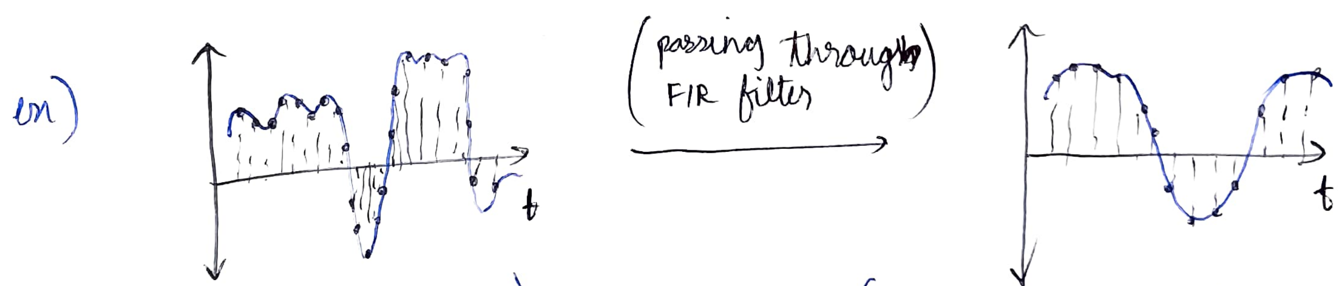
EE2016 EE19B070 ANUVITH PABBA

PART-C (MIDSEM) REPORT :

1) Theory of FIR & FIR filters :

FIR stands for "Finite Impulse Response",
and FIR filter is a "Finite Impulse response filter"

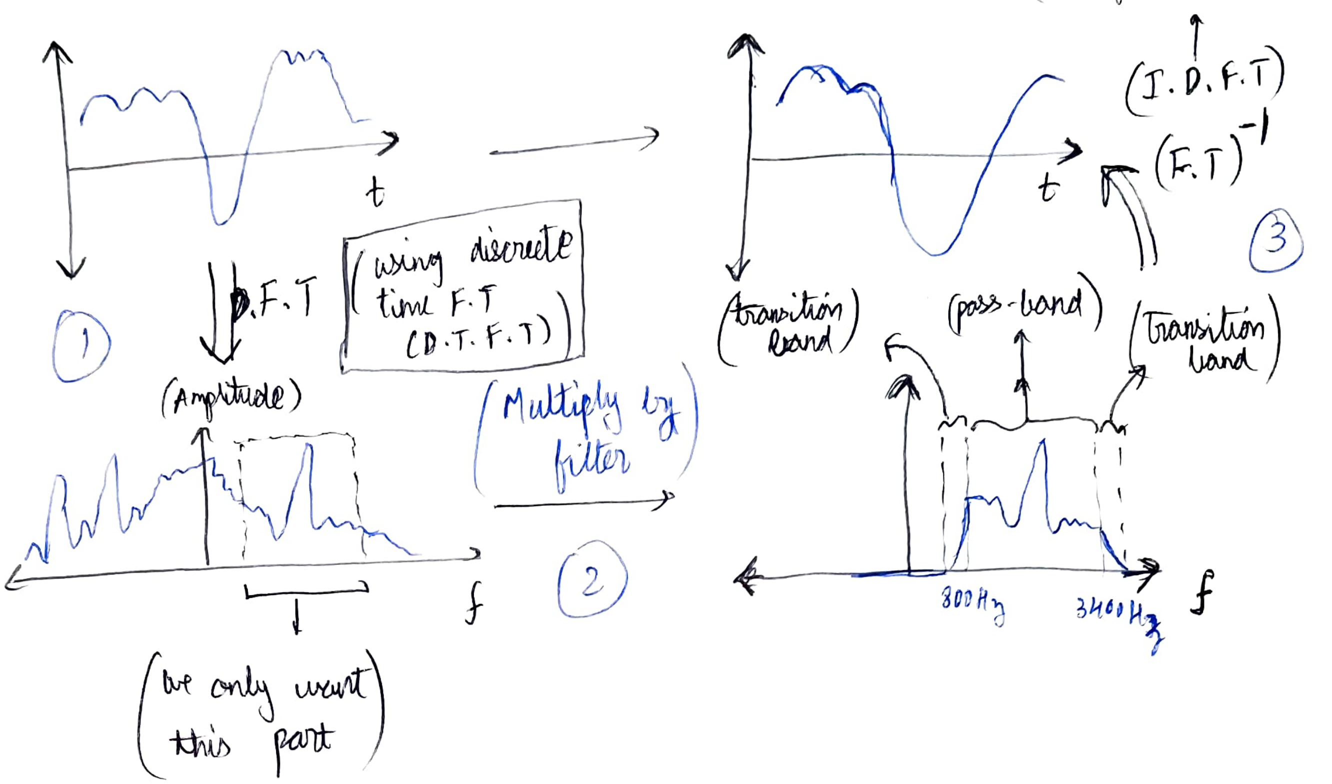
given a signal which is a superposition of various sinusoids with random frequencies, we can pass this signal (ONLY AFTER SAMPLING) through an FIR filter and the output sampled signal would only retain those sinusoids with frequencies WE desire.



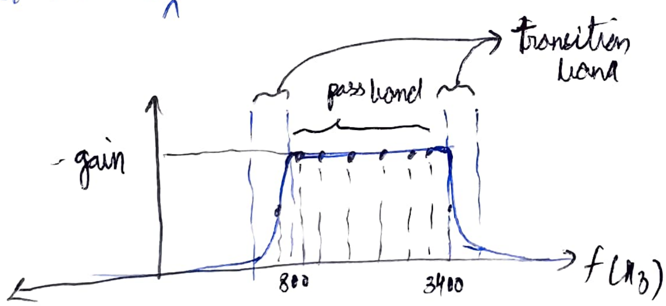
(frequency range is not known)

(frequencies only lie b/w 300 Hz to 3400 Hz)

So the way we can do this is by,



the filter in freq. domain would be,



We also know that multiplication in freq. domain is the same as CONVOLUTION in the time domain.

In the above schematic of realising a filter, the steps are:

1. fourier transforming Input into freq. domain (SAMPLED INPUT) (using DTFT)
2. multiplying it with the required filter
3. Inverse fourier transforming the result. (GIVES SAMPLED OUTPUT)

∴ the DT convolution,

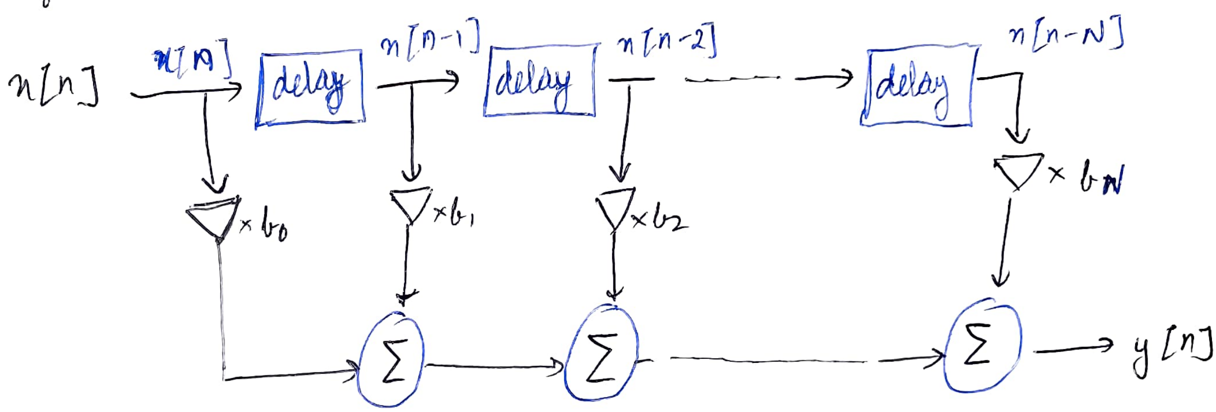
$$y[n] = \sum_{k=0}^{N} x[k] \times h[n-k] \equiv \sum_{k=0}^{N} x[n-k] \times h[k]$$

$$= x[0] \times h[n] + x[1] \times h[n-1] + x[2] \times h[n-2] \text{ ---} + x[N] \times h[n-N]$$

(commonly represented as

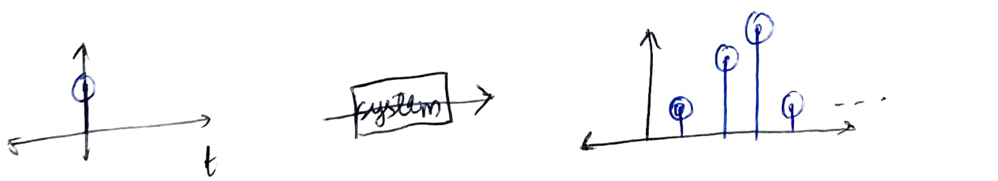
$$= b_0 x[n] + b_1 x[n-1] \text{ ----} + b_N x[n-N])$$

∴ N-Tap FIR filter general design =



FIR = finite impulse response

= the response of the system when the simplest ~~unit~~ impulse function is given as the input



input = impulse response

output = impulse response

Discrete time convolution :

is an operation on 2 discrete time signals defined by the integral

$$y[n] = (f * g)[n] = \sum_{k=-\infty}^{\infty} f[k] g[n-k]$$

Discrete Fourier transformation (D.F.T) :

if we sample an input func. at ω_k, s.T

$$\omega_k = \frac{2\pi}{N} k, \quad k = 0, 1, 2, \ldots, N-1$$

$$\therefore X[k] = X\left(e^{j \times \frac{2\pi}{N} k}\right) \qquad (\text{DFT of } x[n])$$

$$= \sum_{n=0}^{N-1} x[n] e^{-j \times \frac{2\pi}{N} \times k \times n}$$

& IDFT (inverse discrete fourier transformation) :

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\left(\frac{2\pi}{N} k\right) n} \qquad (\text{IDFT of } X[k])$$

2) On the Fixed Point Arithmetic used:

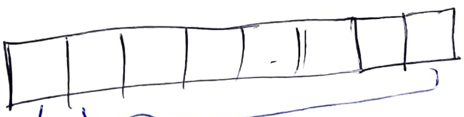
While searching for a suitable scaling factor, I had 2 goals in mind.

1) Altough fractional F.P multiplication is also available, I only try to use integer multiplications, (i.e) after the inputs / coeff are multiplied with suitable scaling factor then they should be integers (then we can ignore the fractional part)

2) I need to try and use as many bits as possible so that we get the maximum precision of the filter.

∴ as we can see, the coeff are all lesser than "1" & when we give the input signal, we generally use an amplitude of "1" ⇒ the sampled input values are also lower than "1".

∴ in a byte, there are 8 bits



(sign bit = 1) (mag^n bits = 7)

∴ since all inputs & coeff^n are less than 1, we can SCALE THEM using 2^7, so that the mag^n of the scaled output is always lower than 7 bits (2^7), and then the first bit can be used for the sign.

M bit of output = $(N_1 + N_2) + \log_2(N) + 1 \rightarrow$ sign

N_1 = inputs = ~~32~~ 7 bits + 1 sign

$\therefore N_1 = 7$

$N_2 = N_1 = 7$ (N_2 = coeff = 7 bits + 1 sign)

N = no. of additions = 5 (5 coeff.)

$\therefore M = 7 + 7 + \log_2 5 + 1$

$= 7 + 7 + 2.322 + 1$

$= 17.322$

$\therefore M$ = 18 bits = output

= 3 bytes (but we dont fully use one of the bytes)

4) Actual implementation Details:

a) INPUTS into the AVR,

we have 2 inputs.

1st Input set = COEFF, In the program I've submitted, there are 5 1byte inputs given. (signed)

this is given in a .db form, ∴ its : stored in the program memory.

When we use Z pointer to find the memory location its stored in, we see that its stored in

"0x 0072, prog" in the "prog FLASH"

as there are 5 inputs in COEFF, they are stored in

0x0072, prog upto 0x 0076, prog

2nd INPUT set = INPUTS, in the prog. I submitted, we have 10 1byte signed inputs.

they are stored in

"0x0078, prog" in the "prog FLASH"

0x 0078, prog upto 0x 0082, prog

I then store the COEFF inputs into the SRAM, I store them from 0x0060 to 0x0064, data in the "prog FLASH"

Outputs: many 3 byte outputs which are stored successively, with the 1st byte of the 1st output stored at 0x008C.

no. of 3 byte outputs = (No. of INPUTS) − (No. of COEFF.) + 1

as,

b_0 b_1 b_2 b_3 b_4

I_9 I_8 I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0 $t = 1$ unit

→ $t = 2$ unit

→ → $t = 3$ unit

⋮

(pls. refer FIR diagram in page ②)

I_9 I_8 I_7 I_6 I_5 ---- I_0 $t = 6$ unit

Output = 3 bytes,

b) No, I have not used circular buffers ⑤

c) Initially, the COEFF inputs are given in the .db form & they are initially stored in "0x0072, prog." which is the program memory, (0x0072 to 0x0076)

Next, I copied these value into the SRAM in the "0x0060, data" location

(0x0060 to 0x0064)

d) Multiplication:

In the Small Loop, 5 multiplications take place and ~~they~~ all need to be added together.

for this use MULS, ADD and ADC

In the first multiplication, we multiply the value pointed by Z and that by X (points to 0x0064 first time), we put them in registers R22 & R21, then we MULS R22, R21 which is the signed multiplication of R22 & R21, the resulting bytes (signed) are stored in R1, R0.

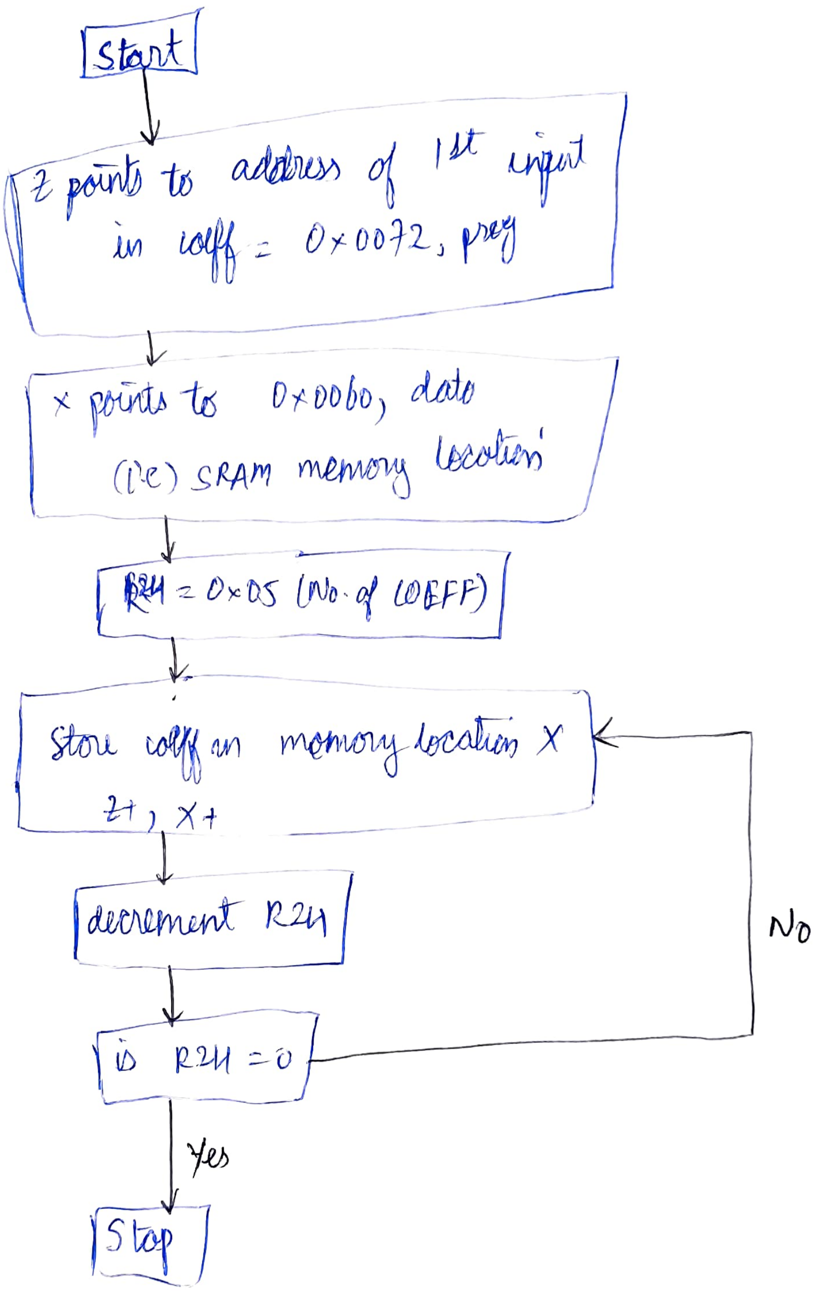
then we ADD R0, $\underbrace{\text{R16}}_{\text{(in my program)}}$, then we ADC R1, R17. (ADD with prev. carry)

if there's any overflow

∴ now, the sums are stored in R18, R17, R16

Flow - Chart:

a) to store COEFF memory in SRAM,



b) Program flow chart