EXPERIMENT - 3 ADC DAC INTERFACE and Digital Filter Implementation on 8-Bit Controllers

1. **OBJECTIVE :** Complete the ADC-DAC Interface by programming ATMEGA32, on Proteus Simulator. Also 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. **METHODOLOGY :**

The algorithm for ADC-DAC Interface is as follows.

We define Port-A as input port and Port-B as output port.

Next we need to enable the ADC, set the ADC clock conversion speed= ck/128 using ADCSRA register.

Use the ADMUX register, to select left adjustment, put Vref=2.56V, and set ADC1 as input channel.

Keep the polling till the ADIF bit is equal to 1.

Read ADCL and ADCH to the resisters then to the output port.

Algorithm for Digital Filter Implementation

Define Port-A as input port and Port-B as output port.

Follow the ADC-DAC interfacing algorithm from above.

Next we multiply the present value with the coefficient, H0 and store the result in two GPR's.

Continue to do this for past five values and keep adding the result.

Output is obtained at PORTB.

Shift the x[n] value to x[n-1] GPR and x[n-1] value to x[n-2] GPR and x[n-2] to x[n-3] GPR.

10)

Set Multiplicative result Registers to zero before going to the next sample, then proceed to the next input sample

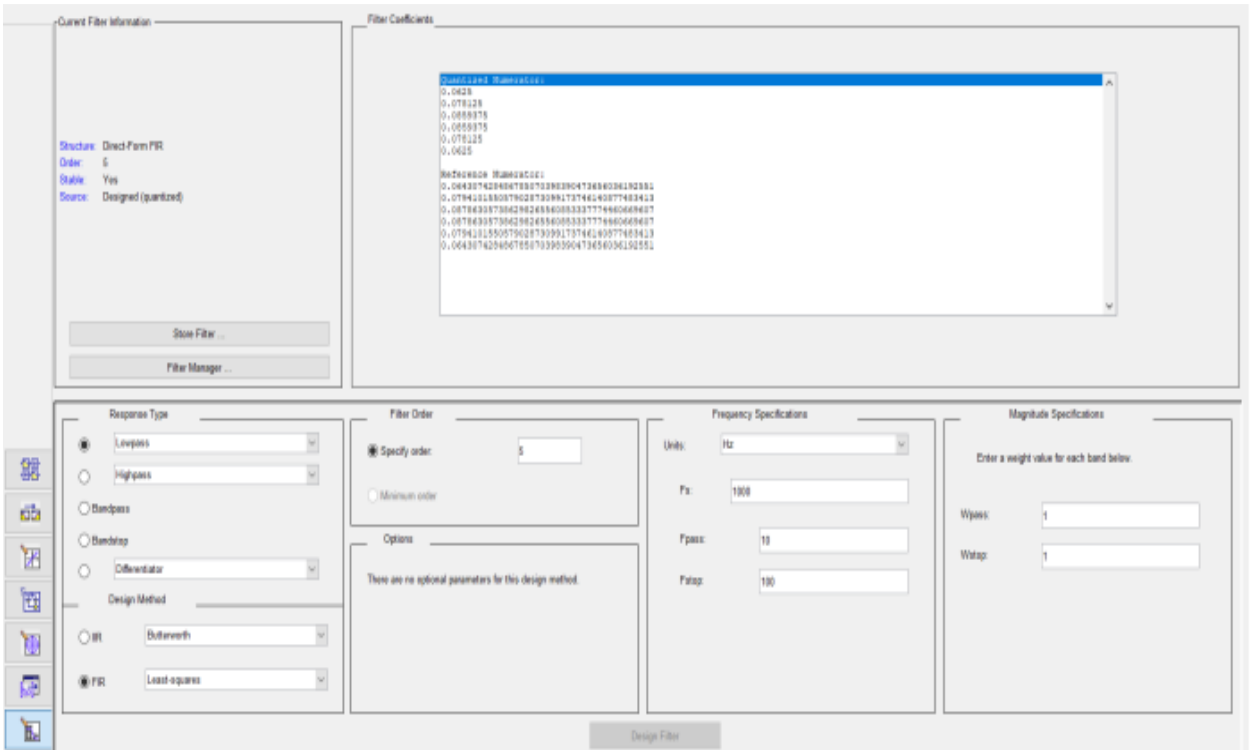
DESIGN THE FILTER ON MATLAB

Convolution Code

```
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
```

Then we use the fiterDesigner tool inbuilt inside Matlab to generate the filter coefficients for low pass digital Filter.

FSampling = 1000Hz FPass = 10Hz FStop = 100Hz



The filter coeffieints obtained here are converted to hexadecimal s.t. We can use them in 8-bit registers of ATMEGA32. Filter coefficients obtained are : 10, 14, 16, 16, 14, 10 (base 16).

3. ASSEMBLY LANGUAGE CODES :

ADC-DAC Interfacing Code

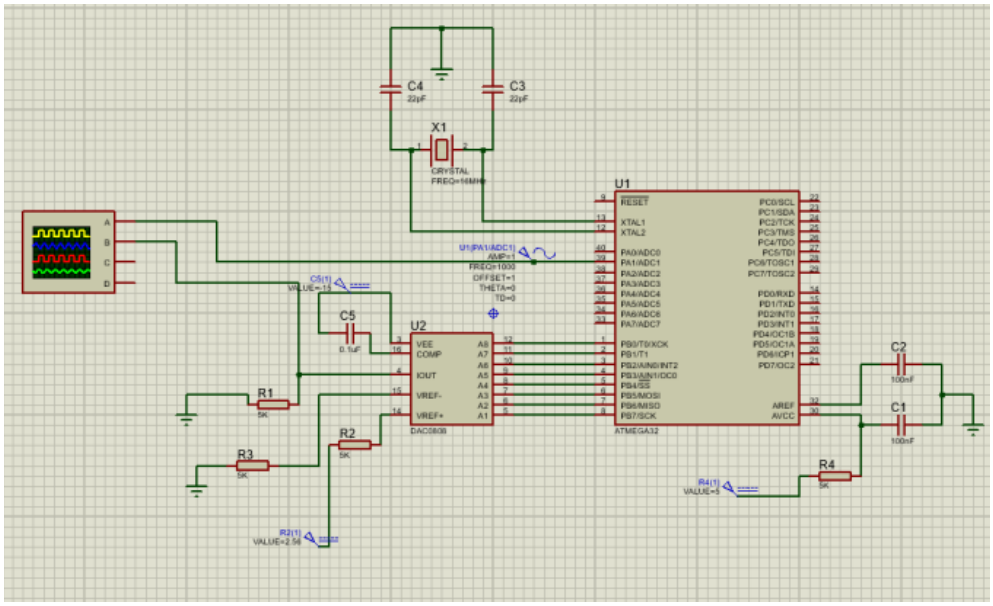
```
;
; AssemblerApplication4.asm
;
; Created: 18-01-2022 12:10:10
; Author : ARUNDHUTI
;


; Replace with your application code
.INCLUDE "M32DEF.INC" ;ADD Atmega32 definition
.ORG 00;Origin at 0x00
;-----------------------------------------------------------------------------------------------
LDI R16, 0x00 ; define portA input
OUT DDRA, R16
LDI R16, 0xFF ;define portb OUTput for DAC
OUT DDRB, R16
LDI R16, 0x87
OUT ADCSRA, R16 ; enable ADC, ADC clock = ck/128
LDI R16, 0xE1
OUT ADMUX, R16 ; ADC1, Left adjustment result, Vref = 2.56V and ADC1 select
READ_ADC: NOP
SBI ADCSRA, ADSC ; Start ADC Conversion
KEEP_POLLING: NOP ; Wait the end of conversion
SBIS ADCSRA, ADIF ; Is it end of conversion yet?
RJMP KEEP_POLLING ; Keep polling until END of 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
;-----------------------------------------------------------------------------------------------
OUT PORTB, R21 ;Put result at DAC input port which is connected to PORTB IN this case
;The whole result is in R21(Most significant Byte) and R20(Least significant Byte)
;-----------------------------------------------------------------------------------------------
RJMP READ_ADC
```

DIGITAL FILTER IMPLEMENTATION CODE

```
;
; AssemblerApplication3.asm
;
; Created: 17-01-2022 10:23:44
; Author : ARUNDHUTI
;


; Replace with your application code
.INCLUDE "M32DEF.INC" ;ADD Atmega32 definition
.ORG 00 ;Origin at 0x00
.EQU H0 = 0x10 ;filter coefficient from matlab fda tool
.EQU H1 = 0x14 ;original coeff multiplied by 256
.EQU H2 = 0x16
.EQU H3 = 0x16
.EQU H4 = 0x14
.EQU H5 = 0x10

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

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

LDI R16, 0x00 ;define PORTA input
OUT DDRA, R16
LDI R16, 0xFF ;define PORTB OUTput for DAC
OUT DDRB, R16

LDI R16, 0x87
OUT ADCSRA, R16 ;enable ADC, ADC clock = ck/128
LDI R16, 0xE1
OUT ADMUX, R16 ;ADC1, Left adjustment result, Vref =2.56V and ADC1 select

READ_ADC: NOP
SBI ADCSRA, ADSC ; Start ADC Conversion
```

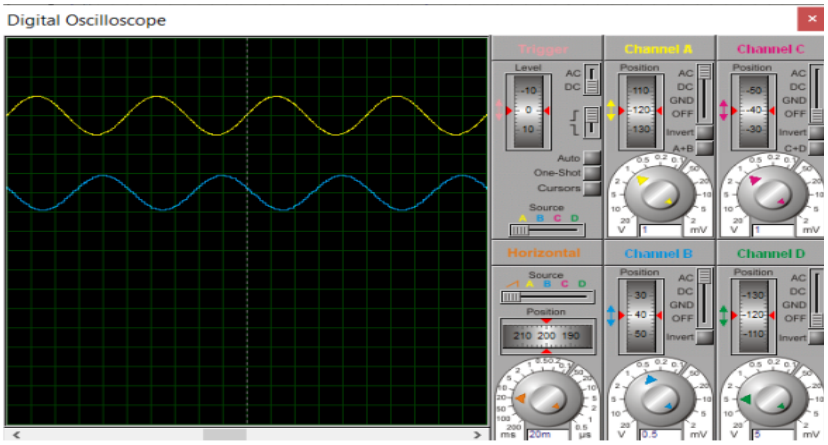
```
KEEP_POLLING: NOP ; No operation instruction just consumes a clock cycle without doing anything
    SBIS ADCSRA, ADIF ; If it is the end of conversion skip the next instruction and come out of the loop
    RJMP KEEP_POLLING ; Keep polling until END of conversion
    SBI ADCSRA, ADIF ; write 1 to 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
    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*R25
    ADD R29, R0
    ADC R30, R1

    LDI R28, H5
    MUL R28, R27 ; 2 Clock cycle Multiplication R1:R0 = R28*R25
    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 the 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
```

4. PROTEUS SIMULATION DIAGRAM

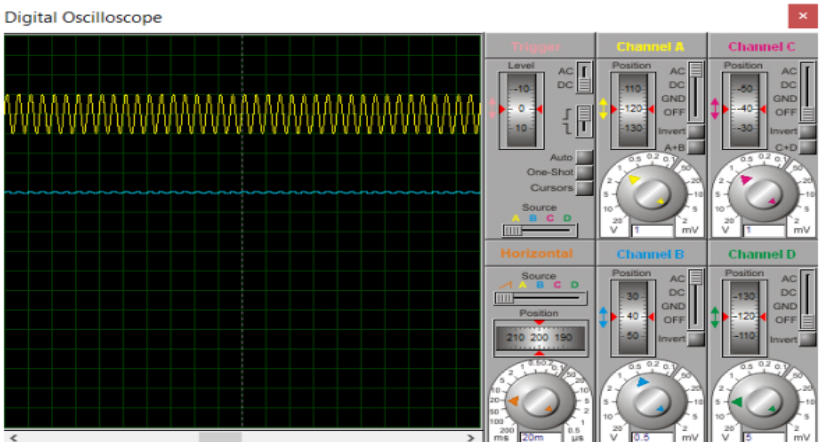


5. RESULTS :

Input Signal Frequency = 10 Hz (Channel A Yellow Signal). Output: Channel B, Blue signal



Input Signal Frequency = 100 Hz (Channel A Yellow Signal)



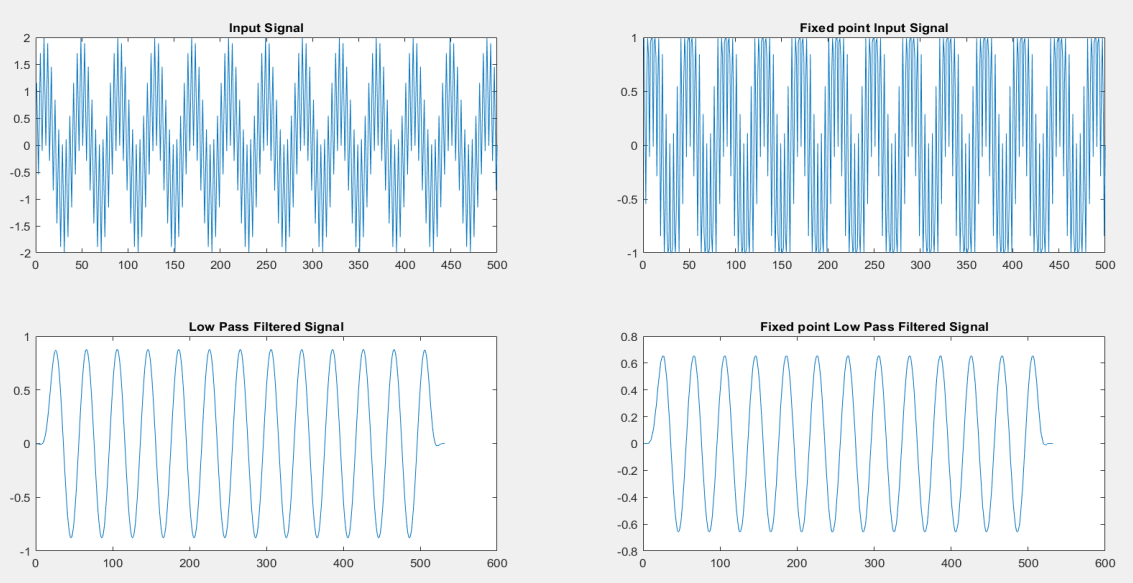
SUPERPOSITION OF INPUT SIGNAL (f1=50 Hz f2=500 Hz**)**

FSampling = 2000Hz Fcut-off = 100Hz (in all the subsequent cases, this is kept constant.)

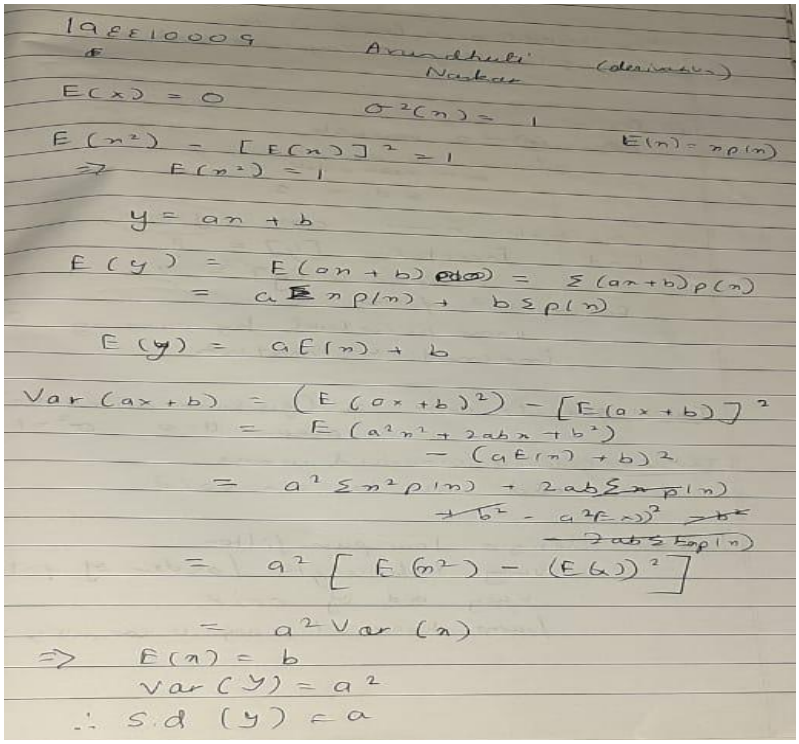
Matlab Code

```
16
17    %% Input Signal
18 -  L=500; f1=50; f2=500;
19 -  sig=zeros(L,1);
20 -  for i=1:L
21 -      sig(i)=sin(2*pi*f1*i/fs)+sin(2*pi*f2*i/fs);
22 -  end
23 -  sig_fixed=fi(sig, 1, 8, 7);
24    %% Do convolution to get output signal
25 -  y=conv(sig,h);
26 -  y_fixed=conv(sig_fixed,h_fixed);
```

Graphs



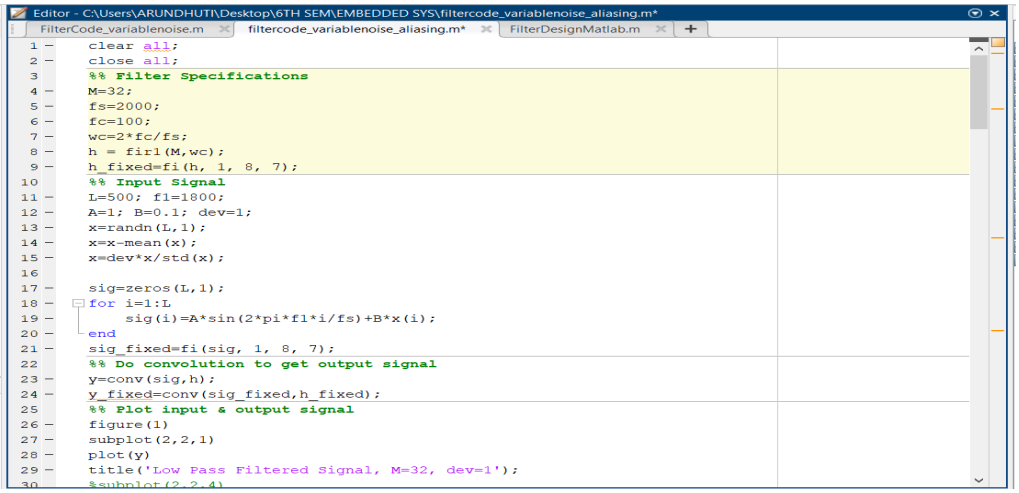
CALCULATIONS :



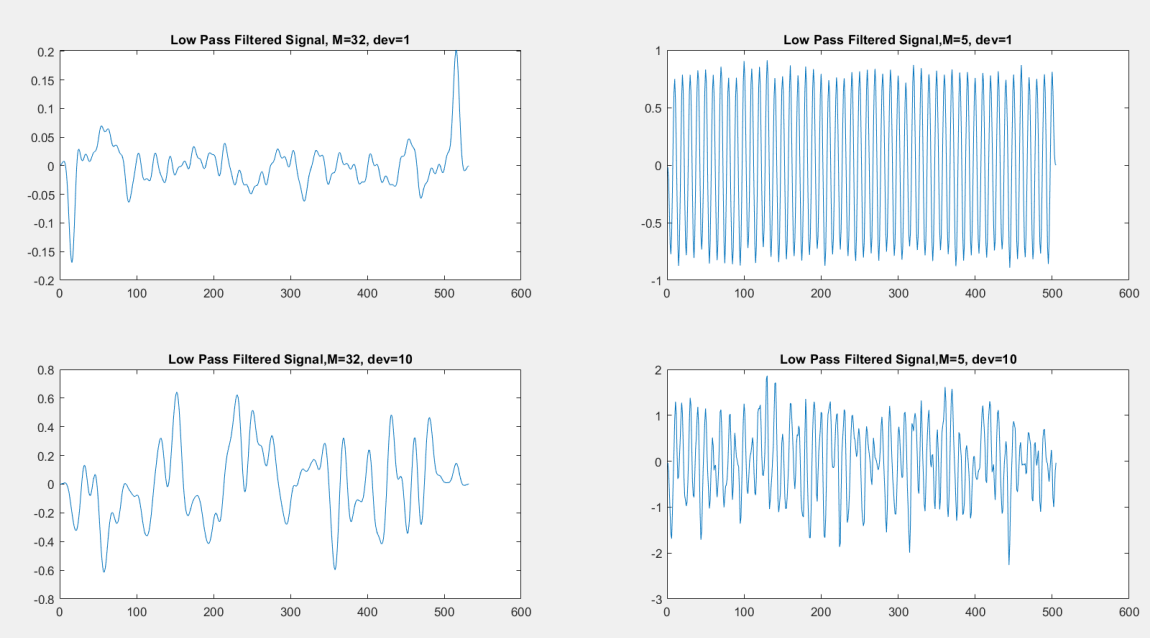
NOISY INPUT SIGNAL (with Aliasing Effect - Input Frequency f1=1800 Hz)

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

Matlab Code for m=32 and deviation = 1



Graphs for all 4 cases of low pass filtered signal. Due to aliasing effect (since Input frequency > 2*Fsampling), it is not able to filter the noise properly.



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

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

Matlab Code for m=5 and dev = 1

Matlab Code (same as above)

Graphs