Arithmetic Circuits:

Circuits which perform arithmetic function.
(adding 2 binary nos: standard half adder).

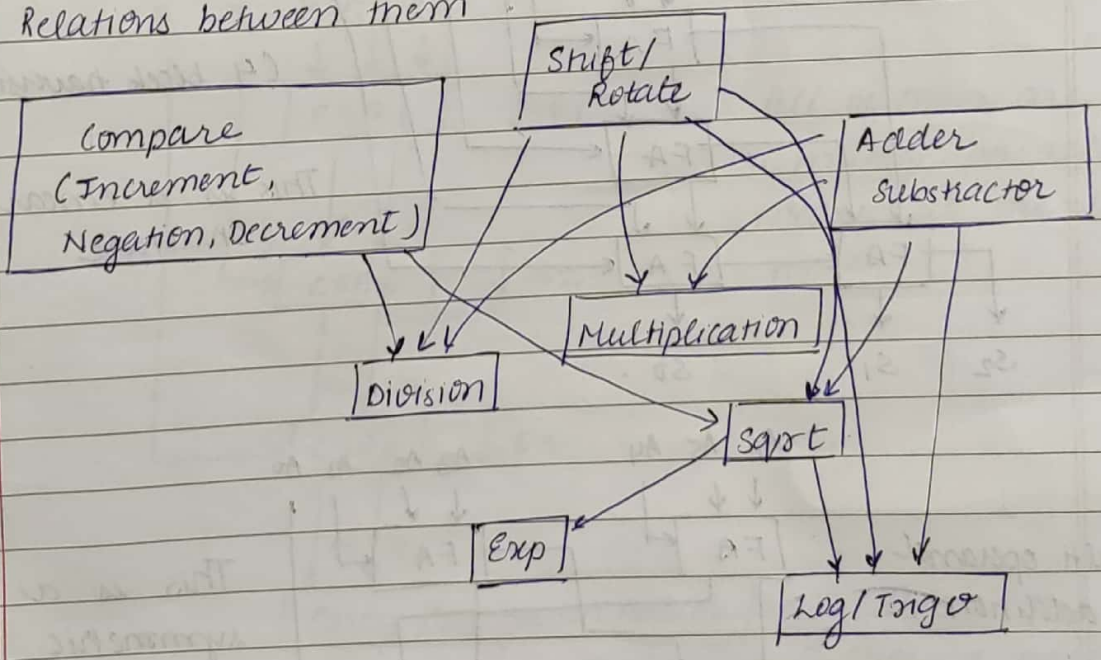
What and why do we use arithmetic circuits?

- They are the heart of every digital circuit.
- Optimize to improve performance.
- Most micro-controllers have ALU.
  Multiplication, Addition ALU, Shifter (incrementor, decrementor, etc).

Types of Arithmetic functions:

min complexity: shift, rotation, incrementor, decrementor, compare, negation, add, sub, -, *, ÷, sqrt, exp, log, trig max complexity.

Relations between them:



Most important block: acdder substractor.

1 1 0 1 1 1 0 0 1 1

no. of ones = 7

∴ I can infer that my add operation is nothing but a counter

Half adder is a (2,2) counter.

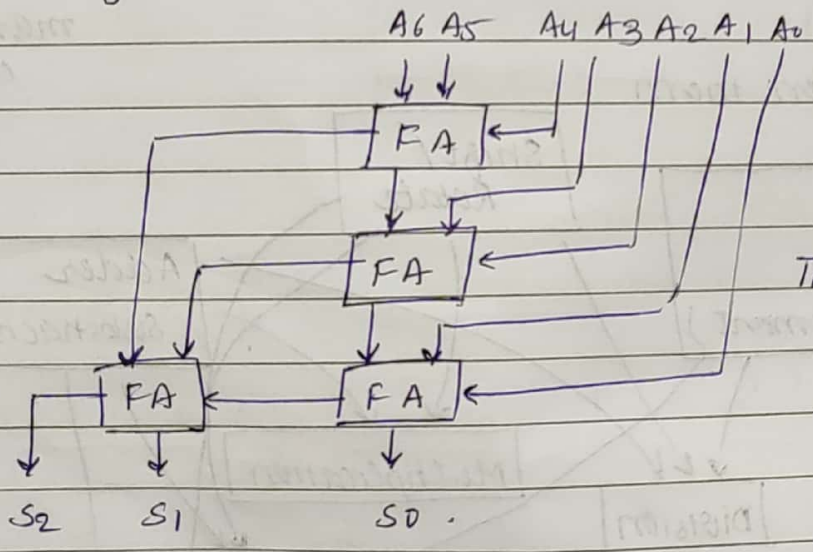
It can add two 1 bit numbers and the result is a 2 bit number

What is a (3,2) counter?

It can add 3 1 bit numbers and present it in a 2 bit number (full adder)

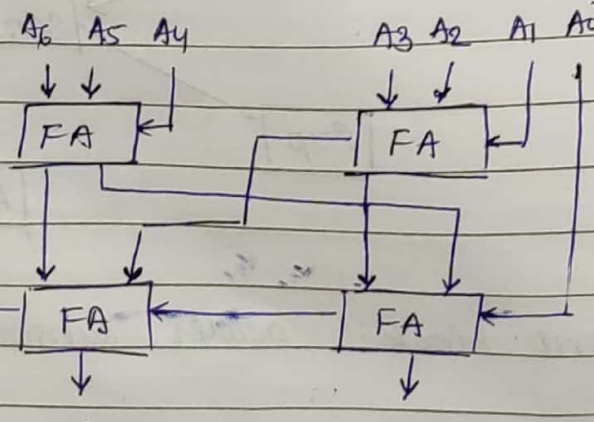
They are compressor type counters (3,2)

Adding multiple 1 bit numbers:



(4 block traversing)

This is a linear structure



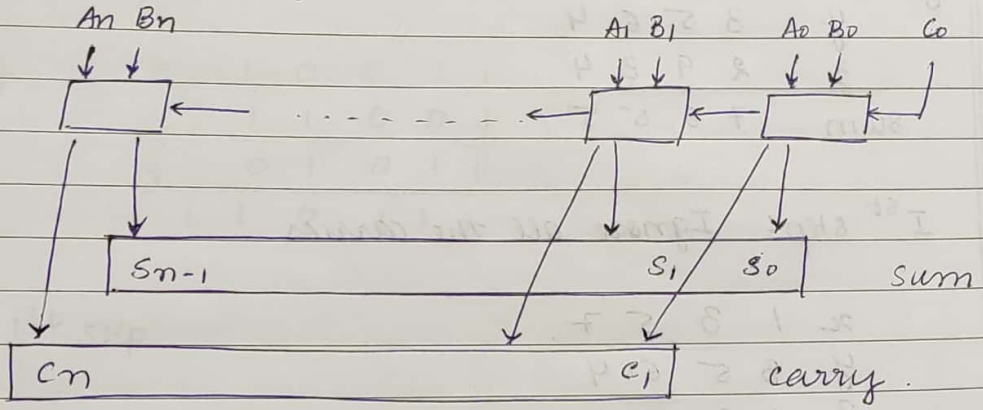
Multi operand addition.

This is a symmetric structure

(3 block traversing)

They are popularly called as carry propogate adder.

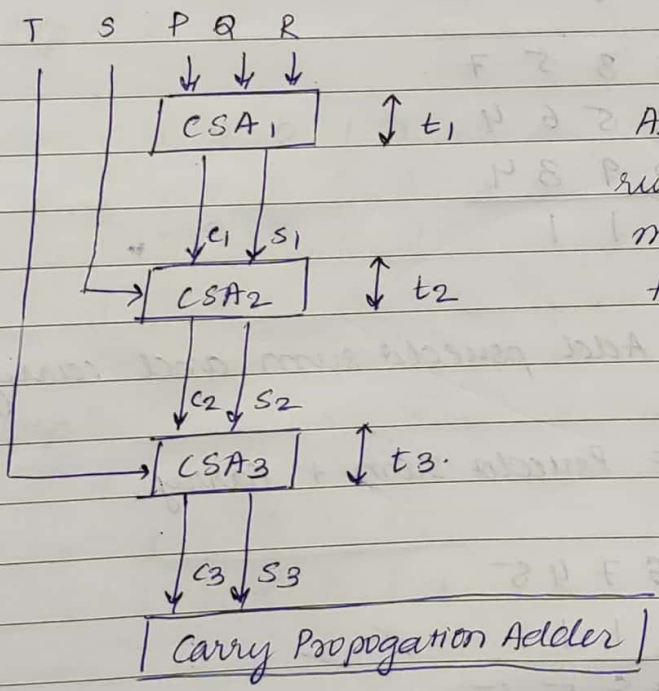
Instead of carry to be propogated we can use carry save adelers to reduce multiple i/p to 2 and then single CSA is used.



carry + sum = A + B + C.

$C_1 + S_0 = A_0 + B_0 + C_0$.

C + S = P + Q + R + S + T



All of them are running parallel, means at the same time.

This is the shorted path.

max (t1, t2, t3)

They are inherently parallel to each other. Generated Asynchronously.

Carry Save Adder:

eg
x	1	3	5	7
y	3	5	6	4
z	2	9	3	4
Sum	7	8	5	5

I st step: Ignore all the carries

x	1	3	5	7
y	3	5	6	4
z	2	9	3	4
Pseudo Sum	6	7	4	5

II nd step: Only write the carries.

x	1	3	5	7
y	3	5	6	4
z	2	9	3	4
Carry	1	1	1	

III rd step: Add psuedo sum and carry

Final Sum = Psuedo sum + Carry

Psuedo sum	6	7	4	5
Carry	1	1	1	
Final Sum	7	8	5	5

Carry and Psuedo sum can be computed independently.

This gives us a parallel implementation. This can give a better and faster implementation than carry ripple adder.

eg

$$\begin{array}{r} x \quad 1\ 0\ 0\ 1\ 1 \\ y \quad 1\ 1\ 0\ 0\ 1 \\ z \quad 0\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 1\ 1 \end{array}$$

1st step:

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1 \\ 1\ 1\ 0\ 0\ 1 \\ 0\ 1\ 0\ 1\ 1 \\ \hline 0\ 0\ 0\ 0\ 1 \end{array}$$

2nd step:

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1 \\ 1\ 1\ 0\ 0\ 1 \\ 0\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 1 \end{array}$$

3rd step:

$$\begin{array}{r} 0\ 0\ 0\ 0\ 1 \\ 1\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 1\ 1\ 1 \end{array}$$

I take 3 operands (to build my hardware)

A+B+C+D.

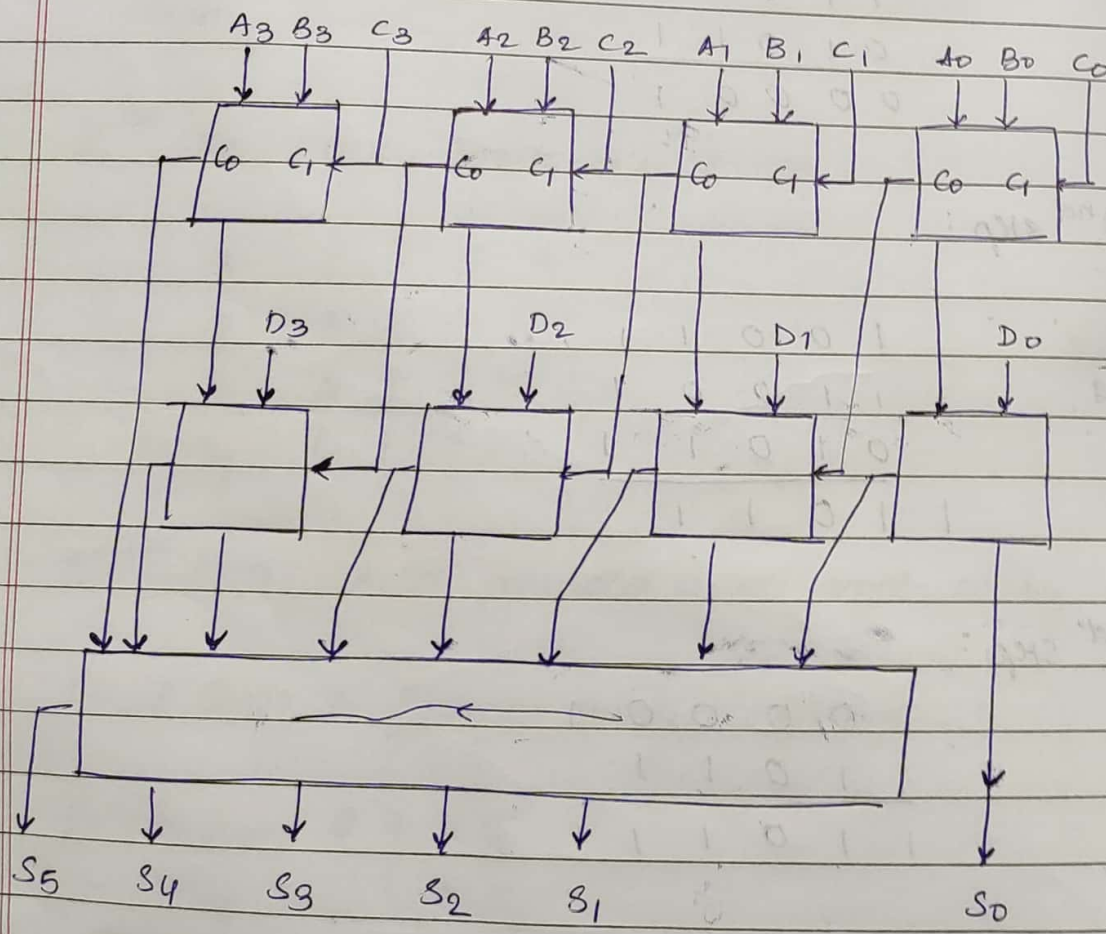
	A_3	A_2	A_1	A_0
A	0	1	1	1
	B_3	B_2	B_1	B_0
B	1	0	1	0
	C_3	C_2	C_1	C_0
C	1	1	1	1
	0	0	1	0 $\quad S_1'$
	1	1	1	1 $\quad C_0'$
	D_3	D_2	D_1	D_0
D	1	0	1	1
	1	0	1	1 $\quad S_2$
	1	0	1	0 $\quad C_0^2$
	1	0	1	0 1 1

A+B+C. 1st CS sum

2nd CS sum.

delay

Final sum.



P7

For m integer operands, involves (m-2) CSA additions

delay
- in RCA,
  $(m-1)\ t_{RCA}$
- in CSA,
  $(m-2)\ t_{CSA} + t_{RCA}$ $\quad m \geq 3$

For multiplication:

$A_3\ A_2\ A_1\ A_0$ — multiplicand A_j

$B_3\ B_2\ B_1\ B_0$ — multiplier B_j

$P_{03}\ P_{02}\ P_{01}\ P_{00}$ (and)

$P_{13}\ P_{12}\ P_{11}\ P_{10}$ (HA)

$P_{23}\ P_{22}\ P_{21}\ P_{20}$ (FA)

$P_{33}\ P_{32}\ P_{31}\ P_{30}$ (FA)

$P_7\ P_6$ (and) P_5 (HA) P_4 (FA) P_3 (FA) $P_2\ P_1\ P_0$

Partial Products P_{ij}

$P_0 = P_{00}$

$P_1 = P_{01} \oplus P_{10}$

$P_2 = P_{02} \oplus P_{11} \oplus P_{20} \oplus C_{12}$

$P_3 = P_{03} \oplus P_{12} \oplus P_{21} \oplus P_{30} \oplus [C_{23} \oplus C_{23}']$

$P_4 = P_{13} \oplus P_{22} \oplus P_{31} \oplus [C_{34} \oplus C_{34}' \oplus C_{34}'']$

$P_5 = P_{23} \oplus P_{32} \oplus [C_{45} \oplus C_{45}' \oplus C_{45}'']$

$P_6 = P_{33} \oplus [C_{56} \oplus C_{56}']$

$P_7 = C_{67}$

$P_{00} = A_0 B_0$

$P_{10} = A_0 B_1$

$P_{01} = A_1 B_0$

$P_{20} = A_0 B_2$

$P_{11} = A_1 B_1$

$P_{02} = A_2 B_0$

$P_{03} = A_3 B_0$

$P_{12} = A_2 B_1$

$P_{21} = A_1 B_2$

$P_{30} = A_0 B_0$

$P_{13} = A_3 B_1$

$P_{22} = A_2 B_2$

$P_{31} = A_1 B_3$

$P_{23} = A_3 B_2$

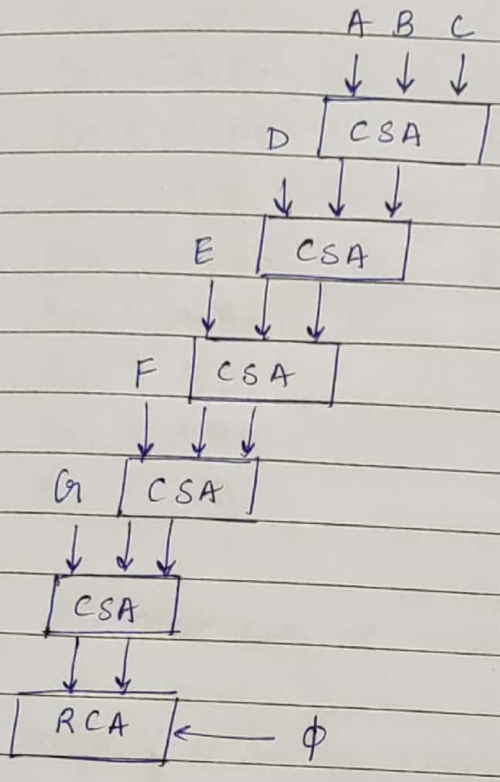
$P_{32} = A_2 B_3$

$P_{33} = A_3 B_2$

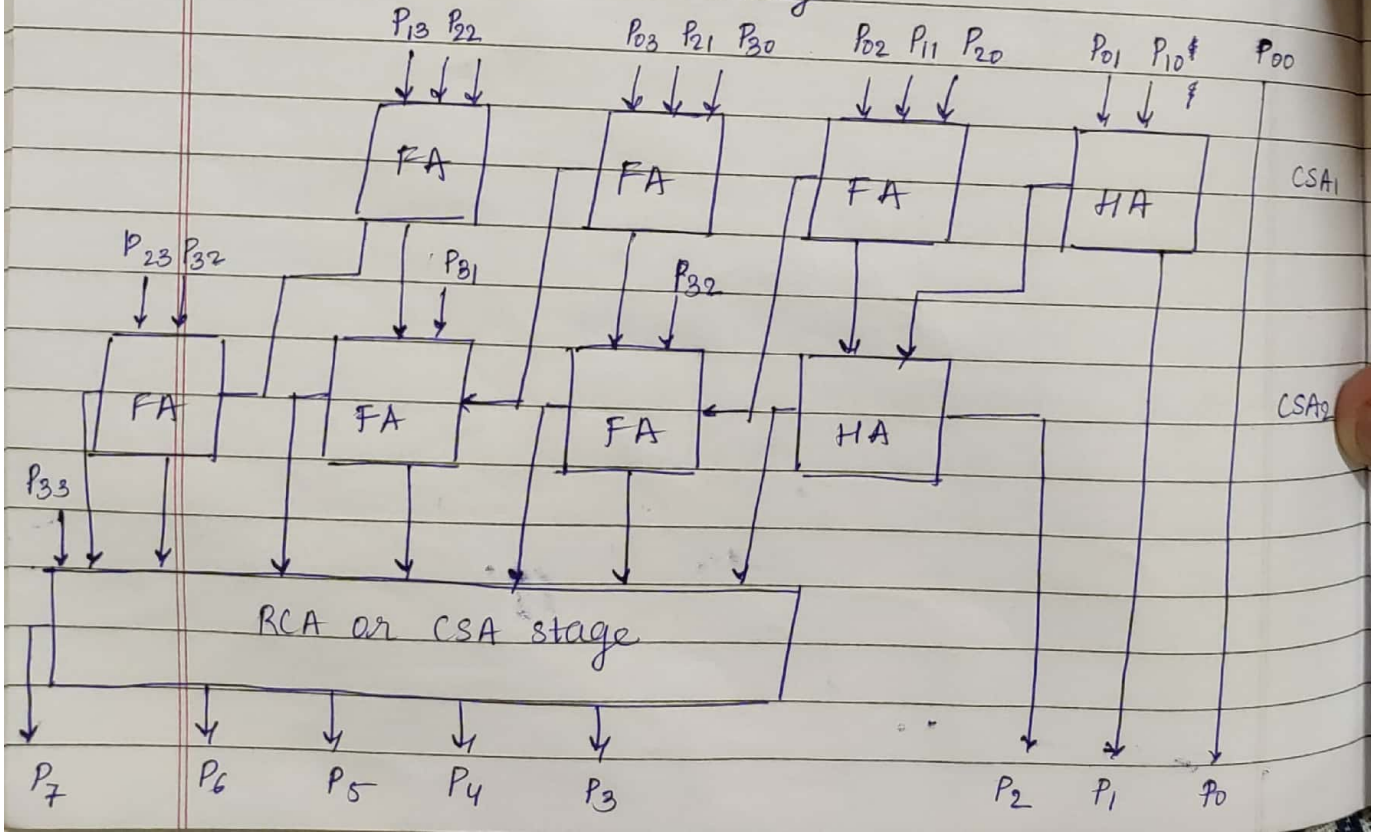
All these can be realized with 'AND'.

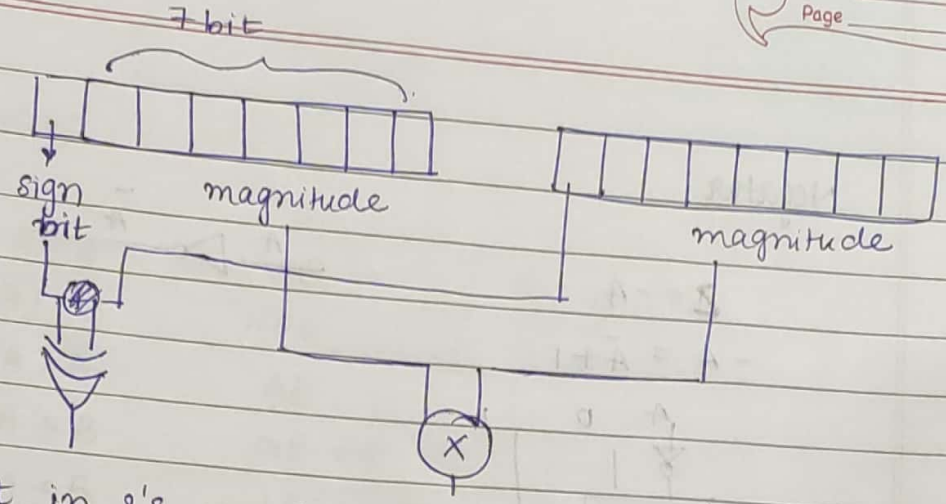
$m \times n$ full adder stages : Array multiplier

$A+B+C+D+E+F+G$



Sure, I can replace a FA or RCA stage to CSA stage. I can replace my parrallel multiplier with CSA stage.





Represent in 2's complement and then multiply.

eg

```
  0 0 1 1   3
  1 0 1 1  -5
```

Put n (no. of digits) of 1 in front on 2's complement

```
  0 0 0 0  0 0 1 1   3
  1 1 1 1  1 0 1 1  -5
```

Operations : (Adder)

- Incrementor
- Negator
- Subtractor
- Adder Subtractor
- Comparator

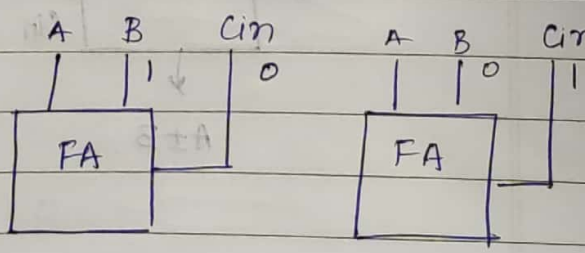
Incrementor :

$Z = A + 1$.

$\therefore Z = A + B + Cin$

$B = 1 \quad Cin = 0$

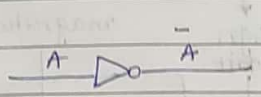
$B = 0 \quad Cin = 1$

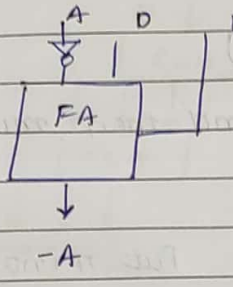


Negator:

$Z = -A$

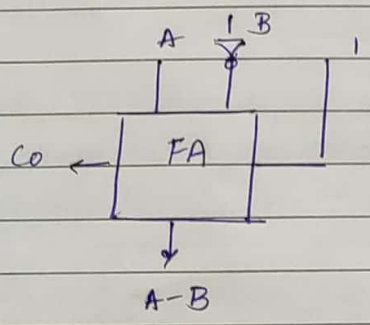
$-A = \bar{A} + 1$



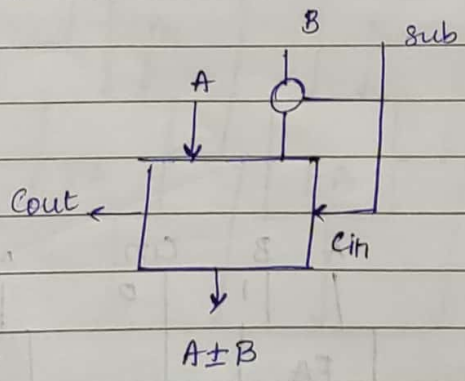


Subtractor:

$Z = A - B$



Adder Subtractor:



Cin is used to complement B

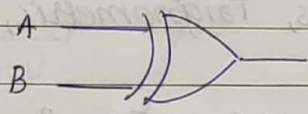
Comparator:

$A = B$	EQ	} Parity
$A != B$	NEQ	
$A > B$	GE	
$A \geq B$	GE . EQ	
$A < B$	LT / $\overline{GE}$	
$A \leq B$	$\overline{GE}$ + EQ	

$A > B$: $A - B \Rightarrow$ Cout = 1.

$\therefore A > B$

Cout = 0.



$\therefore A = B$.

else.

$A < B$

Without adders:

- Shifters
- Rotators

Logical shifter:

$11001 >> 2 = 00110$

Arithmetic shifter:

$11001 >>> 2 = 11110$

Same as LS but fills the empty space with the MSB.

Rotator:

11001 ROR 2 = 01110

Rotates bits in circle. Bits shifted off one end are shifted to the other end.

we can use the shifter as multiplier or divider.

Adder, shifter, comparator

<<2	$\times 2^2$
>>>2	$\div 2^2$

Dividers, Logarithms, Trignometric, Exponential

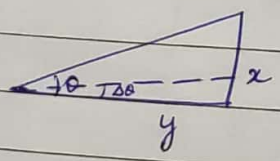
$$\exp(x) \text{ or } e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

x^2 Squaring fn → *

$x^2/2$ Dividing fn → +, −, comp, *

$x^2/2 +$ Adding fn → +

Division is expensive, thus we do multiplication instead of division by storing ($1/p$) values in the Look Up Table (LUT) / ROM / Memory.



$$\tan\theta = \frac{x}{y}$$

$$\theta = \tan^{-1}\frac{x}{y}$$

→ Linear transformation (2^{-1})

Fetch / D

8
16
32
64

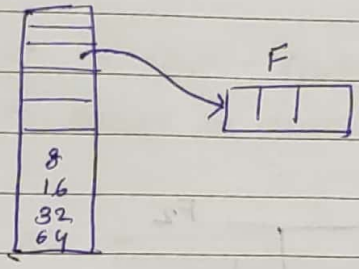
Fetch fr
some
to kno

F →

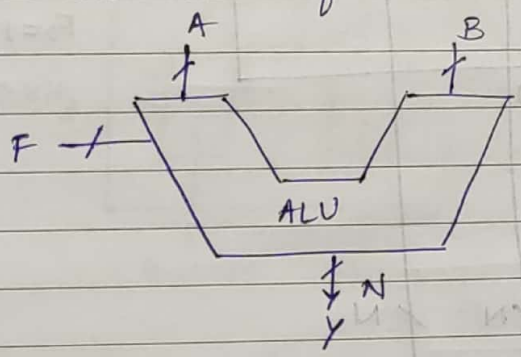
C

1

Fetch / Decode / Execute:

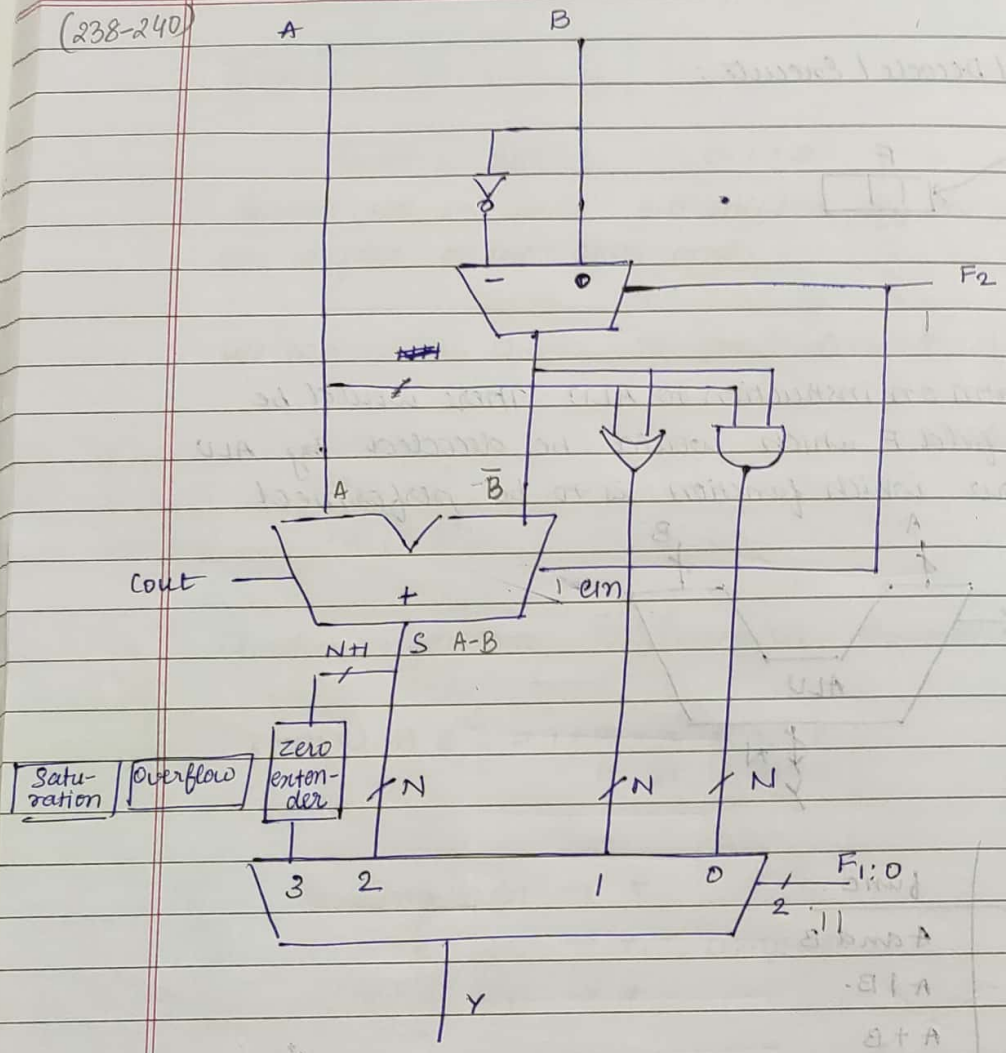


Fetch from an instruction to ALU. There would be some field F which would be decoded by ALU to know which function is to be performed.



$f_{2:0}$	func.
000	A and B
001	A | B.
010	A + B
011	X
100	AB - B
101	A| - B
110	A - B
111	SL

(238-240)



A = 25

B = 32

Compute this 32 bit ALU for shift left (SL) operation

A - B = 25 - 32 = -7

∴ The 2's complement represents -7, the MSB is 1

∴ S_{31} or Y_3 = 1

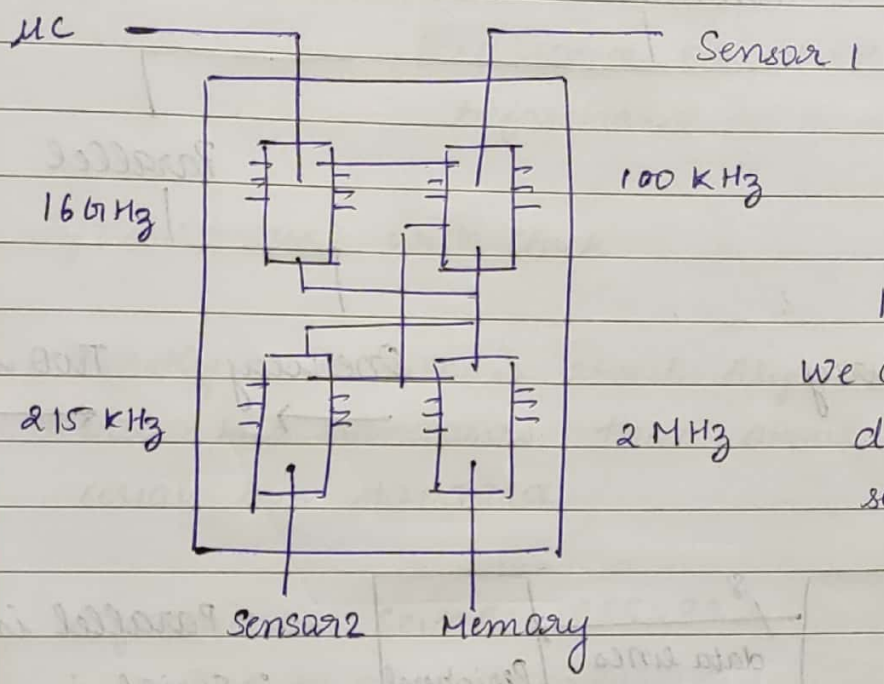
A<B

zero extender : 000.... 1 (32 bits)

31 zeroes added as $F_{1:0}$ = 11, 3rd line is selected.

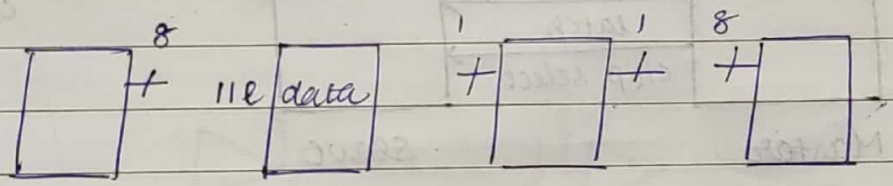
Communication Protocol / Interface

→ On chip - chip to chip
→ Device to device



I^2C : standard communication between the 4.

We can transfer our data parallel or serial

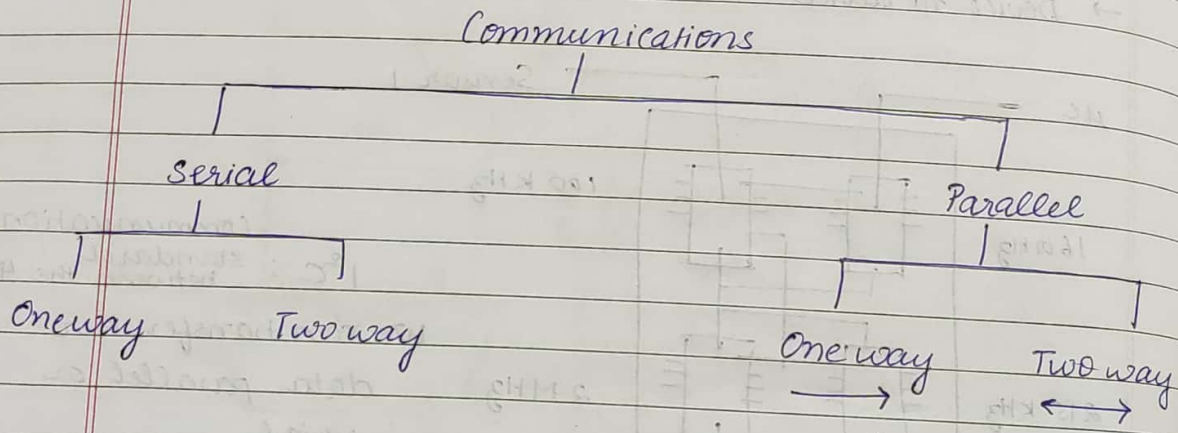


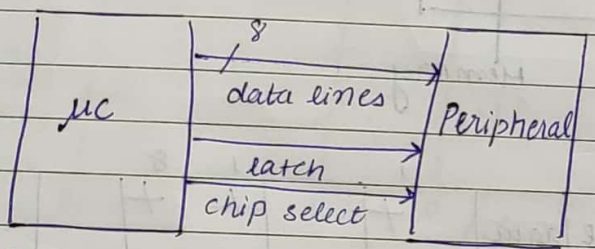
Serial - UART (Universal Asynchronous Receiver Transmitter)
- SPI (Serial Peripheral Interface)
- I^2C (Inter Integrated Circuit Communication)
- I2S (Inter Integrated Sound Bus)

Device to Device:
- USB
- Ethernet
- VGA
- SCSI
- HDMI
- PS2
- Sonet

Two ways of communication:

- Serial
- Parallel





Master or Controller

Slave

- Parallel is fast.
- Serial is cumbersome.

Parallel	Serial
Fast	Slow
Pin (Real estate) is expensive	Cheap.

Serial is usually used as there are many pins on the chip.

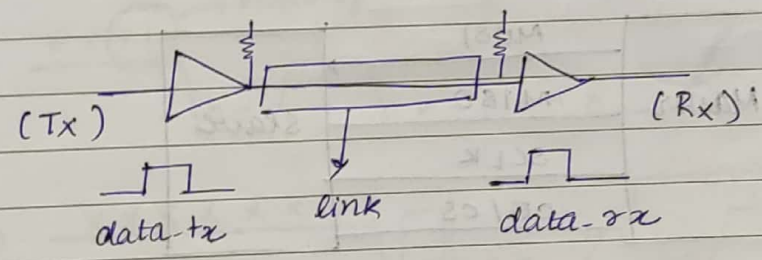
Serial connection - Due to the lesser no. of pins the complexity of communication is high.

Asynchronous : without a clock.
But speed has to be agreed beforehand

Synchronous : with clock

Firstly, we discuss about Asynchronous Serial bus standards then about synchronous serial bus standard.

Asynchronous Serial (RS232)
- One to one communication.



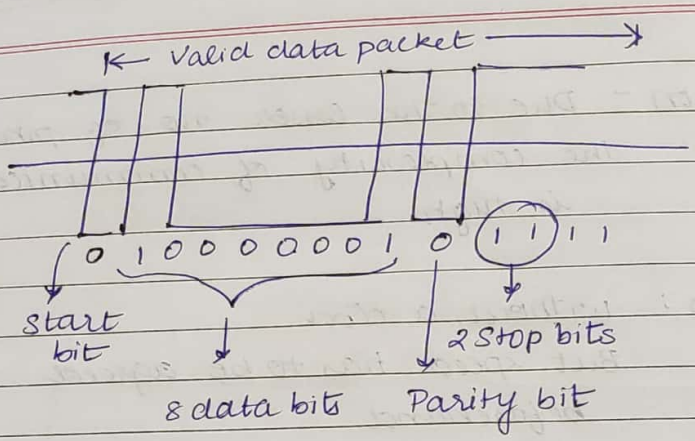
- one Tx, one Rx
- Transmission process (9600 baud rate)
freq at which data is transmitted

$\frac{1}{9600}$ = 0.104 ms = 104 µs.

Every 104 µs, you get 2 data.

- Transmit idles high
- It goes for 1 bit.
- Sends LSB first MSB last
- Even or odd parity

- Start and stop bit for communication

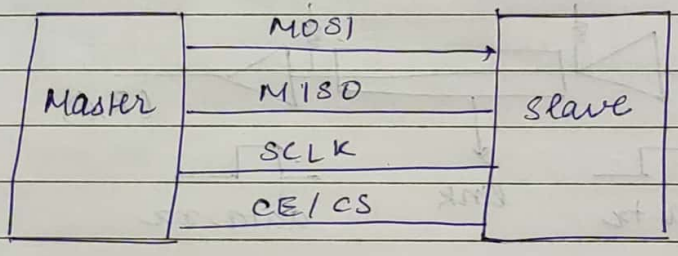


Key points of serial communications:

- 10-30 Gbps
  Typically it started with < 115.2 kbps.

SPI (Serial - Peripheral Interface):

4 wires



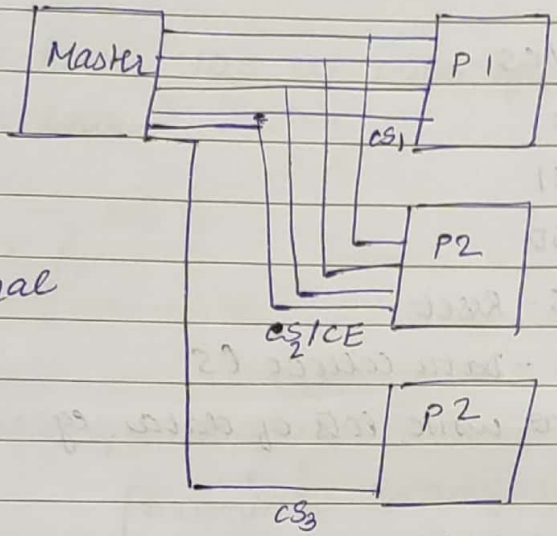
High data rate (1 MHz to 70 MHz clk)
MOSI - Master Out slave In } Can be shared between any no. of peripherals.
MISO - Master In Slave Out }
SCK/SCLK - Sync./Serial Clock
CE/CS - Chip Enable/Chip select

difference UART - (SCK)

wires / Lines are unidirecti

MOS

MOSI:



CS: This cannot be shared

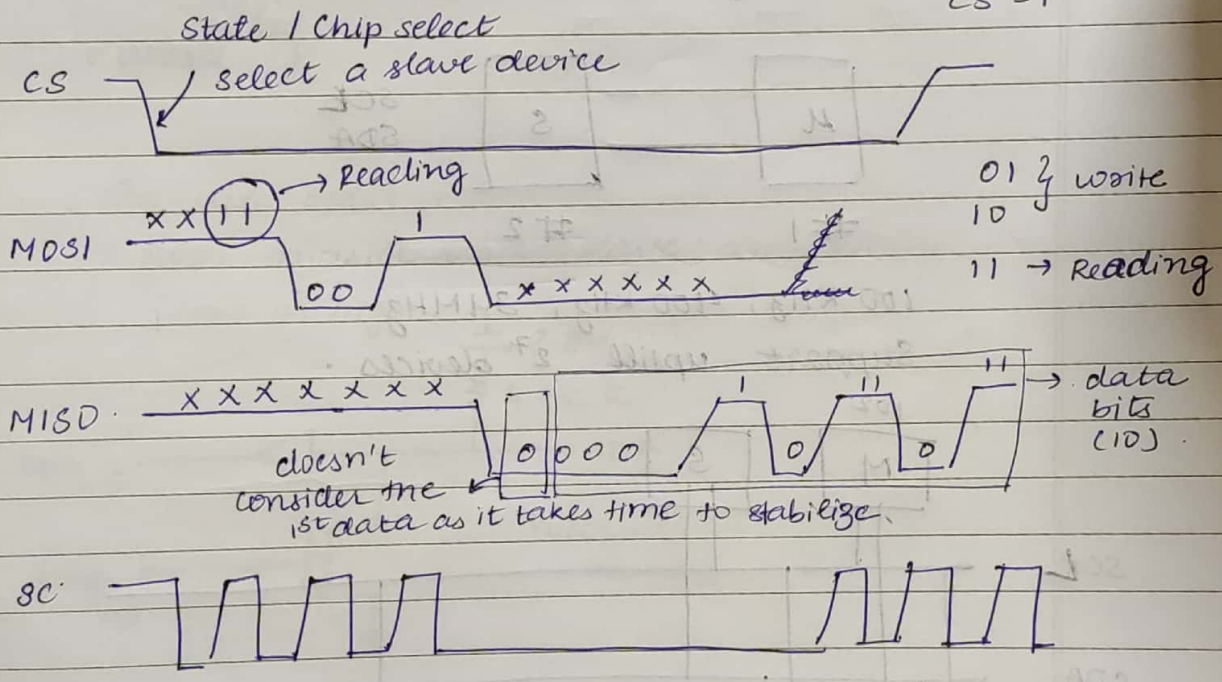
wires / Lines are unidirectional

Q Why do the radio devices have SPI interface?

SPI:

$\overline{CS} = 0$.

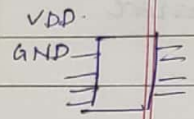
$CS = 1$



MOSI:

000
001
:
101
111
} < 7 channels.

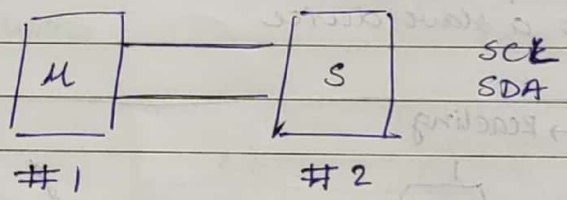
5-6 channels.



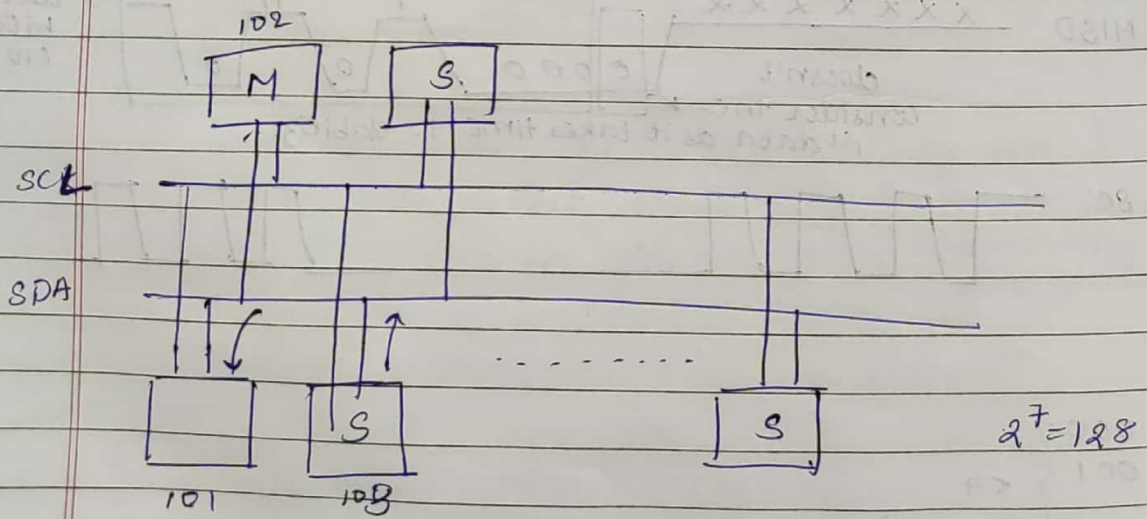
6 wires:
1. CE/CS
2. SCK
3. MOSI
4. MISO
5. RES - Reset
6. D/C - Data collect CS decreases to write lots of data eg

3 for 2 wires : drop MISO
drop CE/CS (only one device)

I^2C: Inter Integrated Circuit
Communication Protocol developed in 1980 by Philips



100 KHz, 400 KHz, 34 MHz
Support uptill 2^7 devices.

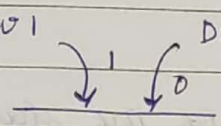


$2^7 = 128$

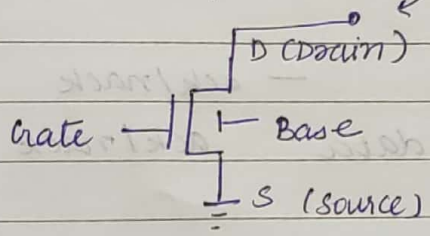
Video: Fun and Easy I2C

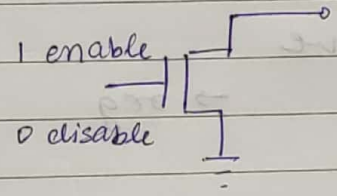
classmate
Date
Page

Select these ID's by pulling external pin high or low.

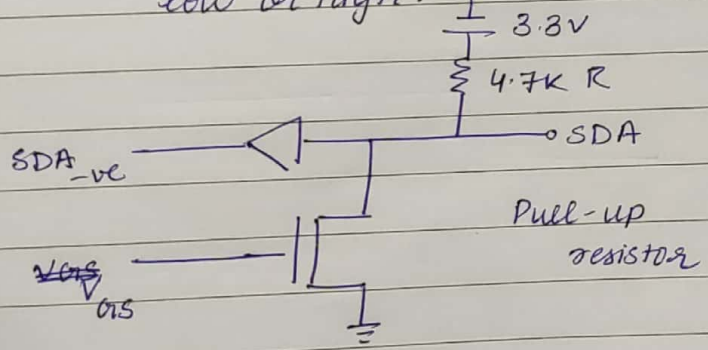


I^2C uses open drain:





I^2C uses open drain:
→ both master and slave are either low or high.



I^2C:

1. Master is incharge of SCL frequency (MHz)
   ↳ 3.4 MHz

   100 kHz
   400 kHz

2. Master generates the address of the slaves.
   If ——— is ack/nack
   128/1024

3. If sends a RD/WR — ack/nack
4. It sends/receives data ack/nack

Sequence is as following:

Master is to write to a slave

a. Start → write → . → reg

b. Master to read from slave.

Start →