INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
EMBEDDED SYSTEM LAB REPORT
Course Code – EE39004

Experiment - Digital Low Pass FIR Filter Using ATMEGA32

Name: Ayush Pandit
Roll No.: 19EE10067
Department: Electrical Engineering

Experiment 2:

1. Aim of the Experiment:
To Program ATMEGA32 to generate voltage signal which filters the input signal received in the ADC port of the microcontroller and gives a filtered output when passed through a Digital-to-Analog converter.

2. Requirements:
1. ATMEGA32
2. DAC0808(Digital to analog converter)
3. Resistors - 4 (5kΩ),2 (6kΩ),1 (30kΩ)
4. Capacitor - 1 (1µF),2 (22pF),2 (100nF)
5. Quartz crystal clock
6. Op-amp(LM392)
7. DC voltage generator - 5
8. AC voltage generator - 2
9.Oscilloscope

3. Procedure:
To perform the experiment we need to build the required digital filter parameters in MATLAB and use the parameters in assembly code to program the ATMEGA32 microcontroller.
MATLAB code:-
Code for convolution function-

```
function Y = myconv(x,h)
m=length(x);
n=length(h);
for i=1:n+m-1
Y(i)=0;
for j=1:m
if(i-j+1>0)&&(i-j<n)
Y(i)=Y(i)+x(j)*h(i-j+1);
else
end
end
end
end
```

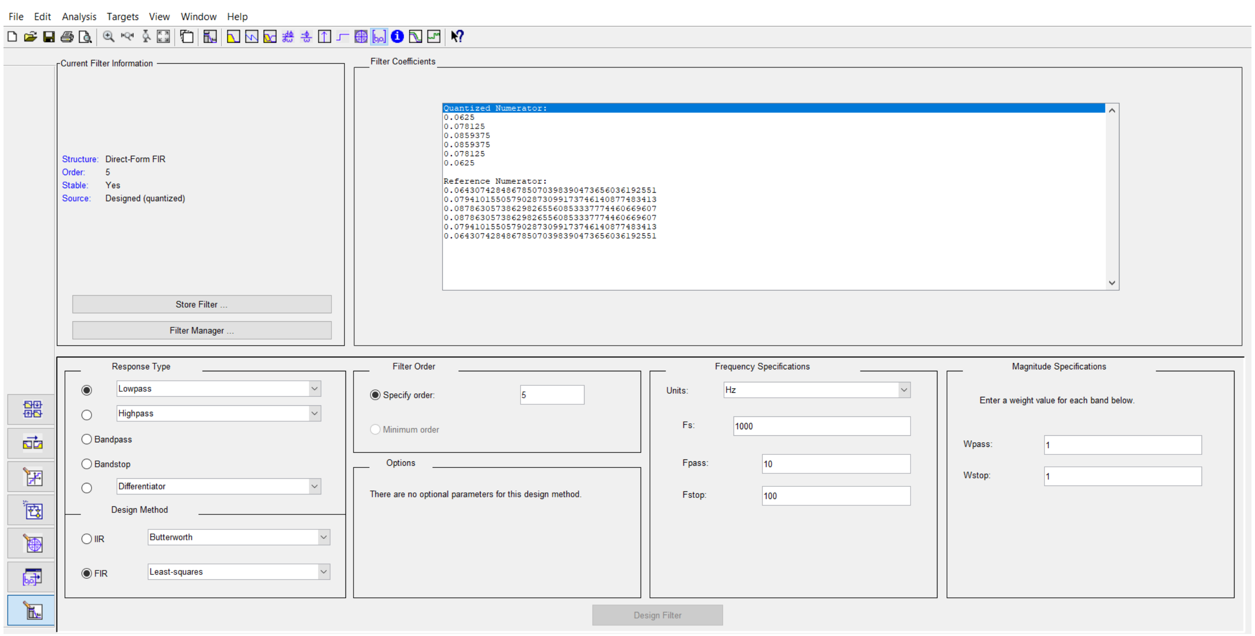
Now we can use the inbuilt filterDesigner of MATLAB to generate the low pass digital filter parameters.

Frequencies used in the design:-

$F_{Sampling}$ = 1000Hz

F_{Pass} = 10Hz

F_{Stop} = 100Hz



We need to convert the filter coefficients to hexadecimal to use them in 8-bit registers available in the ATMEGA32 microcontroller. The filter coefficients we get from the chosen frequencies are 10, 14, 16, 16, 14, 10 (All are of base 16).

Now we can write the assembly code for the ATMEGA32 microcontroller to perform the digital filtering on the provided input signal mixed with high-frequency noise.

4. Assembly code:

```
;
; DigitalFilter.asm
;
; Created: 21-01-2022 14:00:50
; Author : ayush
;


; Replace with your application code
.INCLUDE "M32DEF.INC" ;add ATmega32 definition
.ORG 00 ;origin at 0x00

.EQU H0 = 0x10 ;filter coefficients from matlab tool
.EQU H1 = 0x14 ;originl coefficients multiplied by 256
.EQU H2 = 0x16
.EQU H3 = 0x16
.EQU H4 = 0x14
.EQU H5 = 0x10

LDI R16, HIGH(RAMEND) ; load SPH with the high byte of maximum available RAM
OUT SPH, R16
LDI R16, LOW(RAMEND) ; load SPL with the low byte of maximum available RAM
OUT SPL, R16

LDI R22, 0x00 ; initialize registers to be used to hold the sample
LDI R23, 0x00
LDI R24, 0x00
LDI R25, 0x00
LDI R26, 0x00
LDI R27, 0x00
LDI R29, 0x00

LDI R16, 0x00 ; define port A as input
OUT DDRA, R16
LDI R16, 0xFF ; define port B as output
OUT DDRB, R16

LDI R16, 0x87 ;enable ADC, ADC clock = crystal clk/128
OUT ADCSRA, R16
```

```
LDI R16, 0xE1 ;ADC1 selected, left adjustment, Vref =2.56V
OUT ADMUX, R16

READ_ADC:
	NOP ; No operation instruction just consumes a clock cycle without doing anything
	SBI ADCSRA, ADSC  ;start ADC conversion

KEEP_POLLING:
	NOP
	SBIS ADCSRA, ADIF ; if it is the end of conversion, skip the next instruction and come
out of the loop
	RJMP KEEP_POLLING ; keep polling until the END of the conversion
	SBI ADCSRA, ADIF ; write 1 to clear ADIF flag

	IN R20, ADCL ;ADCL register should be read first
	IN R21, ADCH ;read ADCH after ADCL

	LDI R28, H0 ;load filter coefficient H0
	MOV R22, R21 ;copying the value in R21 to R22
	MUL R28, R22 ;2 Clock cycle Multiplication R1:R0 = R28*R22
	ADD R29, R0 ;adding the values into registers R29 and R30
	ADC R30, R1

	LDI R28, H1 ;load filter coefficient H1
	MUL R28, R23 ;2 Clock cycle Multiplication R1:R0 = R28*R23
	ADD R29, R0
	ADC R30, R1

	LDI R28, H2 ;load filter coefficient H2
	MUL R28, R24 ;2 Clock cycle Multiplication R1:R0 = R28*R24
	ADD R29, R0
	ADC R30 ,R1

	LDI R28, H3
	MUL R28, R25 ;2 Clock cycle Multiplication R1:R0 = R28*R25
	ADD R29, R0
	ADC R30, R1

	LDI R28, H4
	MUL R28, R26 ;2 Clock cycle Multiplication R1:R0 = R28*R26
	ADD R29, R0
	ADC R30, R1

	LDI R28, H5
```

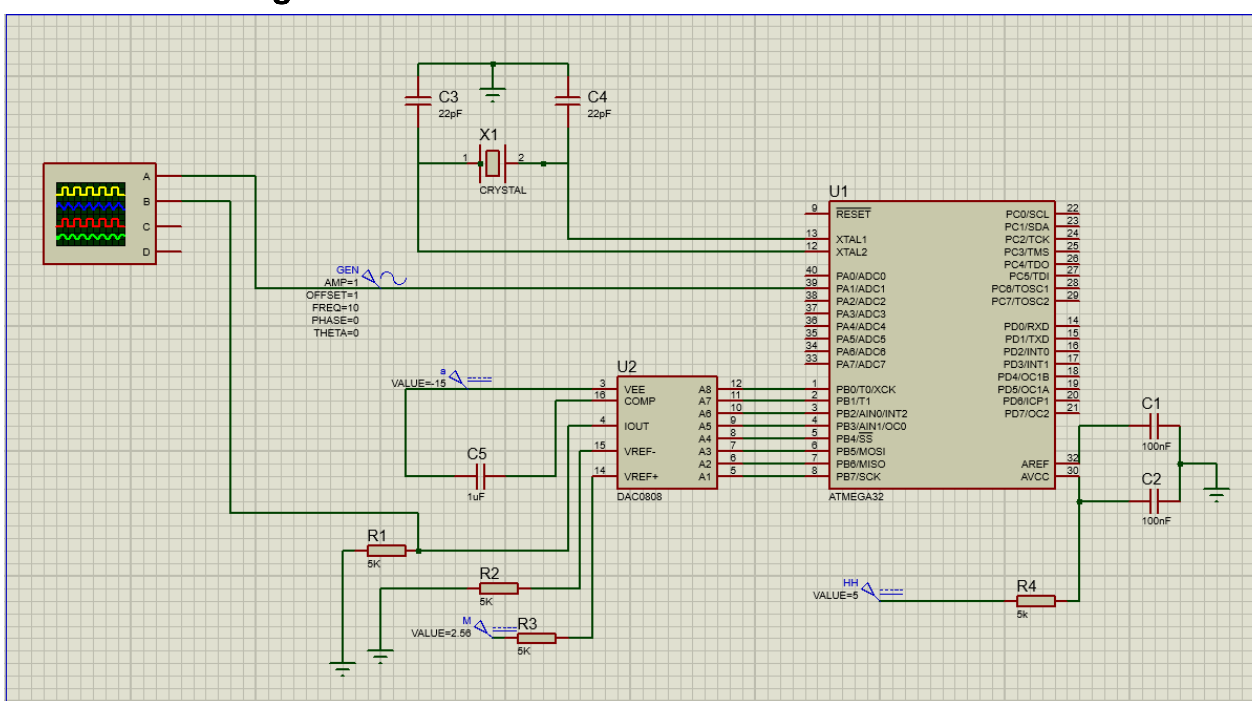
```
MUL R28, R27 ;2 Clock cycle Multiplication R1:R0 = R28*R27
ADD R29, R0
ADC R30, R1

OUT PORTB, R30 ; output the value in the register R30 to PORTB

MOV R27, R26
MOV R26, R25
MOV R25, R24 ; move the values across the registers for storing them for further valuations
MOV R24, R23
MOV R23, R22
LDI R29, 0
LDI R30, 0
RJMP READ_ADC ; jump to READ_ADC to do the same again
```

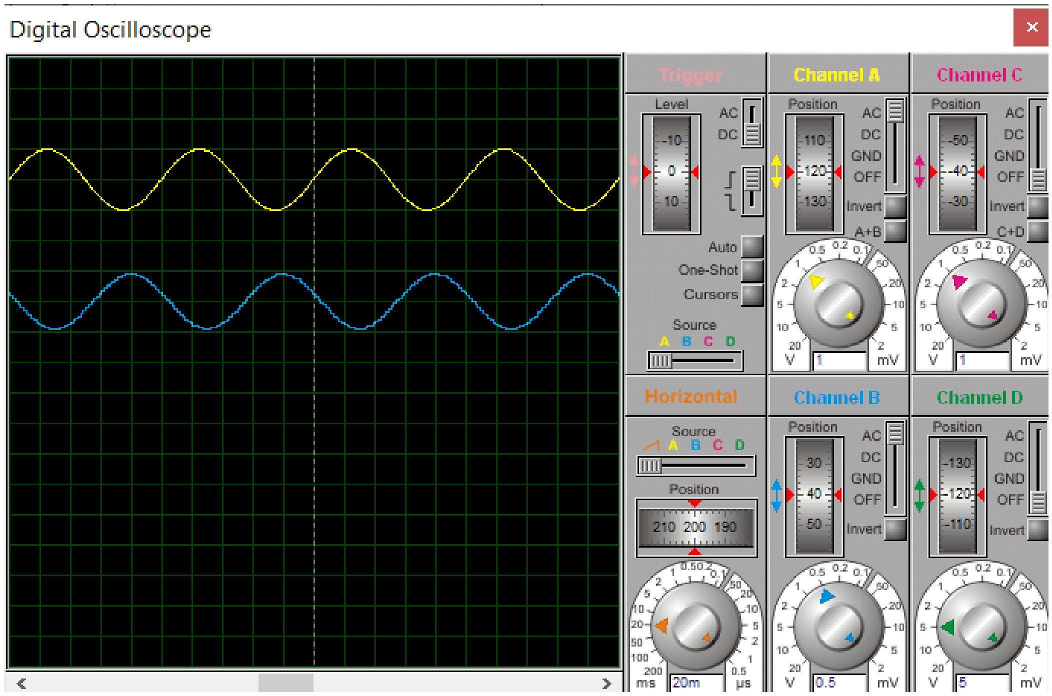
This code will produce the digital signal which when passed through a Digital-to-Analog converter will produce a filtered signal that will omit the high-frequency noise.

5. Schematic diagram in Proteus:

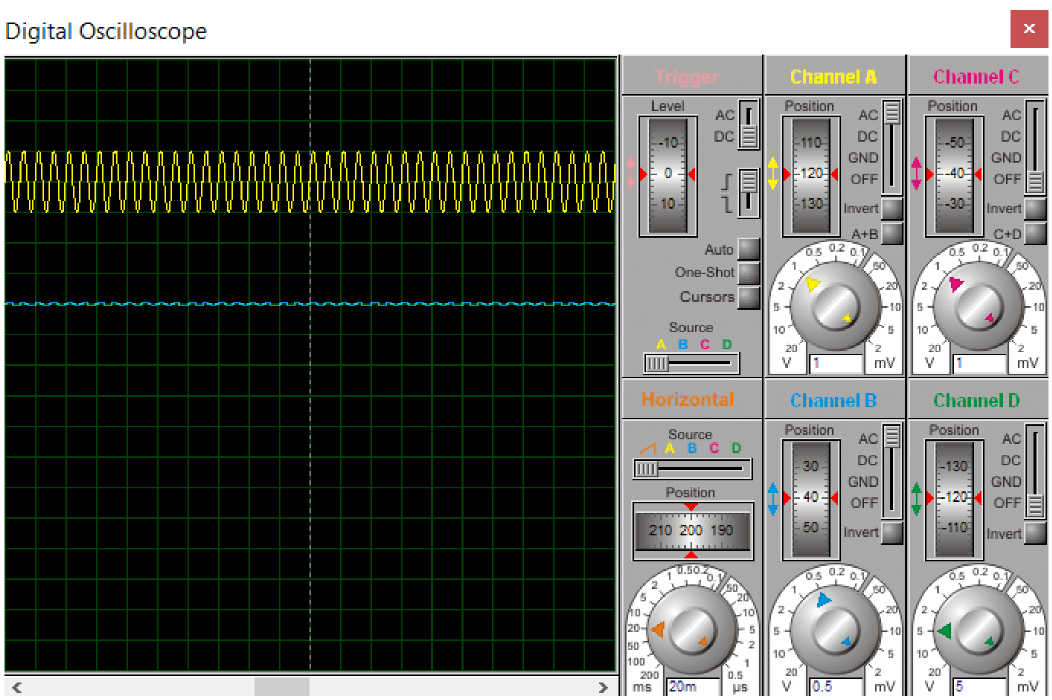


5. Results:

When an input signal(yellow one) of frequency 10 Hz is given:



When an input signal(yellow one) of frequency 100 Hz is given:



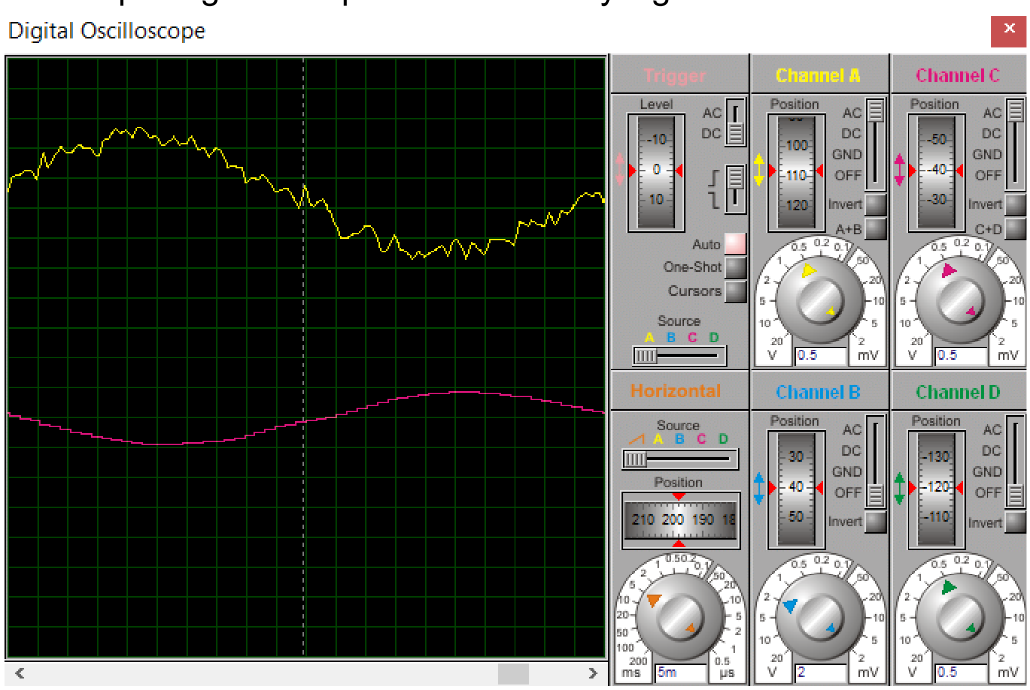
Then we use a noisy signal as the input. We write the input file with the help of the following matlab code:

```
Fs = 1000;
F = 10;
t = 1:100000;
x = 1 + sin(2 * pi * F * t / Fs);
noiseAmp = 0.05;
y = noiseAmp * randn(1, length(t));
x_noise = x + y;

%to visualize both the signals
figure
subplot(2, 1, 1);
plot(t, x, 'b', 'LineWidth', 1);
xlabel('time');
ylabel('sinusoidal signal');
subplot(2, 1, 2);
plot(t, x_noise, 'r', 'LineWidth', 0.1)
xlabel('time');
ylabel('sinusoidal signal with noise');

%creating a file for the noise signal
file=fopen('x_noise.txt', 'w');
for i = 1 : length(x_noise)
    ti = i / Fs;
    fprintf(file, "%d %d\n", ti, x_noise(i));
end
fclose(file);
```

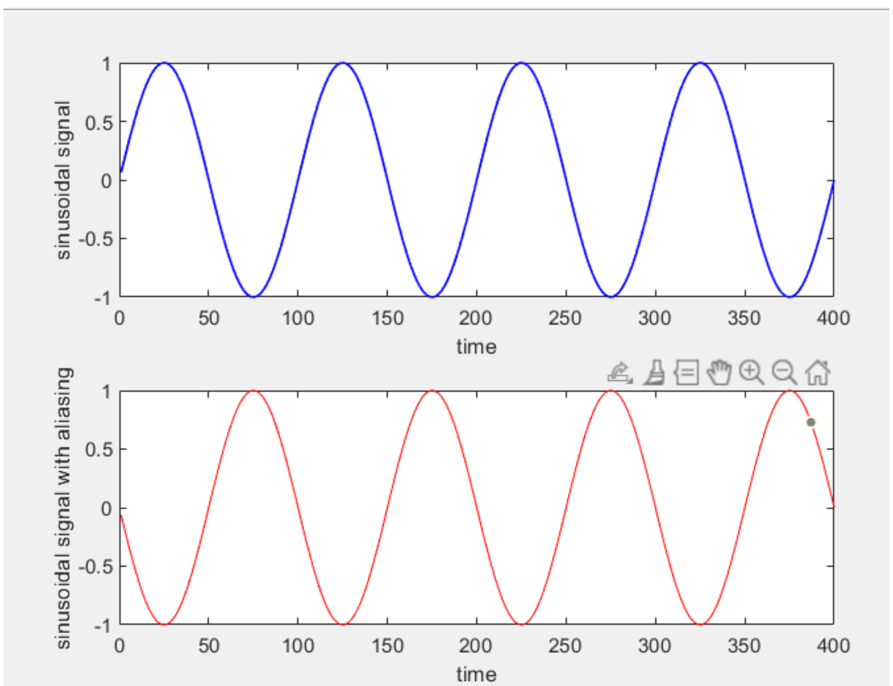
The output signal compared to the noisy signal looks as follows:



7. Visualizing Aliasing:

```
% Input specification
NoP = 1:400;
f1 = 10;
f2 = 990;
f_s = 1000; %sampling frequency
input_sig1 = sin(2 * pi * f1 * NoP / f_s);
input_sig2 = sin(2 * pi * f2 * NoP / f_s);

%Visualizing the two signals
figure
subplot(2, 1, 1);
plot(NoP, input_sig1, 'b', 'LineWidth', 1);
xlabel('time');
ylabel('sinusoidal signal');
subplot(2, 1, 2);
plot(NoP, input_sig2, 'r', 'LineWidth', 0.1)
xlabel('time');
ylabel('sinusoidal signal with aliasing');
```



8. Discussions:

- We have used only 8 bits to generate the filter coefficients. With higher precision and higher sampling frequency, we could have achieved a better-filtered signal with a lesser amount of noise in the output signal. In a similar way we can produce high-pass and band-pass FIR filters but due to higher frequency outputs, we need a higher sampling frequency for more noise-free output.

- If the sampling frequency is comparable with the frequency of the input signal, then the input signal looks like a signal with a much lower frequency. This event is known as aliasing. We can see as the 990 Hz is sampled with a 1000 Hz sampling rate, it almost looks like a 10 Hz signal with a 180° phase shift.
