

EL 308 – Term Project Report

Audio Spectrum Analyser

Objective:

In this project, the goal to build a audio spectrum analyser (ASA) system using 8086 assembly language programming and by implementing the hardware. It is thought that the sampling rate could be 8192 Hz and the samples can be 8 bit quantized (256 different levels). It is considered that only some frequency components will be displayed on the screen (such as in the case of the spectrum analyser in the Winamp or Windows Media Player). After investigations it is also seen that the frequency scale of these programs are not linear but logarithmic. Hence it is concluded that it would be a good idea to use a similar scheme in this project.

In the literature survey it is seen that in order to build a spectrum analyser the frequency components of the sampled input should be derived. Because the sampled data is quantized, a digital Fourier Transform (DFT) technique is needed for this task. DFT could be computed directly or using a Fast Fourier Transform (FFT) algorithm. FFT is the general name given to various DFT algorithms in order to calculate the transforms much faster. The complexity of DFT is N^2 whereas that of FFT is $N \log N$. For example, for a 128 unit sampled input will be computed after 128^2 multiplications and additions with DFT but only $128 \log 128$ instructions. The ratio is $128/7$. It was first decided to use a FFT algorithm to calculate the Fourier Transform. After a few trials it is seen that the FFT algorithm can not be easily implemented using loops and require huge amount coding time. Then it is decided to compute the DFT directly.

The formula for the DFT is given in Eq. 1.

$$f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk} \quad j = 0, \dots, n-1 \quad (1)$$

This can be also written as in Eq. 2:

$$f_j = \sum_{k=0}^{n-1} x_k \left(\cos\left(\frac{2\pi}{n} jk\right) - i \sin\left(\frac{2\pi}{n} jk\right) \right) \quad j = 0, \dots, n-1 \quad (2)$$

Looking at the previous equation, it is seen that two separate addition for both sine and cosine term should be evaluated with the corresponding x_k value and at the end of each summation, the magnitude should be taken by taking the square of both sine(imaginary) and cosine(real) terms and the square root of the addition of two. The number of clock cycles for sine and cosine for the math co-processors after 80487 are approximately 350. The multiplication clock cycle is 16 clock cycles and for addition 8-20 clock cycles. So it is obvious that the major time will be spent by taking sine and cosine. To calculate one frequency component processing 256 samples, 512 sine or cosine instructions are needed. If 16 frequency components will be shown, then a total number of $16 \cdot 512 \cdot 350$ clock cycles for sine and cosine instructions will be needed. (These computations will take the most time). We are estimating that to calculate 16 frequency components approximately 0.01 seconds will be needed. If it is assumed that 0.01 seconds also needed to display the results on the screen and

$\frac{256}{8192} \approx 0.03$ seconds are needed to take the sampled data, a total of 0.05 will be needed to

take the data, process it and then to display it.

To do mathematical computations, the math coprocessor's abilities will be used. We are assuming that mostly fsin, fcos, fadd, fsub, fmul, fsqrt will be the tools.

The general block diagram of the hardware is shown in Fig 1.

Block Diagram Schematics of Audio Spectrum Analyser

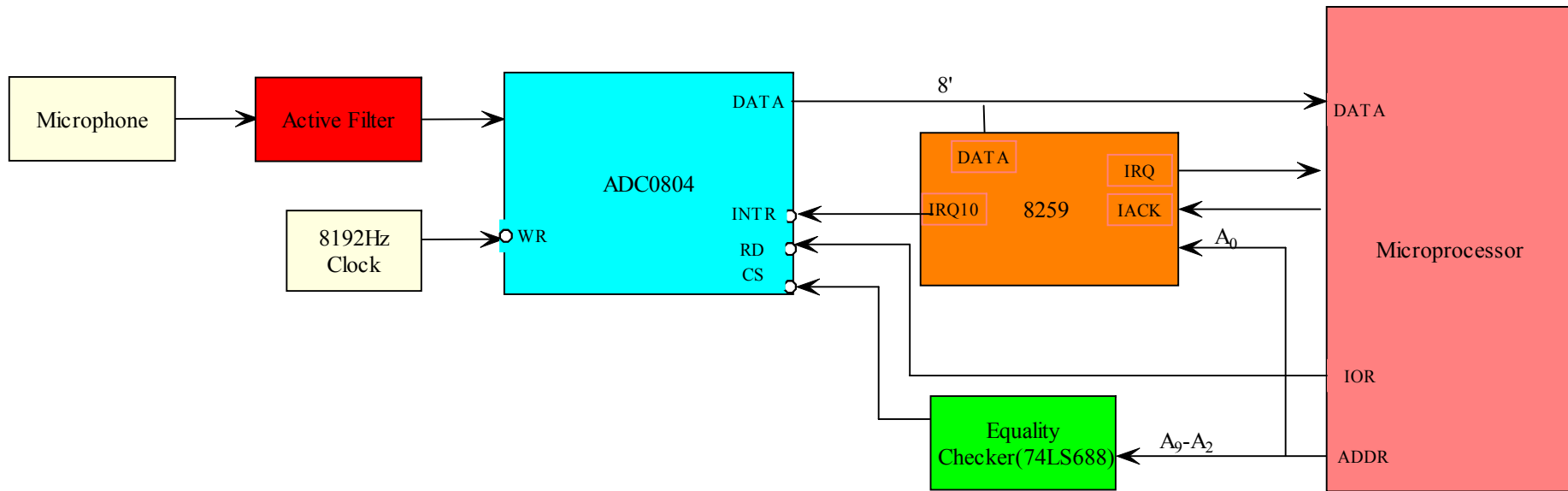


Fig 1. Block Diagram Schematics of the Audio Spectrum Analyser

AMPLIFICATION and FILTERING:

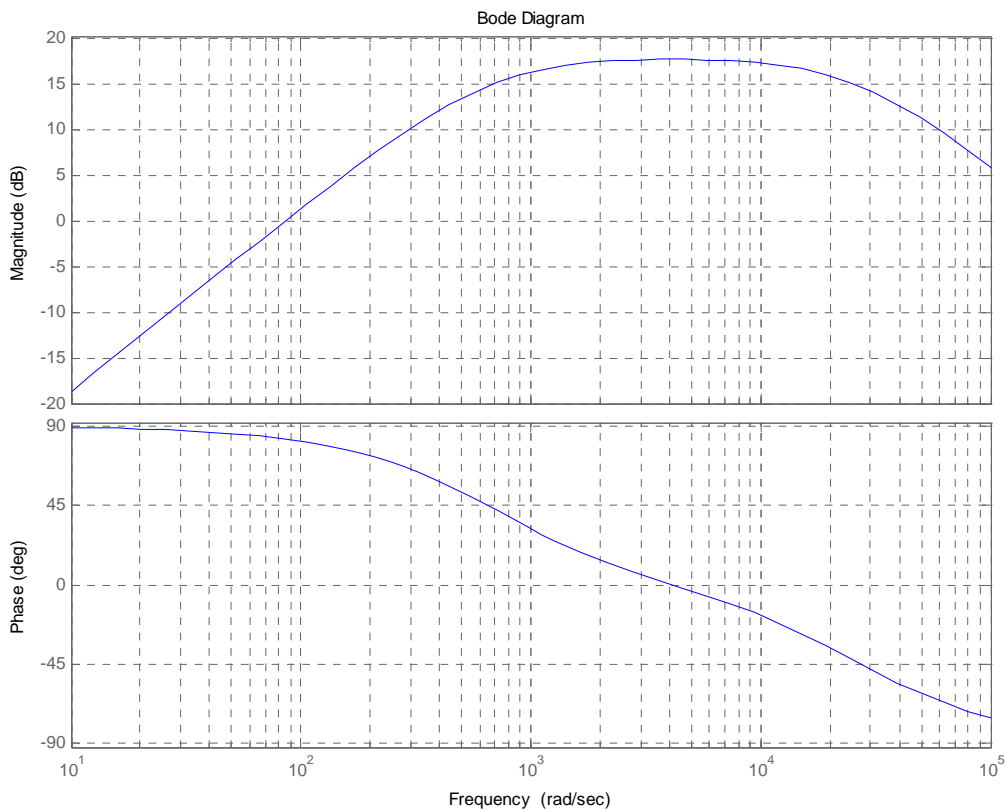


Figure 2: Bode plot of the frequency response of the amplifier and filter

The audio is converted in electrical signal by the microphone. The microphone is connected to the input of the filter and amplifier. The first filter is high pass filter with a bandwidth of 120Hz. The next stage is the amplification part which brings a gain of 8 (17dB). At the end of the system there is a low pass filter with a 3dB frequency of 4100Hz. As we see from the figure 2, the gain is approximately 17dB and it passes from 750 rad/sec frequency value, which corresponds to the 120Hz. The sloping down reaches the 3dB value around 25000rad/sec, which is about 4.1 kHz.

ADC0804 is used to interface analog output from the external world to digital inputs which will be connected to 8 bit datapath.(8 bit mean 256 different quantized level.)

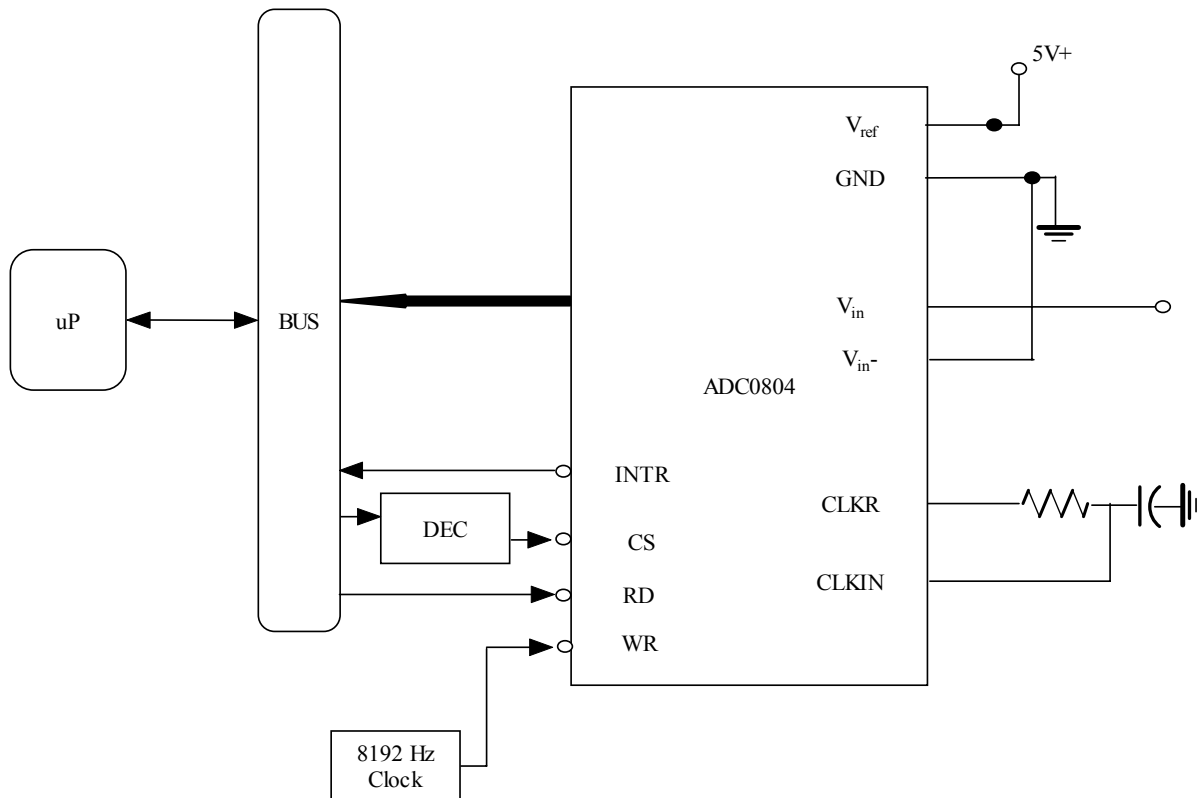


Figure 3: The basic structure of the ADC0804 interfacing to microprocessor

The above figure shows how the ADC0804 can be used and interfaced to the bus thus the microprocessor. Analog input is taken by the V_{in} and the V_{in-} pins and the reference voltage of 5V is applied to V_{ref} . Data is directly connected to the microprocessor via the bus line. INTR is connected to the bus line also but it is ended in the IRQ10 of the 8259 (Interrupt Priority Resolver). CS is enabled whenever an input of 0300h via the address line in the bus. This is achieved by the equality checker 74LS688 (here named as decoder). One input to the 74LS688 will be the A_9-A_2 of the inputs of the address line and the other input is 11000000 in binary. CLKR and CLKIN is required to be connected to those electronic components in order to constitute the clock frequency, which will be required to activate the internal register and sequential components of the ADC0804.

As it has been discussed, when the following systems are interfaced, conversion will take place whenever clock generator sends a low asserted signal that means at a frequency of 8192 Hz. Because one analog to digital conversion is performed in 70 clock cycles

determined by the capacitor and resistor values in the CLKIN and CLKR inputs. Thus we need a frequency of at least 574 kHz, but we set it to 830 kHz in order to have no problem. This low asserted signal is directly connected to the IRQ15 of the ISA bus. When the generator sends a pulse to the IRQ 15, CPU is programmed to send write signal to the write enable of the ADC. This write command activates the conversion. After each conversion is completed the the interrupt is send to the IRQ 10 by the ADC. When CPU is invoked with this interrupt, it sends a read command to the ADC and take the quantized data at the output latches of the ADC. When the each data is taken, with the assistance of the interrupt service routine it is sent to a specific location in the data segment. Until a total of 256 samples have been taken, soft_int sends an interrupt to the to the main program indicating the status of the port and settles data to a appropriate position in the data segment of the main programm. After all the 256 data samples have been taken, the DFT part is to activated and the data taken is processed to give out the frequency components. After the DFT is done, the output values are taken, displayed on the screen. Every time 256 samples have been taken the process starts again, does what has been stated.

8192 Hz Clock generator:

TLC555 timer is used for this task. By adjusting the frequency we send a square wave pulses to the IRQ 15.

Analog input:

The analog input of the ADC is directly connected to the output of the amplifier-filter system. The frequency behaviour is in Figure 2. The MATLAB code and the design of the filter is as follows:

Appendix:

MATLAB CODE:

```
R3 = 68e3  
R4 = 10e3  
C2 = 150e-9  
R2 = 10e3  
C1 = 4.7e-9  
R1 = 8.2e3
```

```
NUM = [ (1+R3/R4)*(C2*R2) 0 ]  
DEN = [ R1*R2*C1*C2 (R1*C1+R2*C2) 1 ]  
sys= TF( NUM,DEN)
```

```
bode(sys , {1e1 , 1e5})  
grid on
```

--1.The ISR Program:

;Atakan Varol 6386
;Berkehan Ciftcioglu 6342

;This program invokes write instruction whenever IRQ15 is invoked
;and after conversion finishes takes the data and sends a soft_int
;invoke to the main program

```
.MODEL SMALL
.STACK 100h
.DATA
.386
INTA0 EQU 20h
INTA1 EQU 21h
EOI EQU 62h
PPDATA EQU 0300h ; Printer Port DATA

.CODE
Start:
    mov ax,@Data
    mov dx,ax
; Enable interrupt on master 8259 (IRQ2)
    mov dx,INTA1
    in al,dx
    and al,11111011b
    out dx,al
; Enable interrupt on slave 8259 (IRQ10 and IRQ15)
    mov dx, 0A1h
    in al, dx
    and al, 01110011b
    out dx, al

; Zero ES
    mov ax,0000h
    mov es,ax

; Install hardware interrupt routine pointer
    mov ax,OFFSET HardInt1
    mov es:[4*72h+0],ax
    mov ax,cs
    mov es:[4*72h+2],ax

; Install hardware interrupt routine pointer
    mov ax,OFFSET HardInt2
    mov es:[4*77h+0],ax
    mov ax,cs
    mov es:[4*77h+2],ax

; Install software interrupt routine pointer
```

```
mov ax,OFFSET softint
mov es:[4*255+0],ax
mov ax,cs
mov es:[4*255+2],ax
```

```
; Stay resident and terminate
mov ah,31h
mov dx,OFFSET Last
inc dx
int 21h
```

```
pdata DB 0
pstatus DB 0
```

```
HardInt1:
```

```
push ax
push dx
push es
```

```
mov ax, 0B800h
mov es, ax
```

```
mov dx,0300h
in al,dx
mov cs:[pdata],al
mov al,1
mov cs:[pstatus],al
```

```
mov dx,INTA0 ; master end of interrupt
mov al,EOI
out dx,al
```

```
mov dx,0A0h ; slave end of interrupt
mov al,EOI
out dx,al
```

```
pop es
pop dx
pop ax
iret
```

```
softint:
```

```
mov ah,cs:[pstatus]
mov al,0
mov cs:[pstatus],al
mov al,cs:[pdata]
iret
```

```
HardInt2:
```

```
push ax
push dx
push ds
```

```
push es
```

```
mov dx, PPDATA ;sends write instruction to the ADC  
out dx, al
```

```
mov dx,INTA0 ; master end of interrupt  
mov al,EOI  
out dx,al
```

```
mov dx,0A0h ; slave end of interrupt  
mov al,67h  
out dx,al
```

```
pop es  
pop ds  
pop dx  
pop ax  
iret
```

Last:

```
END Start
```

--2.The Main Program:

;Berkehan Ciftcioglu 6342
;Atakan Varol 6342

; This program takes the data sampled by the ADC and
; takes the DFT of the 256 sampled datas, and then give
; out the result and displays it on the screen.

DOSSEG

.MODEL SMALL

.STACK 100h

.DATA

.386

input_db DB 256 DUP (0)

input DW 256 DUP (0)

real DT 0.

imag DT 0.

result DT 0.

result1 DT 0.

output DW 16 DUP (0)

op1 DT 0.

jfake DW 0

j DT 0.

N DT 256. ; resolution rate

k DW 0 ; the pointer of the summation operand

mid DT 0.

ratio DT 70000.

two DW 2

LookUp DB 1,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120

results DB 16 DUP (0)

six DB 8

xpos DW 0

hsixty DB 160

lineoffset DW 800

Title_main DB 'Audio Spectrum Analyser','\$'

Title_sub DB 'copyright @ Atakan and Berkehan', '\$'

.CODE

Start:

mov ax, @Data ; set data

mov ds, ax

mov ax, 0B800h ; set video memory

mov es, ax

call Disp_format

General_loop:

```
    call Take_DATA
    call DFT
    call Disp_DFT
```

jmp General_loop

Terminate:

```
    mov ah, 4Ch      ; end the program
    int 21h
```

; -----

Disp_format Proc

```
    ; clear the screen
    mov ax, 0600h
    mov bh,07
    mov cx, 0
    mov dx,184fh
    int 10h
```

```
    ; set cursor to the begin of the title
    mov ah,02h
    mov bh,00
    mov dl,20
    mov dh,2
    int 10h
```

```
    ; Print the Title
    mov ah,09
    mov dx, Offset Title_main
    int 21h
```

```
    ; set cursor to the begin of the title
    mov ah,02h
    mov bh,00
    mov dl,25
    mov dh,4
    int 10h
```

```
    ; Print the Title
    mov ah,09
    mov dx, Offset Title_sub
    int 21h
```

```
    ret
```

Disp_format ENDP

; -----

Take_DATA Proc

```
mov cx, 0
mov si, 0
```

Check_data:

```
mov ah,0Bh
int 21h
int 255
cmp ah,1
jne Check_data
mov input_db[si],al
```

```
inc si
inc cx
cmp cx,256
jne Check_data
```

```
mov si, 0
```

fill_data:

```
mov ah, 0
mov al, input_db[si]
mov input[si], ax
inc si
cmp si, 256
jne fill_data
```

```
ret
```

ENDP Take_DATA

; -----

DFT Proc

```
mov di,0
```

loop2: ; calculates the selected freq components

```
mov ah, 00h
mov al, LookUp[di]
mov jfake,ax
```

```
finit
fild jfake
fstp j ; the freq comp to be calc. loaded in j
fwait
```

```
mov cx, 255 ; sampling is 256
```

summation:

```
mov k, cx
mov bx, cx
add bx, bx ; bx points the DW of the corresponding addition pointer
```

```

finit
fldpi
fadd st(0), st(0) ; obtaining 2*pi
fild k
fmul ; obtaining 2*pi*k
fstp mid
fld N
fld mid
fdiv st(0), st(1) ; (2*pi*k)/n loaded in st(0)
fld j
fmul ; (2*pi*k*j)/n loaded in st(0)
fstp op1 ; result stored in op1
fwait

; cosine part
fld op1
fcos
fild input[bx]
fmul
fld real
fadd
fstp real
fwait

; sine part
fld op1
fsin
fild input[bx]
fmul
fld imag
fadd st(0), st(1)
fstp imag
fwait
loop summation

finit
fld real
fmul st(0), st(0)
fstp real
fwait
fld imag
fmul st(0), st(0)
fstp imag
fwait
fld real
fld imag
fadd st(0), st(1)
fsqrt
fstp result1
fwait
fldz
fstp imag ; make imag 0 for next frequency

```

```

fwait
fldz
fstp real      ; make real 0 for next frequency
fwait

```

; divides by ratio for printing screen purposes

```

mov bx, di
add bx, bx

finit
fld ratio
fld result1
fdiv st(0), st(1)
fstp result1
fwait
fld result1
fbstp result
fwait
fld result1
fistp output[bx]
fwait

```

; test for bytes

```

mov ax, 0B800h
mov es, ax
mov ax, di
mul two
mov si, ax
mov al, byte ptr output[bx]
mov results[di], al

```

```

inc di
cmp di, 16
jne Loop2      ; end of Discrete time Fourier Transform

```

```

ret
DFT ENDP

```

```

;-----
Disp_DFT PROC

```

```

    mov cx, 16
Dsploop:
    mov di, cx
    dec di      ; because first op is pointed by 0
    mov al, cl
    mul six
    mov xpos, ax      ; find xpos of the chr to be displayed
    add xpos, 8

```

```

mov bh, 0
mov bl, results[di]
mov ax, 15
sub ax, bx
mov si, ax      ; si contains the stop line
mov bx, ax      ; bx contains the stop line

Vertcount:      ; prints the green spaces
mov ax, si
mul hsixty
add ax, xpos
mov si, ax

mov al, 32      ; ascii code of space
add si, lineoffset
mov es:[si], al
mov al, 00100111b
mov es:[si+1],al

inc bx
mov si,bx
cmp bx, 16
jne Vertcount

mov bh, 0
mov bl, results[di] ; bx contains the stop line
mov ax,16
sub ax,bx
mov bx,ax

mov dx,0
Vertcount2:    ; prints the black spaces
mov ax,dx
mul hsixty
add ax, xpos
mov si, ax

mov al, 32      ; ascii code of space
add si, lineoffset
mov es:[si], al
mov al, 00000111b
mov es:[si+1],al

inc dx
cmp dx, bx
jne Vertcount2
loop Dsploop

ret
ENDP Disp_DFT
; -----
END

```