Name: MANSI UNIYAL
Roll No.: 19EE10039
Experiment 1
Title - LED blinking and Programmable Waveform Generation

Experiment 1a:

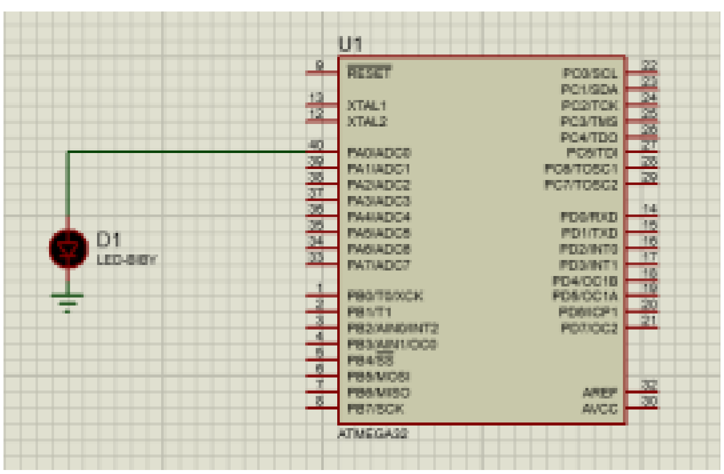
Objective:
To Program ATMEGA32 to produce voltage signal pulse used for the blinking of a LED

Requirements:
1. ATMEGA32
2. LED

Methodology:
The Algorithm and pseudo code are mentioned here:-

Set output at port A
Infinite loop:
    Set port A output to high
    Delay function
    Set port A output to low
    Delay function
Delay function:
    Nested loop
    Return subroutine

Schematic diagram in Proteus:



Code:

```
.INCLUDE "m32DEF.INC"
.ORG 0X00

LDI R16, HIGH(RAMEND)
OUT SPH, R16
LDI R16, LOW(RAMEND)
OUT SPL, R16

LDI R16, 0XFF
OUT DDRA, R16
LOOP:
        LDI R16, 0XFF
        OUT PORTA, R16
        CALL DELAY
        LDI R16, 0X00
        OUT PORTA, R16
        CALL DELAY

        DELAY:
        LDI R16, 0
        LDI R17, 0
        LOOP1:
        DEC R16
        BRNE LOOP1
        DEC R17
        BRNE LOOP1
        RET
        RJMP LOOP
```

Results:

The simulated output as observed in the lab is produced. The LED blinks as long as the simulation runs, due to the infinite loop programmed in the assembly code.

Discussion:

Our aim of the experiment is to program ATMEGA32 for the blinking of the LED, so we must produce a square wave voltage output from ATMEGA32 to produce the desired output. The clock frequency of ATMEGA32 is 1MHz. So, if we use normal instructions for switching, the program will use only a few clock cycles for the blinking of the LED, so we use the extra DELAY function to increase the computing time by nested loops. This makes the blinking of the LED noticeable.

Experiment 1b:

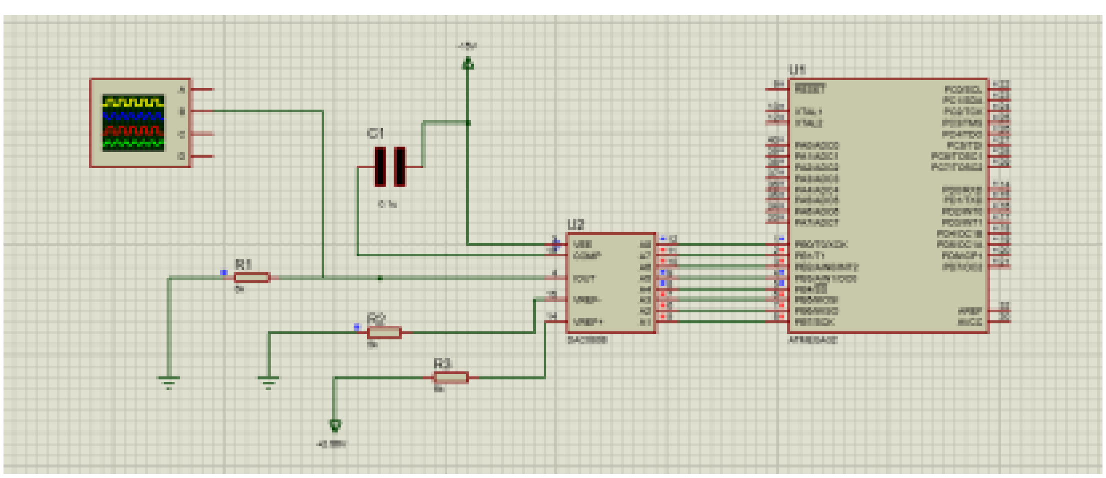
Objective:
To Program ATMEGA32 to generate voltage signal which produces sawtooth waveform when passed through a digital to analog converter.

Requirements:
1. ATMEGA32
2. DAC0808(Digital to analog converter)
3. Resistors - 3 (5kΩ)
4. Capacitor - 1 (0.1μF)
5. Oscilloscope

Methodology:
The Algorithm and pseudo code are mentioned here:-

```
Set output at port B
Infinite loop:
    Set port B output to high(11111111)
    Fall:
        Delay function
        Decrease the output of port B by 1
        Call Fall function till output of port B is low(00000000)
Delay function:
    Nested loop
    Return subroutine
```

Schematic diagram in Proteus:

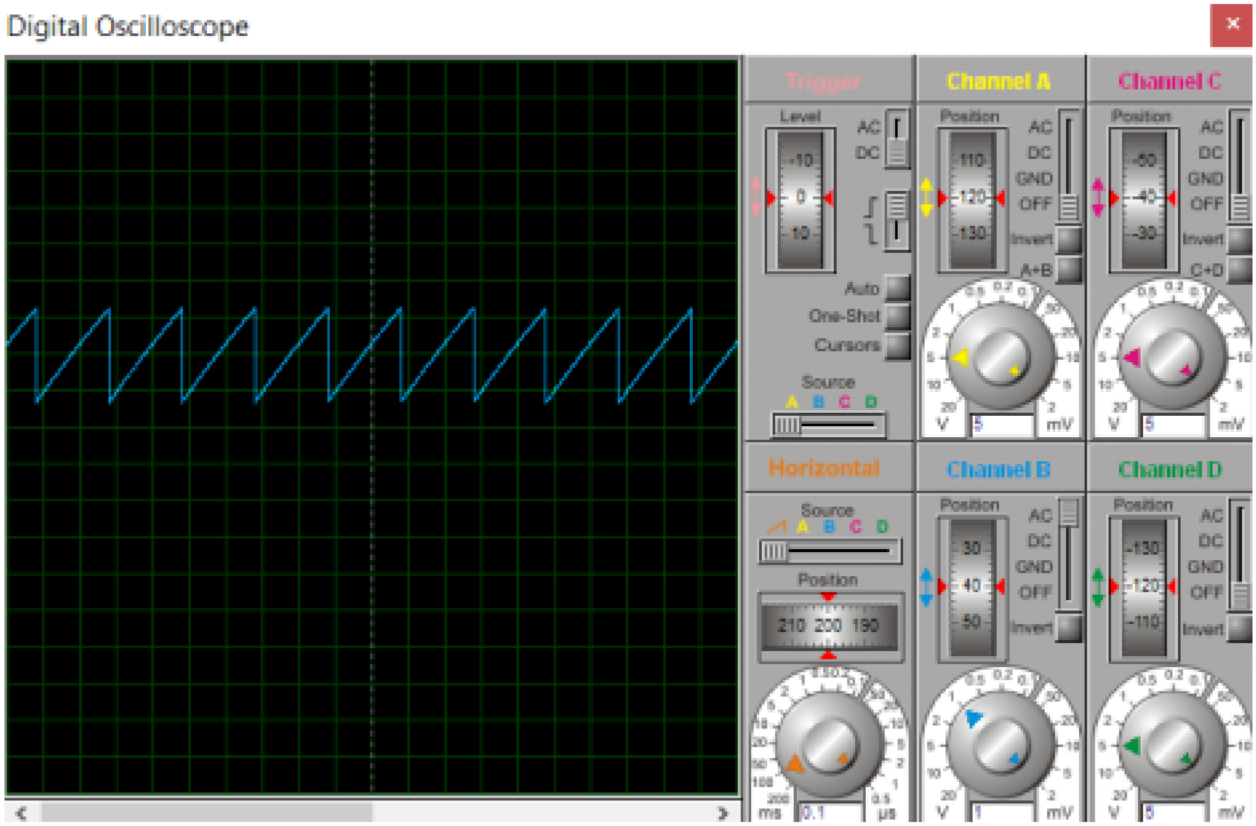


Code:

```
.INCLUDE "m32DEF.INC"
.ORG 0X00

LDI R16, HIGH(RAMEND)
OUT SPH, R16
LDI R16, LOW(RAMEND)
OUT SPL, R16

LDI R16, 0XFF
OUT DDRB, R16

LOOP:
        LDI R16, 0XFF
        FALL:
                OUT PORTB, R16
                CALL DELAY
                DEC R16
                BRNE FALL

DELAY:
        LDI R17, 0
        LOOP1:
        DEC R17
        BRNE LOOP1
        RET
        RJMP LOOP
```

Results:

The simulated output as observed in the lab is produced. The DAC output can be observed in an oscilloscope, which shows a sawtooth wave output.



Discussion:

We give input 0XFF to 0X00 to DAC with a delay. DAC produces minimum -ve voltage output when the input is high(0XFF) which gradually increases and becomes 0V as the input becomes 0X00. Hence the output voltage increases from minimum to maximum output voltage gradually because of the delay introduced in the program. But it jumps back to 0XFF in very few clock cycles as there is no delay involved. Thus a sawtooth wave is generated.

Name: MANSI UNIYAL
Roll No.: 19EE10039
Experiment 2
Title - Digital Low Pass FIR Filter Using ATMEGA32

Objective:
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.

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

Methodology:
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-

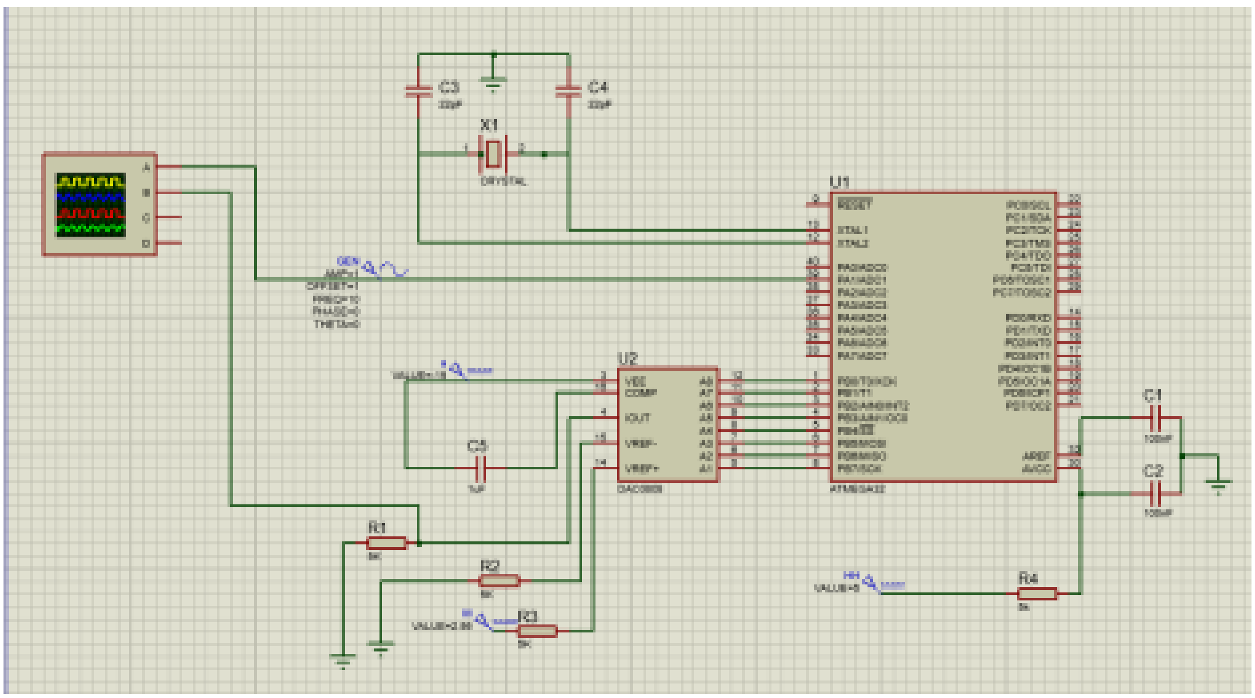
```
function Y = myconv(x,h)
m=length(x);
n=length(h);
for i=1:n+m-1
Y(i)=0;
for j=1:m
if(i-j+1>0)&&(i-j<n)
Y(i)=Y(i)+x(j)*h(i-j+1);
else
end
end
end
end
```

Now we can use the inbuilt filterDesigner of MATLAB to generate the low pass digital filter parameters.
Frequencies used in the design:-
FSampling = 1000Hz

FPass = 10Hz
FStop = 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.

Schematic diagram in Proteus:



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
```

Calculations:

$E(x) = 0$ $\qquad \sigma^2(x) = 1$

$E(x^2) - [E(x)]^2 = 1$ $\qquad E(x) = x\,p(x)$

$\Rightarrow E(x^2) = 1$

$y = ax + b$

$E(y) = E(ax+b)\,p(x) = \Sigma (ax+b)\,p(x)$

$= a \Sigma x\,p(x) + b \Sigma p(x)$

$E(y) = aE(x) + b$

$Var(ax+b) = (E(ax+b)^2) - [E(ax+b)]^2$

$= E(a^2x^2 + 2abx + b^2) - (aE(x)+b)^2$

$= a^2 \Sigma x^2 p(x) + 2ab \Sigma x\,p(x) + \cancel{b^2} - a^2(E(x))^2 \cancel{- b^2}$
$\cancel{- 2ab \Sigma x\,p(x)}$

$= a^2 [E(x^2) - (E(x))^2]$

$= a^2 Var(x)$

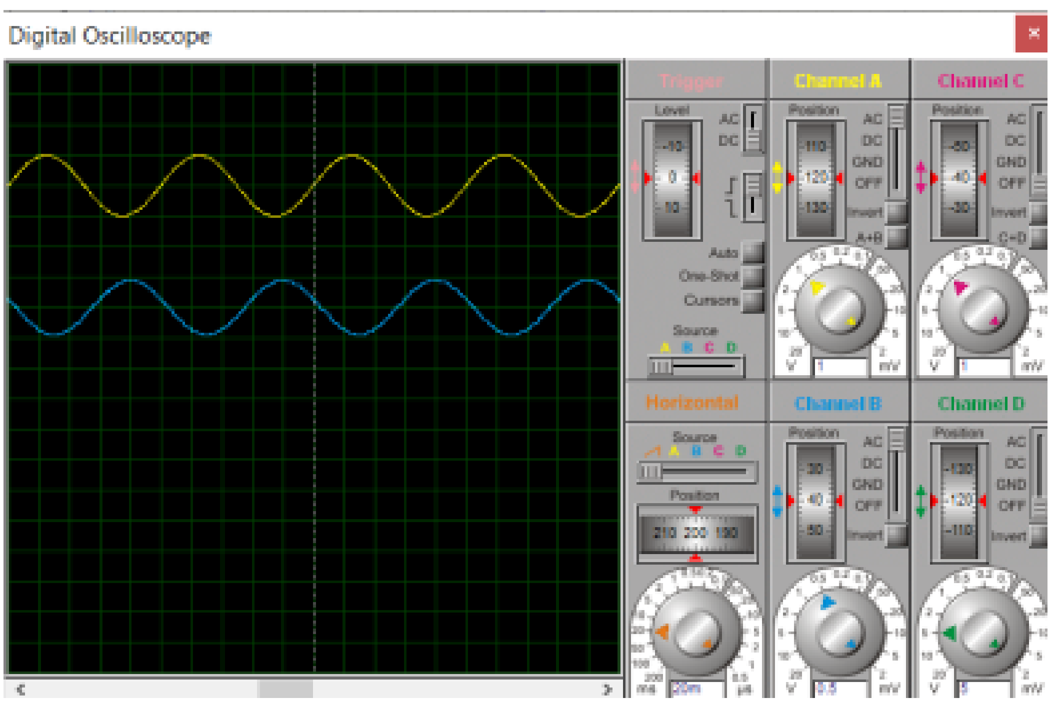
$\Rightarrow E(x) = b$

$Var(y) = a^2$

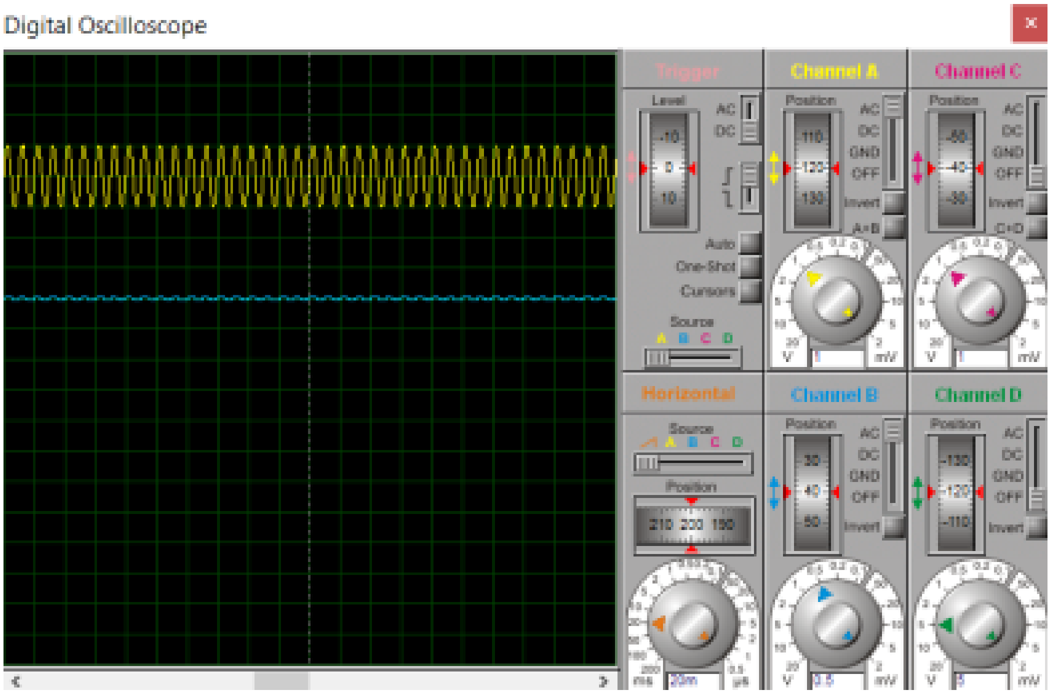
$\therefore s.d(y) = a$

Results:

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



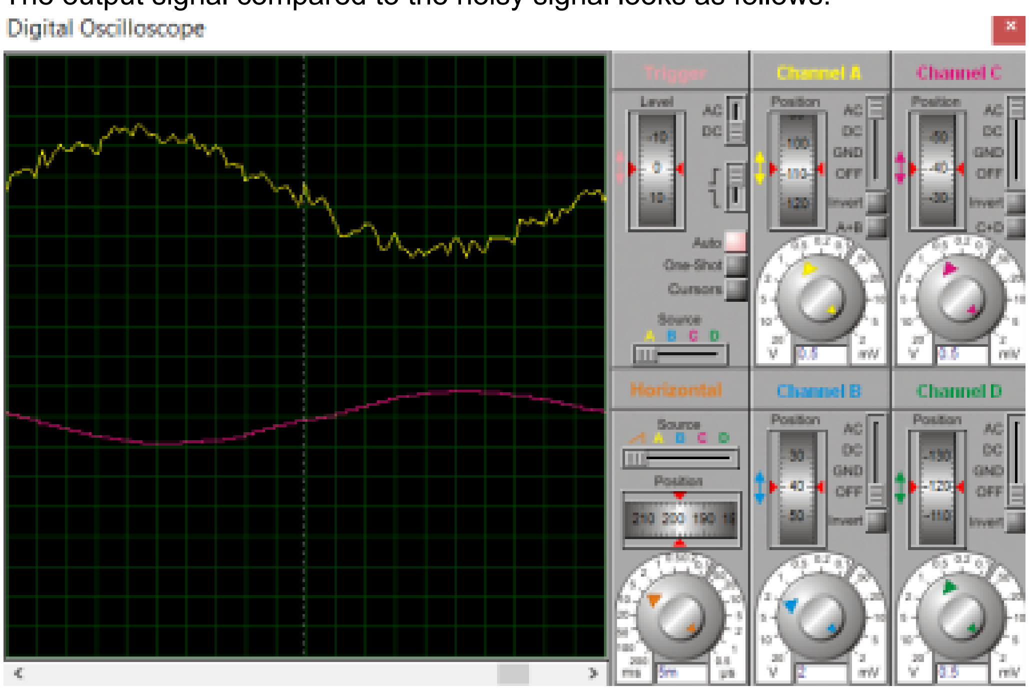
When an input signal(yellow one) of frequency 100 Hz is given:



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

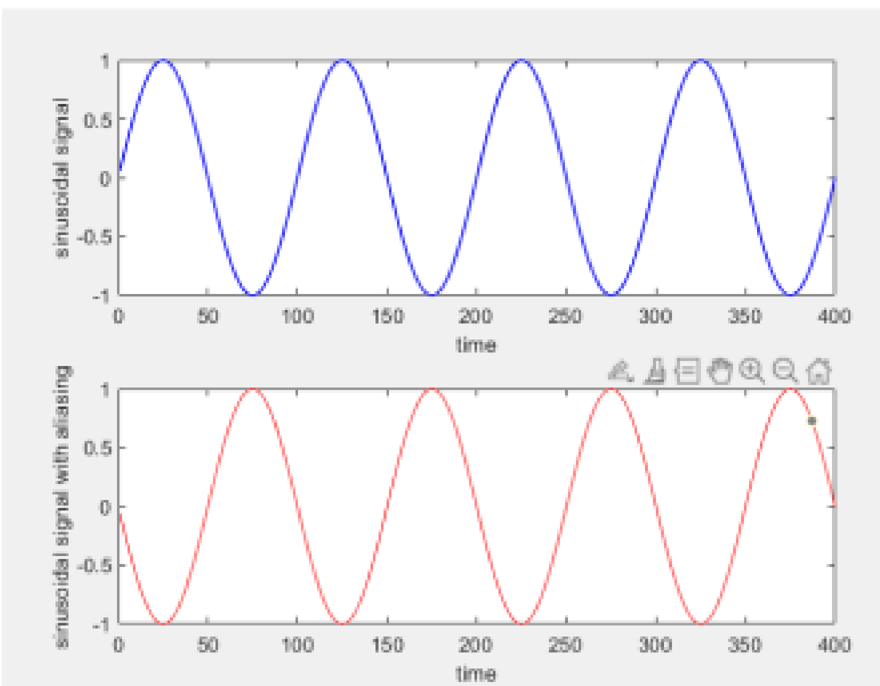
```
Fs = 1000;
F = 10;
t = 1:100000;
x = 1 + sin(2 * pi * F * t / Fs);
noiseAmp = 0.05;
y = noiseAmp * randn(1, length(t));
x_noise = x + y;

%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:



Visualizing Aliasing:

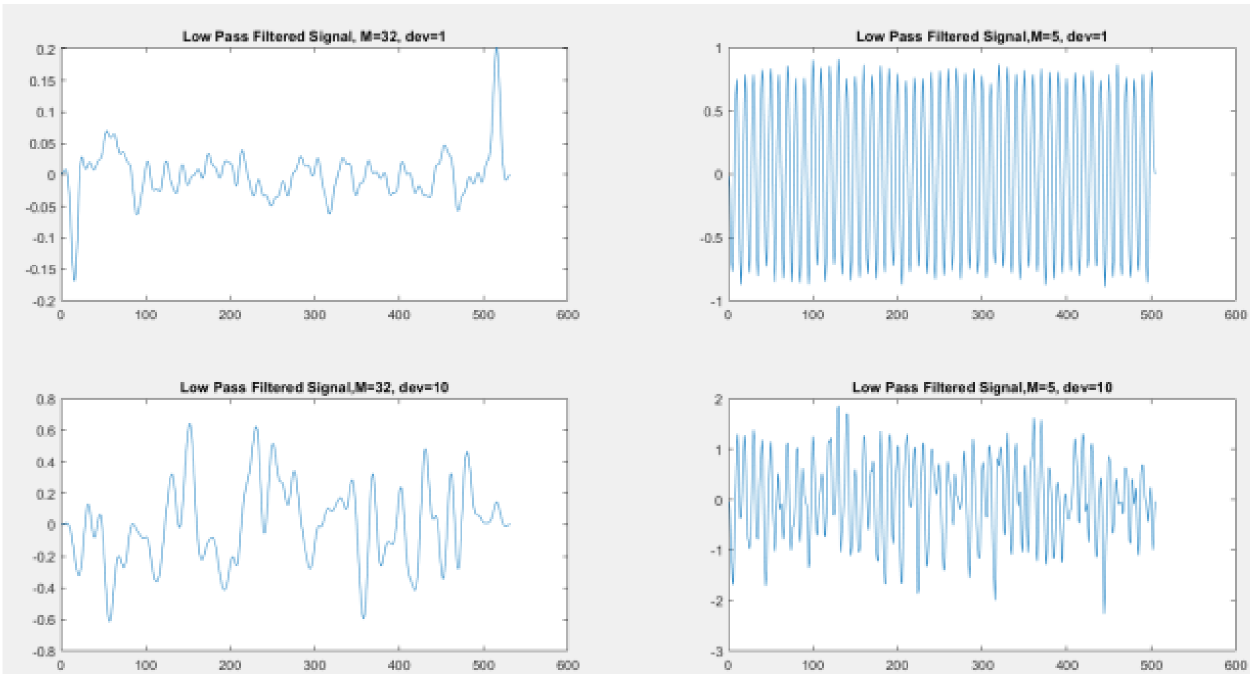
```
% Input specification
NoP = 1:400;
f1 = 10;
f2 = 990;
f_s = 1000; %sampling frequency
input_sig1 = sin(2 * pi * f1 * NoP / f_s);
input_sig2 = sin(2 * pi * f2 * NoP / f_s);

%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');
```



NOISY INPUT SIGNAL

(with Aliasing Effect - Input Frequency f1=1800 Hz)

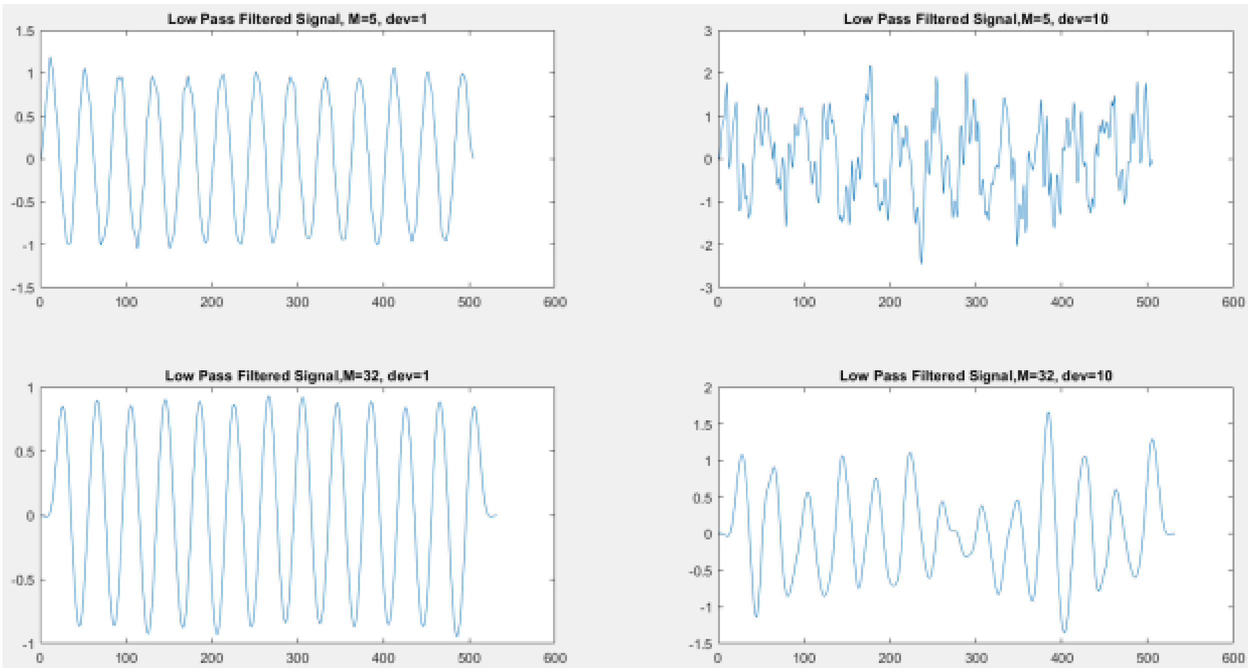
4 different cases of Filter order(m) and Std. Deviation of Guassian noisy signal is considered



NOISY INPUT SIGNAL

(without Aliasing Effect. Input Frequency f1=50 Hz)

The same 4 different cases of Filter order(m) and Std. Deviation of Gaussian noisy signal is considered here too for comparison.



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.

Name: MANSI UNIYAL
Roll No.: 19EE10039
Experiment 3
Title - Digital low pass FIR filter implementation on Arduino Hardware

---

Objective:
To Program ATMEGA328p in Arduino Uno to filter the input signal received in the ADC port of the microcontroller and gives a filtered output which we can observe in the serial plotter as a virtual output.

Requirements:
1. Arduino Uno Board
2. USB A to USB B cable
3. Arduino IDE

Methodology:
Build the required digital filter parameters in MATLAB and use the parameters in assembly code to program the ATMEGA328p microcontroller in Arduino.

Frequencies used in the design:-
FSampling = 2000Hz
Fcutoff = 50Hz

MATLAB code:-

```
%% Filter Specifications
M=32;
fs=2000;
fc=100;
wc=2*fc/fs;
h = fir1(M,wc);
h_fixed=fi(h, 1, 8, 7);
%% Plot filter gain vs frequency response
fvtool(h);
title('Designed filter');
fvtool(h_fixed);
title('Fixed point filter');
%% Input Signal
L=500; f1=50; f2=500;
sig=zeros(L,1);
for i=1:L
sig(i)=sin(2*pi*f1*i/fs)+sin(2*pi*f2*i/fs);
end
```

```
sig_fixed=fi(sig, 1, 8, 7);
%% Do convolution to get output signal
y=conv(sig,h);
y_fixed=conv(sig_fixed,h_fixed);
%% Plot input & output signal
figure(1)
subplot(2,2,1)
plot(sig)
title('Input Signal');
subplot(2,2,2)
plot(sig_fixed)
title('Fixed point Input Signal');
subplot(2,2,3)
plot(y)
title('Low Pass Filtered Signal');
subplot(2,2,4)
plot(y_fixed)
title('Fixed point Low Pass Filtered Signal');
%% Write fixed-point filter coefficients & input signal in hex format file1=fopen('Filter
Co-efficients from MATLAB.txt', 'w'); for i=1:1:length(h_fixed)
hh=h_fixed(i)*2^8;
if i<length(h_fixed)
fprintf(file1, '%d, ', hh);
else
fprintf(file1, '%d', hh);
end
end
fclose(file1);
file2=fopen('Input Signal Data from MATLAB.txt', 'w'); for i=1:length(sig_fixed)
si=sig_fixed(i)*2^8;
if i<length(sig_fixed)
fprintf(file2, '%d, ', si);
else
fprintf(file2, '%d', si);
end
end
fclose(file2);
```

This code generates two files that contain input signal data(50Hz) with noise(500Hz) and the filter parameters. Now we can write the C++ code in Arduino IDE for the ATMEGA328p microcontroller to perform the digital filtering on the provided input signal mixed with high-frequency noise.

Code:

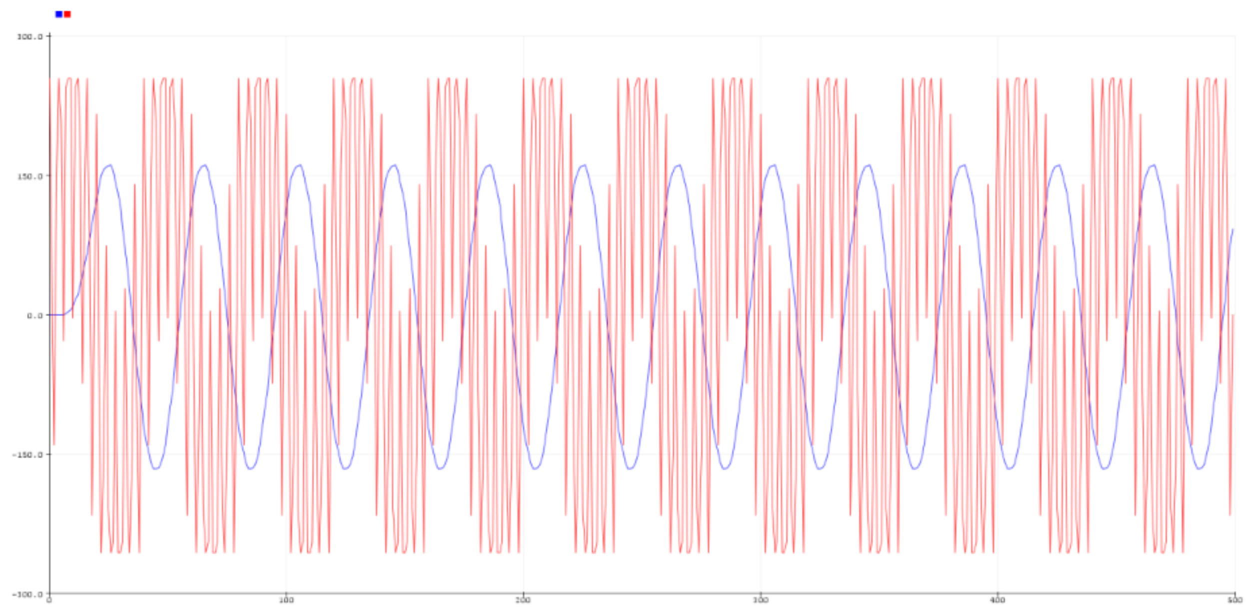
Arduino IDE code:-

```
int m = 500, n = 30;
int x[500] = {254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74,
150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4,
244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4,
-256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208,
-28, 244, 254, 254, -4, 244, 254, 208, -74,
150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4,
244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4,
-256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208,
-28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150,
74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254,
80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216,
0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150,
140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208,
-74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256,
-244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244,
254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208,
-256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140,
150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216,
-80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80,
-256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150,
254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4,
244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4,
-256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208,
-28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150,
74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254,
80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216,
0};
int h[33] = {0, 0, 0, 0, 0, 0, 0, 2, 4, 6, 10, 14, 16, 20, 24, 26, 26, 26, 24, 20, 16, 14, 10, 6,
4, 2, 0, 0, 0, 0, 0, 0, 0}; //lowpass
long int y = 0;
void setup() {
// initialize serial communication with 9600 bits per second:
Serial.begin(115200);
//convolution
for (int i=0; i<m+n; i++){
y = 0;
for (int j=0; j<n; j++){
if ((i-j)>=0 && (i-j) < m)
```

```
y = y + (h[j]*x[i-j])/256;
}
Serial.print(y);
Serial.print(", ");
if (i < m)
Serial.println(x[i]);
delayMicroseconds(500); //0.5ms delay
}
}
void loop() {
// put your main code here, to run repeatedly:
}
```

Results:

The simulated output as observed in the lab is produced. The Arduino output can be observed in the serial plotter, which shows a filtered output signal of 50Hz frequency with very little noise in the output.



In the above diagram, the blue curve is the input signal and the red curve is the 20Hz filtered output signal. We can see that the input signal has both low and high frequencies while the output signal only has the low-frequency component. Also, we can observe a little delay in the input and output signal as well as a little clipping in the peak values of the input signal.
The observation confirms that the low pass filter implemented on the Arduino board is working correctly as only waves of low frequencies are allowed to pass through and appear in the output of the filter.

Discussions:

As we can observe, there is a little clipping in the input signal near the peak values. As we use

8-bit registers to store values, whenever the values of the input signal reach above 255 or below -256 the clipping occurs.
Also from observation, there is a little delay between input and output signal. Whenever a signal is passed through a filter it causes a delay in the filtered output which is termed a group delay. If the delay is constant for all frequencies then we can compensate for the delay with proper programming. Reducing the number of filter parameters reduces accuracy but it reduces the delay too.

Name: MANSI UNIYAL
Roll No.: 19EE10039
Experiment 4
Title - Digital Highpass FIR filter implementation on Arduino Hardware

---

Objective:
To Program ATMEGA328p in Arduino Uno to filter the input signal received in the ADC port of the microcontroller and gives a filtered output which we can observe in the serial plotter as a virtual output.

Requirements:
1. Arduino Uno Board
2. USB A to USB B cable
3. Arduino IDE

Methodology:
Build the required digital filter parameters in MATLAB and use the parameters in assembly code to program the ATMEGA328p microcontroller in Arduino.

MATLAB code:-

```
%Filter Specifications
M=32;
fs=2000;
fc=500; % high pass filter
wc=2*fc/fs;
h = fir1(M,wc, 'high'); % for highpass
h_fixed=fi(h, 1, 8, 7);

%Plot filter gain vs frequency response
fvtool(h);
title('Designed filter');

fvtool(h_fixed);
title('Fixed point filter');

%Input Signal
L=500; f1=50; f2=500;
sig=zeros(L,1);
for i=1:L
   sig(i)=sin(2*pi*f1*i/fs)+sin(2*pi*f2*i/fs);
end
```

```
sig_fixed=fi(sig, 1, 8, 7);

%Do convolution to get output signal
y=conv(sig,h);
y_fixed=conv(sig_fixed,h_fixed);

%Plot input & output signal
figure(1)
subplot(2,2,1)
plot(sig)
title('Input Signal');
subplot(2,2,2)
plot(sig_fixed)
title('Fixed point Input Signal');
subplot(2,2,3)
plot(y)
title('High Pass Filtered Signal');
subplot(2,2,4)
plot(y_fixed)
title('Fixed point High Pass Filtered Signal');

%Write fixed-point filter coefficients & input signal in
hex format file1=fopen('Filter Co-efficients from
MATLAB.txt', 'w'); for i=1:1:length(h_fixed)
    hh=h_fixed(i)*2^8;
    if i<length(h_fixed)
        fprintf(file1, '%d, ', hh);
    else
        fprintf(file1, '%d', hh);
    end
end
fclose(file1);
file2=fopen('Input Signal Data from MATLAB.txt', 'w');
for i=1:length(sig_fixed)
    si=sig_fixed(i)*2^8;
    if i<length(sig_fixed)
        fprintf(file2, '%d, ', si);
    else
        fprintf(file2, '%d', si);
    end
end
fclose(file2);
```

This code generates two files that contain input signal data(50Hz) with noise(500Hz) and the filter parameters. Now we can write the C++ code in Arduino IDE for the ATMEGA328p microcontroller to perform the digital filtering on the provided input signal mixed with high-frequency noise.

Code:

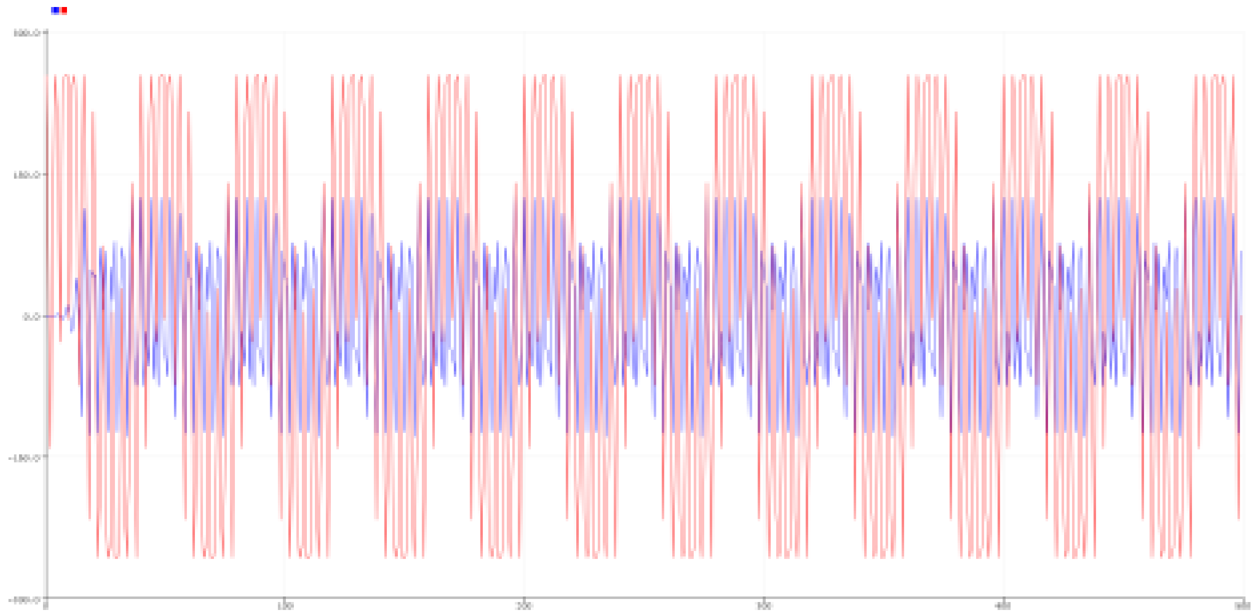
Arduino IDE code:-

```
int m = 500, n = 30;
int x[500] = {254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74,
150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4,
244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4,
-256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208,
-28, 244, 254, 254, -4, 244, 254, 208, -74,
150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4,
244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4,
-256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208,
-28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150,
74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80,
-140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0,
216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140,
-80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74,
150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4,
244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4,
-256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80, -140, 150, 254, 208,
-28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216, -80, -256, -150,
74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80, -256, 0, 254, 80,
-140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0,
216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140,
-80, -256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74,
150, 254, 80, -216, 0, 216, -80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28,
-208, -256, -150, 140, -80, -256, 0, 254, 80, -140,
150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150, 254, 80, -216, 0, 216,
-80, -256, -150, 74, -208, -256, -244, 4, -256, -256, -244, 28, -208, -256, -150, 140, -80,
-256, 0, 254, 80, -140, 150, 254, 208, -28, 244, 254, 254, -4, 244, 254, 208, -74, 150,
254, 80, -216, 0};

int h[33] = {0, 0, 0, 0, 0, 2, 0, -4, 0, 8, 0, -12, 0, 26, 0, -80, 128, -80, 0, 26, 0, -12, 0, 8, 0,
-4, 0, 2, 0, 0, 0, 0, 0}; //highpass
```

```
long int y = 0;

void setup() {
  // initialize serial communication with 9600 bits per second:
  Serial.begin(115200);

  //convolution
  for (int i=0; i<m+n; i++){
    y = 0;
    for (int j=0; j<n; j++){
      if ((i-j)>=0 && (i-j) < m)
        y = y + (h[j]*x[i-j])/256;
    }
    Serial.print(y);
    Serial.print(", ");
    if (i < m)
      Serial.println(x[i]);
    delayMicroseconds(500); //0.5ms delay
    }
}

void loop() {

}
```

Results:

The simulated output as observed in the lab is produced. The Arduino output can be observed in the serial plotter, which shows a filtered output signal of 500Hz frequency with very little noise in the output.



Discussions:

In the case of the High-pass filter, of the two signals(50 Hz and 500 Hz) the 50 Hz signal is blocked. The cutoff frequency of the filter being 500Hz, the 500Hz signal is allowed to pass

Name: MANSI UNIYAL
Roll No.: 19EE10039
Experiment 5
Title - Low Pass Filter implementation through Interrupt and pooling implementation on Arduino Hardware

---

Objective:
To Program ATMEGA328p in Arduino Uno to filter the input signal received in the ADC port of the microcontroller and give a filtered output using interrupt and pooling filtering which we can observe in the serial plotter as a virtual output.

Requirements:
1. Arduino Uno Board
2. USB A to USB B cable
3. Arduino IDE

Methodology:
Build the required assembly code to program the ATMEGA328p microcontroller in Arduino. Write the code in Arduino IDE and connect the Arduino Board and observe the signal

Code:
Interrupt filterring:
```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

volatile int analogVal;
const byte adc_pin = A0;
int n=33;
int x[33]={0};
//filter coefficients
int h[33] ={0, 0, 0, 0, 0, 0, 0, 2, 4, 6, 10, 14, 16, 20, 24, 26, 26, 26, 24, 20, 16, 14, 10, 6,
4, 2, 0, 0, 0, 0, 0, 0, 0};
//current filter output sample value
float yi;
//filtered output sample value
float yv;
//current input sample value
float xv;

void setup(){
  Serial.begin(9600);
  ADCSRA = 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0;
  ADCSRA |= 1<<ADEN;
```

```
  ADMUX |= 1<<ADLAR;
  ADMUX |= 1<<REFS0;
  ADMUX |= ((adc_pin -14) & 0x07);
}

void loop(){
  ADCSRA |= 1<<ADSC;
  while((ADCSRA & (1<<ADIF)) == 0);
  analogVal = ADCH;
  for(int i=0;i<n-1;i++){
    x[i] = x[i+1];
  }
  x[n-1] = analogVal;

  yi = 0;
  for(int j=0;j<n;j++){                    //convolution
    yi+=(h[j]*x[n-1-j]*1.0)/270;
  }

  yv = 5*((yi*1.0)/255);
  xv = 5*((x[n-1]*1.0)/255);
  Serial.print(xv);
  Serial.print(",");
  Serial.println(yv);
}
```

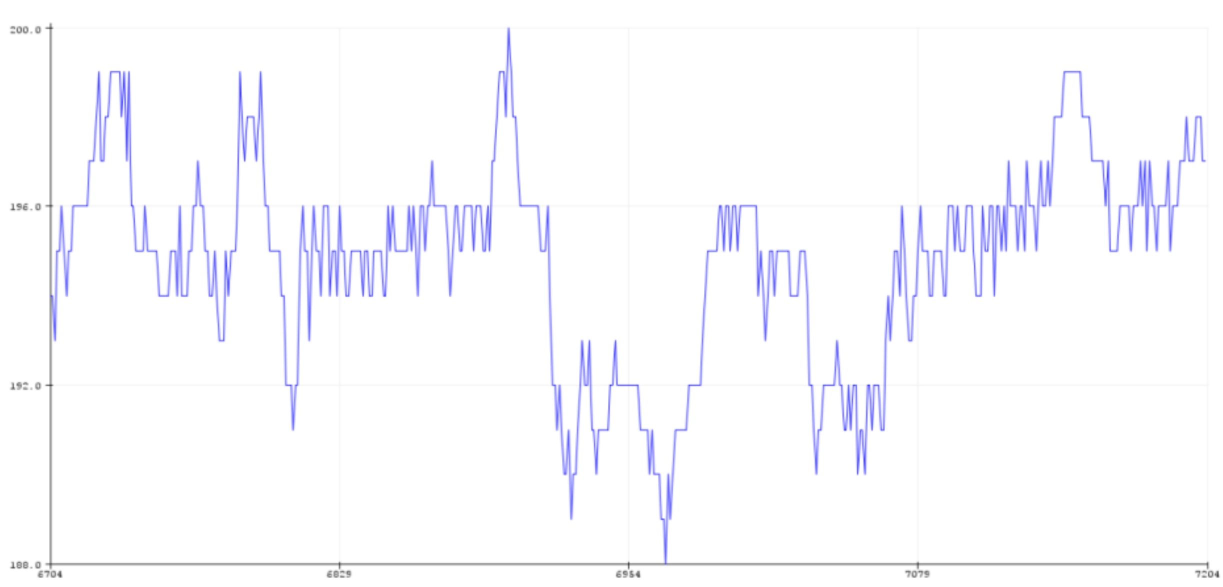
Pooling filtering:

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

volatile int analogVal;
const byte adc_pin = A0;                    //A0 is to be used as input pin
int n=33;
int x[33]={0};
int h[33] ={0, 0, 0, 0, 0, 0, 0, 2, 4, 6, 10, 14, 16, 20, 24, 26, 26, 26, 24, 20, 16, 14, 10, 6,
4, 2, 0, 0, 0, 0, 0, 0, 0};
float yi;          //variable to store the current filter output sample value
float yv;          //variable to store the filtered output sample value in volts
float xv;          //variable to store the current input sample value in volts

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);
```

```
  ADCSRA = 1<<ADPS2 | 1<<ADPS1 | 1<<ADPS0;        //set clock prescaler to 128
  ADCSRA |= 1<<ADEN;                              //enable ADC
  ADMUX |= 1<<ADLAR;                              //setting output to be left adjusted
  ADMUX |= 1<<REFS0;                              //Vref set to VCC value
  ADMUX |= ((adc_pin -14) & 0x07);                //sending arduino pin value to ADC

}

void loop()
{
  ADCSRA |= 1<<ADSC;                              //start conversion
  while((ADCSRA & (1<<ADIF)) == 0);               //wait for conversion to complete
  analogVal = ADCH;                               //obtain value after conversion
 //shifting the x array to store the latest 33 input samples
  for(int i=0;i<n-1;i++)
  {
    x[i] = x[i+1];
    delay(0.01);
  }
  x[n-1] = analogVal;
  yi = 0;
  //performing convolution based on the formula y(i) = (sum for j = 0 to n-1) x(n-1-j)*h(j)
and scaling appropriately
  for(int j=0;j<n;j++)
  {
    yi+=(h[j]*x[n-1-j]*1.0)/270;
  }
  yv = 5*((yi*1.0)/255);//converting input and output sample values to volts
  xv = 5*((x[n-1]*1.0)/255);
  Serial.print(xv); //printing input and output samples to serial monitor
  Serial.print(",");
  Serial.println(yv);

}
```

Results:

The simulated output as observed in the lab is produced. The Arduino output can be observed in the serial plotter, which shows a filtered output signal of 50Hz frequency with very little noise in the output.



Discussions:

The real-time signal can be produced by swirling the open terminal of the Male-to-Female jumper wire where the generated signal is basically noise. We can also generate real-time signals using a potentiometer.

The interrupt method is beneficial over the polling-based method because in the interrupt method the processor can execute its code without checking for the completion flag in regular intervals. When the filtering is complete, the interrupt signal is sent to the CPU and the CPU executes the necessary ISR while storing the previous PC location at the stack. Thus computational time is saved output pin to D5, +ve terminal to 3V, and -ve pin to ground

Name: MANSI UNIYAL
Roll No.: 19EE10039
Experiment 6
Title - Sending Temperature and Humidity sensing data to a web server using ESP8266

Objective:
To send temperature and humidity data (sensed by DHT22) to the ThingSpeak web server using ESP8266.

Requirements:
1. DHT22 temperature sensor
2. ESP8266
3. OTG cable
4. Arduino IDE
5. ThingSpeak web server

Methodology:
We connect the DHT22 to the ESP8266 board with the output pin to D5, +ve terminal to 3V, and -ve pin to ground.
We download all the necessary libraries and boards in Arduino IDE. Then we upload the following code to the ESP8266 board using Arduino IDE:

Code:
Arduino IDE code:
```
#include <DHT.h>
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>

const char * apiKey = "U451HU8PF4VCUJ57";
unsigned long Channel_ID = 1685254;
const char *ssid = "moto g(6) plus 1137";
const char *pass = "laluprasad";
const char* server = "api.thingspeak.com";
#define DHTPIN D5 //pin where the dht22 is

connected DHT dht(DHTPIN,

DHT22);WiFiClient client;

void setup()
{
```

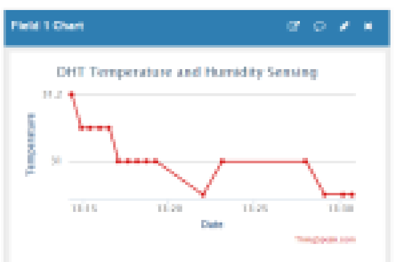
```
 Serial.begin(115200);
 ThingSpeak.begin(client);
 delay(10);
 dht.begin();
 Serial.println("Connecting to ");
 Serial.println(ssid);
 WiFi.begin(ssid, pass);
 while (WiFi.status() != WL_CONNECTED)
 {
   delay(500);
   Serial.print(".");
 }
 Serial.println("");
 Serial.println("WiFi connected");
}
void loop()
{
 float h = dht.readHumidity();
 float t = dht.readTemperature();
 if (isnan(h) || isnan(t))
 {
   Serial.println("Failed to read from DHT
   sensor!"); delay(1000);
   return;
 }
 Serial.print(F("Humidity: "));
 Serial.print(h);
 Serial.print(F("% Temperature: "));
 Serial.print(t);
 Serial.print(F("°C "));
 ThingSpeak.writeField(Channel_ID, 1, String(t),
 apiKey); delay(15000);
 ThingSpeak.writeField(Channel_ID, 2, String(h),
 apiKey); delay(15000);
 Serial.println("Waiting...");
}
```

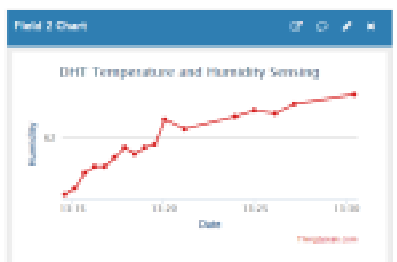
Results:

The serial monitor shows the readings after every 15s.

```
moto g(6) plus 1137
..................
WiFi connected
Humidity: 60.20% Temperature: 31.20°C Waiting...
Humidity: 60.40% Temperature: 31.10°C Waiting...
Humidity: 60.90% Temperature: 31.10°C Waiting...
Humidity: 61.10% Temperature: 31.10°C Waiting...
Humidity: 61.10% Temperature: 31.10°C Waiting...
Humidity: 61.40% Temperature: 31.00°C Waiting...
Humidity: 61.70% Temperature: 31.00°C Waiting...
Humidity: 61.50% Temperature: 31.00°C Waiting...
Humidity: 61.70% Temperature: 31.00°C Waiting...
Humidity: 61.80% Temperature: 31.00°C Waiting...
Humidity: 62.60% Temperature: 31.00°C Waiting...
Humidity: 62.00% Temperature: 31.00°C Waiting...
Humidity: 62.30% Temperature: 31.00°C Waiting...
Humidity: 62.80% Temperature: 31.00°C Waiting...
Humidity: 62.30% Temperature: 30.90°C Waiting...
Humidity: 62.50% Temperature: 31.00°C Waiting...
Humidity: 62.40% Temperature: 31.00°C Waiting...
Humidity: 62.70% Temperature: 31.00°C Waiting...
Humidity: 62.80% Temperature: 31.00°C Waiting...
Humidity: 62.90% Temperature: 31.00°C Waiting...
Humidity: 62.80% Temperature: 31.00°C Waiting...
Humidity: 62.80% Temperature: 31.00°C Waiting...
Humidity: 63.00% Temperature: 30.90°C Waiting...
Humidity: 63.10% Temperature: 30.90°C Waiting...
Humidity: 62.90% Temperature: 30.90°C Waiting...
Humidity: 63.00% Temperature: 31.00°C Waiting...
Humidity: 63.20% Temperature: 30.90°C Waiting...
Humidity: 63.30% Temperature: 30.90°C Waiting...
Humidity: 63.30% Temperature: 31.00°C Waiting...
Humidity: 63.40% Temperature: 30.90°C
```

In the ThingSpeak server, our specified channels will show





Discussions:

ThingSpeak only allows data transfer at a minimum interval of 15s. So, we need to introduce a delay of at least 15s between two consecutive readings.

The DHT22 connection should be made very carefully(output pin to D5, +ve terminal to 3V, and -ve pin to ground). Otherwise, it gives NaN results.

Sometimes, despite connecting the OTG cable, the port isn't available. We need to update the driver to solve this problem.