
for i = 1 to j
    a = a + N[i]
```

Control path and Data path




```
0 vector x, y
1 while (1) {
2 while (go_i);
3 x = x_i;
4 y = y_i;
5 while (x != y) {
6 if (x < y)
7 y = y - x
8 else
8 x = x - y
9 }
9 do = x
}
```

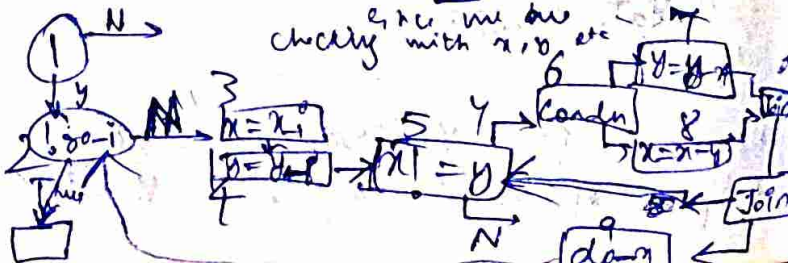


This is hardware

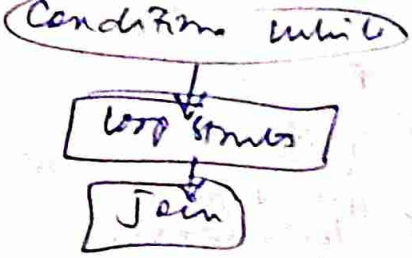
single purpose processor

Control path for GCD:

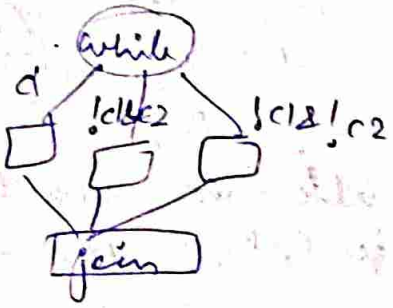
For Q1 the FSMD is



while (cond) {
    loop stmts;
}



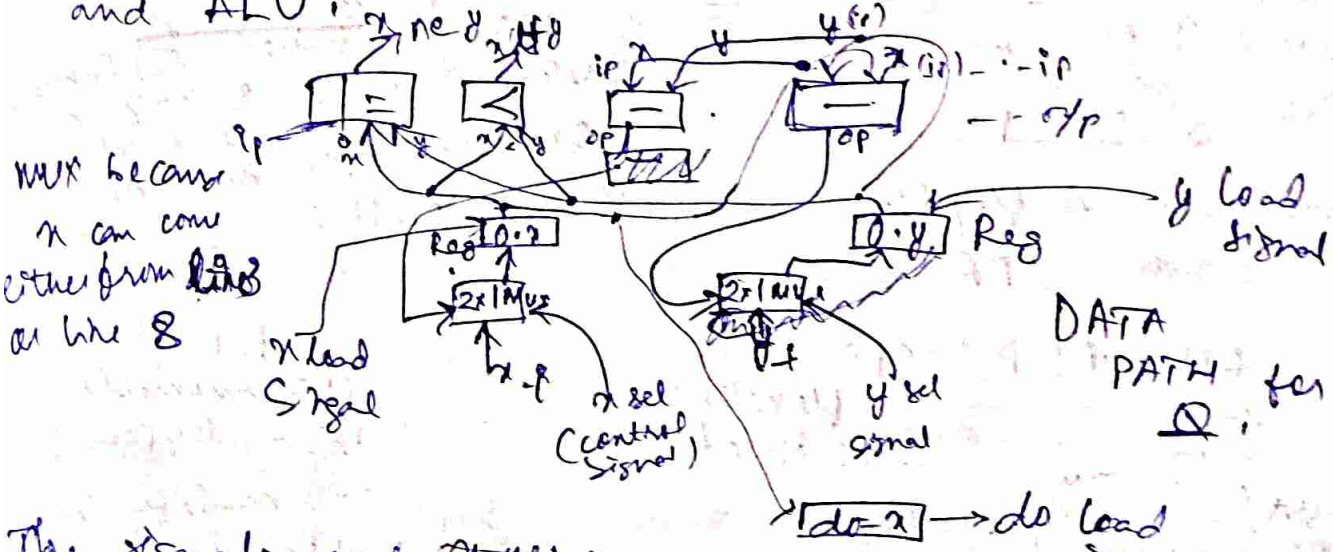
while () {
if c1 {
} else c2 {
}
}



FSMD: Finite State Machine with Data

Data path: a + b * c

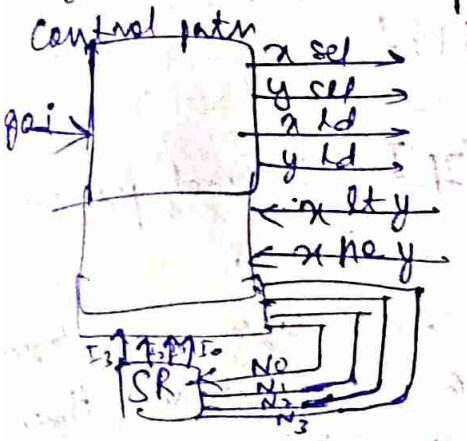
contains registers and ALU.



mux because x can come either from line or like 8

The signals and x neq y ... go to control path.

go from as x load.

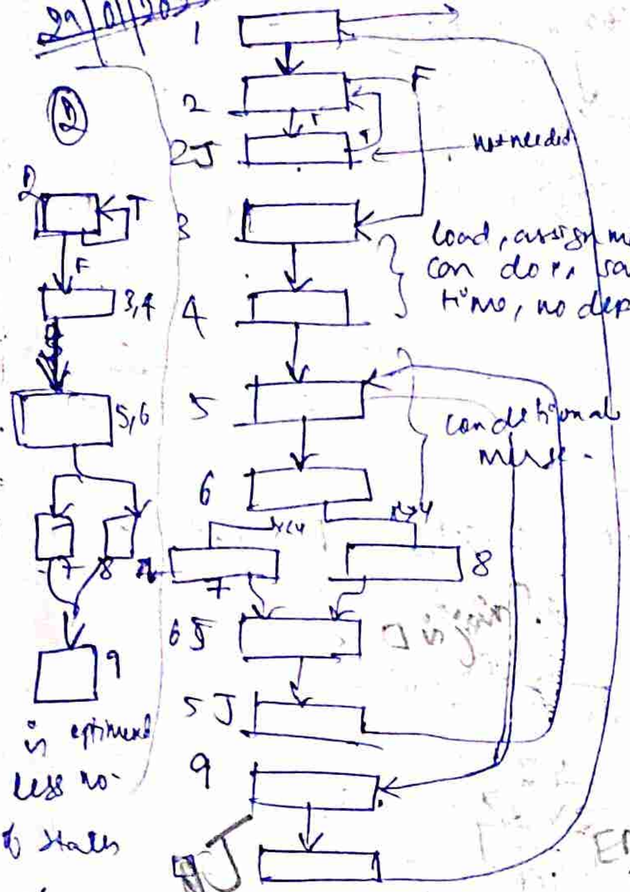


the SR is state register which takes in the current state and gives next state.

In the FSMD, 1 is 0000, 2 is 0001 like - these for each state, assign 4 bit nums, and then all are the transitions

I/P $N_3N_2N_1N_0$	$x<y$	$x!<y$	go	$\hat{I}_3\hat{I}_2\hat{I}_1\hat{I}_0$
0 0 0 0	X	X	X	0 0 0 1
0 0 0 1				

29/01/2025



↳ 12 states : 4 flipflops
6 states : 3 FF

we can optimise the lines 5 to 9

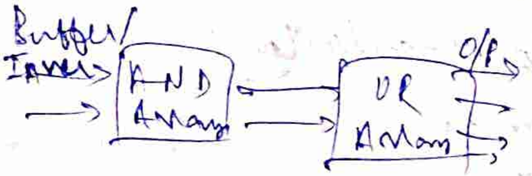
as r = x % y
x = y ;
y = r ; } ①

Earlier - there were 9 iter for (42, 8) now there are just 3

Optimise
① change code.
② Modify FSMD, reduce no of states
③ Data Path:
Have a single [-] subtractor instead of 2 : as x - y or y - x only.
Once will happen at a time.
OR use ALU.

GPP ASIP SPP
⟵ flexibility
⟶ speed

PLD: programmable logic devices!
simple can modify logic
SPLD → PROM, PLA, PAL, GAL
complex CPLD

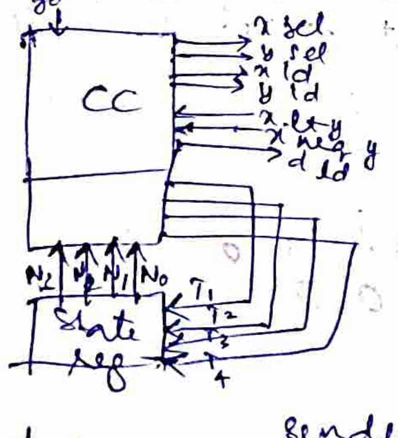


	AND	OR
PROM	F	P
PLA	P	P
PAL	P	F
GAL	P	F

F : fixed
P : programmable

FPGA = IC. programmability generic
(IP) Intellectual property

30/01/2025

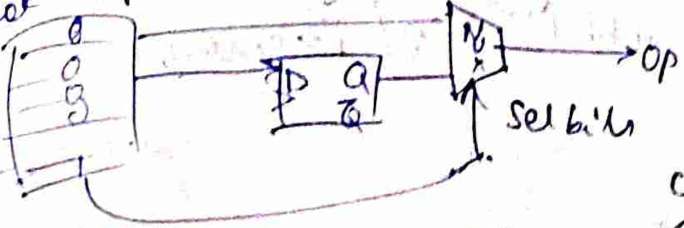


Control logic and datapath for ... down timer
verilog.

Q. sender and rcvr, sender sends 2 4 bit numbers, how to design control path and data path?

31/01/2025 CLB: configurable logic blocks contained in FPGA's. performs logical ops & can be reprogrammed

look up table — building block of FPGA. Configurable / Programmable



{ * Lookup table
{ * Mux.
→ users can implement custom circuits by loading configurable data onto the FPGA.

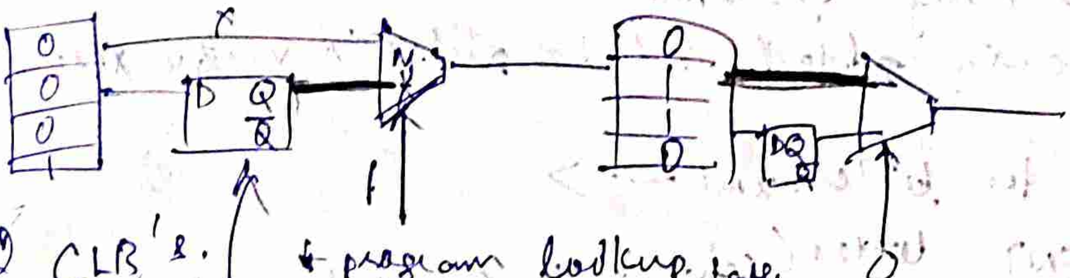
LUT is programmable.

Interconnection networks:

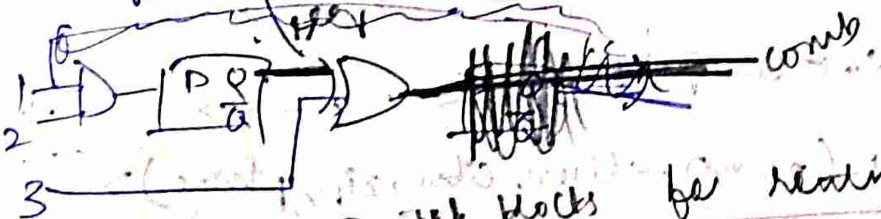
* Antifuse: non programmable. separate voltage ? …
* SRAM: programmable

Connection b/w CLB: set the bit at req pos.

A-C A-B A-D …
1 0 0 0 0 0
means A & C has connection

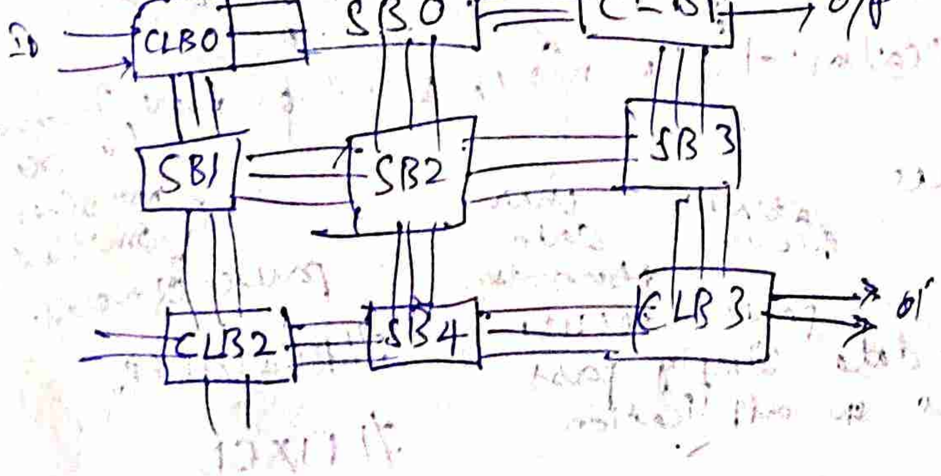


2 CLB's. Is this for this
* program lookup table
* program selection bits based on this



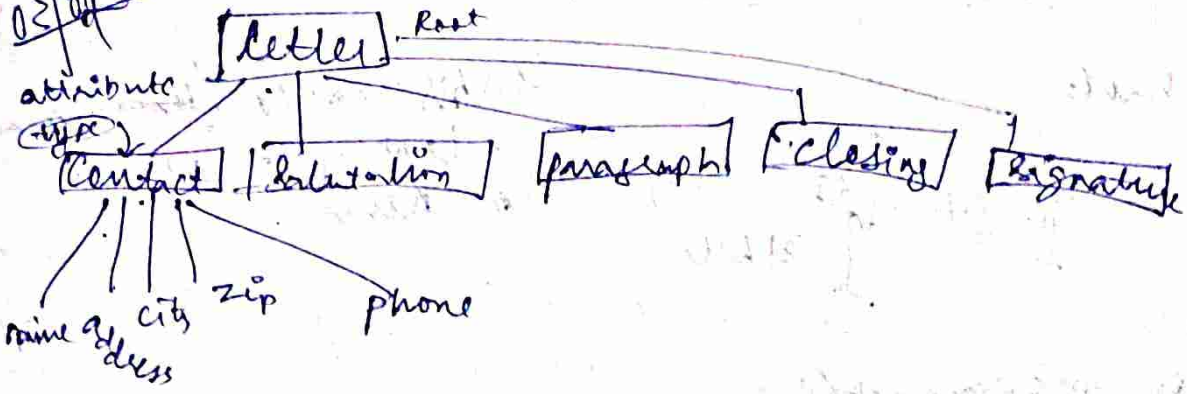
Switch blocks

for reading: Konstantinos Tatas, Intro to FPGA



FPGA is an IC.

02/09/2025 letter.xml XML Tree



XML Validation:

- Validate XML document 1. DTD
  2. XML Schemas

DTD:

- enables the XML parser to validate the given XML document
- verifies the syntax of XML doc
- Uses EBNF grammar
- File extension: .dtd rules to verify XML syntax

letter.dtd

<!-- DTD for letter.xml -->

<!ELEMENT letter (contact+, salutation, paragraph+, closing, signature)>

<!-- define individual child elements -->

<!ELEMENT Contact (name, address, city, zip, phone)>

<!ATTLIST Contact type CDATA #IMPLIED>

Occurance Indicator: + : 1 or more, * : 0 or more, ? : once or not given (omitted)

- + : 1 or more
- * : 0 or more
- ? : once or not given (omitted)

Contact: element name

type: attribute of Contact

CDATA: char data

#IMPLIED: parser ignores.

#REQUIRED

#FIXED

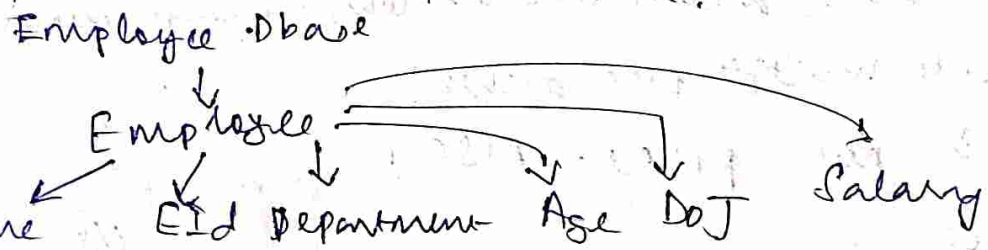
Process the data: parser doesn't process the data simply pass data to the application

<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT salutation (#PCDATA)>
<!. . . paragraph . . .>
. . . closing . . .
. . . signature . . .

parsed character data : ones ~~font~~, doesn't contain markup

To validate the XML document
http://www.w3schools.com/dom/dom_validate.asp
http://www.w3schools.com/xml/xml_validator.asp.

Consider the following XML Tree:



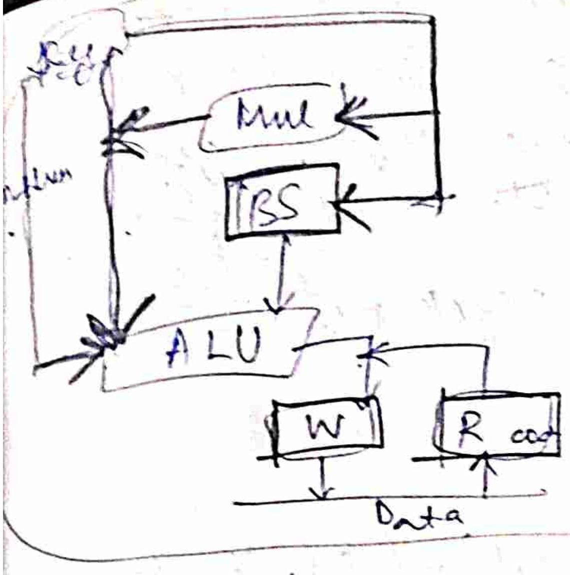
Create a XML document for this XML tree with 5 no. of employees (emp.xml)
Also write an external DTD (emp.dtd) to validate emp.xml.

05/02/2025 ARM : Advanced RISC Machine
32 bit * 70% of 32 bits ?
* low power consumption
* good performance.

Design :
* Small
* Low power
* High code density (code stored takes less memory after optimizing).
* Interface with slow & low cost memory system
* Reduced die size.

ARM-7 : 3 stage pipeline F D E
ARM9 : 5 stages F D E M W
ARM11 : 6 stages F P D E M W
Design features (RISC)
Instruction : Reduced set / Single cycle / Fixed length
Pipeline : Decode in one stage, converts inst in microinstruction.
Reg : General Purpose Regs. CISC takes several stages.
Load & store architecture. Why? because memory is.
CISC needs multiple RISC archis!
ARM is not pure RISC.

ARM : 2
* Variable cycle, multiple registers load/store
* Inline barrel shifter : a h/w unit to perform shift and some other op in same cycle.
shift and operate.
* Thumb : 16 bit
* Conditional execution.
* MOVEQ Ri, #0 add if zero is —?
It will mov if eq else it just doesn't.
like in one inst itself checking (EQ) and doing else (MOV)
* SDP Enhanced ins. Scalable data processing?
* No separate I/O inst.



Throughput = no. of ins per sec

$$\frac{n}{(n-1+k)\tau}$$

Efficiency : $\frac{\text{speedup}}{\text{theoretical max speedup}}$ (k stages is expected to give speedup of k).

$$= \frac{\frac{nk}{k-1+n}}{k} = \frac{n}{n+k-1}$$

Latches : for buffer b/w stages b/m res

~~Max stage~~ delay : τ: clk period, t_i: time delay in stage S_i, d: latch delay. Max stage delay: max τ_i

Total time to process N data sets : NPL : $N\tau$

PL : $(N+k-1)\,T/k$

For ~~NPL~~ ~~speed up~~ non-pipelined : $T_k = [(k-1)+N]\tau$

T = sum of times of all stages

For Pipelined : $S_k = \frac{T_1}{T_k} = \frac{Nk\tau}{((k-1)+N)\tau}$

τ = time for one stage.

ARM-7 : | Fetch | Decode | Execute (Reg Rd) (Shift) (ALU) (Reg W) |

parallely?

ARM-9 : | F | (D) (Reg Rd) | Execute Shift ALU | Mem Access | Write reg W |

$(N-1)\tau$

$1 \cdot k\tau$

Processor Mode : 7 Modes.

* User Mode.
* High priority mode (interrupt high prio)
* Low priority mode
* Supervised mode
* Abort mode
* Undefined (trying to exec an undefined instruction)
* System (privileged tasks).

Register : 37 registers. 30 : General Purpose (GP), 1 : PC,

1 : CPSR , 5 : SPSR

Current/Save prog status reg
C S P S R

Save prog while an interrupt comes.

No Stack ptr, no push, pop ops.

Interrupt : move data from CPSR → SPSR
finish subroutine, then move data SPSR → CPSR

No stack ptr in ARM, (but we can create using any reg)
only 16/37 regs are visible (Usable) $R_0 - R_{12}$ GP

R_{13} : to create our own stack ptr.
R_{14} : link reg.
R_{15} : PC and CPSR.

Data types : (signed, unsigned) × (8b, 16b, 32 bit)
Can access only lower order bits.
Only for data ops, not for arithmetic ops.
Arithmetic : 32 bit only.

ARM mode : 32 bit (4B), word accessible last 2 bits 0 always.
Thumb mode : 16 bit (2B), half word accessible.

ARM mode :
0000
0100
1000
1100

Thumb mode :
0000
0010
0100
0110
1000
1010
1100
1110

CPSR :

N	Z	C	V	Q		IRQ	FIR	T	4	3	2	1	0

Load interrupt req

Interrupt req

Mode

T - 0 : ARM
1 : Thumb

0 - disable
1 - enable.

11111 : Sys
11011 : undef
10000 : USR
10001 : FIR
10010 : IRQ
10011 : super
10111 : abort

Interrupt : load all data to SPSR.
Make FIR as 1?
Make IRQ as 1? high priority

Link reg: to store return addr.

27/01/2025 Types of ins? in ARM

- Data Processing — only registers, arithmetic
- Data Transfer — regs and mem
- Control flow — PC updated.

Data processing: adc: add w/ carry.

adc r0, r1, r2   r0 = r1+r2+C

sbc r0, r1, r2 : r0 = r1-r2+C-1

rsb r0, r1, r2 $r_0 = r_2 - r_1$

rsc r0, r1, r2 $r_0 = r_2 - r_1$

add, sub are normal.

Bitwise ops: and, or, xor, BIC (bitwise clear)

BIC {S} {cond} Rd, Rn, operand   clear the lower some bits of Rn
— to set conditions

Reg-Reg: MOV r0, r2   Rd ← Rn AND ¬operand

MVN r0, r2 = r0 = not r2
negation

Comparison: CMP r1, r2   sets r1-r2

CMN r1, r2   (r1+r2 ???) compare negative

TST r1, r2   r1 and r2

TEQ r1, r2   r1 ⊕ r2

Immediate:

AND r1, r2, #2   SUB r1, r2, #2   AND r6, r4, #&0F
(ADD?)   HEX

lower ins, 4th is copied to R6.

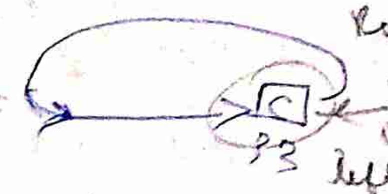
Shift:

ADD r1, r2, r3, LSL #3

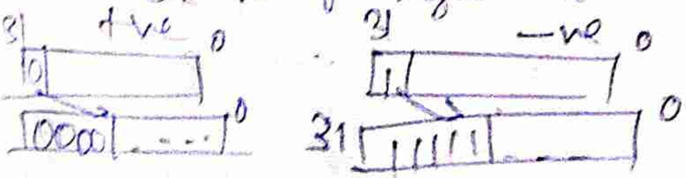
r1 = r2 + (r3 << 3).

r1 = r2 - 2

ROR 
Rotate right

RRX = 
Rotate with carry — This flag is used as leftmost bit for the result

Arithmetic shift right w/ +ve number or -ve number.



Multiplication

MUL r1, r2, r3 , r1 = r2 * 3

MLA r1, r2, r3, r4 r1 = r2 × r3 + r4

Multiply and Augment ?

address relative

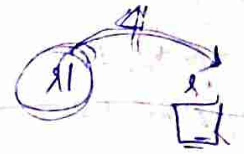
ADRL R1, Table : table's addr is calc and loaded into R1.

LDR r0, [r1] BTR : r0, [r1]

R0 ← r1 at this addr

LDR r0, [r1, #4] → like offset ig

r0 ← *(r1 + 4)



Auto Indexing :

LDR r0, [r1, #4]! r0 ← *(r1+4) then r1 = r1 + 4

32 b

r0 ← r1+4 loaded into it.

Post Indexing :

load LDR r0, [r1], #4 first load r0 ← r1 then add r1 ← r1+4

LDRB 8 bits ✓

LDRSH 16 bits ✓

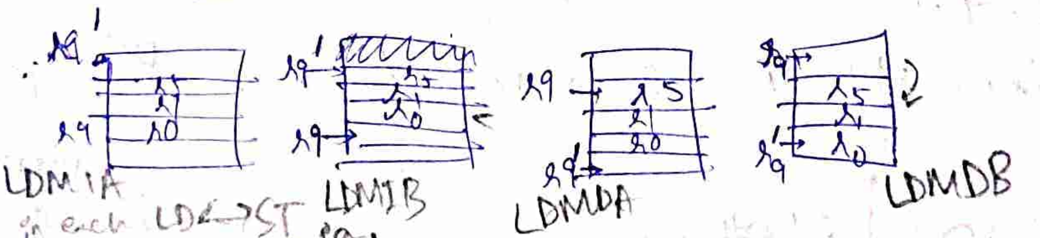
LDMIA r1, {r3, r5, r6} Load mem inc r after

r1, {r2 – r5}

r3 = [r1], r5 = [r1+4], r6 = [r1+8]

LDMIB before

diff b/w in ins explain



in each LD→ST eg.

diag find what ins

Q. Copy block of 128B write piece of code

use 8 registers, in one time we can copy 32B ? 8×4B = 32B.

we need to exec $\frac{128}{32}$ = 4 time 4B (32 B) regs

load from, store into mem. Comp if it is the end,

11/02/2025 Control Flow Instr:

+ (unconditional) + (condition) (Branch to Link)
+ (Conditional exec)

Branch to link : to move to the subroutine while saving the current PC (address of next ins) in the Link reg
CPSR to SPSR

BL SUB1
SUB1 MOV PC, R14. ← After finishing subroutine, R14 is link reg, has X addr (next ins) to PC ← R14 at end

Conditional Execution.

if (r_1 != 5) $r_2 = r_2 + 5 - r_3$.

beq r_1, 5, outside → CMP r_1, #5
addi r_2, r_2, #5 (beq loop outside) this can be skipped in ARM.
sub r_2, r_2, r_3
outside:

In ARM,
CMP, r_1, #5
ADDNE, r_2, r_2, #5
SUBNE r_2, r_2, r_3

ARM Branches

BAL		Branch
BEQ	BNE	
BPL	BMI	Result +ve or -ve
BCC	BCS	Carry set or clear
BVC	BVS	Overflow
BGT	BGE	>, >=, <, <=
BLT	BLE	

FPGAs
Basic features,
Micro proc comp, comp,
Types,
Real time
Design methodology
models
(...)
Design controller
datapath
state table
ARM vs RISC
Ins set ISA
Pipeline
speedup, efficiency,
cost, ...

1. if((r_1 == r_3) && (r_5 == r_0)) $r_7 = r_7 + 10$
cmp, neq go somewhere   cmp, neq go somewhere.

Conditional exec:
CS carry set, EQ, VS overflow? , GT, GE,
PL +ve, HI higher than, HS : higher or same,
CC : carry clear, NE, VC overflow clear?,
LT, NE, MI, LO, LS.

BL : Branch w/ link,
call subroutine/func
and remember where to return.

2. if(r2 & r3 != 0) $r_3 = r_3 + r_4 / r_5$

3. Expl. functionality of this:
BL SUB1
SUB1 STMFD r13 ! (r_1 - r_5, r_{14}) ← show multiple
BL SUB2 full descending
LDMFD r13! (r1-r5, PC) LD or ST here??
SUB2 MOV PC, R14

w/o ~~ARM~~ MIPS

1. cmp $1, $3
   branch neq
   cmp $5, $0
   branch neq
   add

ARM only 3 lines ?

cmp R1, R3
~~Cmpeq~~ R5, R0 ?
addeq R7, R7, #10 ?

2.

3. nested call. subroutine calls another subroutine

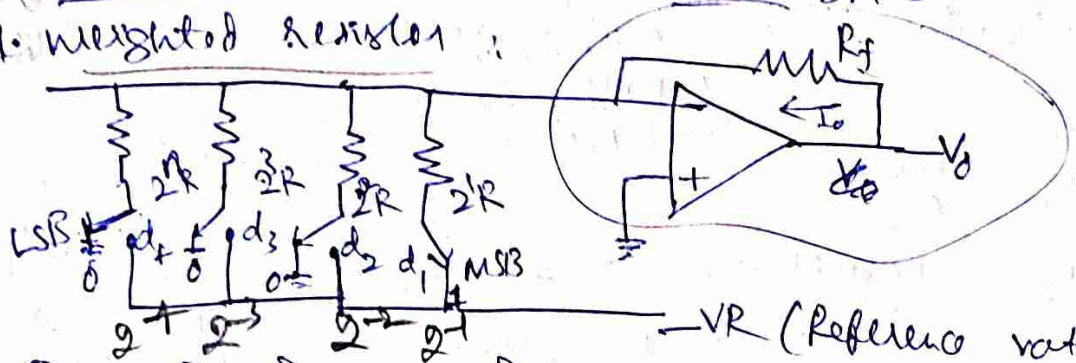
> cmp R2, #0
cmpne R3, #0
subne R8, R4, R5
addne R2, R3, R8

conv pdn
data path
state table
C → MIPS → ARM
Pipeline
Speedup, CPI
efficiency
cost norm

22/02/2025 Types of DAC (digital to analog converter)

Vector resistor? DAC, ladder DAC

1. Weighted resistor :



$I_0 = I_1 + I_2 + \cdots + I_n$

$I_0 = \frac{V_R}{2^1 R} d_1 + \frac{V_R}{2^2 R} d_2 + \frac{V_R}{2^3 R} d_3 + \frac{V_R}{2^4 R} d_4$ for 4-bit ckt & switches

$I_0 = \frac{V_R}{R}\left(\frac{d_1}{2^1} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \frac{d_4}{2^4}\right)$

o/p voltage: $V_0 = I_0 R_f = \frac{R_f V_R}{R}\left(\frac{d_1}{2^1} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \frac{d_4}{2^4}\right)$

If $R_f = R$ then

$= V_R (d_4 \times 2^{-1} + d_2 \times 2^{-2} + d_3 \times 2^{-3} + \ldots)$

Here $V_R = V_{FS}$

V_{FS} = full scale (eg. 0–8V means $V_{FS} = 8V$)

Eg. Take $d_4 d_3 d_2 d_1$ as 0010

then $V_0 = V_{FS}\left(0 + \frac{1}{2^2} + 0 + 0\right) = \frac{1}{4}$

If $V_{FS} = 8V$, $V_0 = \frac{8}{4} = 2$ (which is 0010_2)

presently: keep the no. of weighted resistors. If we find the number of bits.

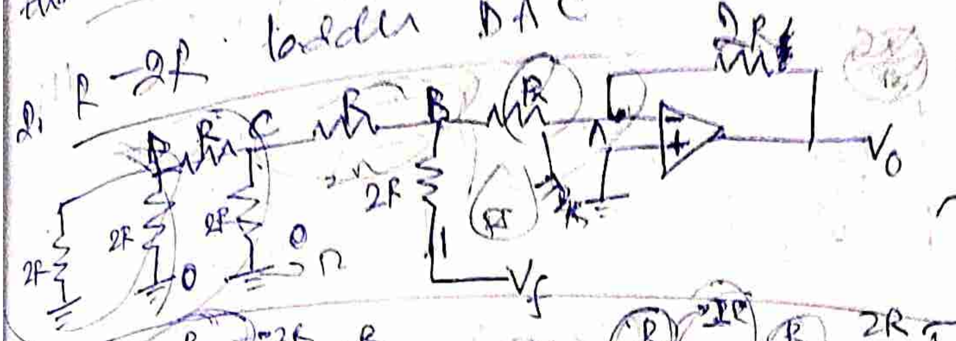
weighted resistor only used for 8 bit values, beyond that capacity it more work.

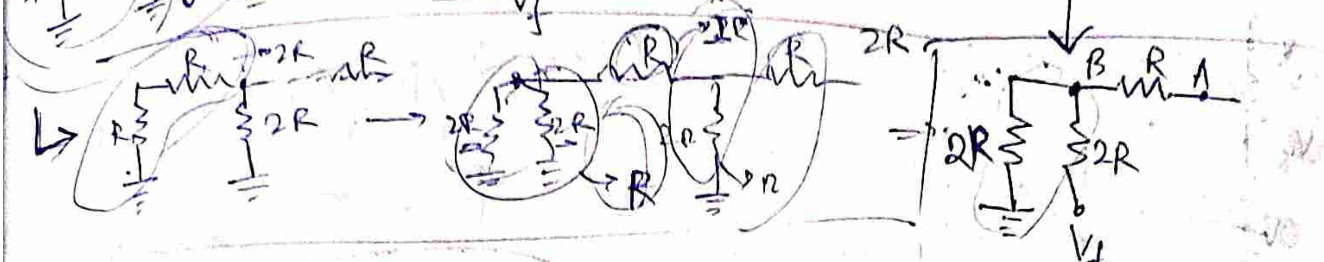
2. R-2R ladder DAC

For binary 100 (4)

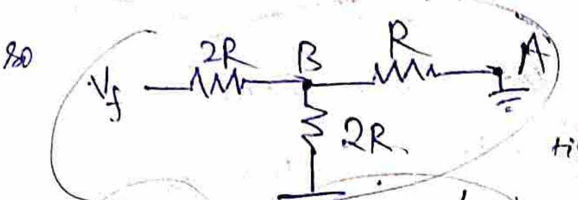
1 : V_f

0 : GND





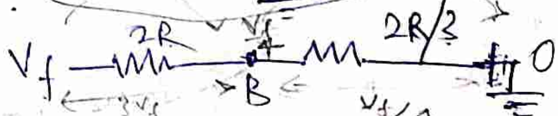
so



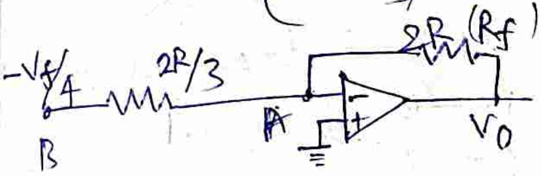
A is also connected to GND since that

till this ok.

so 2R, R in ||el



So $V_B = \frac{-V_f \times \frac{2}{3}}{\left(2 + \frac{2}{3}\right)} = \frac{-V_f \times \frac{2}{3}}{\frac{8}{3}} = -\frac{V_f}{4}$ (why $-$?)



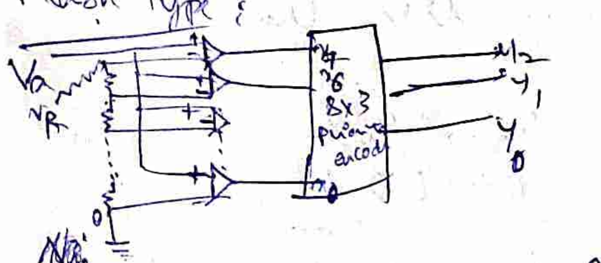
$\frac{2R}{2R/3}$ $\quad V_0 = \frac{R_f}{R_1} \times V_{in}$

so $V_0 = \frac{2R}{2R} \times \frac{-V_f}{4} = \frac{V_f}{2} = \frac{8}{2} = 4$???...

For more bits (like 8 (1000) ⇒ 4 bit so add one more resistor).

ANALOG to Digital

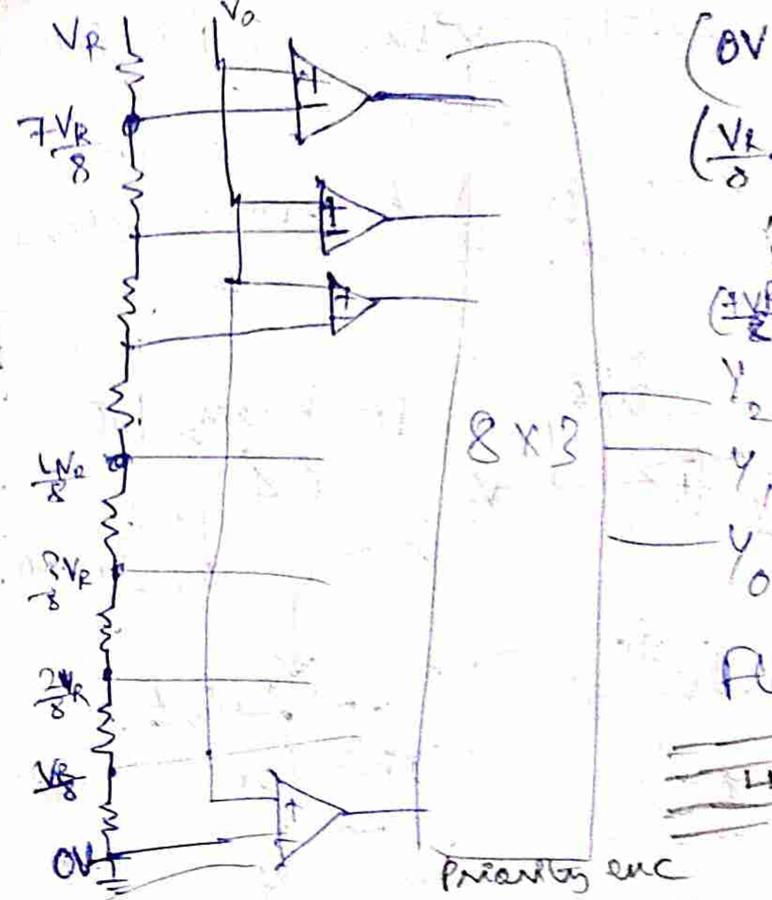
Flash Type :



V_a = analog voltage,

no. of comparators needed is $2^n - 1$, those ▷ things.

Here in x_0, we connected ▷ so there is 8 ▷, but it can be directly connected to GND also, so only $2^n - 1 = 2^3 - 1 = 7$ is required. (no need for x_0)



$\left(0V - \frac{V_R}{8}\right)$ $x_7 \ldots x_1\, x_0$
0 ... 0 0 1

$\left(\frac{V_R}{8} - \frac{2V_R}{8}\right)$ ... 0 1 1

⋮

$\left(\frac{7V_R}{8} - \frac{8V_R}{8}\right)$ 1 1 1 1 1 1 1 1 111

Flash Type ADC

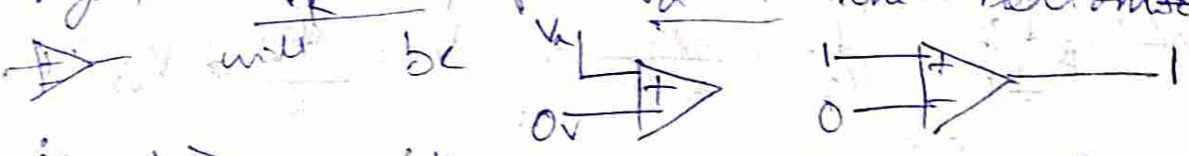
0 0 0 1	0 0
0 0 1 X	0 1
0 1 X X	1 0
1 X X X	1

No. of comparators: $2^n - 1$ are required

→ Here we have 8 but 7 also OK we can connect this last directly to GND

V at a point is $\frac{\text{total voltage} \times \text{corresponding resistance (from GND?)}}{\text{total resistance}}$

Eg. take $V_R = 8$, for $V_a = 1$ the bottom ~~comp~~ comparator will be

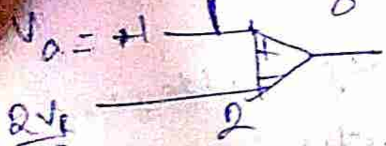


if $+ > -$ then o/p is 1 else 0.

~~Now for~~ for other comparators, like the second from bottom

0, as $1 \not< 1$

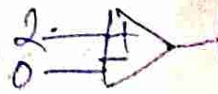
$\frac{V_R}{8} = 1$



0 $1 \not< 2$ ... like that so

it will be $x_7 \ldots x_2 x_1 x_0$
for $\left(0V - \frac{V_R}{8}\right)$, 0 ... 0 0 1

like if we take $V_a = 2$, it will be

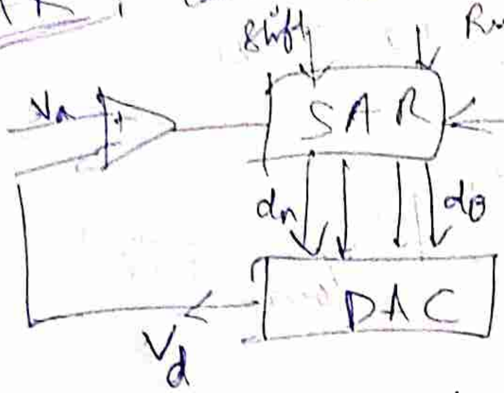


and

0 $2 \not< 2$

we that so ($\frac{}{8} - \frac{20_n}{8}$) is 00..011
→ is

SAR: (Some other types of ADC)).



	o/p of comp
$V_a > V_d$	1
$V_a < V_d$	0
$V_a = V_d$	Previous Value

Eg. take $n=3$, $V_a = 13$

d_3	d_2	d_1	d_0	V_d	comp o/p	
0	0	0	0	0	1	(since $V_a > V_d$) 13>0
✓ 1 (try setting)	0	0	0	8	1	13>8
1	1 (try setting)	0	0	12	1	13>12
1	1	1 (try setting)	0	14	0 → setting this is bad so make 0, set next bit	13<14
1	1	0	1	13	0	13=13 so prev value

whenever comp o/p is 0, rightmost 1 becomes 0 and in next (msb) 0 becomes 1? of LSB?

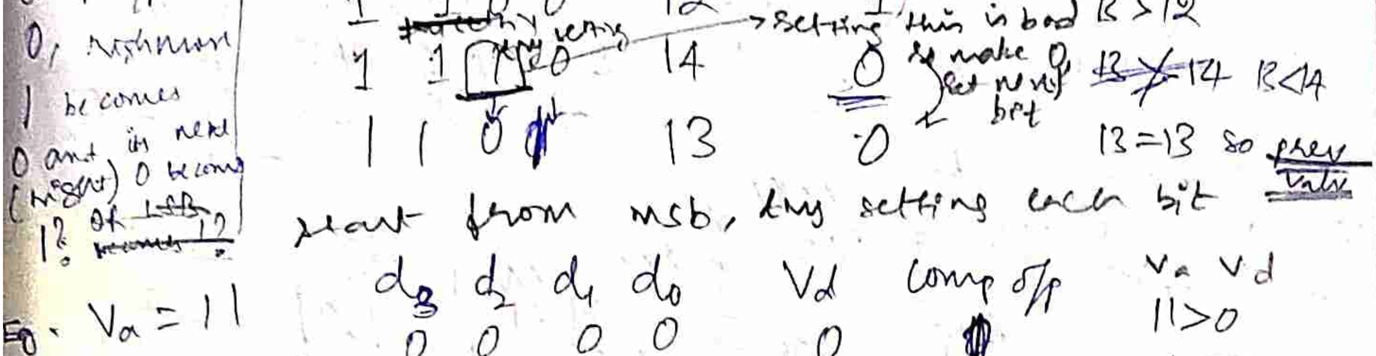
start from msb, try setting each bit

Eg. $V_a = 11$

d_3	d_2	d_1	d_0	V_d	comp o/p	V_a V_d
0	0	0	0	0	1	11>0
1	0	0	0	8	1	11>8
1	1	0	0	12	0	11<12
1	0	1	0	10	1	11>10
1	0	1	1			

PWM: modulate the width of a digital signal's pulses

25% of the time ON
$V_{ON} = 50\%$



$= \frac{t_{ON} \times 100}{T}$

Average ~~time~~ etc (how ~~many~~ much times it is ON)

$$\frac{1}{T}\int f(t)\,dt = \text{avg}$$

$$= t_{ON} V_H + (1 - t_{OFF}) \cdot V_L \qquad V_L = 0 \text{ usually}$$

so

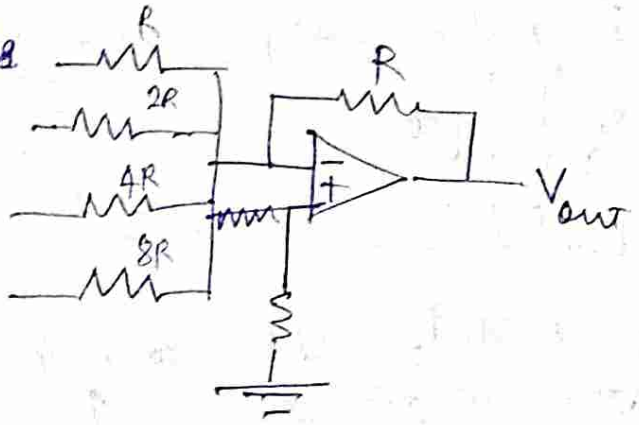
$$= t_{ON} \cdot V_H$$

power efficient, motor control, speed and torq. control, adjust LED brightness

14/03/2025

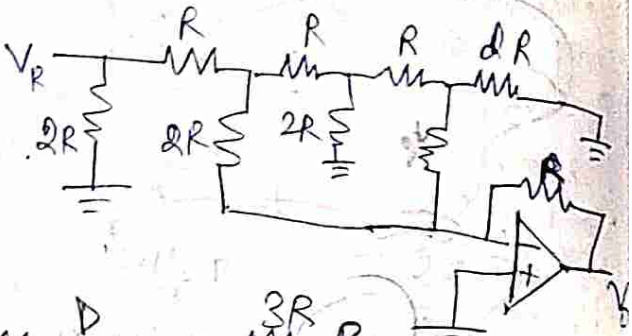
① i/p, o/p 0 5 V resp. OP AMP is ideal, all the resistance, 5V i/p have tolerance ±10%. Spec for the tolerance of DAC is

4 bit DAC is



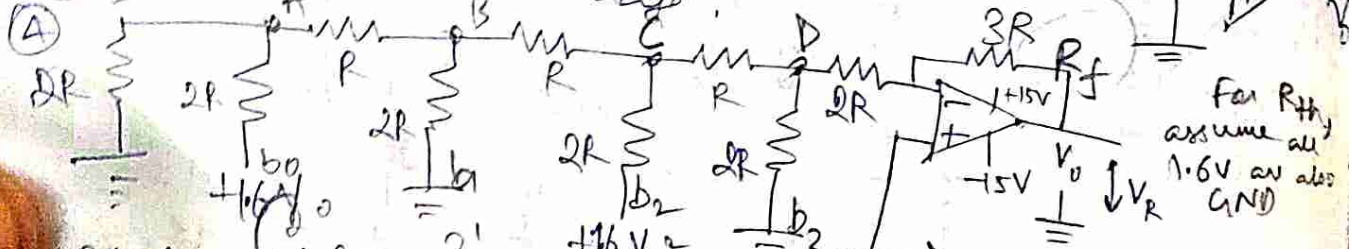
② $V_R = 10V$; $R = 10K\Omega$ V_R

The current is ___?



③ same as 2, find voltage

④



Find 5V?

$V_{th} = 2^{-4}$ $\div 2^4 = 0.5V = V_{th}$

$V_0 = -\left(\frac{R_f}{R_{th}}\right) \times V_{th}$

⑤ Design 2-bit ~~cache~~ flash ADC using logic gates

11/03/2025 Scheduling Algorithms

Rate monotic scheduling RMS algorithms in embedded sys

- Static scheduling
- Higher priority to tasks with shorter period
- Hard-real-time system

Key features:

- Static priority assignment
- Preemptive scheduling
- Deterministic feasibility
- Optimal for fixed priority

Schedulability condition:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$

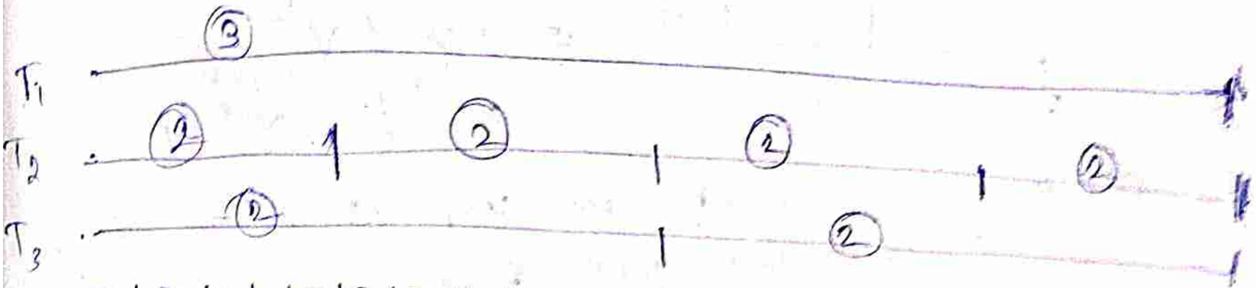
max CPU

where C_i = ... of task, t_i = period of task
U = total CPU utilization

eg:	Capacity (C)	period (P)
T_1	3	20
T_2	2	5
T_3	2	10

1. LCM(20, 5, 10) = 20 prio is highest for shortest period

priority $T_2 > T_3 > T_1$



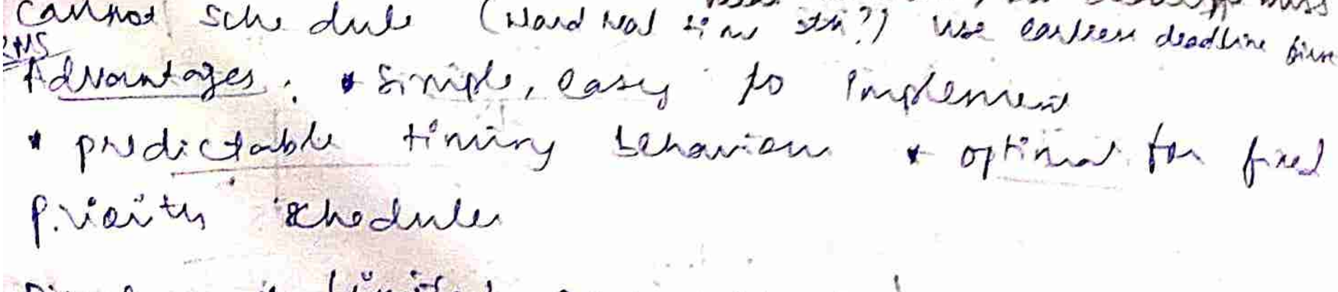
|T_2|T_2|T_3|T_3|T_1|T_2|T_2|T_1|T_1| //// |T_2|T_2|T_3|T_3| //// |
0 1 2 3 4 6 7 8 9 10 11 12 13 14 16 17 18 19 20

//// : CPU idle, cannot schedule anything.

eg: Task	Exec time or Capacity (C)	Arrival time or period
T_1	1	4 at 0, 4, 8 ...
T_2	2	5 at 0, 5, 10, ...
T_3	4	10 at 0, 10, 20 ... instance arrives

priority: $T_1 > T_2 > T_3$

LCM (4, 5, 10) = 20



needs this second, but deadline miss

cannot schedule (would not time still?) use earliest deadline first

RMS Advantages: * simple, easy to implement
* predictable timing behaviour * optimal for fixed priority scheduler

Disadv: * limited CPU utilization * strictly periodic tasks required * priority inversion is possible

Appl: Robotics, automation

12/4/2025 Power Aware Systems

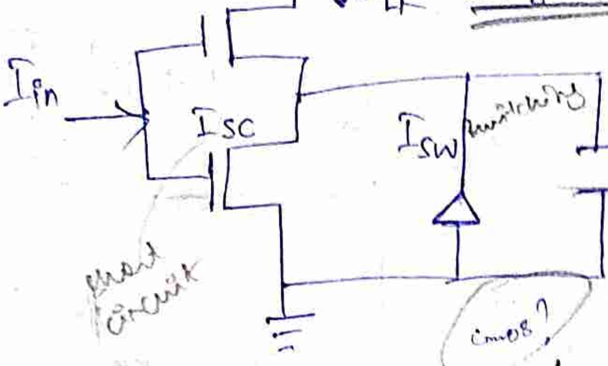
$E = \int P \, dt$

Static and dynamic power

quiescent state

I_{LK} leakage (static)

switching current



1060
1000

$E \propto C_L f V_{dd}^2$
load freq supply voltage

Gated clock: control/disable clk signals for some components of proc when not used
Power gating: disconnect power supply to parts of chip when not needed, saves both dynamic & leakage power
(reduces dynamic power consumption)

2.5V Slow

Standard 4V

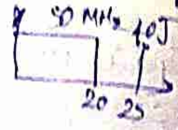
Fast 5V

Vdd	5V	4V	2.5V
Energy/cycle nJ	40	25	10
MHz	50	40	25
T ms	20	25	40

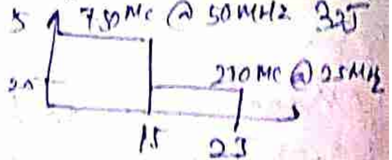
Reduce Static
* Power gating
* Body biasing
* Multi threshold CMOS

Dynamic
* Red supply freq voltage
* Red capacitance
* Clock gating
* DVFScaling

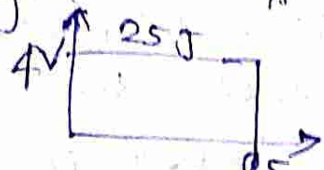
① 10^9 cycles with 5V, 50MHz, calculate energy consumption
$= 10^9 \times 40 \times 10^{-9} = 40$ J



② 750 Mc @ 50 MHz, 250 Mc @ 25 MHz
$750 \times 10^6 \times 40 \times 10^{-9} + 250 \times 10^6 \times 10 \times 10^{-9}$
= 32J ?



③ 10^9 cycles @ 40 MHz
$= 10^9 \times 25 \times 10^{-9} = 25$ J
= 25J



① same task op at full voltage, finish exec before deadline
② discrete voltage levels, try to reduce energy.
③ same voltage level, deadline is met, energy consumption is less.

energy

n: no. of tasks, EC_j: no of E cycles of task j
L: no. of voltages of target processor

V_i : the i^{th} voltage where $1 \le i \le L$;

f : clk freq for S_i ; V V_i : supply voltage

$d = d_L$ global deadline, C_L and α

X_{ij} = no of clk cycles task T_i executed at V_i ; SC_j : switching capacitance

LP sch:

$$E = \alpha C_L f V_{dd}^2$$

$$\sum_{i=1}^{L} \sum_{j=1}^{n} SC_j \times X_{ij} \times V_i^2$$

$$V_j = \sum_{i=1}^{L} X_{ij} = EC_j$$

$$\sum_{j=1}^{n} \sum_{i=1}^{} \frac{X_{ij}}{f_i} \le \text{deadline}$$

f_i → clock frequency at voltage level i

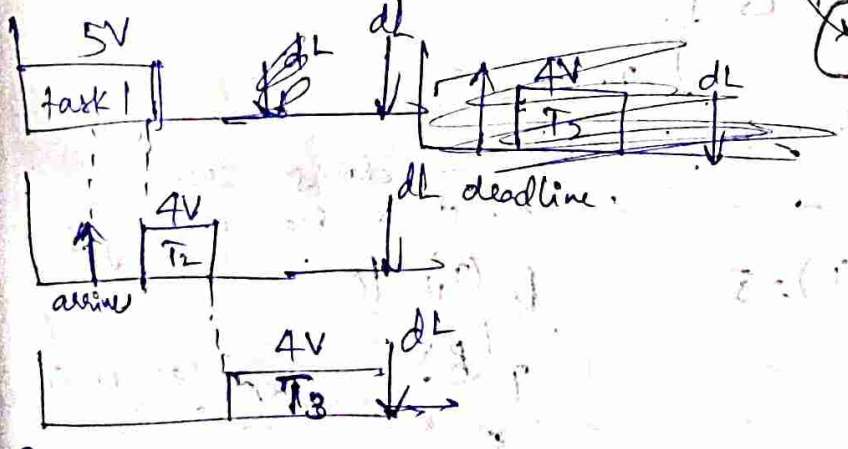
AT: Arrival time

1. Reducing energy consumption

2. Static voltage scheduling, dynamic voltage scheduling

AT of all known → SS & DV, DS & DV

AT if all not known



18/03/2025 compiler ?

20/02/2025 EDF : Earliest deadline first

$T_1(4,1)$, $T_2(5,2)$, $T_3(7,2)$, # sum period = deadline,

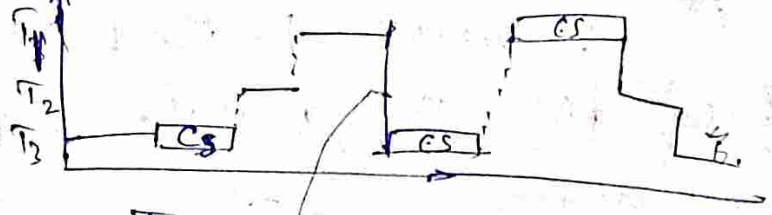
T_1 at $t=4$, T_1 deadline is 8.

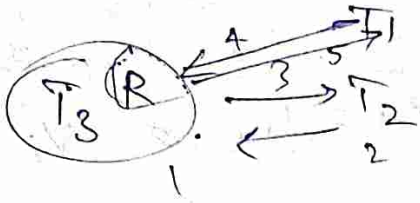
T_2 T_3 deadline is 7.

$\frac{1}{4} + \frac{2}{5} + \frac{2}{7} = \frac{35+56+40}{140} < 1$

condition : $\sum_{i=1}^{n} \frac{e_i}{p_i} \le 1$ necessary & sufficient

$T_1 > T_2 > T_3$



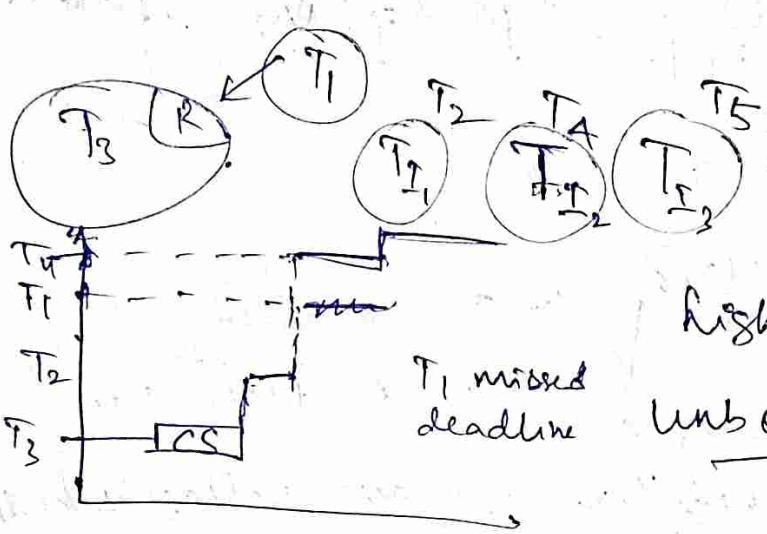


CS: critical section

T_2 : Intermediate task

T_2 preempts T_3.

Priority Inversion → $T_1 > T_3$ but here T_3 executes then T_1

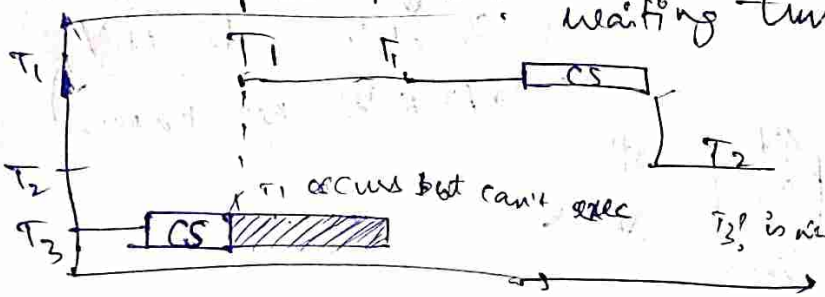


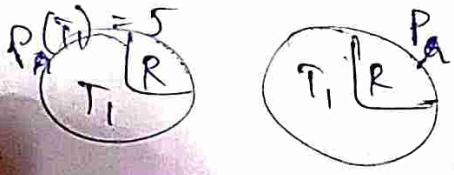
T_1 missed deadline

Intermediate task ~~priority~~ preempts higher priority !

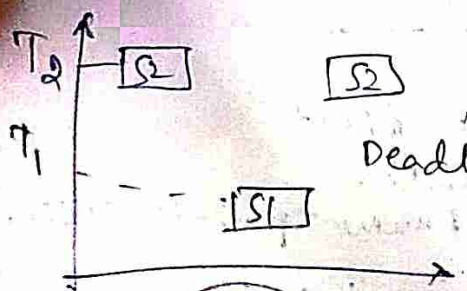
Unbounded priority Inversion

* Disable preemption : waiting time will be lesser.

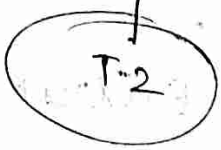




Hard Real Time Systems

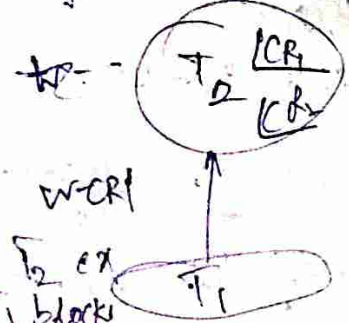


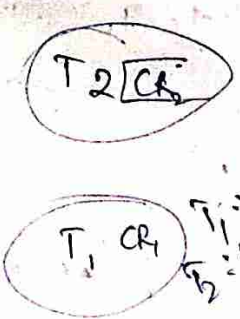
① Deadlock



Inherits priority $P_A(T_2) = 10$

① ② Priority disadv.





② Chain Blocking

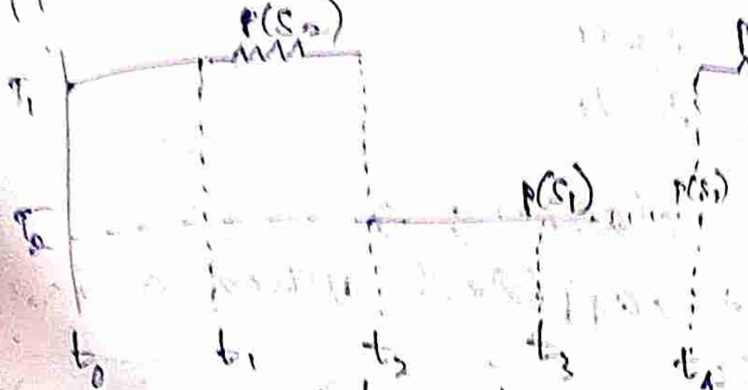
T_1 : Block
T_2 : execute

$T_1 (2,2)$ $T_2 (4,5)$ $T_3 (5,1)$ $T_4 (7,1)$ $T_5 (7,6)$

$T_1 (6,2)$ $T_2 (3,3)$, $T_3 (1,1)$ $T_4 (8,3)$ $T_5 (5,4)$ $T_6 (3,3)$

$T_4 (1,1)$, $(6,2)$, $(8,3)$

Priority Inheritance Protocol

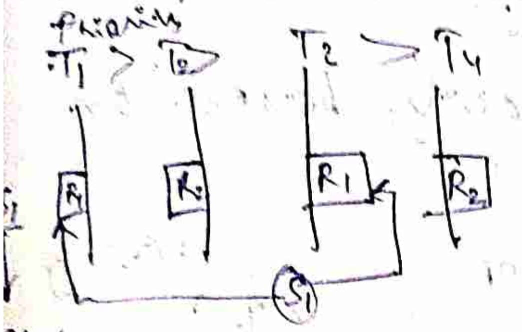


Priority assume $T_2 > T_1$
task holding CS is temporarily given two highest priorities to avoid preemption
two semaphores S_1, S_2

avoids unbounded priority inversion — Drawbacks: Deadlock, chain blocking

Priority Inheritance protocol: may get deadlock

PCP ? Priority Ceiling Protocol



ceiling = priority (T_1)

$C(S_2) = C(S_1) = \text{Priority}(T_2)$

- Each resource is assigned a ceiling priority determined by the highest priorities among all tasks which may acquire the resource
- An OS variable denoting highest ceiling of all locked semaphores is maintained.

Difference b/w PIP and PCP

PIP	PCP
greedy: Resource is promptly allocated if it is free when it is req	Not greedy, Resource may not be alloc. to req even if free
dynamic task priority assigned when blocking occurs	static fixed resc ceiling before running
easier to implement	deadlock, chain blocking
blocking time ∝ $\sum$ duration of lower priority task in critical section	tougher / complex
	blocking time ∝ largest critical section among low priority tasks

PCP : CSC = Current System Ceiling

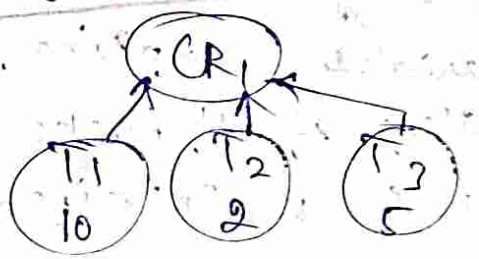
1. CSC = max ({ceil(CR_i) | CR_i is currently in use})
At system start CSC is initialized to 0.

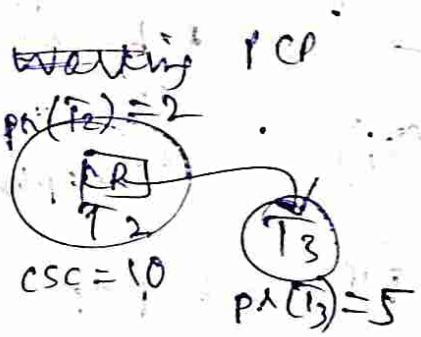
2. Resource sharing among tasks in PCP is regulated by 2 protocols/rules:
   (a) Resource request rule
   (b) Resource release rule

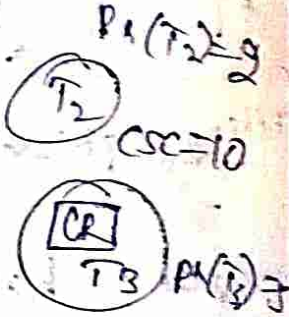
(a) Res req rule has 2 clauses: &
(i) Rsrc grant clause, applied when a task requests a resource
(ii) Inheritance clause: applied when task is made to wait for a resource

(b) Resource release rule: ~~bas~~
check the ceiling fn priority based on they release the resources

Eg.







PCP
+ prevents deadblocks, chainblocking, unbounded priority inversion

Eg1:

Task name	T	priority
A	50	10
B	500	9

Task A
lock(S1)
lock(S2)
:
Unlock(S1)
unlock(S2)

Task B
lock(S2)
lock(S1)
:
unlock(S2)
unlock(S1)

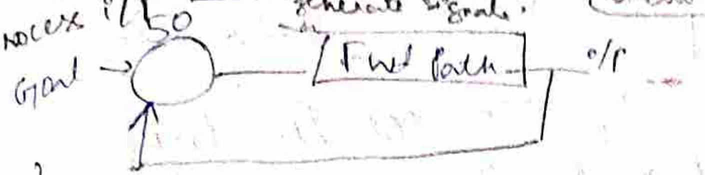
ceil(S1) = ceil(S2) = 10

J requests R in PCP
+ R busy ⇒ J blocked, req denied, Else:
+ priority of J > CSC ⇒ ✓ alloc else:
+ J holds some other rsrc with priority ceil = CSC
then req ✓ alloc else denied

07/07/2025 → processing unit / computational unit, processing data, executes ins, manage ops & IO

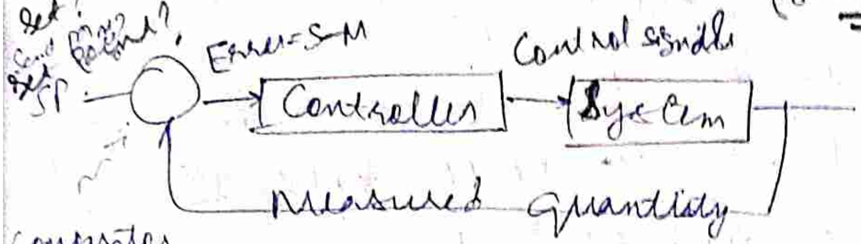
What is Controller - where (brain) does it exist in ES and what is its use? make decisions or respond to events, generate signals. (e.g. single chip, PCB, printed circuit board)

process i/p → Goal



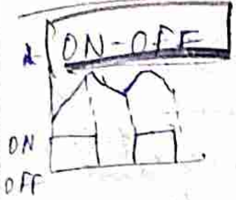
1. Sense the value
2. Corrective action
3. Maintain limits

Set? / Set point? / SP



Types of Controller:

* ON-OFF



overshoot: Not immediately on/off like step by step or sth.

* Proportional : overshoot and steady state error is there

o/p ∝ i/p

P-D Controller

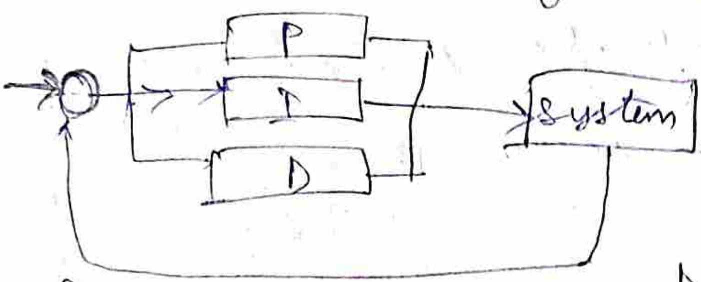
o/p ∝ i/p

o/p ∝ derivative

derivative sth, D predicts error trend and gives damping mechanism

reduced overshoot
steady state error is there

PID : Proportional Integral Derivative



CT-2
ADC, DAC
Scheduling
Power

Response time is less
No overshoot
No steady state error

D: future
I: past
↳ accumulates error and takes corrective measures.

Q RMA

	P	CT
T1	2	1
T2	3	1
T3	6	2

EDF

	P	CT
T1	4	2
T2	6	3
T3	10	4

Q. 2 tasks high priority and low, that share a resource. Low pri acquires the resource. H becomes ready. However, before L releases the resource, a medium pri task M becomes ready an preempt.

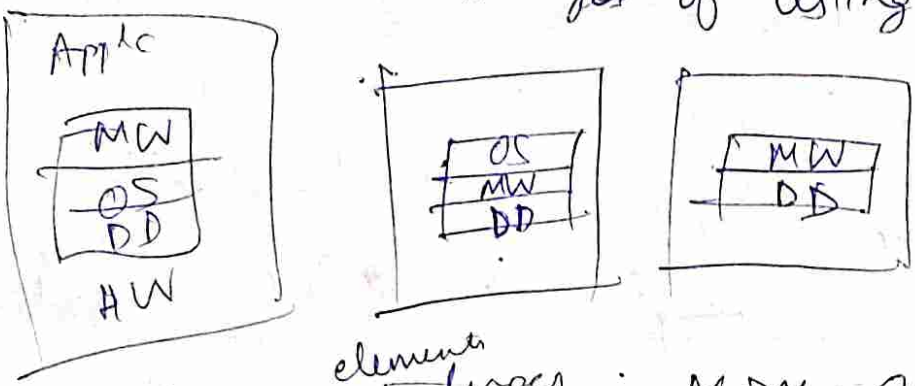
L. what is the problem? How to prevent it?
Priority inheritance & Inversion sth.

Q. 3 tasks: T_1 H, T_2 M, T_3 L — T_1, T_3 tasks need to access a shared resource. T_3 acquires the lock, T_2 becomes ready, preempts T_3.
Later T_1 becomes ready, tries to acquire the lock, But it is blocked. T_3 is still holds L,
Explain the problem and solve it.

11/04/2025

Compiler, interpreter, preprocessor, Middleware — interface b/w high level, low level

~~any~~ we need diff types of testing.



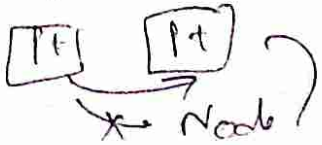
Middleware elements types: MoM, ORBs, Rpcs, DBaccess, N/w protocols

Types: general purpose, vertical specific
OSI, JVM (for TV)

Define the point
PDP protocol in OSI DLL/NW
↓ encapsulate, decapsulate * Synchronous and async
phases & sth?
list dead (establish, auth, N/w layer, link termination

SMTP: MTA, MUA

15/04/2025



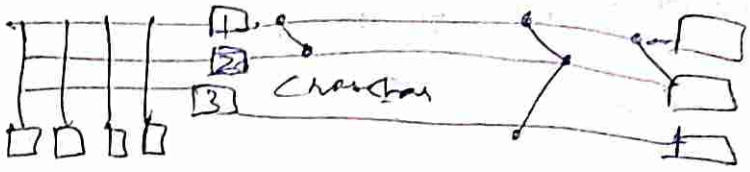
OSI 7 layers
Phys: mech, electrical
DLL: Reliable data transfer
N/W: end to end
transport: connection
session: apply dialog control

Presentation : data format, Application end-to-end User Interface

System Architecture :

* Topology * Scheme of communication
* Routing

Multistage n/w =>
multi-bus



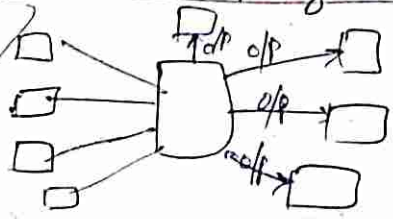
cross path topology

Data Push Programming

Based on o/p Controller triggers
⇒ considered fed data, some action
controller doesn't request.
Eg: heater, submarine

Automobile, serial, high speed, half duplex, high end application : short messages.

Control Area Network : defines standards for phy & DLL



← phys & DLL layer

Transmitter: Sends the msgs

Message : * fixed format * diff length (limited)

Routing : identifier ⇒ tells what type of info is carried type of action that has to be taken, sent to all processing elements.

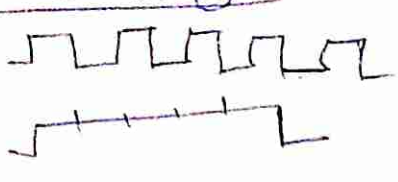
Multicast, Priorities, Remote Data Request
CAN use static priority ↳ src node requests connection, another node acts as transmitter

Physical layer :

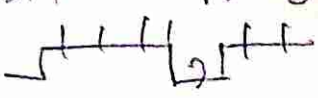
CAN-high must support bit dominance
CAN-low

~~Bus~~ wires operate on differential voltage

Bit Encoding



bit stuffing



SOF	Arbitration	Control	Data	CRC	ACK	EOF

↳ Identifier
↳ Dominance

Data?
Remote frame: 7 bits

Receive bits → ?

CSMA/CD+AMP Collision Dominant bit 0 ↑ high
↓ low

Error Control

Remote frame can transmitted on bus

↳ error detection, error signalling taken care by CAN controller

error frame, all messages are stopped (held), fault recovery, message retransmitted.

Bit error, message error

Bit stuffing: 5 consecutive 1's or 0's

Bit error: error in the data while transmitting?

↳ CRC error, Frame error, ACK Error / did not receive any ACK from the other

CAN Controller Modes

Error active, Error-passive, Bus off

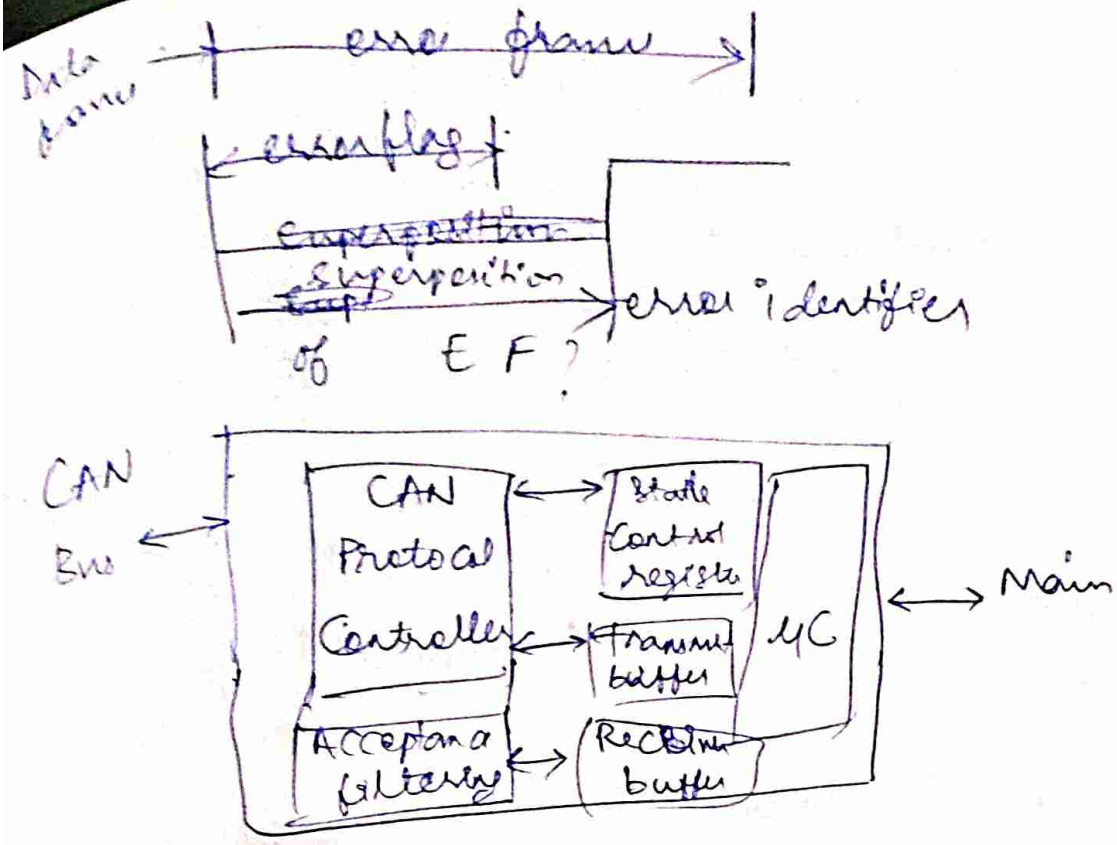
Normal error nodes can't transmit msg

active flag high priority dominance field ~~high~~ low

Error at receiver low priority passive flag

Dominance field high

executes serious issues / reset


Data frame
error frame
error flag
Superposition
Superposition
error identifier
of EF?

CAN
Bus
CAN
Protocol
Controller
Status
Control
register
Transmit
buffer
Acceptance
filtering
Receive
buffer
MC
Main