REAL TIME DIGITAL SIGNAL PROCESSING

A Lab Record

submitted by

SOUMYA SARA JOHN

in partial fulfillment of the requirements
for the award of credits in RTDSP LAB: AVD641

AVIONICS
INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
Thiruvananthapuram - 695547

May 2018

CERTIFICATE

This is to certify that the thesis titled '**Real Time Digital Sigal Processing**', submitted by **Soumya Sara John**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, for the award of the degree of **MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Sheeba Rani J
Supervisor
Avionics
IIST

Dr. Manoj B.S.
Head of Department
Avionics
IIST

Place: Thiruvananthapuram
May, 2018

DECLARATION

I declare that this thesis titled '**Real Time Digital Signal Processing**' submitted in fulfillment of the Degree of MASTER OF TECHNOLOGY is a record of original work carried out by me under the supervision of **Dr. Sheeba Rani J**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Soumya Sara John
SC17M048

Place: Thiruvananthapuram
May, 2018

Acknowledgements

I would like to give special thanks to my guide, Dr.Sheeba Rani J, Associate Professor, Department of Avionics, for giving her enthusiastic views on the preparation of reports and presentations and guiding me throughout. I would like to express my sincere thanks and deep sense of gratitude to my senior Manne Ruchitha , Mtech final year ,DSP,Avionics ,Thomas James Thomas, Research scholar,Avionics and Nidheesh Ravi, Lab-in charge , who have not only helped me in enhancing my technical knowledge about this lab, but along with this, have also taken me up to that zenith where I could effectively present my views among others.

Contents

Chapter 1

Introduction to Code Composer Studio

Question 1:Write a program to display your name and roll number on your console window.

Theory

Stdio.h stands for standard input output and it is included in every c program so that any programmer can use some predefined functions into their program/source code. Some important predefined functions in this header file are printf and scanf which provide the facility to take input from user and to put output on console.

Algorithm

```
Step 1 : Start
Step 2 : Print your name
Step 3 : Print your roll number
Step 4 : Stop
```

Program Code:

```
#include <stdio.h>
int main(void) {
printf("Soumya Sara John\n");
printf("SC17M048");
return 0;}
```

Outputs

Soumya Sara John
SC17M048

Result

Displayed name and roll number on the console window.

Question 2:Write a program to find the sum and product of two user input numbers in Q15 format (fixed point format).

Theory

Q.15 format is commonly used in DSP systems,and data must be properly scaled so that the value lies between -1 and 0.999969482421875.Most fixed point DSP processors use two's complement fractional numbers in different Q formats. However,assemblers only recognize integer values. Thus,the programmer must keep track of the position of the binary point when manipulating fractional numbers in assembly programs.

Algorithm

Step 1 : Start

Step 2 : Declare variables n1,n2,s,m1

Step 3 : Give integer values to n1 and n2

Step 4 : Calculate s=n1+n2

Step 5 : Calculate m1=n1*n2

Step 6 : Display s and m1

Step 7 : Stop

Program Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/*
 * main.c
 */
int main(void) {
int n1=16384;
int n2=8192;
int s;
s=_sadd(n1,n2);
int s1=n1+n2;
int m1=n1*n2;
//int m=_smac(n1,n2);
printf("Sum= %d",s);

printf("\n Product=%d",m1);

return 0;
}
```

Output

Sum= 24576

Product=134217728

Observations

n1=0.5 ; ($0.5*2^{15} = 16384$)

n2=0.25 ; ($0.25*2^{15} = 8192$)

sum=24576 ; ($24576 * 2^{-15} = 0.75$)

product=134217728 ; ($134217728 * 2^{-30} = 0.125$)

Inference

The arithmetic result obtained by a DSP processor is in the integer form. It can be interpreted as a fractional value by dividing by 2^n

Result

Obtained the sum and product of two Q15 format fixed point numbers.

Question 3:Write a program to calculate the linear convolution of a long sequence x[n] and impulse response h[n] sequence.

Theory

Linear convolution is the basic operation to calculate the output for any linear time invariant system given its input and its impulse response. For two signals

x[n] and h[n], linear convolution output y[n]=$\sum_{k=-\infty}^{\infty} x[k]h[n-k]$

Algorithm

Step 1 : Start

Step 2 : Declare variables x,h,y,i,j

Step 3 : Initialize i=0 and repeat till i=12

3.1 : Initialize j=0 and repeat till j=3

3.1.1 : y[i]=y[i]+x[i-j]*h[j]

3.1.2 : j=j+1

3.2 : i=i+1

Step 4 : Display y

Step 5 : Stop

Program Code:

```
#include <stdio.h>
/*
* main.c
*/
int main(void) {
int x[10]={3,-1,0,1,3,2,0,1,2,1};
int h[3]={1,1,1};
int y[13]=0;
int i,j;
for(i=0;i<12;i++)
{for (j=0;j<3;j++)
{
if((i-j)>=0 && (i-j)<10)
y[i]=y[i]+(x[i-j]*h[j]);
}
}printf("[");
```

```
for (i=0;i<12;i++){


printf("%d \t",y[i]);
}
printf("]");
return 0;
}
```

Output

[3 2 2 0 4 6 5 3 3 4 3 1]

Observations

x[n]=[3,-1,0,1,3,2,0,1,2,1]

h[n]=[1,1,1]

y[n]=x[n]*h[n]=[3,2,2,0,4,6,5,3,3,4,3,1]

Inference

The length of the convolved sequence is n1+n2-1,where n1 is the length of x[n] and n2 is the length of h[n].

Result

Obtained the linear convolution result of two sequences x[n] and h[n].

Question 4:Write a program to calculate the circular convolution of a long sequence x[n] and impulse response h[n] sequence.

Theory

Circular convolution of two signals x[n] and y[n] of length N is defined as
$x[n] \textcircled{N} y[n] = \sum_{k=0}^{N-1} x[k]h(([n-k]))_N$
If they are not of the same length ,then do zero padding to make them of length N.

Algorithm

Step 1 : Start

Step 2 : Declare variables x,h,y,i,j

Step 3 : Create $h1 = h(([i=1]))_4, h2 = h(([i-2]))_4, h3 = h(([i-3]))_4$

Step 4 : Initialize i=0 and repeat till i=4

3.1 : y[0]=y[0]+x[i]*h[i]

3.2 : y[1]=y[1]+x[i]*h1[i]

3.3 : y[2]=y[2]+x[i]*h2[i]

3.4 : y[3]=y[3]+x[i]*h3[i]

Step 4 : Display y

Step 5 : Stop

Program Code:

```
#include<stdio.h>
/*
* main.c
*/
```

```
int main(void) {
int x[4]={3,-1,0,1};
int h[4]={1,1,0,1};
int y[4] ,h1[4],h2[4],h3[4];
int i;
for( i=0;i<4;i++)
{
if(i-1>=0)
h1[i]=h[i-1];
else
h1[0]=h[3];
if(i-2>=0)
h2[i]=h[i-2];
else
{h2[1]=h[3];
h2[0]=h[2];}
if(i-3>=0)
h3[i]=h[i-3];
else
{h3[2]=h[3];
h3[1]=h[2];
h3[0]=h[1];
}
}
for(i=0;i<4;i++)
{
y[0]=y[0]+x[i]*h[i];
y[1]=y[1]+x[i]*h1[i];
y[2]=y[2]+x[i]*h2[i];
y[3]=y[3]+x[i]*h3[i];
```

```
}printf("[");
for (i=0;i<4;i++)
{
printf("%d \t",y[i]);
}printf("]");
return 0;
}
```

Output

[3 2 0 4]

Observations

x[n]=[3,-1,0,1]

h[n]=[1,1,0,1]

y[n]=[3,2,0,4]

Inferences

The length of the two sequences must be the same and hence if n1 is the length of x[n],n2 is the length of h[n],N-n1 number of zeros must be added to x[n] and N-n2 zeros must be added to h[n]. This is called zero padding. The length of y[n] would be N.

Result

Obtained the circular convolution output of two sequences x[n] and y[n].

Question 5:Write a program to generate a sine wave and plot it on the graph using i.inbuilt function ii.look up table.

Theory

The simplest method to generate Sine wave is to use Trignometric Sin function. The Sin function will generate the samples from parameters like sampling frequency, number of samples, input frequency. $s(t) = sin(\omega_0 t + \phi)$

Algorithm

```
Step 1 : Start
Step 2 : Declare variables s,sine,si,t
Step 3 : Create a look up table for sine in sine[]
Step 4 : Initialise t=0 and repeat the steps till t=500
4.1 : Create s[t] using sine in-built function
4.2 : Create si[t] using look-up table
Step 5 : Stop
```

Program Code:

```
#include<stdio.h>
#include<math.h>
float s[];
short sine[8]={0,707,1000,707,0,-707,-1000,-707};
short si[];
/*
* main.c
*/
int main(void) {
```

```
int t;
for(t=0;t<500;t=t+1)
{
s[t]=sin(2*3.14*100*t);
si[t]=sine[(t)%8];
printf("%d\t",s);
}
return 0;
}
```

Output

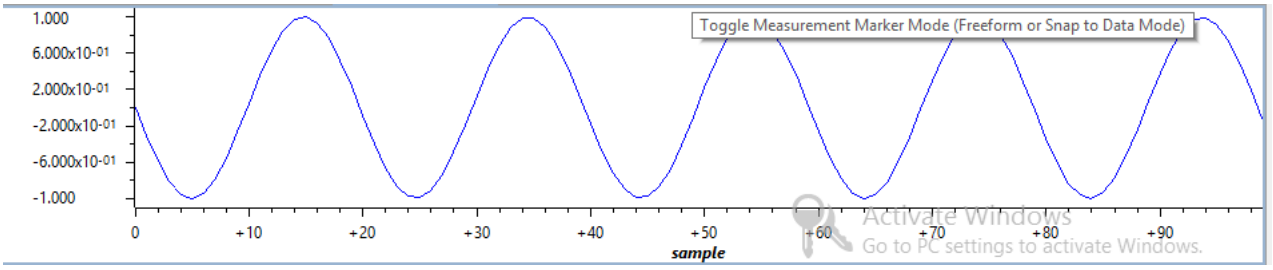


FIGURE 1.1: Sine wave using inbuilt function,f=100 Hz

Result

Obtained the sine waveform using look up table and in-built function.

Question 6:Write a program to generate and plot the following waveforms using look up table: i.Square wave ii.Triangular wave iii.Ramp wave.

Theory

A square wave is a non-sinusoidal periodic waveform in which the amplitude alternates at a steady frequency between fixed minimum and maximum values, with the same duration at minimum and maximum.
A triangle wave is a non-sinusoidal waveform named for its triangular shape. It is a periodic, piecewise linear, continuous real function.
The sawtooth wave (or ramp wave) is a kind of non-sinusoidal waveform which resembles the teeth of a plain-toothed saw.

Algorithm

```
Step 1 : Start
Step 2 : Declare variables sq,squ,r,ram,tr,tri,t
Step 3 : Create a look up table for square in sq,triangle in tr and ramp in r
Step 4 : Initialise t=0 and repeat the steps till t=100
4.1 : Create squ[t] using look-up table
4.2 : Create tri[t] using look-up table
4.3 : Create ram[t] using look-up table
Step 5 : Stop
```

Program Code:

```
#include<stdio.h>
int sq[10]={2,2,2,2,2,-2,-2,-2,-2,-2};
int squ[100];
int r[5]={0,1,2,-2,-1};
```

```
int ram[100];
int tr[8]={0,1,2,1,0,-1,-2,-1};
int tri[100];
/*
* main.c
*/
int main(void) {
int t;
for (t=0;t<100;t++)
{
squ[t]=sq[t%10];
ram[t]=r[t%5];
tri[t]=tr[t%8];
}
return 0;
}
```

Output

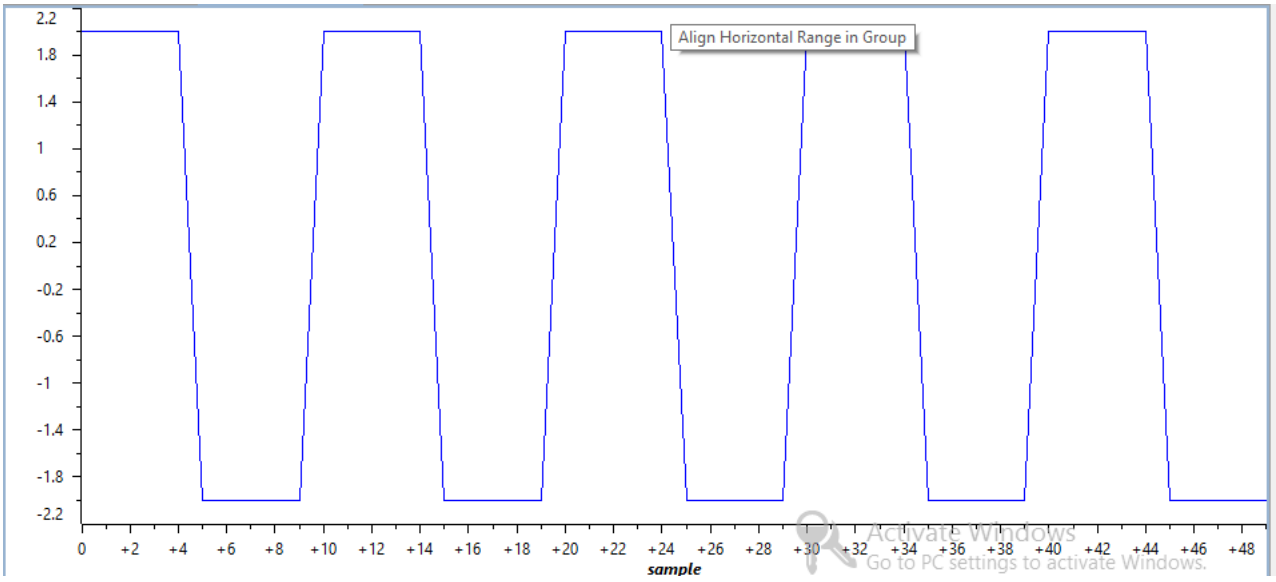


FIGURE 1.2: Square Wave

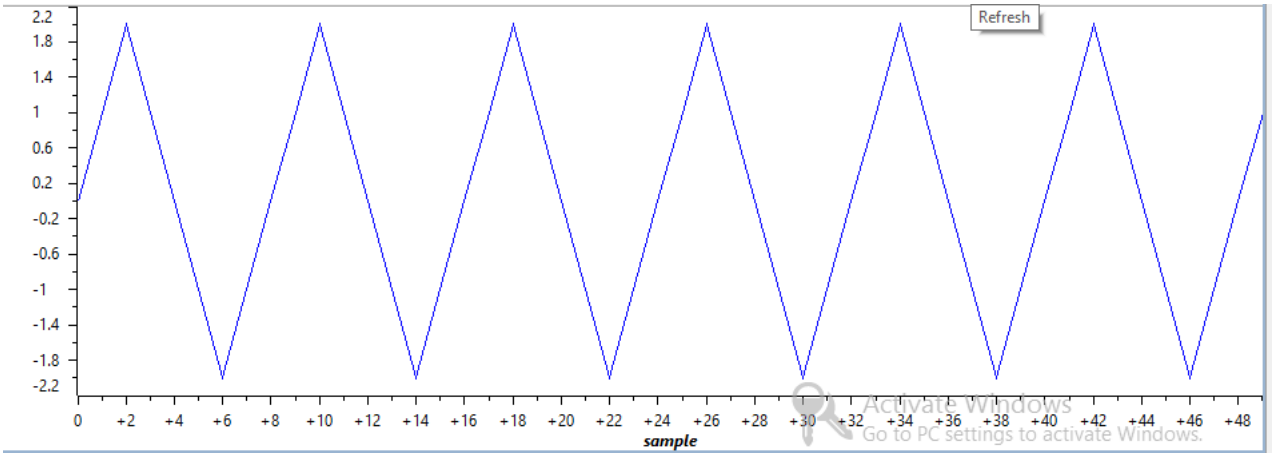


FIGURE 1.3: Triangular wave

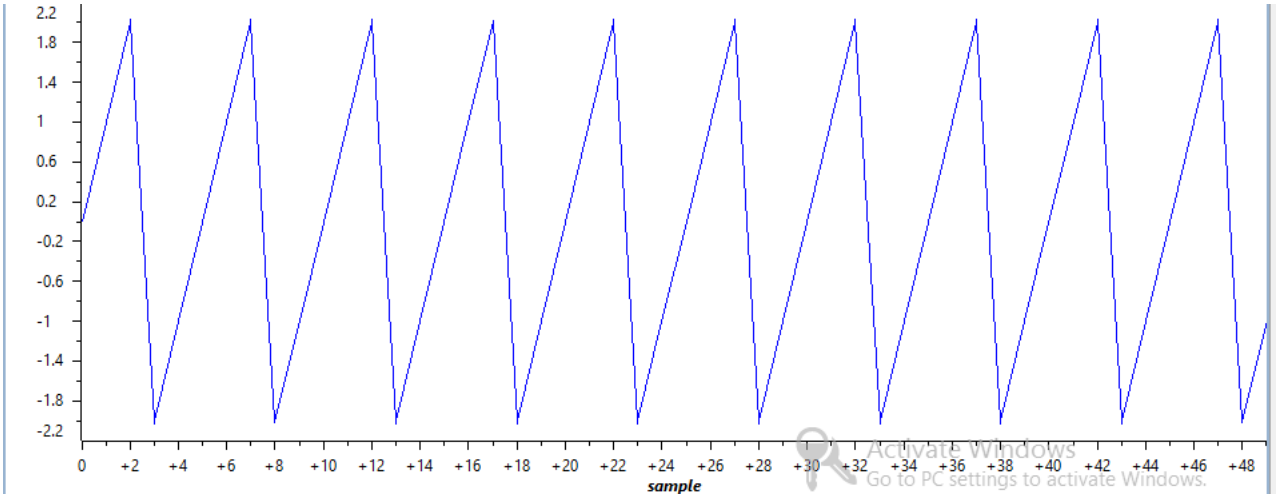


FIGURE 1.4: Ramp wave

Result

Obtained square,triangular and ramp waveforms using look-up tables.

Chapter 2

DIP Switches, LEDs and AIC23 Codec

Question 1. Turn ON an LED when one of the switches is pressed and subsequently turn the LED OFF when the switch is released

Theory

The header files used for LEDs is dsk6713_ led.h and for DIPs is dsk6713_ dip.h
DSK has to be initialized in the beginning using DSK6713_init().
There are four switches and LEDs on the board.
Also,LED and DIP switches must be initialized using DSK6713_LED_init()
and DSK6713_DIP_init() respectively.
To turn ON a LED,we use DSK6713_LED_on(LED number) and to turn it
off,we use DSK6713_LED_off(LED number).
To see whether a switch is pressed, we read the data from the switch using DSK67

Algorithm

Step 1 : Start

Step 2 : Include the necessary header files

Step 3 : Initialise DSK 6713 chip

Step 4 : Repeat the following steps infinitely

   4.1 : If DIP switch 2 is pressed,goto step 4.2

   4.2 : Turn on LED2

   4.3 : Else goto step 4.4

   4.4 : Turn off LED2

Step 5 : Stop

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
int i;

int main(void) {
DSK6713_init();
DSK6713_LED_init();
DSK6713_DIP_init();
while(1){
if(DSK6713_DIP_get(0)==0)
DSK6713_LED_on(2);
else
DSK6713_LED_off(2);
}
return 0;
}
```

Result

LED 2 was turned on when DIP switch 0 was pressed and turned off when switch 0 was not pressed.

Question 2. Toggle an LED ON/OFF for 4 seconds

Theory

To toggle a LED, we use DSK6713_LED_toggle(LED number).

For waiting for t microseconds, we use DSK6713_waitusec(t).

Algorithm

Step 1 : Start

Step 2 : Include the necessary headerfiles

Step 3 : Initialise DSK 6713 chip

Step 4 : Repeat the following steps infinitely

   4.1 : Toggle LED 3

   4.2 : Wait for 4s

Step 5 : Stop

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
int i;
```

```
int main(void) {
DSK6713_init();
DSK6713_LED_init();
DSK6713_DIP_init();
while(1){
DSK6713_LED_toggle(3);
DSK6713_waitusec(4000000);
}
return 0;
}
```

Result

Toggled LED 3 for 4 seconds.

Question 3. Toggle all the LEDs by uneven delays and stop toggling by pressing the switches 0,1,2 and 3.

Algorithm

```
Step 1 : Start
Step 2 : Include the necessary headerfiles
Step 3 : Initialise DSK 6713 chip
Step 4 : Repeat the following steps infinitely
   4.1 : If no switches are pressed
 4.1.1 : Toggle LED 0
 4.1.2 : Wait for 4s
 4.1.3 : Toggle LED 1
 4.1.4 : Wait for 1s
 4.1.5 : Toggle LED 2
```

```
 4.1.6 : Wait for 2s
 4.1.7 : Toggle LED 3
 4.1.8 : Wait for 3s
  4.2 : If swich 0 is pressed
 4.2.1 : Turn off LED 0
  4.3 : If swich 1 is pressed
 4.3.1 : Turn off LED 1
  4.4 : If swich 2 is pressed
 4.4.1 : Turn off LED 2
  4.5 : If swich 3 is pressed
 4.5.1 : Turn off LED 3
Step 5 : Stop
```

Program Code:

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
int i;

/*
 * main.c
 */
int main(void) {
DSK6713_init();
DSK6713_LED_init();
DSK6713_DIP_init();
//DIP at 0,1;LED at 2,3
while(1){
if(DSK6713_DIP_get(0)==0)
DSK6713_LED_off(0);
```

```
else if (DSK6713_DIP_get(1)==0)
DSK6713_LED_off(1);
else if(DSK6713_DIP_get(2)==0)
DSK6713_LED_off(2);
else if(DSK6713_DIP_get(3)==0)
DSK6713_LED_off(3);
else
{ DSK6713_LED_toggle(0);
DSK6713_waitusec(4000000);
DSK6713_LED_toggle(1);
DSK6713_waitusec(1000000);
DSK6713_LED_toggle(2);
DSK6713_waitusec(2000000);
DSK6713_LED_toggle(3);
DSK6713_waitusec(3000000);



}



}
return 0;
}
```

Results

Toggled all the LEDs by controlling using DIP switches and with uneven delays.

Question 4. Generate sinusoidal,ramp,triangle and square signals using look-up table and verify them on the CRO. This generation should be based on switch selection with corresponding LEDs on.

Theory

The values for sinusoidal,square,triangular and ramp signals can be given as a look-up table before the main function and can be called inside the main function to generate real-time signals.

Algorithm

```
Step 1 : Start
Step 2 : Include the necessary headerfiles
Step 3 : Initialise DSK 6713 chip
Step 4 : Set sampling frequency,input source address
Step 5 : Declare variables s,si,sq,squ,r,ram,r1,tr,tri,t.
Step 6 : Call the function comm_poll()
Step 7 : Turn all LEDs off.
Step 8 : Repeat the steps infinitely
 8.1 : If DIP switch 0 is pressed
     8.1.1 : Turn LED 0 on
 8.1.2 : Write outputsamples squ
 8.2 : If DIP switch 1 is pressed
    8.2.1 : Turn LED 1 on
 8.2.2 : Write output samples r1
 8.3 : If DIP switch 2 is pressed
 8.3.1 : Turn LED 2 on
```

```
 8.3.2 : Write output samples tr
 8.4 : If DIP switch 3 is pressed
 8.4.1 : Turn LED 3 on
 8.4.2 : Write output samples si
 8.5 : If no switches are pressed
 8.5.1 : Turn all LEDs off
Step 9 : Stop
```

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"

void DSK6713_init();
void DSK6713_LED_init();
void DSK6713_DIP_init();
extern void comm_poll();
extern void output_sample();
extern void outputsample(int);
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0011;
short s[8]={0,707,1000,707,0,-707,-1000,-707};
short si[1000];
short sq[10]={2,2,2,2,2,-2,-2,-2,-2,-2};
short squ[100];
short r[1000];
float ram[100],r1;
short tr[10]={0,1,2,3,4,5,4,3,2,1};
short tri[100];
```

```
int t;

int main(void) {
DSK6713_LED_off(0);
DSK6713_LED_off(1);
DSK6713_LED_off(2);
DSK6713_LED_off(3);
comm_poll();



while(1)
{
if(DSK6713_DIP_get(0)==0)
{
DSK6713_LED_on(0);
for (t=0;t<100;t++)
{
squ[t]=100000*sq[t%10];
output_sample((short) squ[t]);
}}
else if(DSK6713_DIP_get(1)==0)
{  DSK6713_LED_on(1);
for (t=0;t<200;t++)
{
r1=-20000*t;
output_sample((short) r1);
}}
else if(DSK6713_DIP_get(2)==0)
{ DSK6713_LED_on(2);
for(t=0;t<100;t++)
```

```
{ tri[t]=100000*tr[t%10];

output_sample((short) tri[t]);
}}
else if(DSK6713_DIP_get(3)==0)
{ DSK6713_LED_on(3);
for(t=0;t<100;t++)
{ si[t]=100000*s[t%8];

output_sample((short) si[t]);
}}
else
{ DSK6713_LED_off(0);
DSK6713_LED_off(1);
DSK6713_LED_off(2);
DSK6713_LED_off(3);
for(t=0;t<100;t++)
{
squ[t]=0;
tri[t]=0;
output_sample((short) squ[t]);
}}
}

return 0;
}
```

Outputs

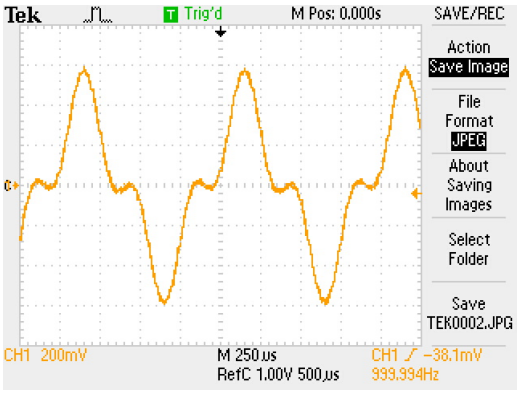


FIGURE 2.1: Sine Wave

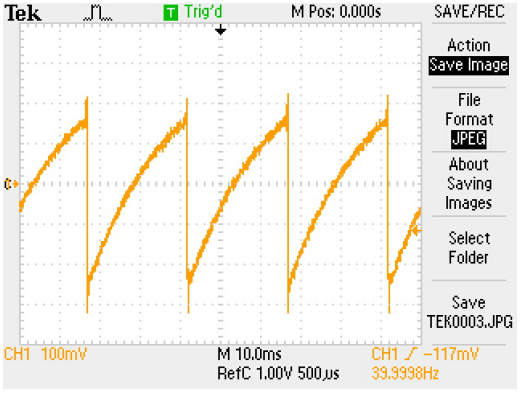


FIGURE 2.2: Ramp Wave

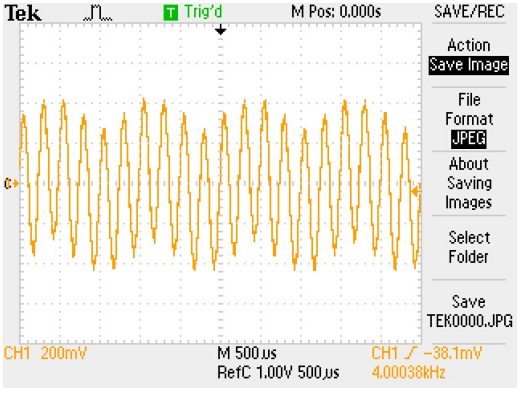


FIGURE 2.3: Triangle Wave

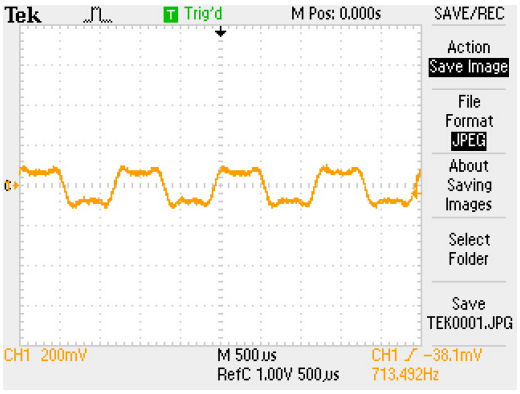


FIGURE 2.4: Square Wave

Observations

Sinusoidal ,ramp,triangular and square signals were seen on CRO when different switches were pressed.

Inferences

By giving the sinusoidal,ramp,triangular and square signal values in a period as look up table,it is possible to create real-time signals that can be visualised on CRO.

Result

Sinusoidal,ramp,triangular and ramp signals are generated using look-up tables and seen on CRO when the four switches were pressed.

Question 5. Record the voice input when DIP switch 1 is pressed and keep LED 1 on. When DIP switch 1 is released,LED1 is OFF and recording is stopped.On pressing the DIP switch 0,the recorded signal is sent as output to the headset.Play back with different decimation factor of 1,2,4.Write your observations/inferences.

Theory

Decimation is a process of reducing the sampling rate of a signal. The decimation factor is usually an integer or a rational fraction greater than one. This factor multiplies the sampling time or, equivalently, divides the sampling rate. For example, if 16-bit compact disc audio (sampled at 44,100 Hz) is decimated to 22,050 Hz, the audio is said to be decimated by a factor of 2. Downsampling alone causes high-frequency signal components to be misinterpreted by subsequent users of the data, which is a form of distortion called aliasing. In this application, the filter is called an anti-aliasing filter,which is a low pass filter.

Algorithm

```
Step 1 : Start
Step 2 : Include the necessary headerfiles
Step 3 : Initialise DSK 6713 chip
Step 4 : Set sampling frequency,input source address
Step 5 : Declare variables i,j,a
Step 6 : Call the function comm_poll()
Step 7 : Repeat the steps infinitely
```

```
   7.1 :If DIP switch 1 is pressed,do steps 7.2 and 7.3
   7.2 :Turn LED 1 on
   7.3 : Read the input to i using input_sample
   7.4 : If DIP switch 0 is pressed,do steps 7.5 , 7.6 and 7.7
   7.5 : Turn LED 0 on
   7.6 : Set decimation rate
   7.7 : Write i as the output using output_sample
Step 8 : Stop
```

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"

void DSK6713_init();
void DSK6713_LED_init();
void DSK6713_DIP_init();
extern void comm_poll();
extern void output_sample();
extern void outputsample(int);
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
Uint16 input_sample();
float i[40000];int j;
float a;
int main(void) {
comm_poll();

DSK6713_LED_off(1);
```

```
DSK6713_LED_off(2);
DSK6713_LED_off(3);
DSK6713_LED_off(0);
while(1)
{
if(DSK6713_DIP_get(1)==0)
{for(j=0;j<40000;j++)
{DSK6713_LED_on(1);

a=input_sample();
i[j]=a;
}}
DSK6713_LED_off(1);

if(DSK6713_DIP_get(0)==0)
{for(j=0;j<40000;j=j+4)
{
DSK6713_LED_on(0);

a=i[j];
output_sample((short) a);
}}
DSK6713_LED_off(0);
DSK6713_LED_off(1);
DSK6713_LED_off(2);
DSK6713_LED_off(3);
}
return 0;
}
```

Observation

When DIP switch 1 was pressed,voice was recorded and when DIP switch 0 was pressed, the recorded voice was played back through the headphone. The voice could be heard properly when the decimation factor was 1.As it was increased to 2 and 4, the recorded voice could not be heard perfectly and was heard as beats. Information content was lost.

Inferences

When the decimation factor is increased,the number of samples that are sent decreased and the interpolation/reconstruction was not done properly.Hence,the information contained in the voice signal was almost lost , i.e. aliasing occured as we didnot use any anti-aliasing filter before downsampling.

Results

When DIP switch 1 was pressed,the sound was recorded through the microphone and when DIP switch 0 was pressed, the sound was played back through the headphone at decimation rates 1,2 and 4.

Question 6. Read sinusoidal signal from the function generator and display on CRO.Demonstrate aliasing using different sampling rates of AIC23 Codec.Write your observations/inferences.

Theory

Nyquist Theorem:

A continuous time signal can be represented in its samples and can be recovered

back when sampling frequency f_s is greater than or equal to the twice the highest frequency component of message signal,
i.e. $f_s \geq 2f_m$. $2f_m$ is called the Nyquist Rate of that signal.

Algorithm

```
Step 1 : Start
Step 2 : Include the necessary headerfiles
Step 3 : Initialise DSK 6713 chip
Step 4 : Set sampling frequency,input source address
Step 5 : Declare variables i,j
Step 6 : Call the function comm_poll()
Step 7 : Repeat the steps infinitely
   7.1 : LED 0 on
   7.2 : Read the input to i using input_sample
   7.3 : Write i as the output using output_sample
Step 8 : Stop
```

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"

void DSK6713_init();
void DSK6713_LED_init();
void DSK6713_DIP_init();
extern void comm_poll();
extern void output_sample();
extern void outputsample(int);
```

```
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0011;
Uint16 input_sample();short i;int j;

int main(void) {
comm_poll();

while(1){
DSK6713_LED_on(0);

i=input_sample();
output_sample((short) i);}
return 0;
}
```

Output

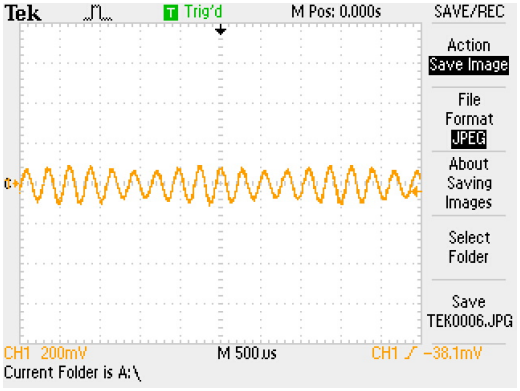


FIGURE 2.5: 4.3kHz Sine Wave

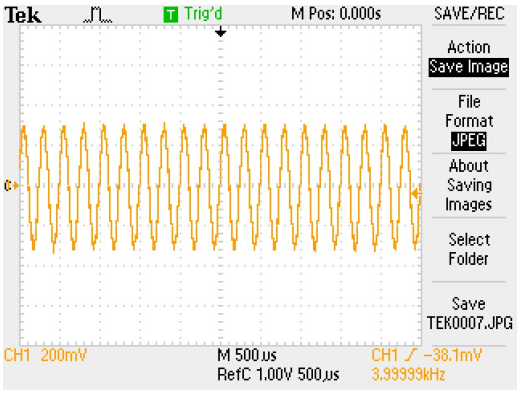


FIGURE 2.6: 4kHz Sine Wave

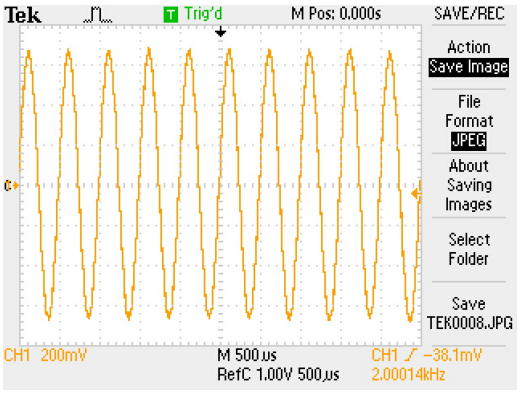


FIGURE 2.7: 2kHz Sine Wave

Observations

The sampling frequency of AIC23 Codec was set to 8kHz and therefore the maximum signal frequency that can be given is 4kHz. When the signal frequency is less than or equal to 4kHz,there is no aliasing effect and the signal could be retrieved without error. When the signal frequency was 4.3kHz,greater than 4kHz, the signal amplitude became small and aliasing could be understood.

Inferences

When the sampling frequency is less than the Nyquist rate of the signal,aliasing occurs and the complete reconstruction of the signal is not possible.Signal information is lost. Therefore,sampling frequency must be greater than twice the

maximum signal frequency present in the signal.

Results

Read the sinusoidal signal from the function generator and displayed it on the CRO. Aliasing was also observed by changing the input frequency.

Chapter 3

FIR Filter Design and Audio Processing

Question1: Implement a real-time digital system using circular buffer for performing four different FIR filtering tasks controlled by four control switches.

(a) Low pass with cutoff at 1.5 kHz

(b) High pass with cutoff at 2 kHz

(c) Band pass with center frequency at 1.75 kHz and bandwidth of 500 Hz

(d) Band stop with center frequency at 2.25 kHz and stop bandwidth of 500 Hz

Theory

A circular buffer is a memory allocation scheme where memory is reused when an index,incremented modulo the buffer size,writes over a previously used location. A circular buffer makes a bounded queue when seperate indices are used for inserting and removing data.

Algorithm

Step1: Start

Step2: Import the necessary headerfiles

Step3: Initialise DSK6713

Step4: Declare pointer variables

Step5: If switch 0 is pressed, BL=BLlpf,B=Blpf and goto step 9

Step6: Else if switch 1 is pressed,BL=BLhpf,B=Bhpf and goto step 9

Step7: Else if switch 2 is pressed,BL=BLbpf, B=Bhpf and go to step 9

Step8: Else if switch 3 is pressed, BL=BLbsf, B=Bbsf and goto step 9

Step9: Get the first sample at the location pointed by pointer p

Step10: Initialise y=0 and p=px

Step11: For i=0,do steps 9.1,9.2 till$i = BL$

11.1: Calculate$y = y + (*p--) * B[i]$

11.2: Output the sample y

Step12: Stop

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"fdacoef.h"
#include"fdacoefshigh.h"
#include"fdacoefs_bpf.h"
#include"fdacoefs_bsf.h"
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
```

```
extern void comm_poll();
extern void output_sample();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 input_sample();
Uint16 inputsource=0x0011;

int low(){
short x[101],*px,*p;
int i;
short y;
px=x;
while(DSK6713_DIP_get(0)==0){
*px=input_sample();
y=0;p=px;
if(++px>&x[BLlpf])
px=x;
for(i=0;i<BLlpf;i++)
{
y=y+(*p--)*Blpf[i];
if(p<&x[0])
p=&x[BLlpf];
}
output_sample((short) y);
}
return 0;
}
int high(){
short x[101],*px,*p;
int i;
short y;
px=x;
```

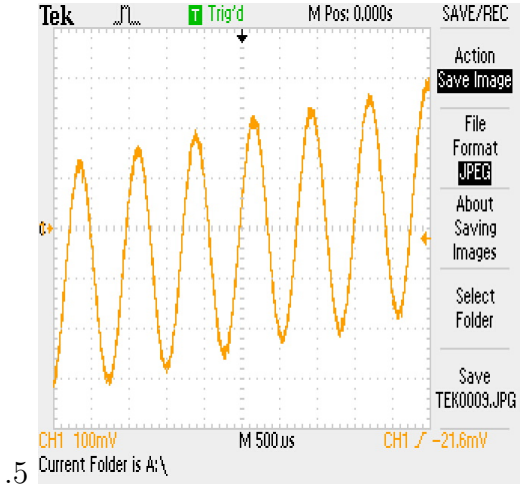
```
while(DSK6713_DIP_get(1)==0){
*px=input_sample();
y=0;p=px;
if(++px>&x[BLhpf])
px=x;
for(i=0;i<BLhpf;i++)
{
y=y+(*p--)*Bhpf[i];
if(p<&x[0])
p=&x[BLhpf];
}
output_sample((short) y);
}
return 0;}

int bpf(){
short x[101],*px,*p;
int i;
short y;
px=x;
while(DSK6713_DIP_get(2)==0){
*px=input_sample();
y=0;p=px;
if(++px>&x[BLbpf])
px=x;
for(i=0;i<BLbpf;i++)
{
y=y+(*p--)*Bbpf[i];
if(p<&x[0])
p=&x[BLbpf];
}
```

```
output_sample((short) y);
}
return 0;}
int bsf(){
short x[101],*px,*p;
int i;
short y;
px=x;
while(DSK6713_DIP_get(3)==0){
*px=input_sample();
y=0;p=px;
if(++px>&x[BLbsf])
px=x;
for(i=0;i<BLbsf;i++)
{
y=y+(*p--)*Bbsf[i];
if(p<&x[0])
p=&x[BLbsf];
}
output_sample((short) y);
}
return 0;}



int main(void) {
comm_poll();
while(1){
if(DSK6713_DIP_get(2)==0)
low();
else if(DSK6713_DIP_get(1)==0)
```

```
high();
else if(DSK6713_DIP_get(2)==0)
bpf();
else if(DSK6713_DIP_get(3)==0)
bsf();
}

return 0;
}
```

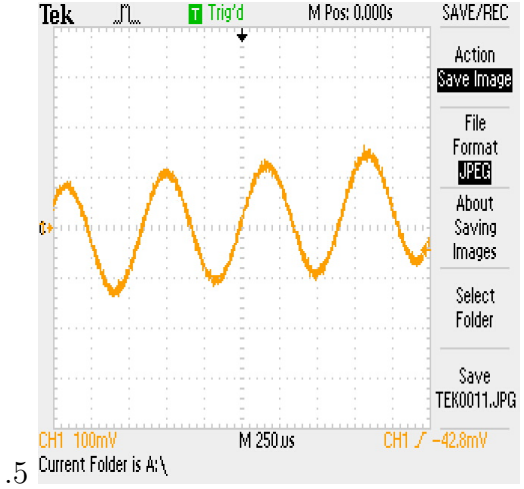
Output

(a)



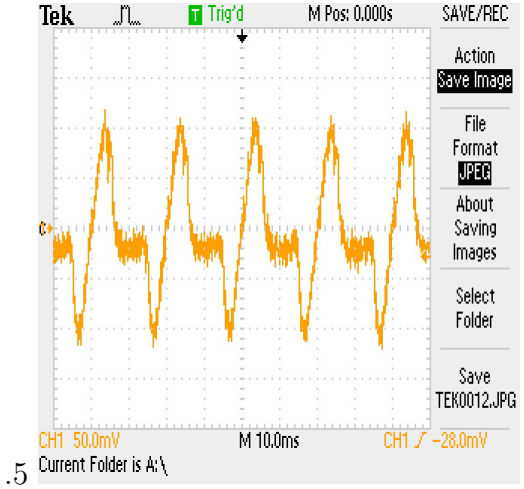
.5

FIGURE 3.1: fin=1.3kHz



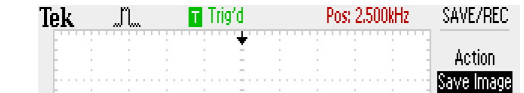
.5

FIGURE 3.2: fin=1.5kHz



.5

FIGURE 3.3: fin=1.6kHz



(b)

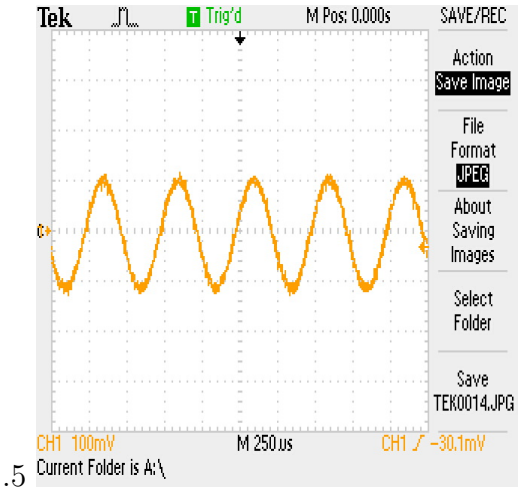


FIGURE 3.6: fin=2kHz

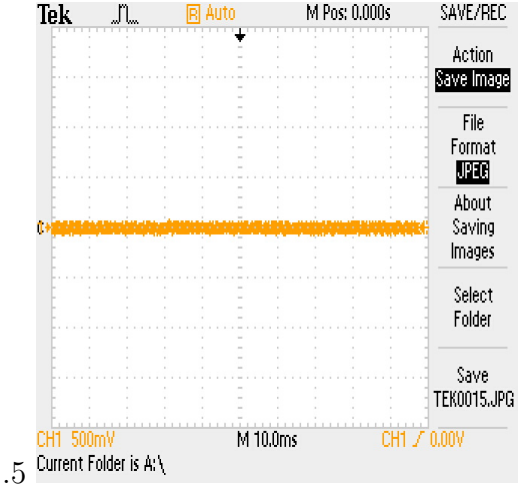


FIGURE 3.7: fin=1.96kHz

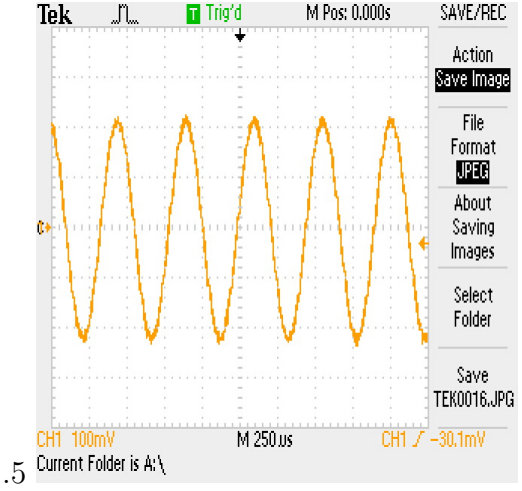
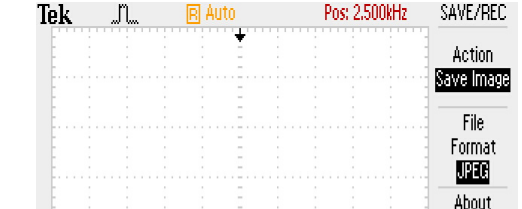
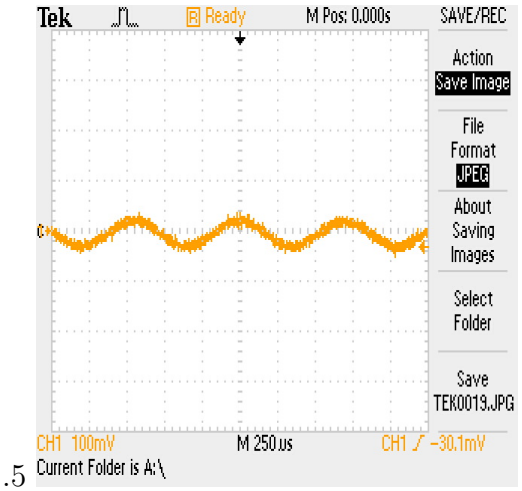


FIGURE 3.8: fin=2.2kHz

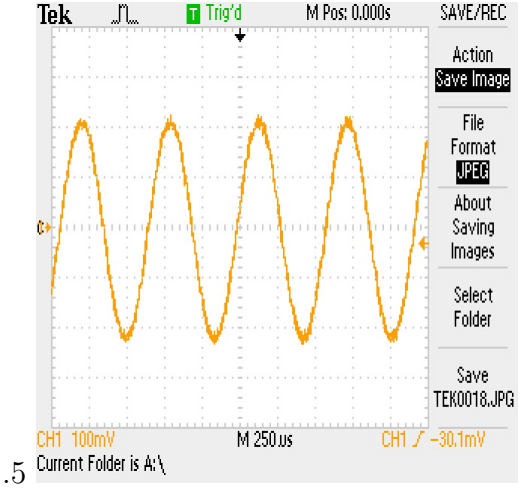


(c)



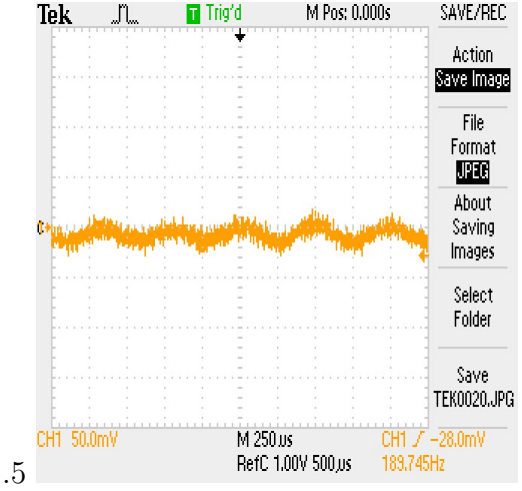
.5

FIGURE 3.11: fin=1.43kHz



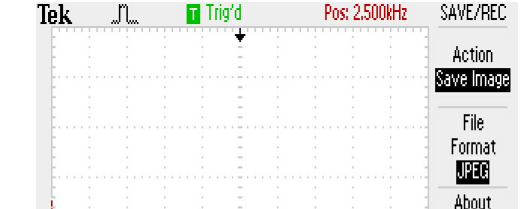
.5

FIGURE 3.12: fin=1.7kHz

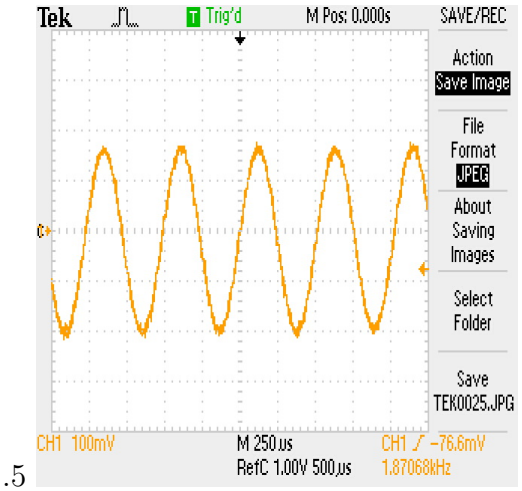


.5

FIGURE 3.13: fin=2.1kHz

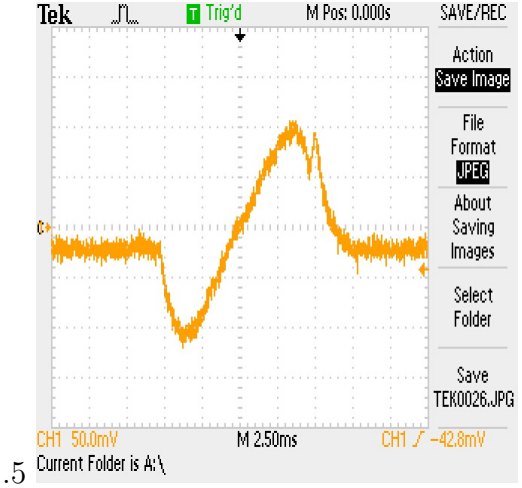


(d)



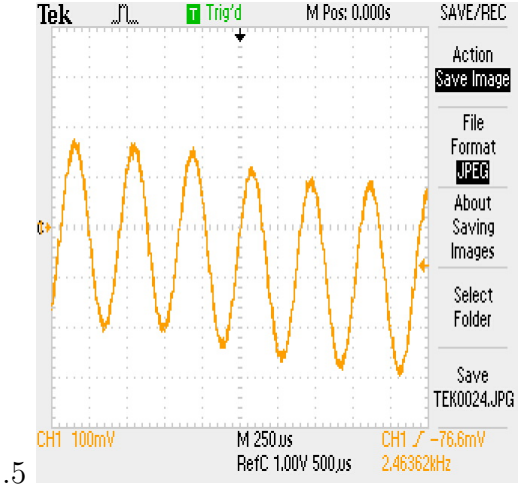
.5

FIGURE 3.17: fin=1.95kHz



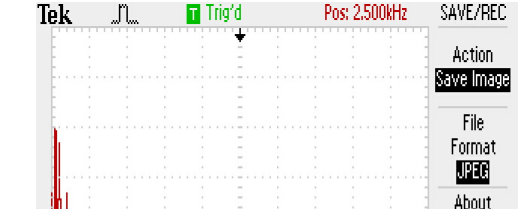
.5

FIGURE 3.18: fin=2.25kHz



.5

FIGURE 3.19: fin=2.55kHz



Observations

Cut-off frequencies were observed as follows:

(a) Low pass filter, fc= 1.63kHz

(b) High pass filter,fc=1.9kHz

(c) Band pass filter, fc1=1.38kHz ,fc2=2.12kHz

(d) Band stop filter,fc1=2.12kHz , fc2=2.38kHz

Results

Designed low pass,high pass,band pass and band stop filters of the given specifications using circular buffer.

Question 2 : Design a suitable filter for the removal of noise from the given audio signal amd listen noise free radio.

Theory

If we know the frequency range of the audio signal,then we can filter out noise to some extent by passing just the frequencies in the audio signal.

Algorithm

Step1: Start

Step2: Import the necessary headerfiles

Step3: Initialise DSK6713

Step4: Initialise i=0 and do steps 4.1 to 4.4 till$i < (N + BL)$

4.1: Initialise y=0

4.2: Initialise j=0 and do steps 4.3 till$j < BL$

4.3: $y = y + h[i - j] * B[j]$

4.4: Output the sample y to CRO

Step5: Go to step 4

Step6: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"fdacoefsbpfinal.h"
#include"noise.h"
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
short i,j; float y;
/*
 * main.c
 */
int bpf(){
for(i=0;i<(N+BL);i++){
y=0;
for(j=0;j<BL;j++)
y=y+h[i-j]*B[j];
y=y*1;//printf("%f \t",y);
output_sample((short) y);
}
```

```
return 0;
}

int main(void) {
comm_poll();
DSK6713_init();
while(1){
bpf();
}
return 0;
}
```

Output

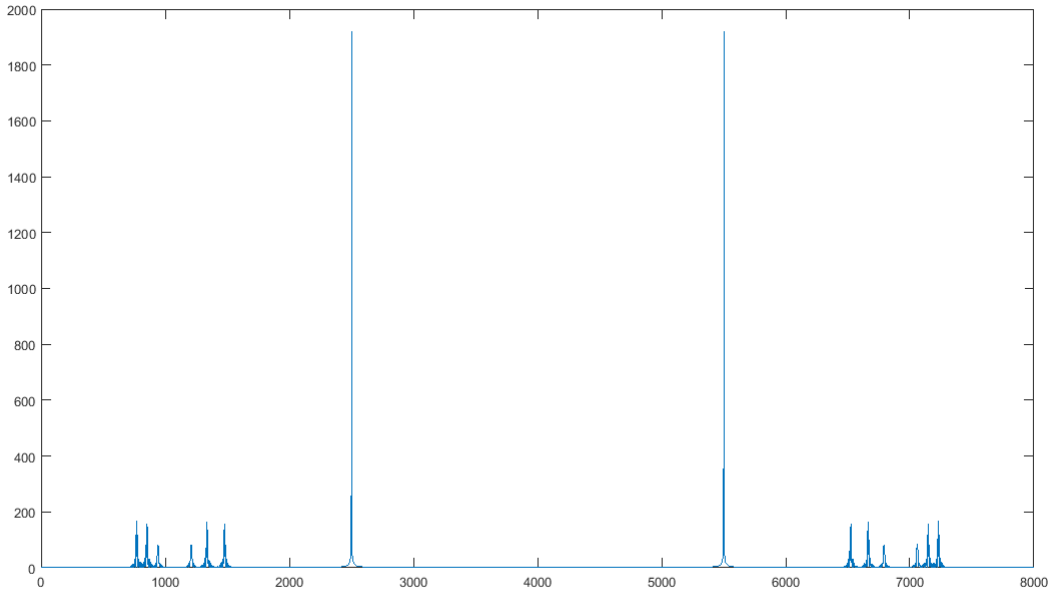


FIGURE 3.23: Spectrum of the noisy audio signal

The spectrum of the noisy audio signal was plotted in MATLAB by taking 8000 samples. From the spectrum, it was clear that the noise was present at frequencies 2200 Hz to 2800 Hz and hence a band stop filter which has cut-off frequencies

fc1=2000 Hz and fc2=3000Hz was designed using fdatool and the values were exported as "fdacoefsbpfinal.h" to CCS. FIR filtering was done in CCS and heard the noise free DTMF sound.

Results

Designed a BSF for the removal of noise from the given audio signal.

Question 3 : Take voice as input to the DSP and apply the discussed audio effects on it in real-time. Write your code to produce the echo effect when SWITCH 1 is pressed and the reverberation effect when SWITCH 2 is pressed.

Theory

Sound is a mechanical wave which travels through a medium from one location to another.Echo and reverberation (or reverb) refers to the effect of sounds reflected off solid objects, such as walls or ceilings in a theater or rocks in a valley. They are differentiated by the length of time between the initial sound and the reflected repetition. When reflected sound travels a greater distance, such as a river valley, and takes more than one-tenth second to return, it is referred to as an echo.
Echo does not add to the original sound as reverberation does, but is perceived as a distinct repetition of the sound, usually slightly fainter than the original. The sound is weaker because of the energy lost as the sound waves travel the greater distance. This is referred to as decay. Echo can be measured by the time lapse between repetitions, strength of the repetitions (i.e., how loud the repetition is) and the decay of the sound.
A reverberation is perceived when the reflected sound wave reaches your ear in less than 0.1 second after the original sound wave. Since the original sound wave is

still held in memory, there is no time delay between the perception of the reflected sound wave and the original sound wave. The two sound waves tend to combine as one very prolonged sound wave.

Algorithm

Step1: Start

Step2: Import required header files

Step3: Initialise DSK6713,the values of α and D

Step4: Check whether switch 0 is pressed, if yes, goto step 5, else, goto step 9

Step5: For $i < 8000$, do steps 5,6,7,8

Step6:Get input sample x[i]

Step7: Calculate y[i]= αx[i-D]+x[i]

Step8: Output the samples y[i]

Step9: For$i < 8000$, do steps 5,6,7,8

Step10: Get input sample x[i]

Step11: Calculate y[i]= αy[i-N]+Dx[i]

Step12: Output the samples y[i]

Step13: Goto step 4

Step14: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
```

```
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
short x[8000];
short y[8000];
short i=0,j;
short k;
int echo(){
for(i=0;i<8000;i++){
x[i] = input_sample();

y[i] = x[i]+ 5*x[i-8000];
//printf("%hi \t",x[i]);
k=1*y[i] ;
output_sample((short) k);
}
return 0;
}
int rev(){
for(i=0;i<8000;i++){
x[i] = input_sample();
// for(i=0;i<8000;i++){
y[i] = x[i]+ 0.25*y[i-400];
//printf("%hi \t",x[i]);
k=1*y[i] ;
output_sample((short) k);
}
```

```
return 0;


}
int main(void) {
DSK6713_init();
DSK6713_LED_init();
DSK6713_DIP_init();
comm_poll();
while(1){
if(DSK6713_DIP_get(0)==0)
{ DSK6713_LED_on(0);
echo();}
else if (DSK6713_DIP_get(1)==0)
{DSK6713_LED_on(1);rev();}
} return 0;
}
```

Results

Heard the echo of the audio signal when switch 0 is pressed and reverberation effect when switch 1 is pressed.

Question 4 : Echo

Experiment with different values of the delay parameter 'D' . What is the minimum possible delay(in ms) for which you can discern the echo?

Theory

Echo sound can be produced by taking an audio signal and playing it after adding it with its own delayed version. The difference equation is given by, y[n]= αx[n-D] + x[n]

Algorithm

Step1: Start

Step2: Import required headerfiles

Step3: Initialise DSK6713,value of D and α

Step4: For$i < 8000$, do steps 5,6,7,8

Step5: Get input sample x[i]

Step6: Calculate y[i]= αx[i-D]+x[i]

Step7: Output the samples y[i]

Step8: Goto step 4

Step9: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"

void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
short x[8000];
short y[8000];
short i=0,j;
```

```
short k;
int echo(){
for(i=0;i<8000;i++){
x[i] = input_sample();

y[i] = x[i]+ 5*x[i-8000];
//printf("%hi \t",x[i]);
k=1*y[i] ;
output_sample((short) k);
}
return 0;
}

int main(void) {
DSK6713_init();
DSK6713_LED_init();
DSK6713_DIP_init();
comm_poll();
while(1){

echo();}
return 0;
}
```

Observations

When the value of D was increased from 1ms to 2s, the effect of echo reduced. When D=1s, the effect of echo was no longer perceived.

Inferences

The minimum possible delay for which we can discern the echo is 1s.

Results

Experimented with different values of D ranging from 1ms to 2s and found out the minimum possible delay for which we can discern the echo to be 1s.

Question 5 : Reverberation

For a given delay 'D', run the program when $\alpha = 0.25$. Then observe the effect of changing 'α' to 0.9. What is the maximum value that 'α' can have?

Theory

In reverberation, each delayed sound wave arrives in a short period of time that we donot perceive each reflection as a copy of the original sound. The effects of combining an original signal with a very short time delayed version of the same signal results in reverberation. The difference equation is y[n]=ay[n-N] + Dx[n]

Algorithm

Step1: Start

Step2: Import required headerfiles

Step3: Initialise DSK6713,value of D and α

Step4: For $i < 8000$, do steps 5,6,7,8

Step5: Get input sample x[i]

Step6: Calculate y[i]= αy[i-N]+Dx[i]

Step7: Output the samples y[i]

Step8: Goto step 4

Step9: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"

void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
short x[8000];
short y[8000];
short i=0,j;
short k;
int rev(){
for(i=0;i<8000;i++){
x[i] = input_sample();
// for(i=0;i<8000;i++){
y[i] = x[i]+ 0.25*y[i-400];
//printf("%hi \t",x[i]);
k=1*y[i] ;
output_sample((short) k);
}
return 0;}}
```

```
int main(void) {
DSK6713_init();
DSK6713_LED_init();
DSK6713_DIP_init();
comm_poll();
while(1){

rev();}
return 0;
}
```

Observations

When the value of α is changed from 0.25 to 0.9 , the effect of reverberation becomes better and better, but when it is increased from 0.9 , the effect of reverberation is very high that it destroys the quality of the signal. D was kept at 400 samples or 50ms.

Inferences

Maximum value that α can have is 0.9.

Results

Experimented on different values of α for a given delay D.

Question 6 : Input a 1kHz sine wave and add random noise to it. Pass it through the moving average filter. Increase the number of taps from 5 to 31 and observe the results.(Use interrupts)

Theory

The moving average filter is a simple Low Pass FIR (Finite Impulse Response) filter commonly used for smoothing an array of sampled data/signal. It takes N samples of input at a time and take the average of those N-samples and produces a single output point.

$y[n] = b0x[n] + b1x[n-1] + b2x[n-2] + ... + bNx[n-N]$

Algorithm

Step1: Start

Step2: Import required headerfiles

Step3: Initialise DSK6713

Step4: For $i < 8000$, do steps 5,6,7,8

Step5: Get input sample x[i]

Step6: Create a random number using rand() and add it to the sample x[i].

Step7: Add the results from steps 5 and 4 to get noisy input xn[]

Step8: Implement for each n from 0 to N-1:

$y[i] = (x[i] + x[i-1] + x[i-2] + ... + x[i-N])/N$

Step9: Output y[i] to CRO

Step10: Goto step 4

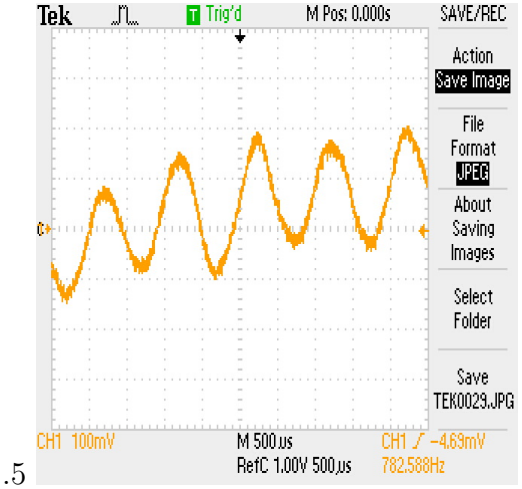
Step11: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"

void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
//extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0011;
float x[8000];
float y[5];
short i=0,j;
float k;

interrupt void c_int11(){
while(i<8000){
x[i]=0;
x[i]=(float) (input_left_sample());
k=(float) (rand()/5);
//printf("%hi \t",i);
x[i]=x[i] + k;

y[i]=0;
```

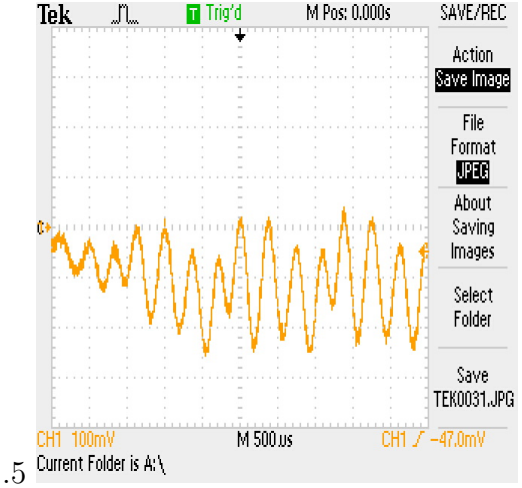
```
for(j=0;j<15;j++){
// printf("%f\t ",x[i-j]);
if(i-j > 0)
y[i] = y[i] + (x[i-j] /15);
else
y[i]=y[i];
}
i++;
//y[i]=1*y[i];
output_left_sample((short) y[i]);}
if(i>8000){
i=0;}
return;
}

int main(void) {

DSK6713_init();
comm_intr();
while(1);
return 0;
}
```

Output



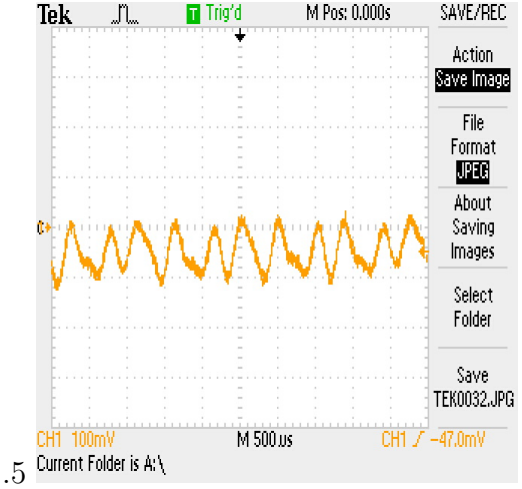
.5

FIGURE 3.24: Output when number of taps=5



.5

FIGURE 3.25: Output when number of taps=15



.5

FIGURE 3.26: Output when number of taps=31

FIGURE 3.27: Increasing the number of taps from 5 to 31 of a moving average filter

Observations

As the filter length increases (the parameter N or the number of taps) the smoothness of the output increases, whereas the sharp transitions in the data are made increasingly blunt.

Inferences

This filter has excellent time domain response but a poor frequency response. The delay involved increases as the number of taps increases.

Results

Created a noisy sine wave and filtered it using a moving average filter.

Chapter 4

IIR Filter Design

1. Design a Type-1 , second order Chebyshev low-pass filter with 2dB of passband ripple and a cut-off frequency of 1500 Hz

Provide a sine wave input from the signal generator to the DSK and observe the output by varying the frequency of the signal. Verify that the filter attenuates the signal beyond the cutoff frequency.

Theory

Chebyshev filters are used to seperate one band of frequencies from another. The primary attribute of Chebyshev filters is their speed. This is because they are carried over by recursion rather than convolution. The Chebyshev response is a mathematical strategy for achieving fastre roll-off by allowing ripple in the frequency response. Chebyshev type 1 filters have ripple only in the passband.

Algorithm

Step1: Start

Step2: Import the required headerfiles

Step3: Initialise DSK6713

Step4: Collect input samples in array x

Step5: Recursively calculate v[n] and y[n] using the equations till n=8000

v[n]=x[n]-a[0][0]*v[n-1]-a[0][1]*v[n-2]

y[n]=b[0][0]*v[n]+b[0][1]*v[n-1]+b[0][2]*v[n-2]

Step6: Output y to CRO

Step7: Stop

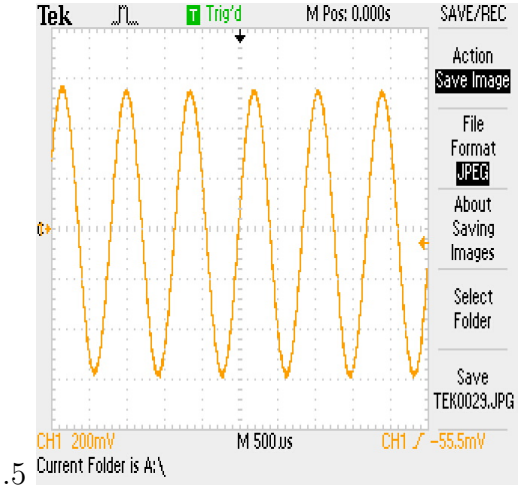
Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"iircheb.h"
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
extern void output_sample();
//extern void outputsample(int);
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 input_sample();
Uint16 inputsource=0x0011;
int i,n;
short x1[8000];
float x[8000],y[8000],v[8000];
/*
 * main.c
 */
```

```
int main(void) {
comm_poll();
while(1){
for(i=0;i<8000;i++)
{
x1[i]=input_sample();
x[i] = (float) x1[i];

}
DSK6713_LED_on(3);
for(n=0;n<8000;n++){
if(n-2 > 0)
{
v[n]=x[n]-a[0][0]*v[n-1]-a[0][1]*v[n-2];
y[n]=b[0][0]*v[n]+b[0][1]*v[n-1]+b[0][2]*v[n-2];
}
else if(n-1 > 0 && n-2<0)
{v[n]=x[n]-a[0][0]*v[n-1];
y[n]=b[0][0]*v[n]+b[0][1]*v[n-1];
}
else
{v[n]=x[n];
y[n]=b[0][0]*v[n];

}

output_sample((short) y[n]);
}

}
```

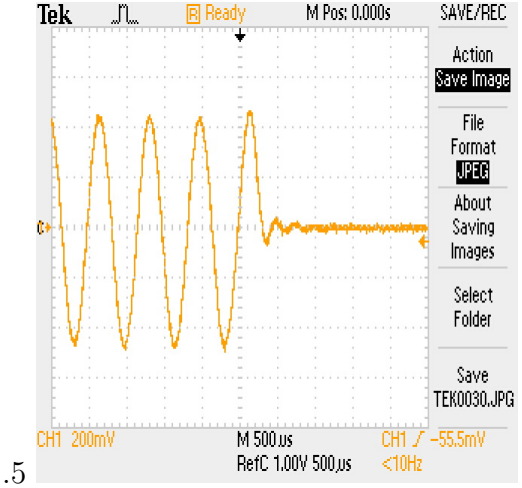
```
return 0;
}
```

Outputs



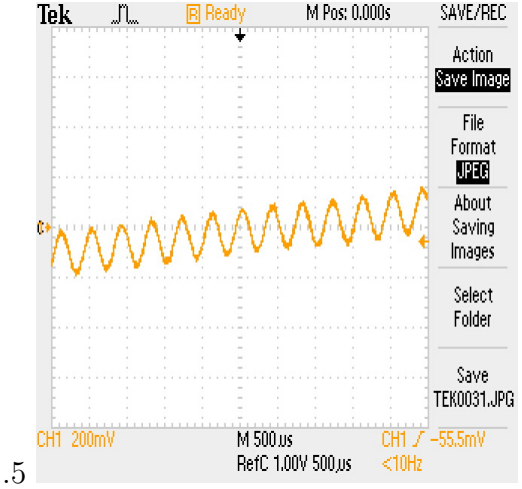
.5

FIGURE 4.1: Frequency=1.178 kHz



.5

FIGURE 4.2: Frequency=1.5 kHz



.5

FIGURE 4.3: Frequency=2.5 kHz

Observations

Output is same as input till an input frequency of 1.5 kHz and it filters out all frequencies above 1.5 kHz.

Inferences

Chebyshev LPF has faster roll-off.

Results

Designed a Type-1 , second order Chebyshev low-pass filter with 2dB of passband ripple and a cut-off frequency of 1500 Hz.

2. Design a 30 order IIR filter with the same specifications as above and comment on its characteristics.

Theory

IIr filters are digital filters with infinite impulse response. Unlike FIR filters, they have feedback and are known as recursive filters. IIR filters have much better frequency response than FIR filters.
The impulse response of an IIR filter can be shown as: $a0y[n] = a1y[n-1] + a2y[n-2] + ... + b0x[n] + b1x[n-1] + b2x[n-2] + ...$

Algorithm

Step1: Start

Step2: Import the required headerfiles

Step3: Initialise DSK6713

Step4: Collect the input 8000 samples into array x

Step5: Recursively calculate v[n] and y[n] using the equations till n=8000

v[n]=x[n]-a[num][0]*v[n-1]-a[num][1]*v[n-2];

y[n]=b[num][0]*v[n]+b[num][1]*v[n-1]+b[num][2]*v[n-2];

Step6: Output y to CRO

Step7: Stop

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"iircheb1.h"
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
extern void output_sample();
//extern void outputsample(int);
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 input_sample();
Uint16 inputsource=0x0011;
int i,n,num;
short x1[8000];
float x[8000],y[8000],v[8000];
/*
 * main.c
 */
int main(void) {
```

```
comm_poll();
while(1){
for(i=0;i<8000;i++)
{
x1[i]=input_sample();
x[i] = (float) x1[i];

}
DSK6713_LED_on(3);
for(num=0;num<15;num++)
{
for(n=0;n<8000;n++){

if(n-2 > 0)
{
v[n]=x[n]-a[num][0]*v[n-1]-a[num][1]*v[n-2];
y[n]=b[num][0]*v[n]+b[num][1]*v[n-1]+b[num][2]*v[n-2];
}
else if(n-1 > 0 && n-2<0)
{v[n]=x[n]-a[num][0]*v[n-1];
y[n]=b[num][0]*v[n]+b[num][1]*v[n-1];
}
else
{v[n]=x[n];
y[n]=b[num][0]*v[n];

}
}
for(n=0;n<8000;n++)
x[n]=y[n];
```

```
}for(n=0;n<8000;n++)
output_sample((short) y[n]);

}



return 0;
}
```

Output

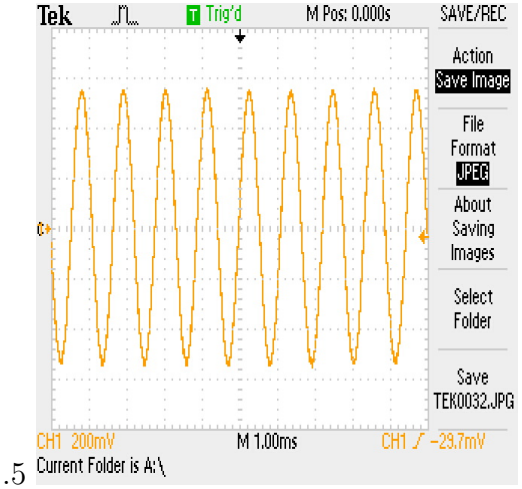


FIGURE 4.4: Input Frequency = 900 Hz

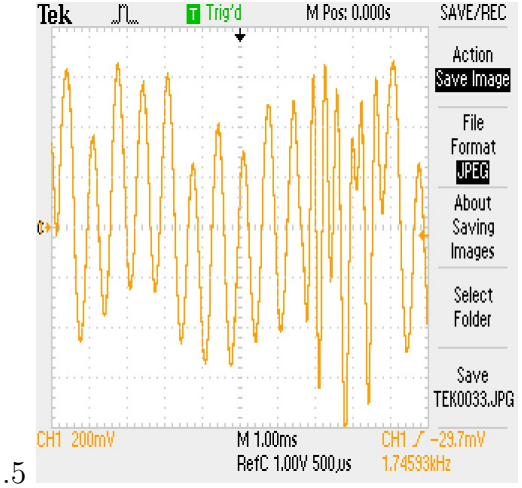


FIGURE 4.5: Input Frequency=1.5kHz

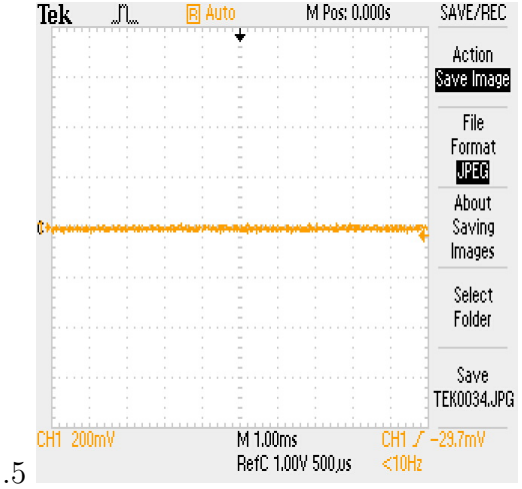


FIGURE 4.6: Input Frequency=1.6kHz

Observations

Input is same as output and the cutoff frequency is less than 1.5 kHz.

Inferences

IIR filters can be used to filter out a band of frequencies(here, more than 1.5kHz) and can be implemented digitally unlike Butterworth and Chebyshev, which are analog filters.

Results

Designed an IIR low pass filter of order 30 and a cutoff frequency of 1.5kHz.

3. Design a suitable IIR filter for the removal of noise from the given audio signal and listen noise free audio. Compare the result with the experimental exercise of last labsheet.

Theory

If we know the range of audio frequency and the range of noise frequency, we can filter out the noise frequencies by using suitable filter whose coefficients can be obtained by using fdatool in MATLAB and then creating a headerfile of it, which can be called in CCS.

Algorithm

Step1: Start

Step2: Import the required headerfiles

Step3: Initialise DSK6713

Step4: Call the function iir() and goto step 5

Step5: Initialise i=0 , goto step 5.1 if$i < 8000$

5.1: Calculate $y[i] = b0 * h[i] + b1 * h[i-1] + b2 * h[i-2] - a0 * y[i-1] - a1 * y[i-2]$

5.2:$i = i + 1$

5.3: Output y[i] to CRO

Step6: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"iirqn3lab4.h"
#include"noise.h"
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
short i;
float v,v1=0,v2,y;
void iir(){
y=0;
y=b[0][2] *h[0]; v2=v1;v1=y;
output_sample((short) 10000*y);
y=b[0][2] * h[1] + b[0][1] * h[0]- a[0][0] *v1 - a[0][1] *v2;
```

```
v2=v1;v1=y;
output_sample((short) 10000*y);
for(i=2;i<N;i++){
//v=h[i]-(a[0][0]*v1)-(a[0][1]*v2);
//y=b[0][0]*v + b[0][1] * v1 + b[0][2]*v2;
y=(b[0][2] * h[i] + b[0][1] * h[i-1] + b[0][0] * h[i-2] - a[0][0] *v1 - a[0][1]
v2=v1;
v1=y;

//v2=v1;
//v1=v;
y=y*10000;
output_sample((short) y);
}
return ;}
int main(void) {
comm_poll();
DSK6713_init();

while(1){
iir();
}
return 0;
}
```

Output and Observations

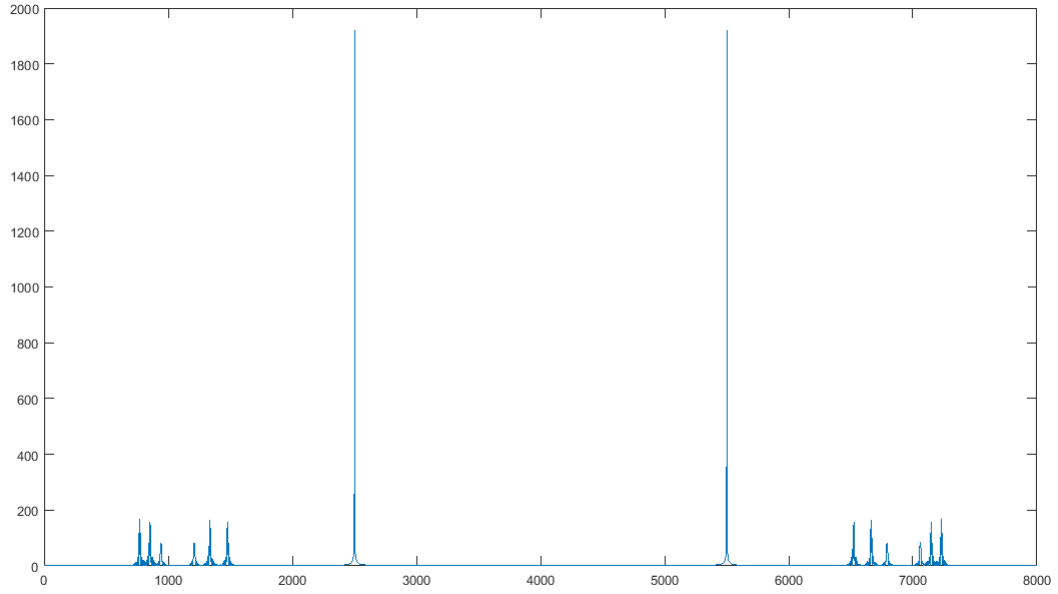


FIGURE 4.7: Magnitude spectrum of the noisy audio sample

As seen in the previous labsheet, the noise is present at frequencies between 2000 Hz and 3000 Hz and by using a band stop filter with cutoff frequencies at fc1=2000Hz and fc2=3000 Hz , we can get back the original audio signal which is a DTMF signal . Here, the filter designed is an IIR band stop Chebyshev type 2 filter, designed using fdatool and the coefficents were included in a headerfile created by the function *dsksosiir*67(). It is then exported to CCS to filter out the noisy signal. The denoised DTMF signal was heard.

Inferences

IIR filter is difficult to implement compared to FIR filter because of its recursive nature. If the initial output was wrong, it will be added again for the next samples and hence error propogation can happen. FIR filters have linear phase. It ensures that signals of all frequencies are delayed by the same amount of time, thereby eliminating the possibility of phase distortion and hence FIR filters are better for

audio applications. The number of coefficients is less in IIR filter and hence , it uses less memory.

Results

Filtered the noisy audio sample and listened to noisefree audio.

Chapter 5

Code Optimization using ASM functions

1. In Lab3, you have designed FIR filter using C program for different filtering operations. Use profiling methods to profile your code.

Theory

CCS supports function profiling and code coverage along with simulator events in all simulators except 28x. Function profiling provides information on number of times functions called,inclusive and exclusive ttal cycle each functions took to execute and number of simulator events within the functions.

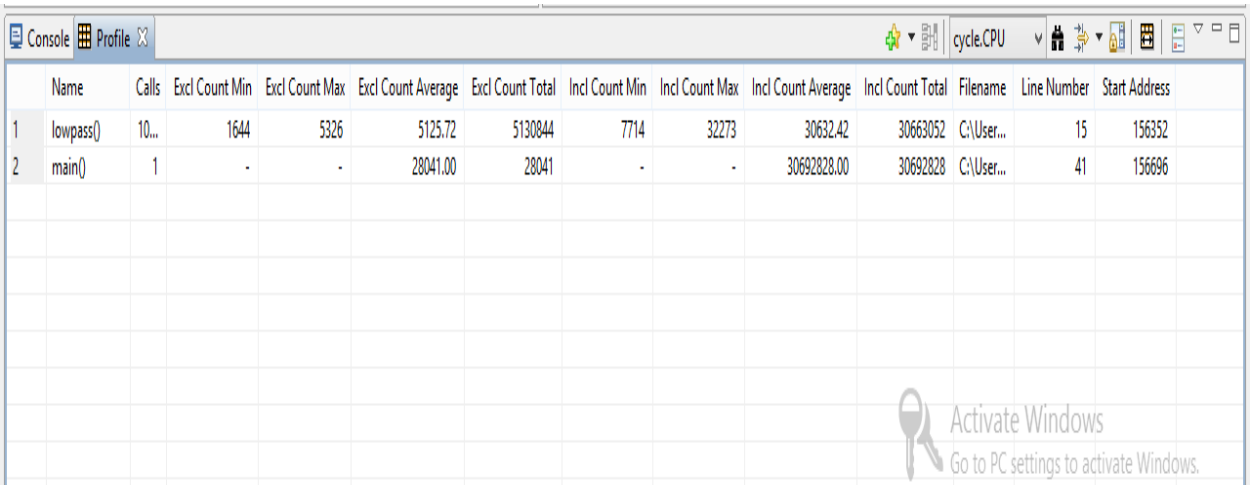
Program

```
#include<stdio.h>
//#include"dsk6713.h"
//#include"dsk6713_led.h"
//#include"dsk6713_dip.h"
//#include"dsk6713_aic23.h"
```

```
#include"fdacoefsvoice.h"
#include"noise.h"
//void DSK6713_init();
//void DSK6713_DIP_init();
//void DSK6713_LED_init();
//extern void comm_poll();
//Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
//Uint16 inputsource=0x0015;
int i=0,j;
int lowpass(){

float y[16100],am;

//DSK6713_LED_off(0);

{//DSK6713_LED_on(0);
y[i]=0;
for(j=0;j<100;j++)
{
//DSK6713_LED_off(0);
//output_sample((float) h[j]);
if(i-j>=0)
{y[i]=y[i]+(h[i-j]*Bvoice[j]);
}}
am=100*y[i];
printf("%f\n",am);
//output_sample((float) am);
i++;
}
```

```
return 0;
}
int main(void) {
// comm_poll();
while(i<=1000){
lowpass();}
return 0;
}
```

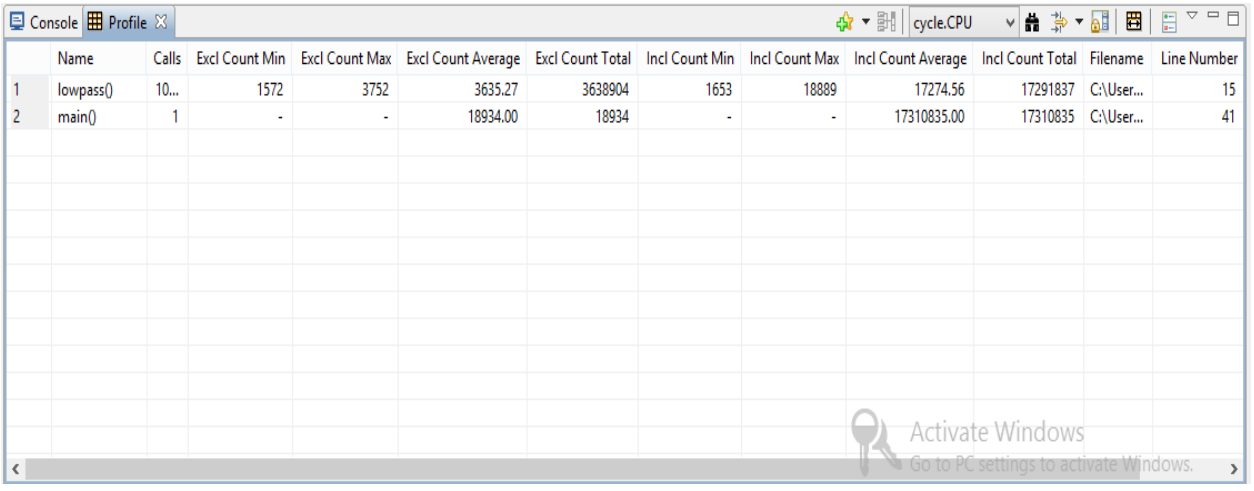
Output



	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filename	Line Number	Start Address
1	lowpass()	10...	1644	5326	5125.72	5130844	7714	32273	30632.42	30663052	C:\User...	15	156352
2	main()	1	-	-	28041.00	28041	-	-	30692828.00	30692828	C:\User...	41	156696

1

FIGURE 5.1: Without optimization



	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filename	Line Number
1	lowpass()	10...	1572	3752	3635.27	3638904	1653	18889	17274.56	17291837	C:\User...	15
2	main()	1	-	-	18934.00	18934	-	-	17310835.00	17310835	C:\User...	41

1

FIGURE 5.2: After optimization

Observations

Counts	Exclusive count average	Inclusive Count average
Before optimization	5125.72	30632.42
After optimization	3635.27	17274.56

Inferences

The number of exclusive and inclusive counts are reduced when code profiling is done. When clock profiling was done, the clock cycles that the main function took were 3, 5799, 699 ,etc ., for each call of the function low pass() inside the main function, which is also low.

Results

Performed code profiling using ASM codes.

2.Write a C callable ASM function that has the same prototype and performs the same task as your C function.

Theory

Assembly instructions and directives can be incorporated within a C program using the asm statement. The asm statement can provide access to hardware features that cannot be obtained using C code alone. When an ASM function is called inside the C program, the arguments are passed through the registers A4,B4,A6,B6, and so on. The assembly function's name is preceded by an underscore . Note that only registers A1,A2,B0,B1 and B2 can be used as conditional registers.

Program

```
#include<stdio.h>
#include<math.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"fdacoefs.h"
#include"noise.h"
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0015;
int yn=0,j=0;
float *dly;
short i=0;
/*
 * main.c
 */
void lpf()

{
*dly=h[j];
yn=fircasmfunc(dly,B,BL);
output_sample((short) yn);
return;
}
```

```
void main() {

comm_poll();
while(1){
for(j=0;j<N;j++){
//printf("%f",h[j]);
lpf();}}

return ;
}
------ASM---------------------------------
;FIRcasmfunc.asm ASM function called from C to implement FIR
;A4 = Samples address, B4 = coeff address, A6 = filter order
;Delays organized as:x(n-(N-1))...x(n);coeff as h[0]...h[N-1]

.def _fircasmfunc
_fircasmfunc: ;ASM function called from C
MV A6,A1 ;setup loop count
MPY A6,2,A6 ;since dly buffer data as byte
ZERO A8 ;init A8 for accumulation
ADD A6,B4,B4 ;since coeff buffer data as byte
SUB B4,1,B4 ;B4=bottom coeff array h[N-1]
loop: ;start of FIR loop
LDH *A4++,A2 ;A2 = x[n-(N-1)+i] i=0,1,...,N-1
LDH *B4--,B2 ;B2 = h[N-1-i] i=0,1,...,N-1
NOP 4
MPY A2,B2,A6 ;A6 = x[n-(N-1)+i]*h[N-1-i]
NOP
ADD A6,A8,A8 ;accumlate in A8
```

```
LDH *A4,A7 ;A7=x[(n-(N-1)+i+1]update delays
NOP 4 ;using data move "up"
STH A7,*-A4[1] ;-->x[(n-(N-1)+i] update sample
SUB A1,1,A1 ;decrement loop count
 [A1]  B loop ;branch to loop if count # 0
NOP 5
MV A8,A4 ;result returned in A4
B B3 ;return addr to calling routine
NOP 4
```

Results

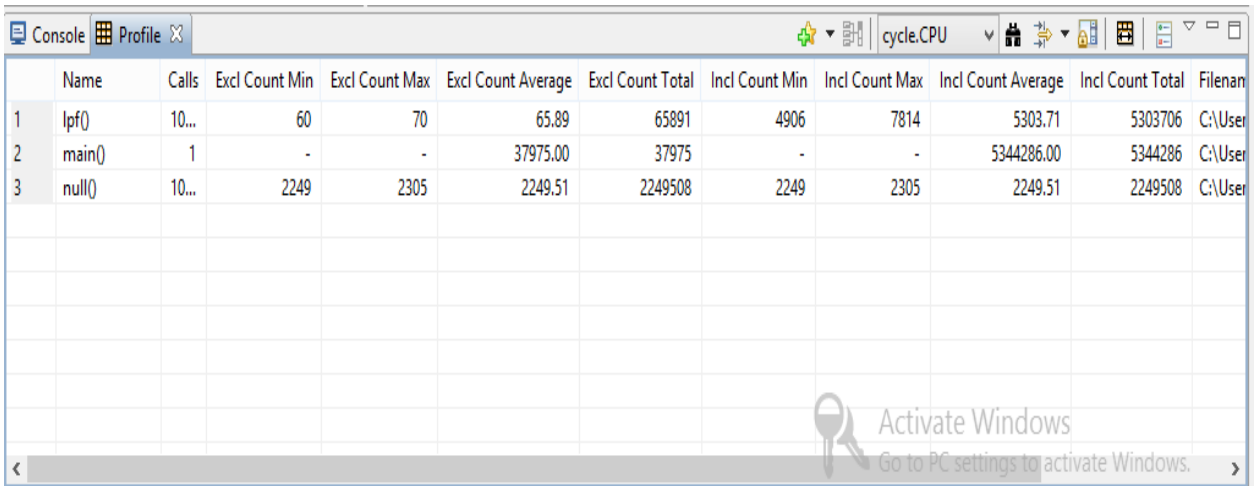
A C callable ASM function was written that performs the same function as the C function in question 1.

3. Now use this mechanism in the first question and obtain the profiling results. Record your observations.

Program

Same as in question 2.

Output



	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filenan
1	lpf()	10...	60	70	65.89	65891	4906	7814	5303.71	5303706	C:\User
2	main()	1	-	-	37975.00	37975	-	-	5344286.00	5344286	C:\User
3	null()	10...	2249	2305	2249.51	2249508	2249	2305	2249.51	2249508	C:\User

1

FIGURE 5.3: Profiling results using ASM function

Observations

By using ASM function, the profiling results are better. The ASM function has an exclusive count average of just 65.89 and an inclusive count average of 5303.71 .

Inferences

Writing ASM code functions inside the C function reduces the exclusive and inclusive counts an dhence is optimal. Hand optimization by writing ASM codes is the best but it is very much time consuming and difficult.

Result

Obtained profiling results for the C callable ASM function written for question 2.

4. Implement a 256 tap FIR band pass filter, with a centre frequency of 15kHz, when sampling at 48 kHz. Implement this filter to achieve 2kHz bandwidth.

Theory

A finite impulse response (FIR) filter is a filter whose impulse response is of finite duration, because it settles to zero in finite time. The FIR filter is designed using windowing. FIR filter requires no feedback is stable and linear phase, and hence, suitable for audio applications.

Program

```
#include<stdio.h>
#include<math.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"fdacoefsbpf48.h"
//#include"noise.h"
void DSK6713_init();
//void DSK6713_DIP_init();
//void DSK6713_LED_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_48KHZ;
Uint16 inputsource=0x0011;
int yn=0,j=0;
short dly[101];
```

```
//short i=0;
/*
 * main.c
 */
void bpf()

{ dly[BL-1]=input_sample();
//dly=h[j];

yn=fircasmfunc(dly,B,BL);

output_sample((short) (yn>>15));

return;
}
void main() {
short i;
DSK6713_init();
for(i=0;i<BL;i++)
dly[i]=0;
comm_poll();
while(1)
{
bpf();}
return ;}
```

Output

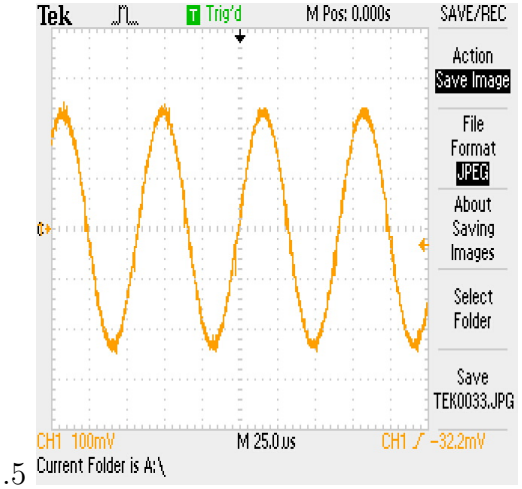


FIGURE 5.4: Input frequency = 15kHz

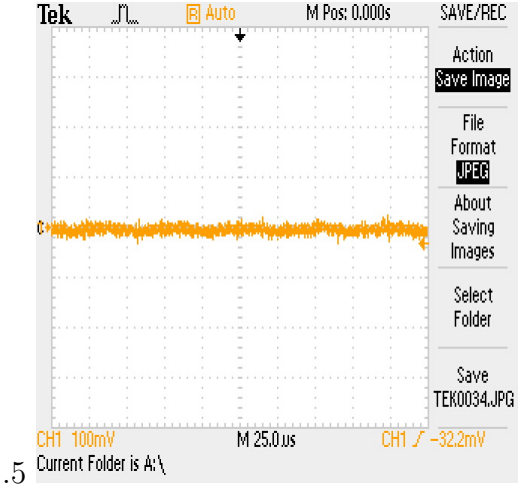


FIGURE 5.5: Outside pass band, fin=19kHz

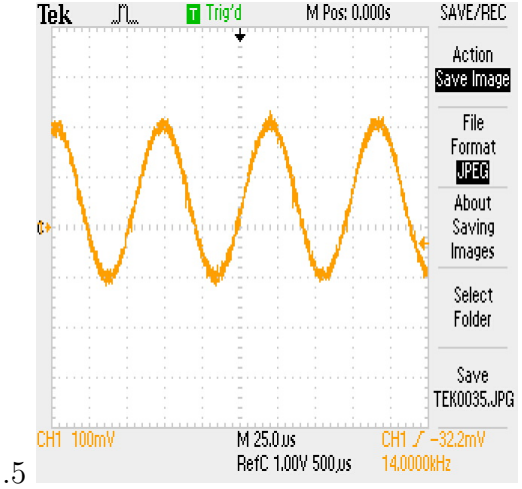


FIGURE 5.6: Inside pass band , fin=14kHz

FIGURE 5.7: Output of the filter at different input frequencies

Observations

The filter passes the signals between 14kHz and 16kHz only.

Inferences

When sampling fequency set for CODEC is high (oversampling) , the number of samples will be high, storage and computations are more. But when the code is written in assembly language, the memory requirement can be compromised as we are sending just the memory locations and not the bytes and hence, the input samples neednot be stored in a fixed size array. Proper output was obtained using C callable ASM function using both interrupt and polling.

Results

Designed and implemented a 256 tap FIR bandpass filter with a center frequency of 15kHz and sampling at 48kHz.

Chapter 6

FIR and IIR filter structure using ASM functions

1. Modify the C-callable ASM for FIR in the previous labsheet to achieve a faster implementation. Verify using the profiling method.

Theory

There are one cross path in each side of the two data paths and hence two instructions can be loaded in a cycl. Also, pipelining feature is there, which helps in faster implementation. By using two data paths in a single clock cycle, the number of clock cycles for operation can be reduced and hence faster implementation is possible. This can be done manually, by rearranging the ASM code such that instructions that can be implemented in different data paths are grouped together by a '—— ' sign before the next instruction. In other words parallel instructions are given.

Program

```
#include<stdio.h>
```

```
#include<math.h>
//#include"dsk6713.h"
//#include"dsk6713_led.h"
//#include"dsk6713_dip.h"
//#include"dsk6713_aic23.h"
#include"fdacoefs.h"
#include"noise.h"
//void DSK6713_init();
//void DSK6713_DIP_init();
//void DSK6713_LED_init();
//extern void comm_poll();
//Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
//Uint16 inputsource=0x0015;
int yn=0,j=0;
float *dly;
short i=0;
/*
 * main.c
 */
void lpf()

{ //dly[100]=input_sample();
*dly=h[j];
//printf("%f",*dly);
yn=fircasmfunc(dly,B,BL);
printf("%d \t",yn);
//output_sample((short) yn);
//j=j+1;
//j=j%16000;
//}
//printf("hai");
```

```
return;
}



void main() {

//printf("hai");
//for(i=0;i<101;i++)

//dly[i]=0;



//comm_poll();
//while(1){
for(j=0;j<1000;j++){
// *dly=h[j];
// yn=fircasmfunc(dly,B,BL);
// printf("%d \t",yn);
//printf("%f",h[j]);
lpf();}
//}
//while(1);
//{j++;
// j=j%16000;
//}
return ;
}
ASM:
;FIRcasmfunc.asm ASM function called from C to implement FIR
```

```
;A4 = Samples address, B4 = coeff address, A6 = filter order
;Delays organized as:x(n-(N-1))...x(n);coeff as h[0]...h[N-1]

.def _fircasmfunc
_fircasmfunc: ;ASM function called from C
MV A6,A1 ;setup loop count
MPY A6,2,A6 ;since dly buffer data as byte
ZERO A8 ;init A8 for accumulation
ADD A6,B4,B4 ;since coeff buffer data as byte
SUB B4,1,B4 ;B4=bottom coeff array h[N-1]
loop: ;start of FIR loop
LDH *A4++,A2 ;A2 = x[n-(N-1)+i] i=0,1,...,N-1
||   LDH *B4--,B2 ;B2 = h[N-1-i] i=0,1,...,N-1
SUB A1,1,A1 ;decrement loop count
LDH *A4,A7 ;A7=x[(n-(N-1)+i+1]update delays
NOP 4
STH A7,*-A4[1] ;-->x[(n-(N-1)+i] update sample
[A1]  B loop ;branch to loop if count # 0
NOP 2
MPY A2,B2,A6 ;A6 = x[n-(N-1)+i]*h[N-1-i]
NOP
ADD A6,A8,A8 ;accumlate in A8
B B3 ;return addr to calling routine
MV A8,A4 ;result returned in A4
NOP 4
```

Output

.5

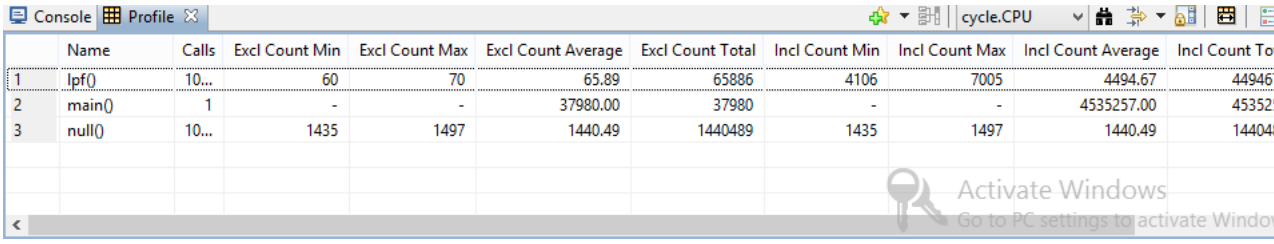


FIGURE 6.1: Profiling output using C-callable ASM

Observations

Counts	Exclusive count average	Inclusive Count average
C callable ASM	65.89	5303.71
Modified C callable ASM	65.89	4494.67

Inferences

Modified C callable ASM function has faster implementation and uses less clock cycles as the ASM codes are regrouped so as to perform parallel implementation.

Results

Verified using profiling that using a C callable ASM function,we can get faster implementation.

2.Write a linear assembly code to replace your C-callable function and verify your results.

Theory

A linear assembly code uses the C code variables in the ASM code and the hardware registers neednot be specified.An assembler optimizer is used with a linear assembly - coded source program to create an assembly source program in much the same way that a C compiler optimizer is used in conjunction with a C - coded source program. The resulting assembly- coded program produced by the assembler optimizer is typically more effi cient than one resulting from the C compiler optimizer.Specifying the functional unit is optional in a linear assembly program as well as in an assembly program.Parallel instructions are not possible in this code.It is saved as .sa file.

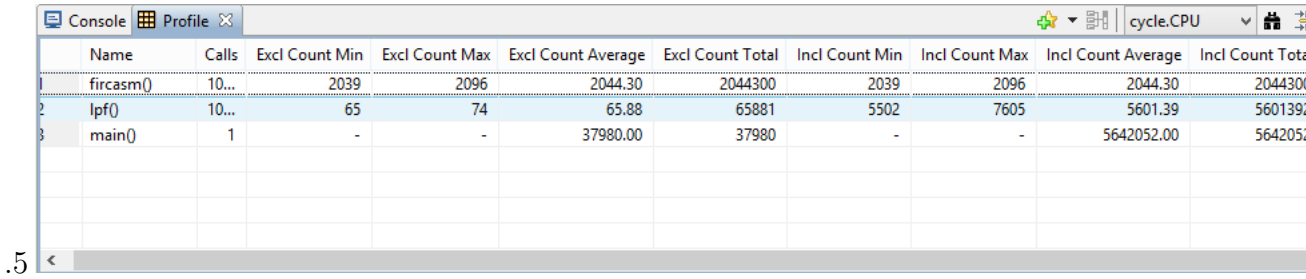
Program

```
C:
#include<stdio.h>
#include<math.h>
//#include"dsk6713.h"
//#include"dsk6713_led.h"
//#include"dsk6713_dip.h"
//#include"dsk6713_aic23.h"
#include"fdacoefs.h"
#include"noise.h"
//void DSK6713_init();
//void DSK6713_DIP_init();
//void DSK6713_LED_init();
//extern void comm_poll();
//Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
```

```
//Uint16 inputsource=0x0015;
int yn=0,j=0;
float *dly;
short i=0;
/*
 * main.c
 */
void lpf()

{ //dly[100]=input_sample();
*dly=h[j];
//printf("%f",*dly);
yn=fircasm(dly,B,BL);
printf("%d \t",yn);
//output_sample((short) yn);
//j=j+1;
//j=j%16000;
//}
//printf("hai");

return;
}



void main() {

//printf("hai");
//for(i=0;i<101;i++)

//dly[i]=0;
```

```
//comm_poll();
//while(1){
for(j=0;j<1000;j++){
// *dly=h[j];
// yn=fircasmfunc(dly,B,BL);
// printf("%d \t",yn);
//printf("%f",h[j]);
lpf();}
//}
//while(1);
//{j++;
// j=j%16000;
//}
return ;
}

.ref _fircasm
_fircasm: .cproc ap,bp,count
  .reg a,b,prod,sum
  zero sum
loop: ldh *ap++,a
ldh *bp++,b
   mpy a,b,prod
   add prod,sum,sum
   sub count,1,count
[count]    b   loop
   .return sum
   .endproc
```

Output



.5

FIGURE 6.2: Profiling result using linear assembly code

Observations

Counts	Exclusive count average	Inclusive Count average
C code	3635.27	17274.56
C callable ASM	65.89	5303.71
Modified C callable ASM	65.89	4494.67
Linear assembly code	65.88	5601.39

Inferences

Linear assembly codes are better than C codes but not as optimized as assembly codes. It provides a compromise between coding efficiency and coding effort.

Results

A linear assembly code was written in the place of the C- callable function and verified the results.

3.FIR filters can be implemented in direct form structure. Write functions for each of the structures in C for fixed point implementations.

Theory

Finite Impulse Response filters design consists of the approximation of a transfer function with a resulting set of coeffi cients. These coefficients can be created in the fdatool of MATLAB. A direct form structure is the simplest FIR structure using delay units, adders and multipliers. The different filter structures of FIR filters are direct form, cascade and lattice structure. A cascade structure consists of more than one lower order filter structures implemented in direct form, put in cascadeA lattice filter is the most efficient structure for generating at the same time the forward and backward prediction errors. A typical application of lattice filters is in linear prediction for speech processing. They can be used to model the vocal tract with an all-pole structure. Also, the lattice structure is modular,i.e. increasing the order of the filter requires adding only one extra module, leaving all other modules the same.

Algorithm

Step1: Start

Step2: Import the necessary headerfiles

Step3: Initialise DSK6713

Step4: Declare pointer variables

Step5: If switch 2 is pressed and goto step 6

Step6: Get the first sample at the location pointed by pointer p

Step7: Initialise y=0 and p=px

Step8: For i=0,do steps 9.1,9.2 till$i = BL$

8.1: Calculate$y = y + (*p - -) * B[i]$

8.2: Output the sample y

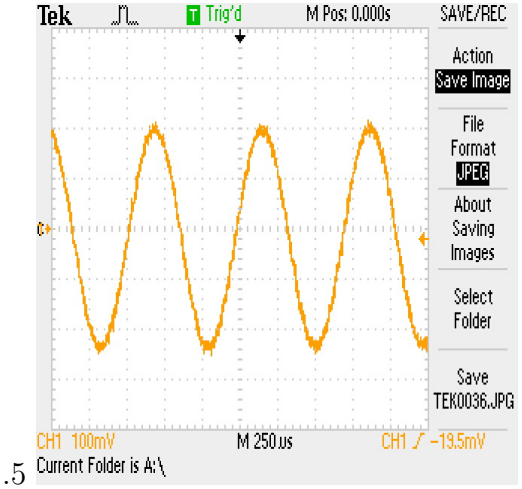
Step9: Stop

Program

```
#include<stdio.h>
#include"dsk6713.h"
#include"dsk6713_led.h"
#include"dsk6713_dip.h"
#include"dsk6713_aic23.h"
#include"fdacoef.h"
#include"fdacoefshigh.h"
#include"fdacoefs_bpf.h"
#include"fdacoefs_bsf.h"

int low(){
short x[101],*px,*p;
int i;
short y;
px=x;
while(DSK6713_DIP_get(2)==0){
*px=input_sample();
y=0;p=px;
if(++px>&x[BLlpf])
px=x;
for(i=0;i<BLlpf;i++)
{
y=y+(*p--)*Blpf[i];
if(p<&x[0])
p=&x[BLlpf];
}
```

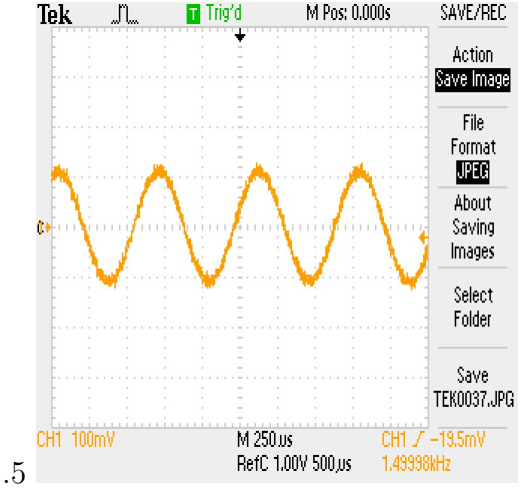
```
output_sample((short) y);
}
return 0;
}
void DSK6713_init();
void DSK6713_DIP_init();
void DSK6713_LED_init();
extern void comm_poll();
//extern void output_sample();
//extern void outputsample(int);
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
//Uint16 input_sample();
Uint16 inputsource=0x0011;

int main(void) {
comm_poll();
while(1){
if(DSK6713_DIP_get(2)==0)
low();
}

return 0;
}
```

Output



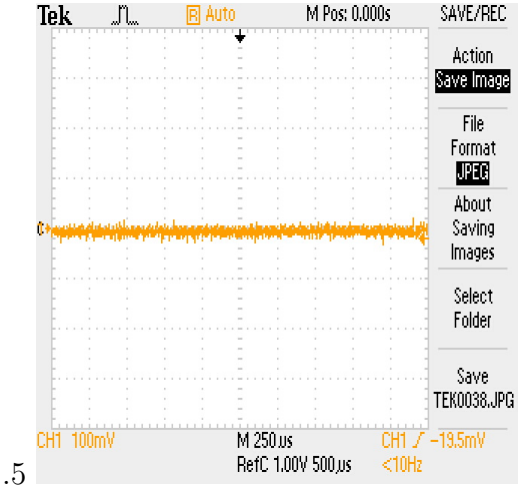
.5

FIGURE 6.3: Input-frequency = 1.4kHz



.5

FIGURE 6.4: Input-frequency = 1.5kHz



.5

FIGURE 6.5: Input-frequency = 1.8kHz

Observations

Low pass filter was implemented at a cut-off frequency of 1.5 kHz. Frequencies below 1.5kHz were passed and all other frequencies were attenuated.

Inferences

Since FIR filters donot have a feedback or a recursive part, it has only direct form structure (no seperate direct form I and II). The filter coefficients were made in fdatool of MATLAB and used in the C code written in CCS.

Results

Implemented FIR Direct form filter structure in CCS.

Chapter 7

Communication System Implementation using Simulink

Introduction to exporting Simulink model to DSK

1. Create the required block model by using Simulink library in Simulink and verify its working using time scope and spectrum analyzer.

2. Replace the input and outputs (if required) with DSK ADC and DAC blocks.

3. Save the model and change the Configuration parameters as shown below:

3.1 $Solver \rightarrow Type = Fixed - step; Solver = Discrete$

3.2 $CodeGeneration \rightarrow Systemtargetfile = idelink_ert.tlc$

3.3 $Hardwareimplementation \rightarrow ChooseTexasInstruments, C6000$

3.4 $CodeGeneration \rightarrow CoderTarget \rightarrow TargetHardwareresources \rightarrow IDE/Toolchains = TICCSv5; Board = SDC6713DSK; OS = None$

4 .Build the model in Simulink. There will be a MK file in that folder that was created when building was done in Simulink.

5. In CCS , create an empty project and give the required specifications.

6. Add the (.H, .C and .cmd) files created by Simulink when building was done.

7. Right click the Project from the Project Explorer window and click on Properties.

7.1 $C600Compiler \rightarrow AdvancedOptions \rightarrow PredefinedSymbols \rightarrow Addthesymbolsgiveninthe$

7.2 $C600Compiler \rightarrow IncludeOptions \rightarrow AddthefilesgivenintheMKfileunder"CODE_GEN_A$

7.3 $C600 Linker \rightarrow Basic Options \rightarrow Stack size = 4096; Dynamocmemory Allocation = 4096$

7.4 $C600 Linker \rightarrow File Search Path \rightarrow Add' rts6700.lib' and' cs16713.lib'$

8. Clean, Build, Run

1(a). Implement AM modulation and demodulation systems in MATLAB for a carrier frequency of 2kHz and a simple message signal of frequency of 100 Hz using basic blocks in Simulink. Verify the working of your blocks using scope and spectrum analyzer.

Theory

Modulation is defined as the process of modifying a carrierwave(radiowave) systematically by the modulating signal (audio).This process makes the signal suitable for the transmission and compatible with the channel . The resultant signal is called the modulated signal. In Amplitude modulation, the amplitude of the carrier wave is changed in accordance with the message signal. If m(t) denotes the message signal , $c(t) = Acos(wt)$ denotes the carrier wave , then the modulated signal is given by, $am(t) = A(1 + m(t))cos(wt)$. We can get back the message signal by passing it through an envelope detector and low pass filter. This process is called demodulation.

Block diagram

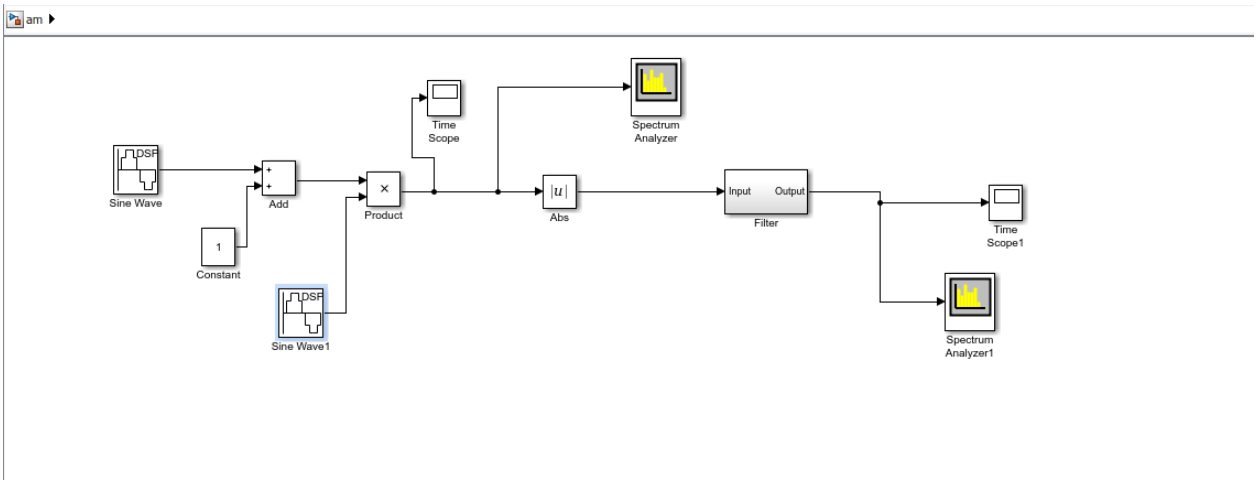
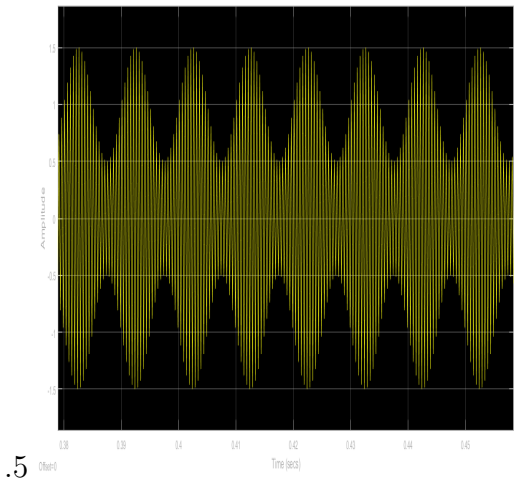


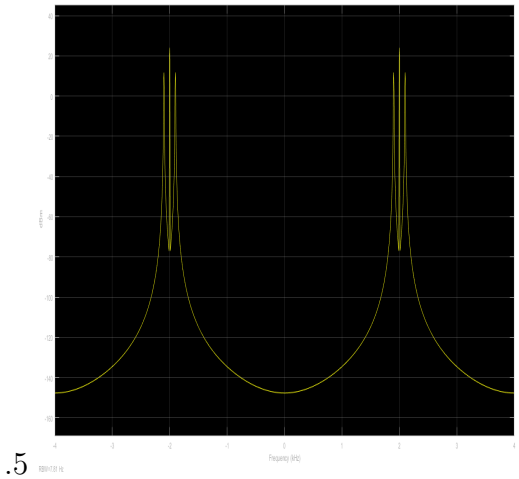
FIGURE 7.1: Block diagram

Output



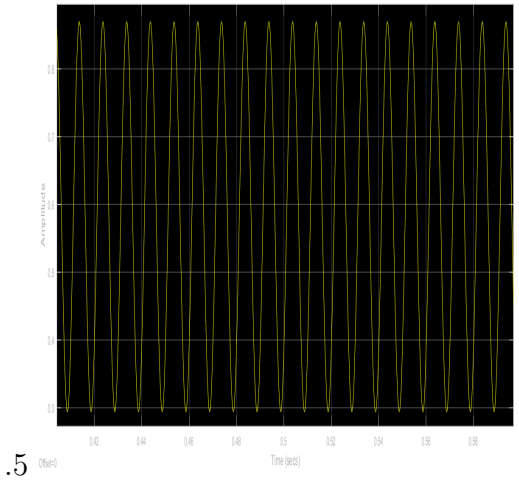
.5

FIGURE 7.2: Modulated wave



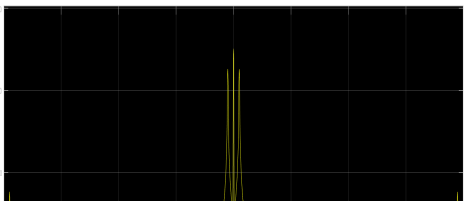
.5

FIGURE 7.3: Modulated wave - spectrum



.5

FIGURE 7.4: Demodulated wave



Results

Implemented AM modulation and demodulation systems in MATLAB for a carrier frequency of 2 kHz and a message signal of frequency of 100 Hz using basic blocks in Simulink and verified using scope and spectrum analyzer.

1(b). After completing (a), customize your design for exporting it onto the DSK hardware kit using CCS and verify its working. Record your results.

Block Diagrams

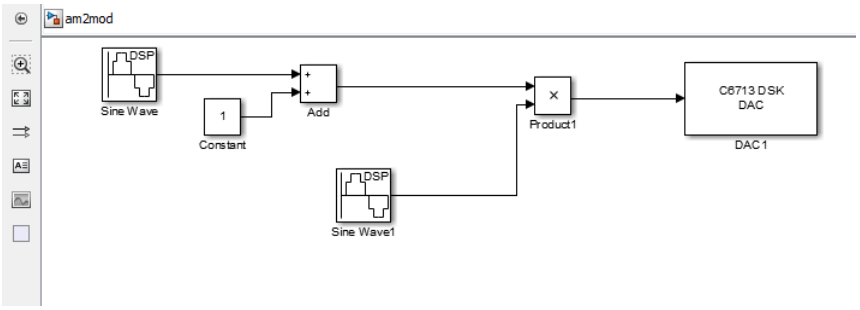


FIGURE 7.6: AM Modulator

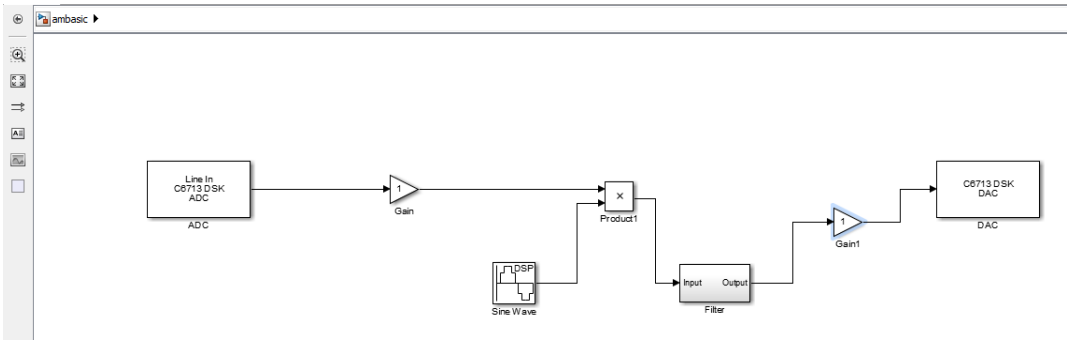


FIGURE 7.7: AM Demodulator

Output

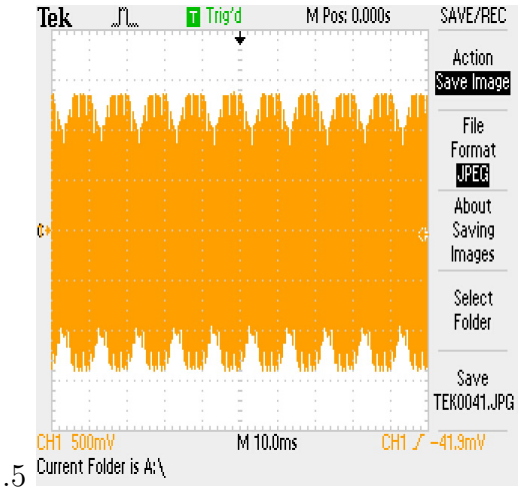


FIGURE 7.8: AM modulated wave

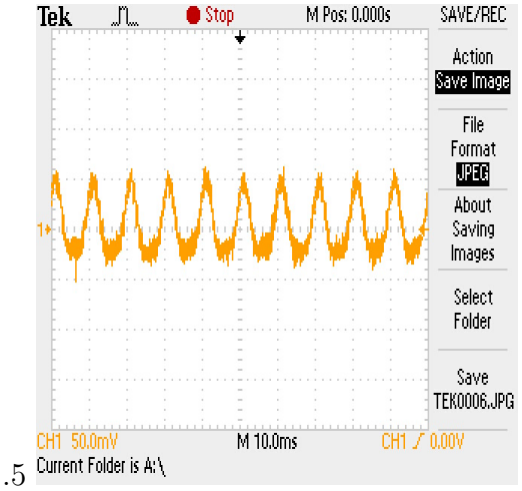


FIGURE 7.9: Demodulated wave

Results

Verified the working of AM modulator and demodulator implemented on two seperate DSK hardware.

1(c). Replace the basic blocks using Simulink's AM modulator and demodulator inbuilt blocks and export them onto the hardware and compare with (b).

Block Diagrams

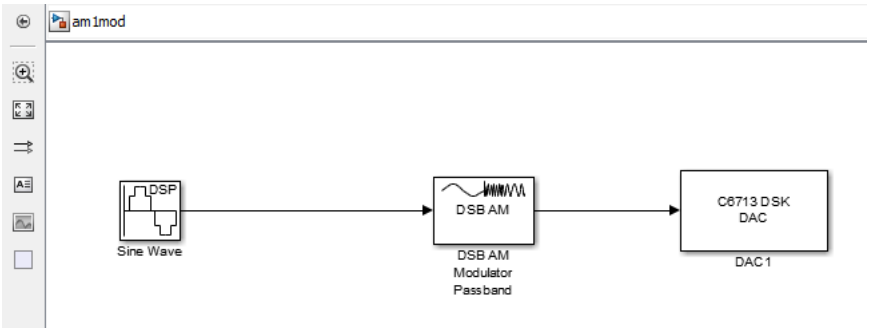


FIGURE 7.10: AM Modulator using inbuilt blocks

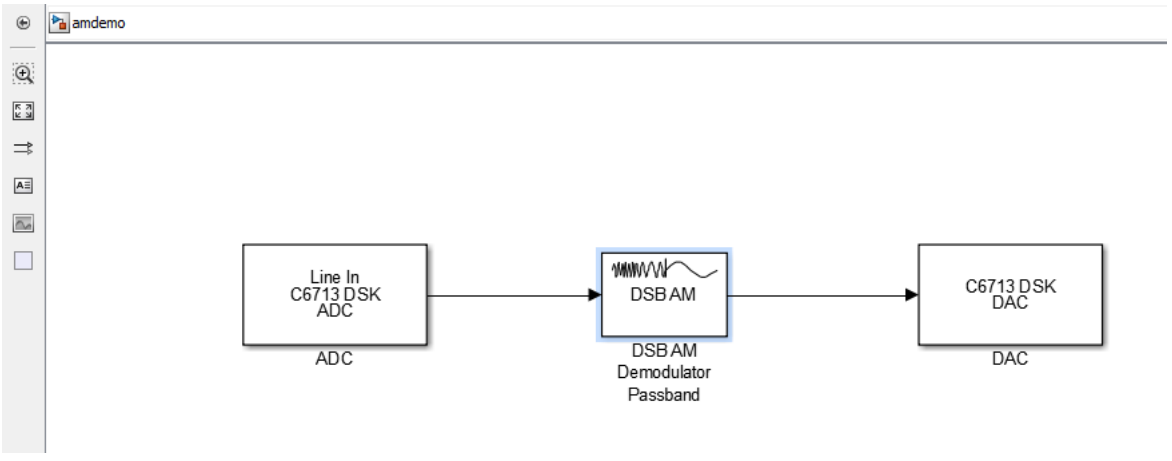


FIGURE 7.11: AM Demodulatorusing inbuilt blocks

Output

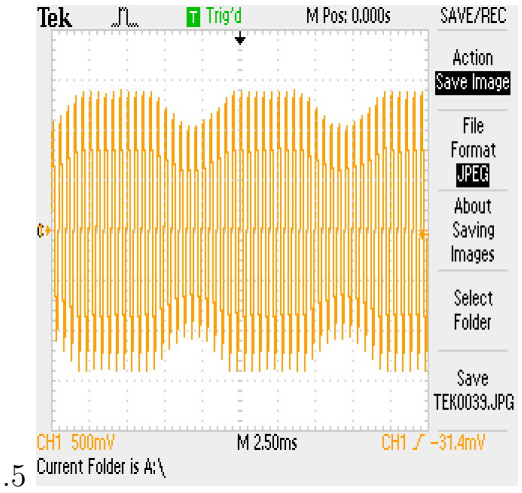


FIGURE 7.12: AM modulated wave

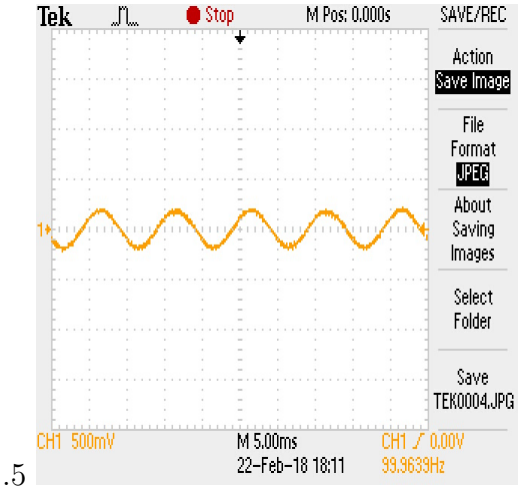


FIGURE 7.13: Demodulated wave

Inferences

As can be seen from the output waveforms, better demodulation is done when using the inbuilt block, probably because these inbuilt blocks were created in an optimized manner. Modulated output is the same for both inbuilt and basic block.

Results

AM modulator and demodulator were designed in Simulink and exported it onto DSK hardware.

1(d). Redesign your blocks to implement a speech/audio transmission and reception system using AM for DTMF signals. The maximum bandwidth of the message signal is 2kHz.

Block Diagram

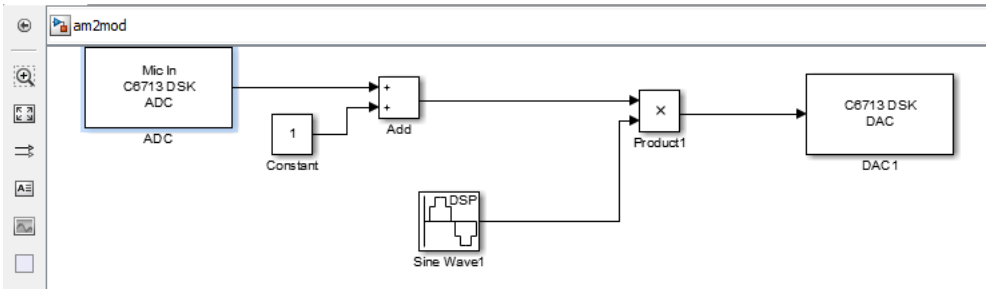


FIGURE 7.14: AM modulator of voice signal

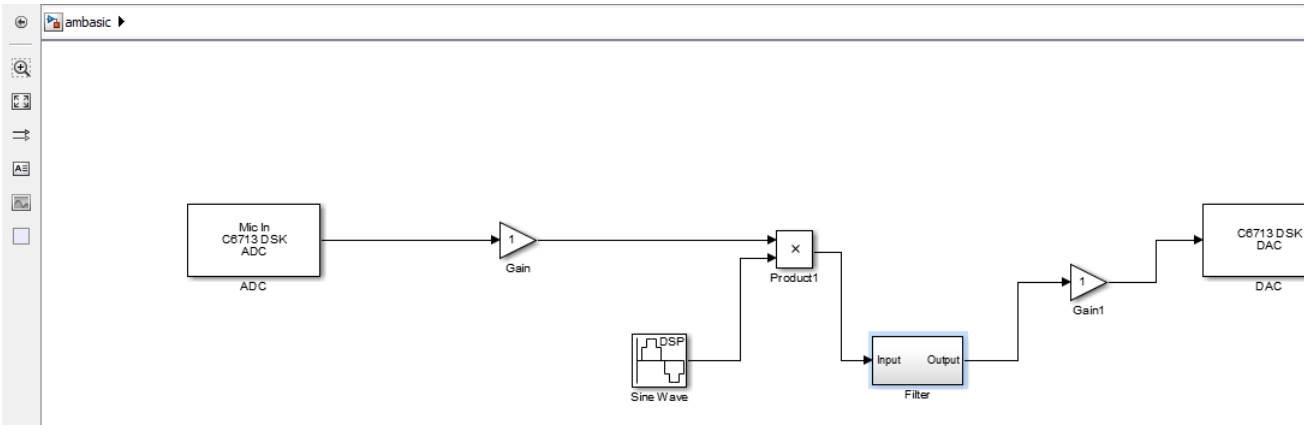


FIGURE 7.15: AM demodulator of voice signal

Observations

The given audio signal when modulated had extra sound heard along with the DTMF tone. When demodulated, this extra sound could not be heard again. This extra sound was at the carrier frequency which was removed during demodulation.

Results

The given audio signal was modulated and demodulated using AM on DSK.

2. Implement the ASK modulation and demodulation scheme in Simulink and verify its real time operation.

Theory

ASK or Amplitude Shift Keying is a digital modulation scheme whereby the strength of the carrier signal is varied to represent binary 1 or 0. Both frequency and phase remain constant while amplitude changes. Commonly one of the amplitudes is zero. It is a simple modulation technique but is highly susceptible to noise. ASK is used to transmit digital data over optical fiber.
For example,if input symbol is s and if Tb is the bit length , then ASK signal is

Block Diagram

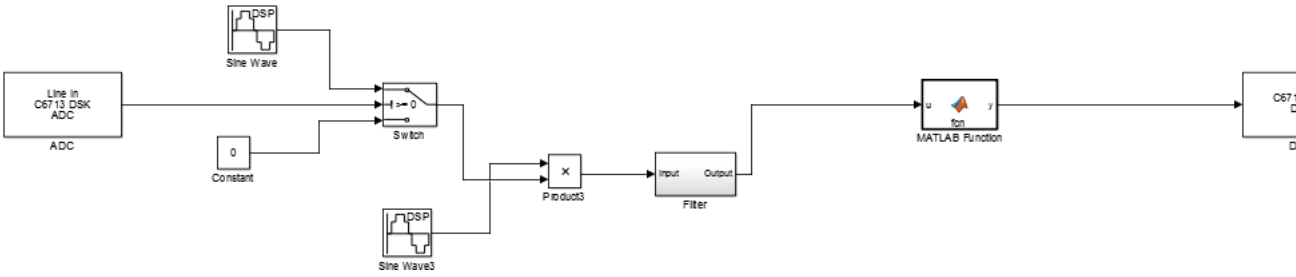


FIGURE 7.16: BASK block diagram

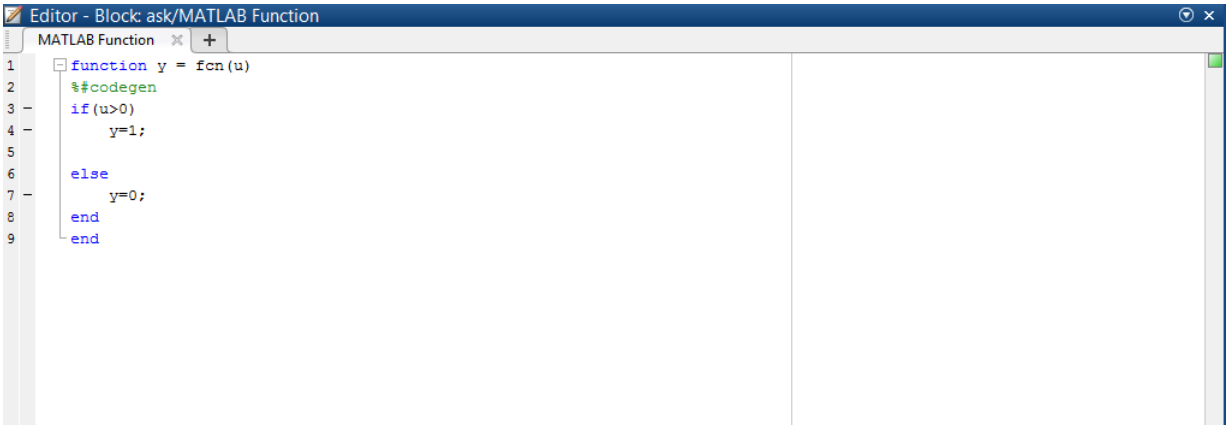


FIGURE 7.17: Decision block - matlab function

Output

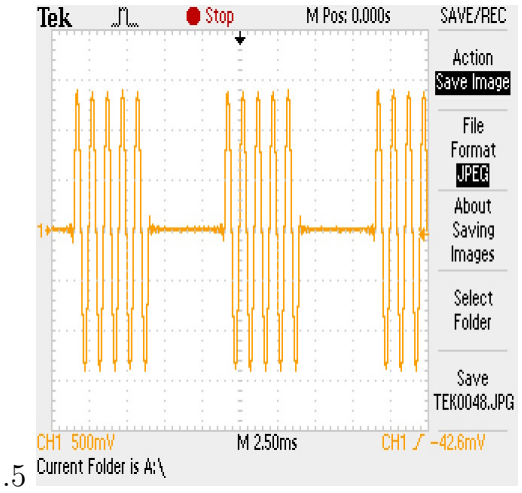


FIGURE 7.18: BASK modulated wave

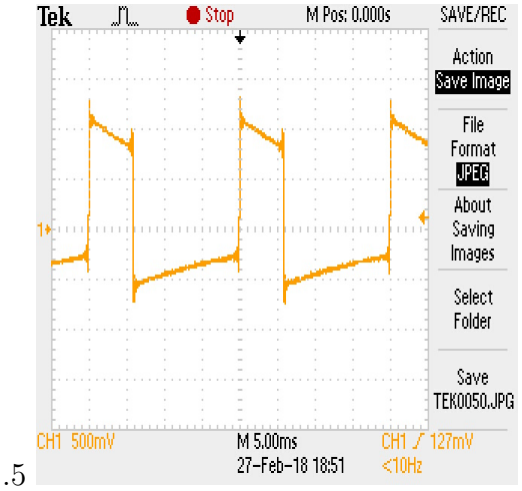


FIGURE 7.19: BASK demodulated wave

Results

Implemented ASK modulation and demodulation scheme in Simulink and verified its real time operation.

3. Repeat the above for the FSK modulation and demodulation scheme.

Theory

Frequency shift keying or FSK is a digital modulation scheme wherein bits 1 and 0 are transmitted by carriers at two different frequencies. To properly demodulate the signal at receiver, the carrier signals must be orthogonal to each other. FSK is less susceptible to noise and is used over voice lines and high frequency radio transmission.

For example, if input symbol is s and Tb is the bit length, then FSK signal is Acos(w1t) if s=1, else FSK signal is Acos(w2t). w1 and w2 are the two frequencies used.

Block Diagram

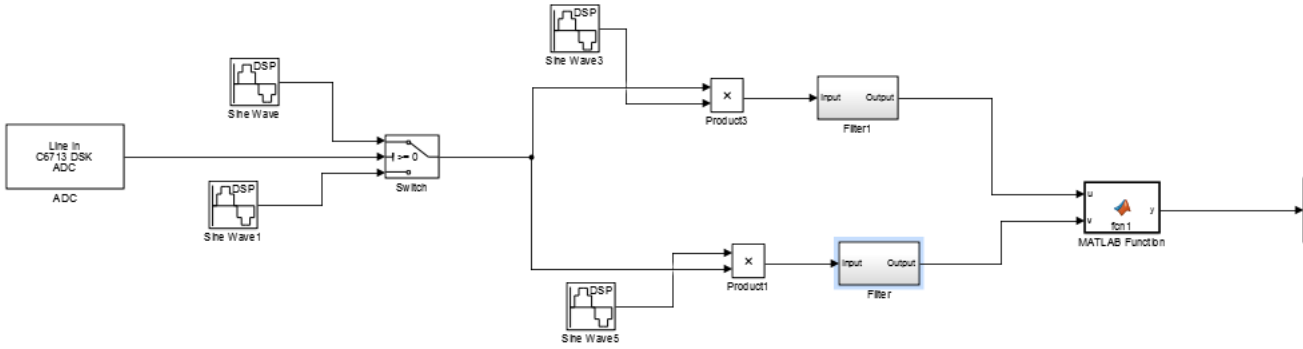


FIGURE 7.20: BFSK block diagram

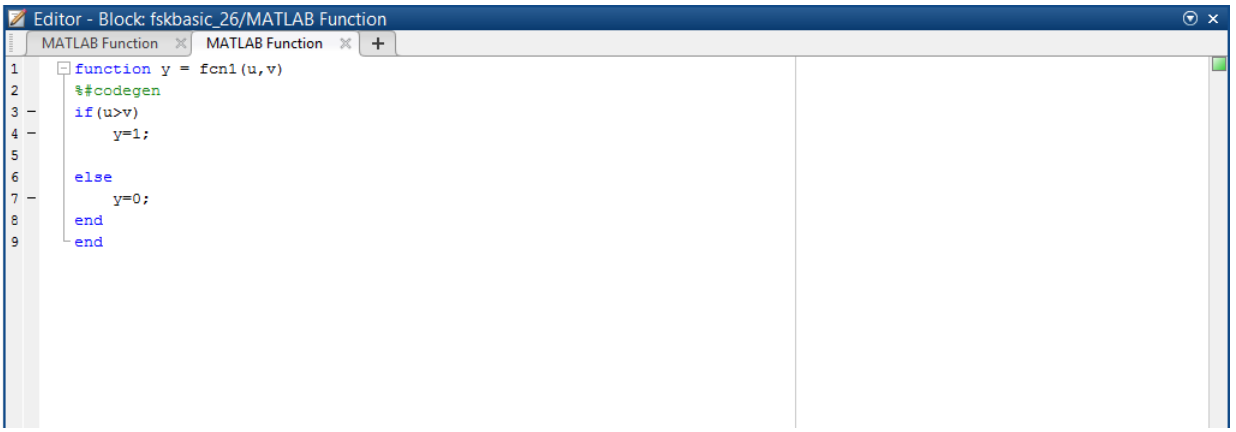
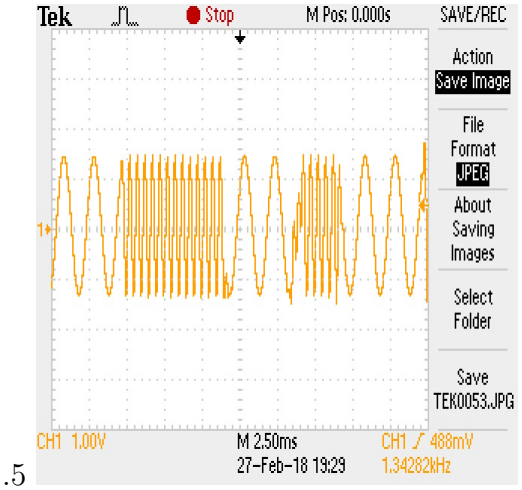


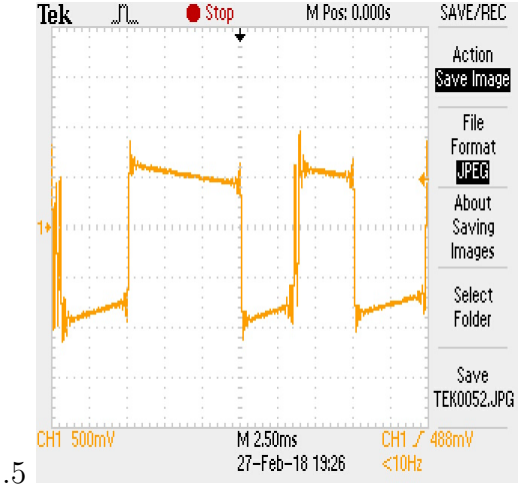
FIGURE 7.21: Decision block - matlab function

Output



.5

FIGURE 7.22: BFSK modulated wave



.5

FIGURE 7.23: BFSK demodulated wave

Results

Implemented FSK modulation and demodulation scheme in Simulink and verified its real time operation.

Chapter 8

Communication System Implementation using Simulink - Part 2

1.Implement an FM modulation system for a carrier frequency of 1kHz, message frequency of 100 Hz and frequency deviation of 200Hz using basic blocks in Simulink. Verify its real time operation in DSK6713.

Theory

In frequency modulation, the frequency of the radio carrier is changed in line with the amplitude of the incoming audio signal.When the audio signal is modulated onto the radio frequency carrier, the new radio frequency signal moves up and down in frequency. The amount by which the signal moves up and down is known as frequency deviation.Broadcast stations in the VHF portion of the frequency spectrum between 88.5 and 108 MHz use large values of deviation, typically 75 kHz. This is known as wide-band FM (WBFM). These signals are capable of supporting

high quality transmissions, but occupy a large amount of bandwidth. Usually 200 kHz is allowed for each wide-band FM transmission. For communications purposes less bandwidth is used. Narrow band FM (NBFM) often uses deviation figures of around 3 kHz.

Block diagram

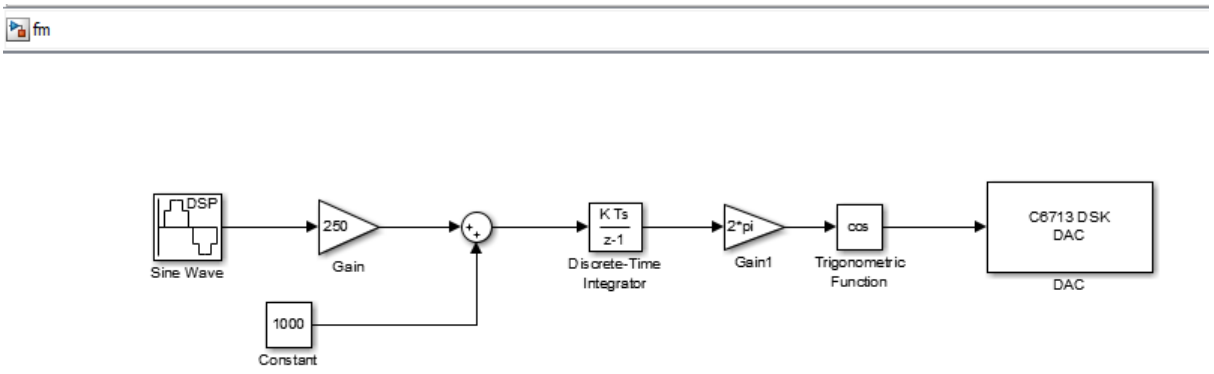


FIGURE 8.1: Block diagram

Simulink Output

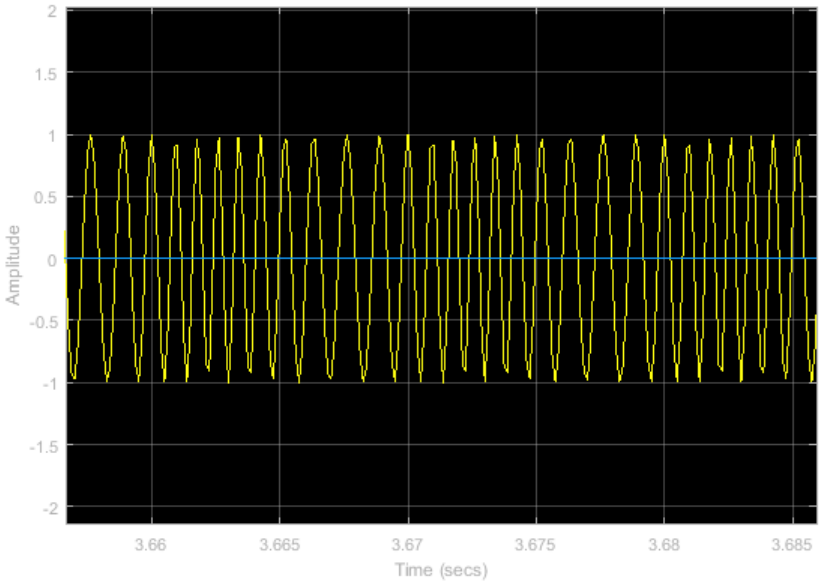


FIGURE 8.2: FM modulated signal

DSK6713 Output

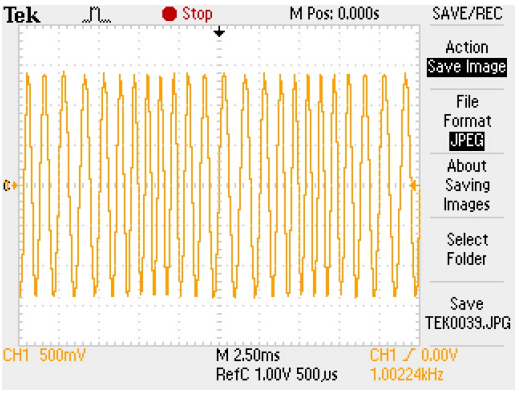


FIGURE 8.3: FM modulated signal

Observations

Observed the FM modulated wave on CRO obtained using a carrier wave of frequency 1kHz , and a message frequency of 100 Hz. Frequency deviation was 200 Hz.

Inferences

FM is resilient to noise but has poorer spectral efficiency compared to other modulation schemes. It requires complicated hardware for demodulation,like Phase Locked Loop or PLL.

Results

Verified the real time operation of a FM modulation system for a carrier frequency of 1kHz , message frequency of 100 Hz and frequency deviation of 200 Hz using basic blocks in Simulink and DSK6713.

2.Design a music equalizer according to the given specifications.

Theory

Equalization is the process of adjusting the balance between frequency components within an electronic signal.The most well known use of equalization is in sound recording and reproduction but there are many other applications in electronics and telecommunications. The circuit or equipment used to achieve equalization is called an equalizer. These devices strengthen (boost) or weaken (cut) the energy of specific frequency bands or "frequency ranges".

Algorithm

Step1: Start

Step2: Create filters and its headerfiles using fdatool

Step3: Initialise DSK6713

Step4: Do the following steps if switch 0 is pressed

4.1: Initialise y=y1=y2=y3=y4=y5=y6=0

4.2: Initialise i=0 and do steps 4.3 till$i < BL$ using $B, B2, ...B7$

4.3: $y = y + (*p) * B[i]$

4.4: $yn = y + 1.3 * y1 + 1.2 * y2 + 4 * y3 + 1.5 * y4 + 0.8 * y5 + 0.6 * y6$

4.4: Output the sample yn to CRO

Step5: Go to step 4

Step6: Stop

Program

```
#include<stdio.h>
#include<math.h>
#include <stdlib.h>
```

```
#include "dsk6713_aic23.h"
#include "fdacoefslpf.h"
#include "fdacoefsbpf.h"
#include "fdacoefsbpf2.h"
#include "fdacoefsbpf3.h"
#include "fdacoefsbpf4.h"
#include "fdacoefsbpf5.h"
#include "fdacoefsbpf6.h"
#include "dsk6713.h"
void DSK6713_DIP_init();
extern void comm_poll();
Uint32 fs=DSK6713_AIC23_FREQ_48KHZ;
Uint16 inputsource=0x0015;

short yn;
short i,j,dly[101],xn[100];
short x[101],*px,*p;
short y,y1,y2,y3,y4,y5,y6;

void lpf()
{
px=x;
while(DSK6713_DIP_get(0)==0){
*px=input_sample();
y=0;p=px;
yn=0;y1=0;y2=0;y3=0;y4=0;y5=0;y6=0;
if(++px>&x[BL])
px=x;
for(i=0;i<BL;i++)
{
y=y+(*p)*B[i];
```

```
y1=y1+(*p)*B2[i];
y2=y2+(*p)*B3[i];
y3=y3+(*p)*B4[i];
y4=y4+(*p)*B5[i];
y5=y5+(*p)*B6[i];
y6=y6+(*p--)*B7[i];
if(p<&x[0])
p=&x[BL];
}
yn=y+1.3*y1+1.2*y2+4*y3+1.5*y4 + 0.8*y5 +0.6*y6;
output_sample((short) yn);
}

return;
}
int main(void) {
DSK6713_init();
DSK6713_DIP_init();
comm_poll();
//for(i=0;i<BL;i++)
// dly[i]=0;
while(1){
lpf();}
return 0;
}
```

Observations

It was very difficult to hear the equalized audio signal when the code was created by building Simulink. The noise present was very high. When the code was directly written in C, better results were obtained.

Inferences

Audio equalisation can be performed by creating appropriate filters which form a filter bank. Here ,the frequencies 0-200 Hz were given a gain of 4 and everything else , less than 1.

Results

Designed a music equalizer according to the given specifications.

Chapter 9

Communication System Implementation using Simulink - Part3

1.Implement a BPSK modulation and demodulation system in DSK6713 from the Simulink model with basic elements.Record your observations.

Theory

This is also called as 2-phase PSK or Phase Reversal Keying. In this technique, the sine wave carrier takes two phase reversals such as 0 and π. It is given as

$$s(t) = \begin{cases} Acos(2\pi f_c t) & \text{,if bit=1} \\ -Acos(2\pi f_c t) & \text{,if bit=0} \end{cases}$$

Demodulation is done by multiplying the received signal with locally generated carrier of the same frequency and phase and then low pass filtering it.

Block diagram

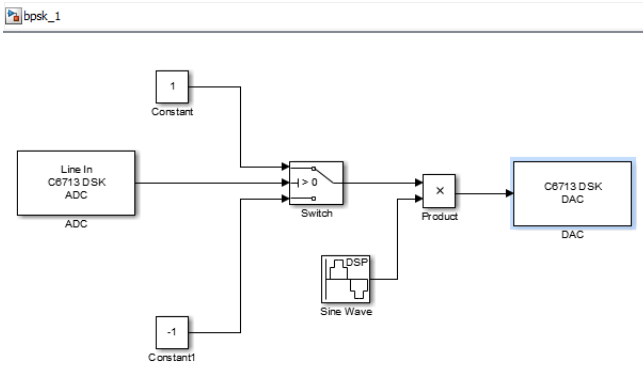


FIGURE 9.1: BPSK modulation

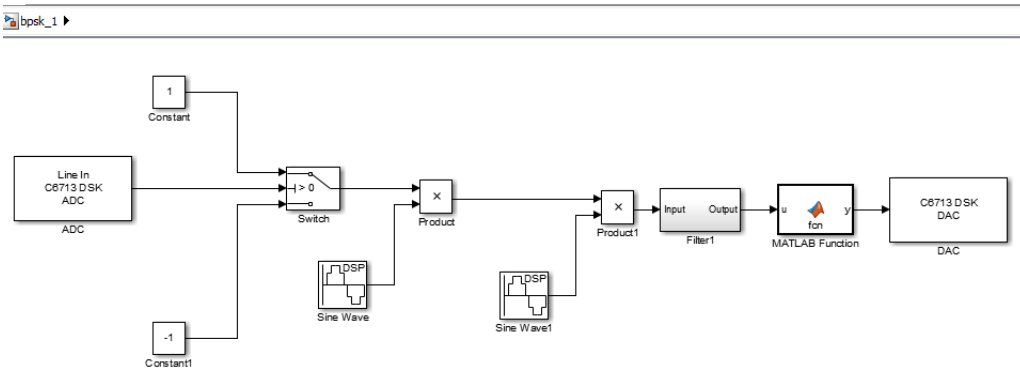


FIGURE 9.2: BPSK demodulation

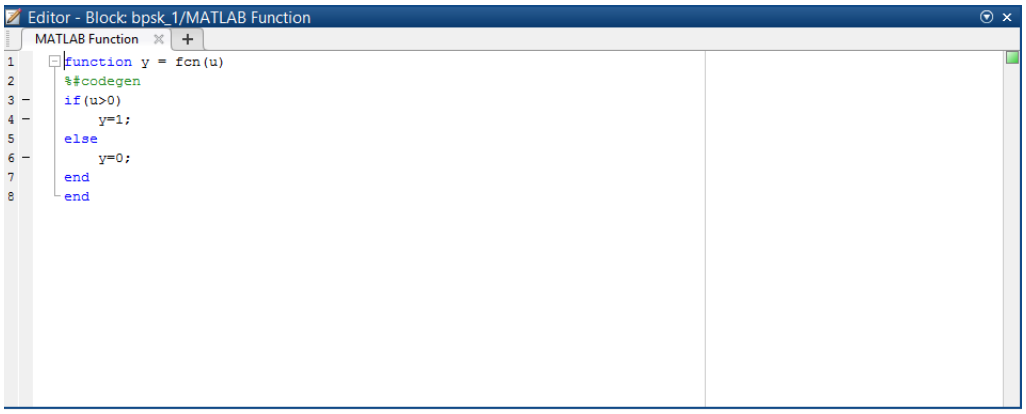


FIGURE 9.3: Decision block Matlab function

Output

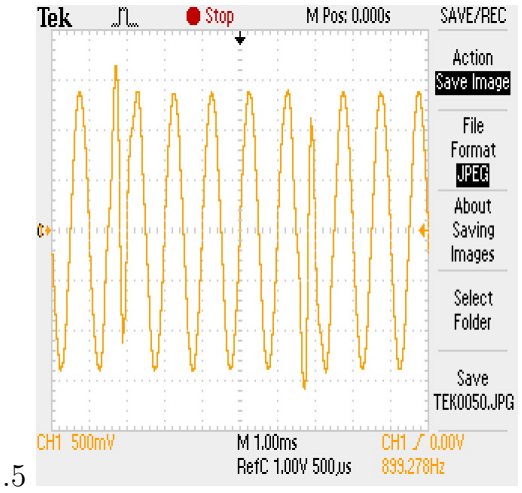


FIGURE 9.4: BPSK Modulated wave

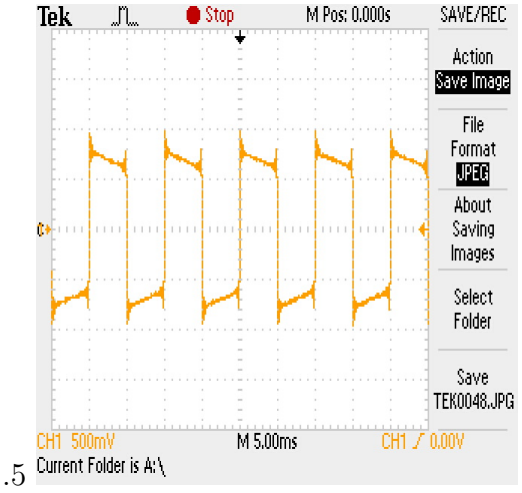


FIGURE 9.5: BPSK Demodulated wave

FIGURE 9.6: BPSK modulation and demodulation outputs

Results

Implemented BPSK modulation and demodulation on DSK6713 using basic blocks from Simulink and observed the output on CRO.

2.Implement a QPSK modulation and demodulation system in DSK6713 from the simulink model with baasic elements. Record your observations.

Theory

QPSK or Quadrature Phase Shift Keying uses two bits per symbol.It is given as: $s_n(t) = Acos(2\pi f_c t + \theta_n)$, where $\theta_n = (2n-1)\frac{\pi}{4}$
This it uses two carriers which are orthogonal to each other($sin(2\pi f_c t) and cos(2\pi f_c t)$),and thus has two components, the in-phase component and the quadrature phase component. At the receiver, the I-channel and Q-channel signals are individually demodulated as in BPSK(i.e. multiplying with the carriers and low pass filtering it). Since the two carriers are orthogonal to each other, when they are multiplied and integrated over one time period, it will be zero.

Block Diagrams

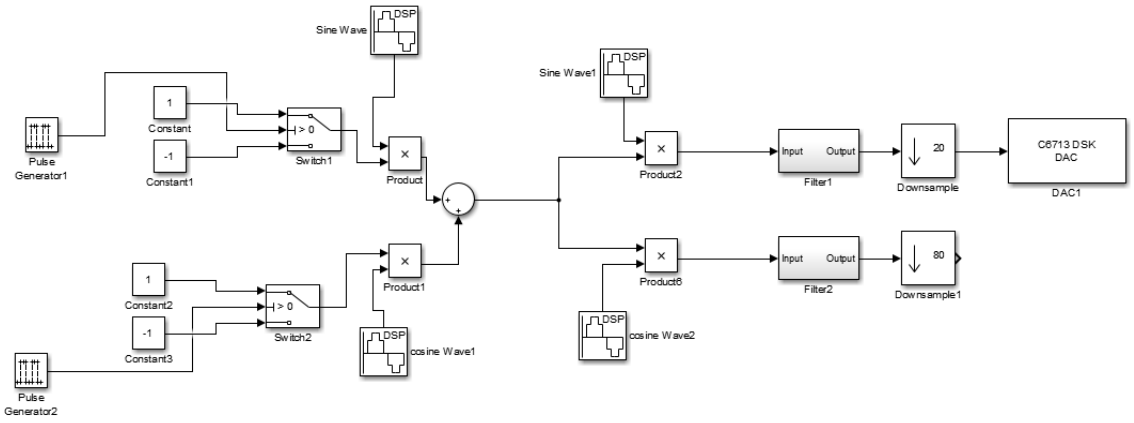
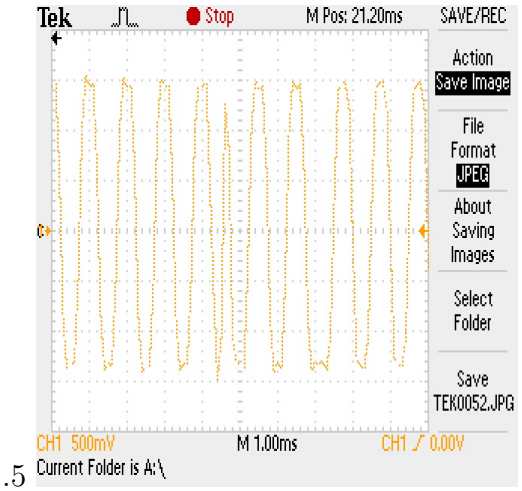


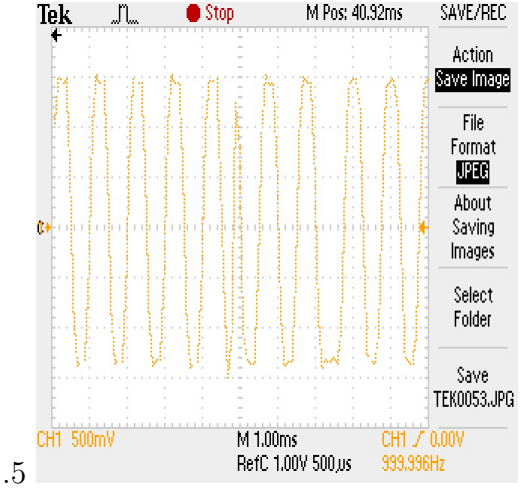
FIGURE 9.7: QPSK modulation and demodulation

Output



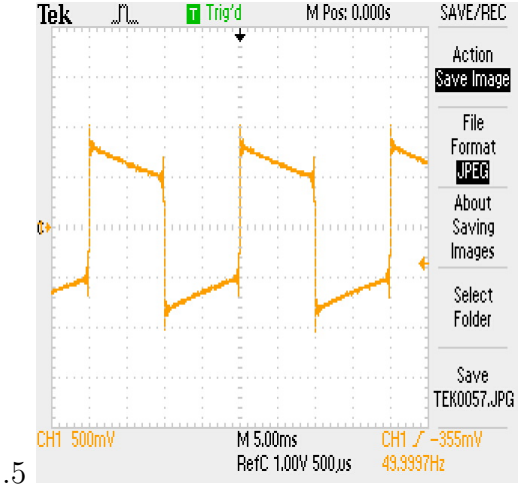
.5

FIGURE 9.8: QPSK modulated wave($phaseshift = (3\pi/4)$)



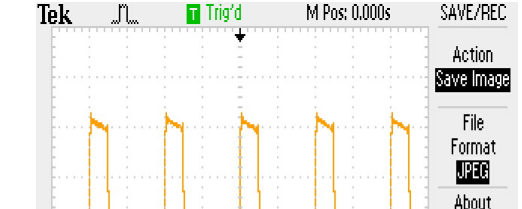
.5

FIGURE 9.9: QPSK modulated wave($phaseshift = (\pi/4)$)



.5

FIGURE 9.10: Demodulated in phase component



Observations

Observed the in-phase and quadrature phase demodulated waves on the CRO and they were same as the even and odd bit sequences. The input bitrate was 400 bps.

Inferences

When compared with BPSK, QPSK has bandwidth efficiency twice that of BPSK.

Results

Implemented QPSK modulation and demodulation in DSK6713 using blocks from Simulink.

3.Implement wavelet denoising on a noisy 1-D signal in Simulink and export the model onto the DSK 6713 and verify its performance in real time.

Theory

Denoising is the process in which we reconstruct a signal from a noisy one.The Wavelet transform performs a correlation analysis, therefore the output is expected to be maximal when the input signal most resembles the mother wavelet.If a signal has its energy concentrated in a small number of wavelet dimensions, its coefficients will be relatively large compared to any other signal or noise whose energy is spread over a large number of coefficients.This means that shrinking the wavelet transform will remove the low amplitude noise or undesired signal in the wavelet domain, and an inverse wavelet transform will then retrieve the desired signal with little loss of details. Dyadic wavelet transforms are scale samples of wavelet transforms following a geometric sequence of ratio 2.It is implemented by perfect reconstruction filter banks.

Block diagrams

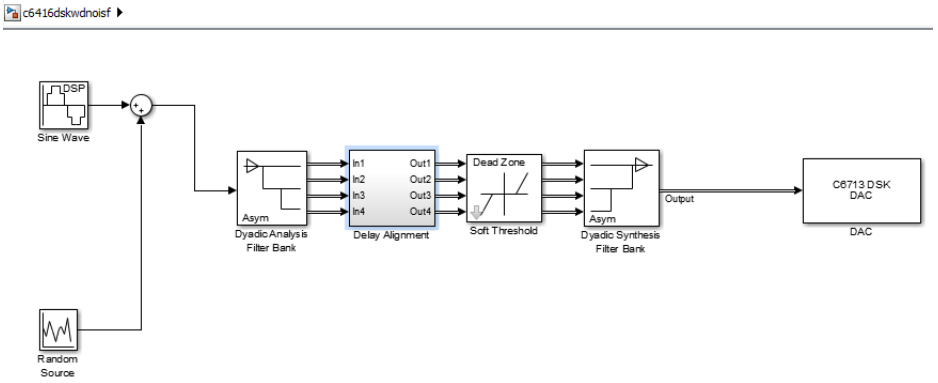


FIGURE 9.13: Wavelet denoising block diagram

Output

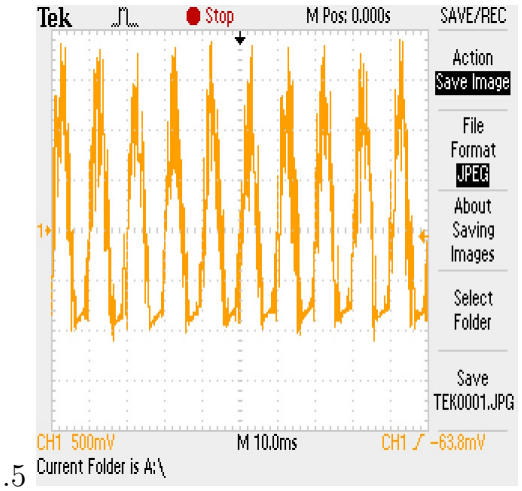


FIGURE 9.14: Noisy signal

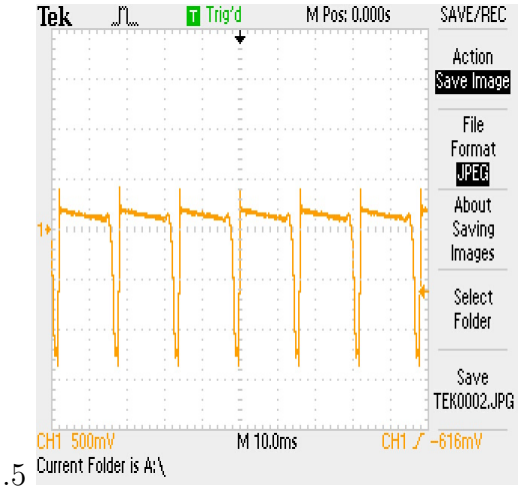


FIGURE 9.15: Denoised signal

FIGURE 9.16: Wavelet denoising

Observations

A noisy sinusoidal signal was created in Simulink and this signal was denoised using wavelet method.

Inferences

It is possible to remove the noise in a noisy signal using Wavelet transform using Dyadic Synthesis and analysis blocks and soft threshold blocks of Simulink.

Results

Implemented wavelet denoising of 1-D noisy signal and exported the model to DSK6713 and verified the performance.

Chapter 10

Fast Fourier Transform

1.Implement a 256-point FFT in real time, using an external input signal 2kHz, 1Vpp sine wave. Record your observations on the oscilloscope and interpret the results.

Theory

The Fast Fourier Transform (FFT) is an efficient computation of the Discrete Fourier Transform (DFT) and one of the most important tools used in digital signal processing applications. Because of its well-structured form, the FFT is a benchmark in assessing digital signal processor (DSP) performance.The DFT is viewed as a frequency domain representation of the discrete-time sequence x(n). The N-point DFT of finite-duration sequence x(n) is defined as $X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$

Program

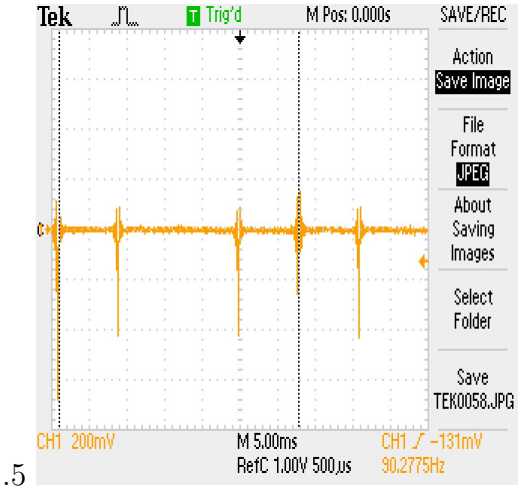
```
#include<stdio.h>
#include<math.h>
#include "dsk6713.h"
#include "dsk6713_led.h"
```

```
#include "dsk6713_dip.h"
#include "dsk6713_aic23.h"
#include "fft.h"

void DSK6713_init();
void DSK6713_LED_init();
void DSK6713_DIP_init();
//extern void comm_poll();
extern void output_sample();
extern void outputsample(int);
Uint16 input_sample();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0011;

#define N 256
#define PI 3.14


float iobuffer[N];
float x1[N];
short i;
short buffercount=0;
short flag=0;
COMPLEX w[N];
COMPLEX samples[N];

void main() {


for(i=0;i<N;i++)
{
```

```
w[i].real=cos(2*PI*i/512);
w[i].imag=-sin(2*PI*i/512);

}
comm_intr();
while(1)
{
while(flag==0);
flag=0;
for(i=0;i<N;i++)
{
samples[i].real=iobuffer[i];
iobuffer[i]=x1[i];
}
for(i=0;i<N;i++)
samples[i].imag=0.0;

fft(samples,N,w);
for(i=0;i<N;i++)
{
x1[i]=sqrt(samples[i].real*samples[i].real+samples[i].imag*samples[i].imag)/16;
}
x1[0]=32000.0;
}
return;
}
interrupt void c_int11()
{
output_sample((short) (iobuffer[buffercount]));
iobuffer[buffercount++]=(float)((short) input_sample());
if(buffercount>=N)
```

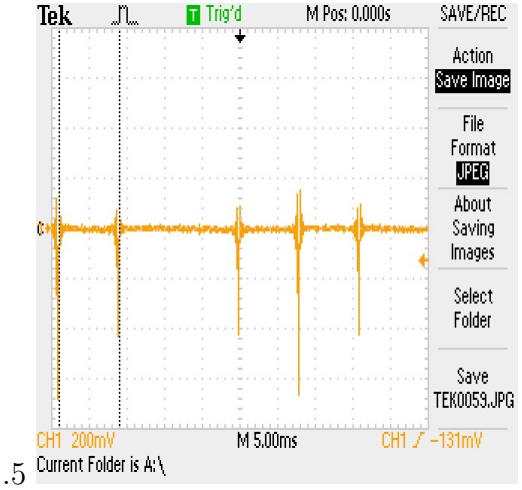
```
{
buffercount=0;
flag=1;
}

}
```

Output



.5

FIGURE 10.1: Pointer at 314.5 Hz



.5

FIGURE 10.2: Pointer at 1.28 kHz

FIGURE 10.3: FFT of the input signal

Observations

For one frequency f in the FFT, there was another frequency corresponding to 4000f also present. The screenshots of the FFT signal with frequency 314.5Hz and 1.28kHz are included in the output section.

Inferences

The DFT of the signal repeats every 2π. When $fs = 8kHz, fs/2$corresponds to π which is equal to 4kHz.Thus if there is an FFT with f Hz, then another will be present at f*4000.

Results

Implemented a 256-point FFT in real time and observed the same on the oscilloscope.

2.Using TI's C-callable optimized Radix-2 FFT function for the same input, implement a 1024-point FFT and record your observations.

Theory

The TMS320C600 digital signal processing library (DSPLIB) provides a set of C-callable, assembly-optimized functions commonly used in signal processing applications, e.g., filtering and transform. The DSPLIB includes several functions for each processing category, based on the input parameter conditions, to provide parameter-specific optimal performance.

Program

```
#include<stdio.h>
#include<math.h>
#include "dsk6713.h"
#include "dsk6713_led.h"
#include "dsk6713_dip.h"
#include "dsk6713_aic23.h"
#include"DSPF_sp_bitrev_cplx.h"
#include"DSPF_sp_cfftr2_dit.h"

void DSK6713_init();
void DSK6713_LED_init();
void DSK6713_DIP_init();
extern void comm_poll();
extern void output_sample();
extern void outputsample(int);
Uint16 input_sample();
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
Uint16 inputsource=0x0011;

#define N 256
#define RADIX 2
#define PI 3.14
#define DELTA (2*PI)/N

short i=0;
short iTwid[N/2];
short iData[N];
float Xmag[N];
typedef struct Complex_tag{float re,im;} Complex;
```

```
Complex W[N/RADIX];
Complex x[N];
#pragma DATA_ALIGN(W,sizeof(Complex))
#pragma DATA_ALIGN(x,sizeof(Complex))

void main()
{
for (i=0;i<N/RADIX;i++)
{
W[i].re=cos(DELTA*i);
W[i].im=sin(DELTA*i);
}
bitrev_index(iTwid,N/RADIX,RADIX);
//void DSPF_sp_cfftr2_dit(float* x, float* w, short n)
DSPF_sp_bitrev_cplx((double*) W,(short*) iTwid,N/RADIX);
comm_poll();
for(i=0;i<N;i++)
Xmag[i]=0;
while(1)
{
output_sample(32000);
for(i=0;i<N;i++)
{
x[i].re=(float)((short) input_sample());
x[i].im=0.0;
if(i>0)
output_sample((short) Xmag[i]);
}
//DSPF_sp_bitrev_cplx(double     * x, short * index, int n)
DSPF_sp_cfftr2_dit((float*) x, (float*) W,N);
bitrev_index(iData,N,RADIX);
```

```
DSPF_sp_bitrev_cplx((double*) x, (short*) iData,N);
for(i=0;i<N;i++)
Xmag[i]=sqrt(x[i].re*x[i].re+x[i].im*x[i].im)/N;

}

}
```

Output

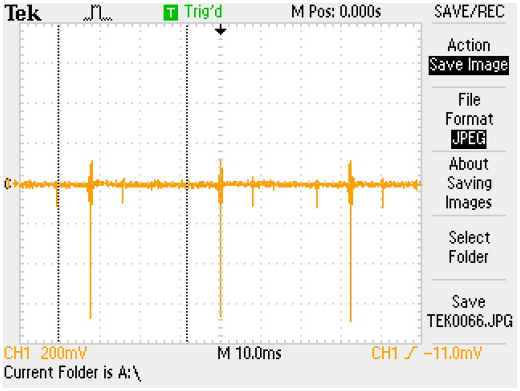
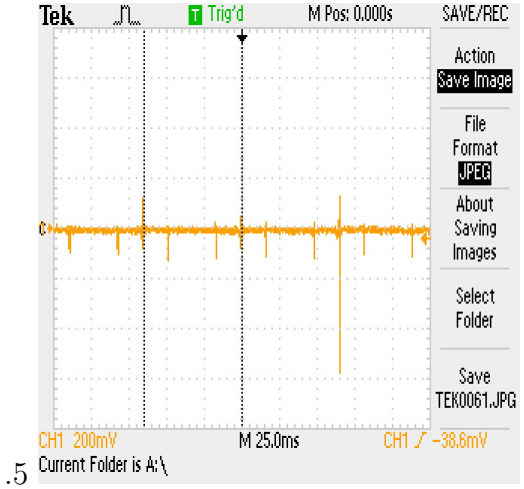
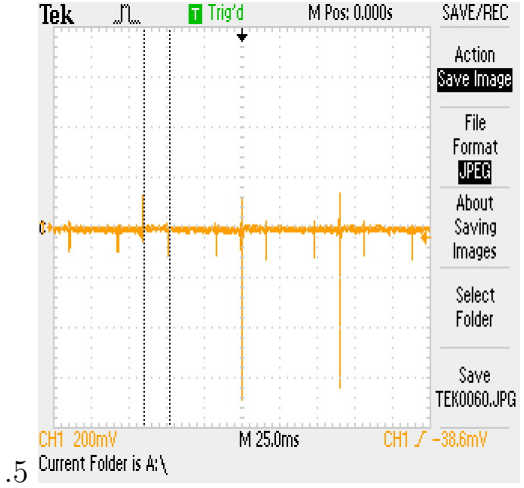


FIGURE 10.4: N=256,fs=8kHz , Pointer at 625Hz



.5

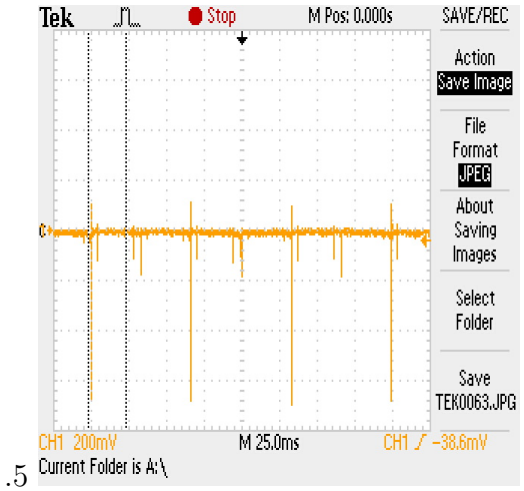
FIGURE 10.5: Pointer at 769.2Hz



.5

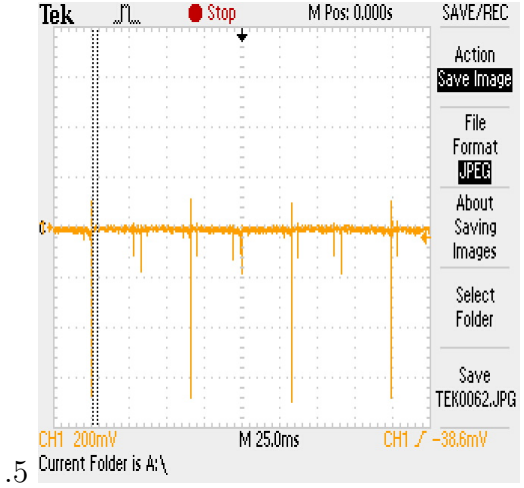
FIGURE 10.6: Pointer at 2.941kHz

FIGURE 10.7: N=512 , fs = 8kHz



.5

FIGURE 10.8: Pointer at 2kHz



.5

FIGURE 10.9: Pointer at 16kHz

FIGURE 10.10: N=512 ,fs = 16kHz

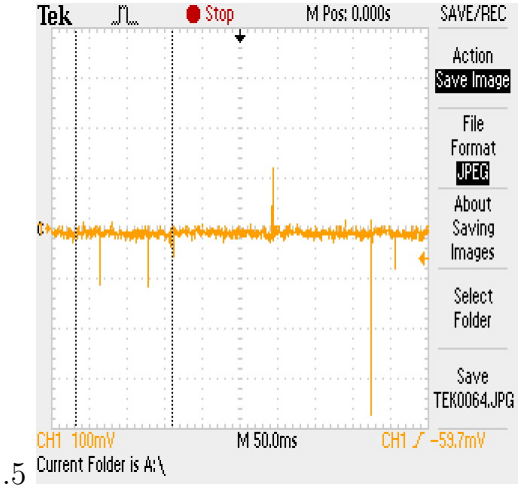


FIGURE 10.11: Pointer at 781Hz

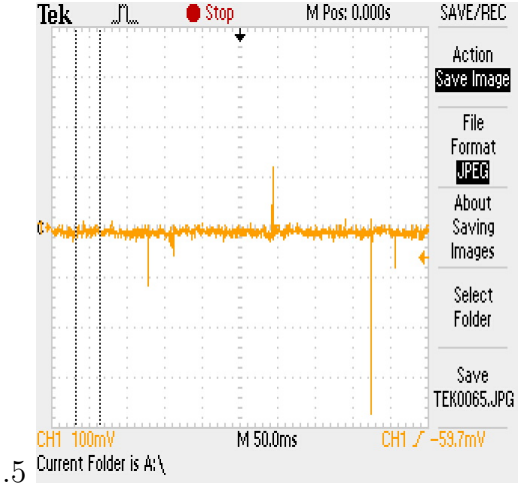


FIGURE 10.12: Pointer at 3.125kHz

FIGURE 10.13: N=1024 , fs=8kHz

Observations

Using TI's callable optimized radix-2 function, FFT was calculated for N=1024,512 and 256 with fs=8 and 16 kHz each. Similar to seen in question1, we could see repeating nature of the FFT. When the sampling frequency was 8kHz, multiplication of 4000 was seen and when the sampling frequency was 16kHz, multiplication of 8000 was observed ; i.e., if there was a frequency f Hz present in the spectrum, then there would another FFT coefficient at a frequency of 4000f or 8000f depending on the sampling frequency.

Inferences

DFT repeats every 2π frequency.

Results

Implemented a 1024-point,512-point and 256-point FFT using TI's C-callable optimized Radix-2 function and observed the output on the oscilloscope.

Chapter 11

Image Processing

I.Basic Image Processing and Spatial Filtering Operations

1. Perform the following operations on a color image(Lena.jpg):

(a) Color image to gray scale image conversion

(b) Binary image conversion

2. Perform the following spatial filtering operations on grey scale image:

(a) Sobel-x , Sobel-y

(b) Box-filter,Gaussian filter

(c) Sharpen,Outline

1.(a) and (b)

Theory

A pixel color in an image is a combination of three colors Red,Green, and Blue(RGB).The grayscale image is represented by luminance using 8 bits value. The luminance of a pixel value of a grayscale image ranges from 0 to 255. The conversion of a color image into a grayscale image is converting the RGB values (24 bit) into grayscale value (8 bit).

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Read the three channels of image hr,hg,hb

Step 4: For i = 0 to 4096 do steps 5,6

Step 5: Calculate the gray value as $gray[i] = 0.21 * hr[i] + 0.72 * hg[i] + 0.07 * hb[i]$

Step 6: If $gray[i] > 128, binary[i] = 255; elsebinary[i] = 0$

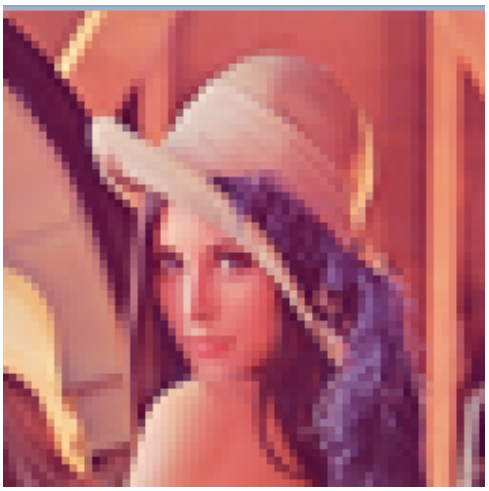
Step 7: Stop

Program

```
# include "lenar.h"
# include "lenag.h"
# include "lenab.h"
# include "math.h"
# include "stdio.h"
int i;
float gray[4096];
unsigned char gray1[4096];
unsigned char binary[4096];
int main(void) {
for(i=0;i<N;i++){
gray[i] = 0.21*hr[i]+0.72*hg[i]+0.07*hb[i];
gray1[i]=(unsigned char) gray[i];
if(gray[i]<128)
binary[i]=0;
else
binary[i]=255;
}
return 0;
```

```
}
```

Output



.5

FIGURE 11.1: Original Image

.5

FIGURE 11.2: Gray scale Image

.5

FIGURE 11.3: Binary Image

Results

Converted the given color image to gray scale and binary images.

2.(a)

Theory

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.
The Sobel kernels are:

$$dx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$dy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the sobel kernels sobelx and sobely

Step 4: Obtain the gray scale image

Step 5: For i=0 to 67, do steps 6

Step 6: For j=0 to 67, do steps 7,8

Step 7: Initialise $sobxout1[i][j] = sobyout1[i][j] = 0$

Step 8: For k,l from 0 to 3 do steps 9 and 10

Step 9: $sobxout1[i][j] = sobelx[k][l] * image[i-k][j-l] + sobxout1[i][j]$

Step 10:$sobyout1[i][j] = sobely[k][l] * image[i-k][j-l] + sobyout1[i][j]$

Step 11:Stop

Program

```
# include "lenar.h"
# include "lenag.h"
# include "lenab.h"
# include "math.h"
# include "stdio.h"
int i,j,k=0,l;
float gray[4096];
unsigned char gray1[4096],image[64][64];
short sobely[3][3] = {1,2,1,0,0,0,-1,-2,-1};
short sobelx[3][3] = {1,0,-1,2,0,-2,1,0,-1};
float sob_x_out1[68][68],sob_x_out_bin1[68][68],sob_y_out1[68][68],sob_y_out_bi
unsigned char sob_x_out[68][68],sob_y_out[68][68];
unsigned char sob_x_out_bin[68][68],sob_y_out_bin[68][68];

int main(void) {
for(i=0;i<N;i++){
gray[i] = 0.21*hr[i]+0.72*hg[i]+0.07*hb[i];
gray1[i]=(unsigned char) gray[i];
}
for (i=0;i<64;i++){
for(j=0;j<64;j++){
image[i][j] = gray1[k];
k++;}}
for(i=0;i<67;i++){
for(j=0;j<67;j++){
```

```
sob_x_out1[i][j]=0;
sob_y_out1[i][j]=0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
if(i-k>=0 && j-l>=0)
sob_x_out1[i][j] = sobelx[k][l]*image[i-k][j-l] + sob_x_out1[i][j];
sob_x_out[i][j] = (unsigned char) sob_x_out1[i][j];

sob_y_out1[i][j] = sobely[k][l]*image[i-k][j-l] + sob_y_out1[i][j];
sob_y_out[i][j] = (unsigned char) sob_y_out1[i][j];

//printf ("%d\t",sob_y_out[i][j]);
}
}
if(sob_x_out[i][j]<228)
sob_x_out_bin1[i][j] = 0;
else
sob_x_out_bin1[i][j]=255;
sob_x_out_bin[i][j] = (unsigned char) sob_x_out_bin1[i][j];
if(sob_y_out[i][j]<205)
sob_y_out_bin1[i][j] = 0;
else
sob_y_out_bin1[i][j]=255;
sob_y_out_bin[i][j] = (unsigned char) sob_y_out_bin1[i][j];
}
}

return 0;
}
```

Output

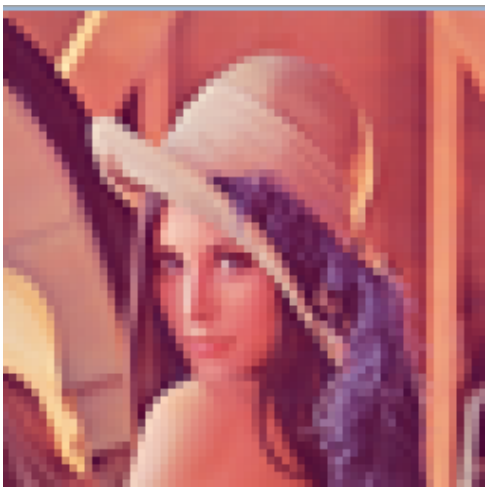
.5 

FIGURE 11.4: Original Image

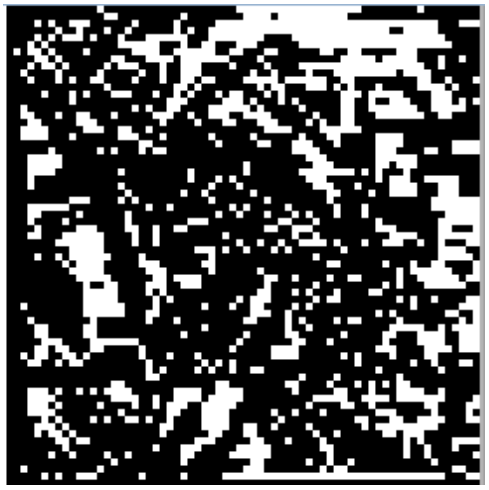
.5 

FIGURE 11.5: Sobel-x filtered image

.5 

FIGURE 11.6: Sobel-y filtered image

Results

Performed filtering using Sobel operator.

2.(b)

Theory

Box filtering is also known as Average filter or mean filter.The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. The Gaussian smoothing operator is a 2-D convolution operator that is used to 'blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian ('bell-shaped') hump.
The kernels used are:

$$box = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

$$gaussian = \begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}$$

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the sobel kernels box and gauss

Step 4: Obtain the gray scale image

Step 5: For i=0 to 67, do steps 6

Step 6: For j=0 to 67, do steps 7,8

Step 7: Initialise $boxout1[i][j] = gaussout1[i][j] = 0$

Step 8: For k,l from 0 to 3 do steps 9 and 10

Step 9: $boxout1[i][j] = box[k][l] * image[i-k][j-l] + boxout1[i][j]$

Step 10:$gaussout1[i][j] = gauss[k][l] * image[i-k][j-l] + gaussout1[i][j]$

Step 11:Stop

Program

```
# include "lenar.h"
# include "lenag.h"
# include "lenab.h"
# include "math.h"
# include "stdio.h"
int i,j,k=0,l;
float gray[4096];
unsigned char gray1[4096],image[64][64];
float box[3][3] = {0.11,0.11,0.11,0.11,0.11,0.11,0.11,0.11,0.11};
float gauss[3][3] ={0.0113, 0.0838 , 0.0113 , 0.0838 , 0.6193 , 0.0838 , 0.0113
float box_out1[68][68],gauss_out1[68][68];
unsigned char box_out[68][68],gauss_out[68][68];

int main(void) {
for(i=0;i<N;i++){
gray[i] = 0.21*hr[i]+0.72*hg[i]+0.07*hb[i];
gray1[i]=(unsigned char) gray[i];
}
for (i=0;i<64;i++){
for(j=0;j<64;j++){
image[i][j] = gray1[k];
k++;}}
```

```
for(i=0;i<67;i++){
for(j=0;j<67;j++){
box_out1[i][j]=0;
gauss_out1[i][j]=0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
if(i-k>=0 && j-l>=0){
box_out1[i][j] = box[k][l]*image[i-k][j-l] + box_out1[i][j];
box_out[i][j] = (unsigned char) box_out1[i][j];
gauss_out1[i][j] = gauss[k][l]*image[i-k][j-l] + gauss_out1[i][j];
gauss_out[i][j] = (unsigned char) gauss_out1[i][j];
//printf("%f \t",box_out[i][j]);
}
}
}
}}

return 0;
}
```

Output

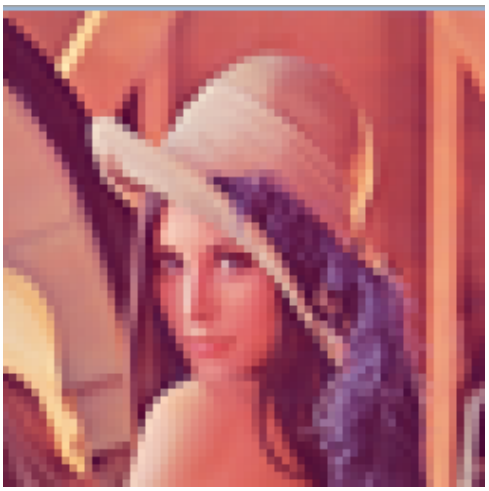


FIGURE 11.7: Original Image

FIGURE 11.8: Box-filtered image

FIGURE 11.9: Gaussian filtered image

Observations

A Gaussian provides gentler smoothing and preserves edges better than a similarly sized mean filter.

Results

Performed filtering using Box and Gaussian filter.

2.(c)

Theory

A high-pass filter can be used to make an image appear sharper. These filters emphasize fine details in the image.Finding the edges or outline can also be done using a high pass filter.
The kernel used are:

$$sharpen = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$outline = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the kernels sharpen and outline

Step 4: Obtain the gray scale image

Step 5: For i=0 to 67, do steps 6

Step 6: For j=0 to 67, do steps 7,8

Step 7: Initialise $shout1[i][j] = outlineout1[i][j] = 0$

Step 8: For k,l from 0 to 3 do steps 9 and 10

Step 9: $shout1[i][j] = sharpen[k][l] * image[i-k][j-l] + shout1[i][j]$

Step 10:$outlineout1[i][j] = outline[k][l] * image[i-k][j-l] + outlineout1[i][j]$

Step 11:Stop

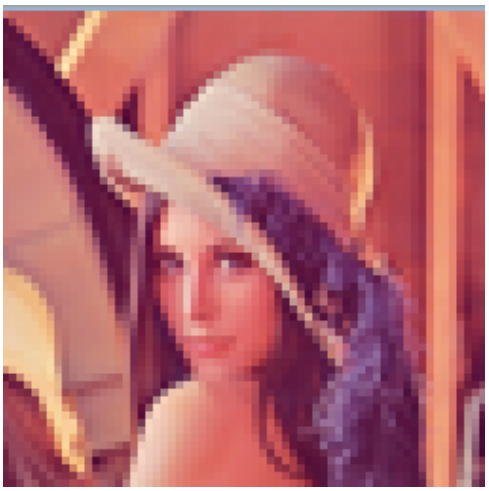
Program

```
# include "lenar.h"
# include "lenag.h"
# include "lenab.h"
# include "math.h"
# include "stdio.h"
int i,j,k=0,l;
float gray[4096];
unsigned char gray1[4096],image[64][64];
float gauss[3][3] ={0.0113, 0.0838 , 0.0113 , 0.0838 , 0.6193 , 0.0838 , 0.0113
float box_out1[68][68],gauss_out1[68][68];
unsigned char box_out[68][68],gauss_out[68][68];

float sharpen[3][3] = {0,-1,0,-1,5,-1,0,-1,0};
float outline[3][3] ={-1,-1,-1,-1,8,-1,-1,-1,-1};
float sh_out1[68][68],outline_out1[68][68],out_bin1[68][68];
unsigned char sh_out[68][68],outline_out[68][68],out_bin[68][68];


int main(void) {
for(i=0;i<N;i++){
gray[i] = 0.21*hr[i]+0.72*hg[i]+0.07*hb[i];
gray1[i]=(unsigned char) gray[i]; }
```

```
for (i=0;i<64;i++){
for(j=0;j<64;j++){
image[i][j] = gray1[k];
k++;}}

for(i=0;i<67;i++){
for(j=0;j<67;j++){
gauss_out1[i][j]=0;
sh_out1[i][j]=0;
outline_out1[i][j]=0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
if(i-k>=0 && j-l>=0){
gauss_out1[i][j] = gauss[k][l]*image[i-k][j-l] + gauss_out1[i][j];
gauss_out[i][j] = (unsigned char) gauss_out1[i][j];}}}}}
for(i=0;i<68;i++){
for(j=0;j<68;j++){
for(k=0;k<3;k++){
for(l=0;l<3;l++){

//printf("%f \t",box_out[i][j]);
sh_out1[i][j] = sharpen[k][l]*gauss_out[i-k][j-l] + sh_out1[i][j];
sh_out[i][j] = (unsigned char) sh_out1[i][j];
outline_out1[i][j] = outline[k][l]*gauss_out[i-k][j-l] + outline_out1[i][j];
outline_out[i][j] = (unsigned char) outline_out1[i][j];

}
}

if(outline_out[i][j]>100 && outline_out[i][j]<200)
out_bin1[i][j] = 255;
```

```
else
out_bin1[i][j]=0;
out_bin[i][j] = (unsigned char) out_bin1[i][j];}
}

return 0;
}
```

Output

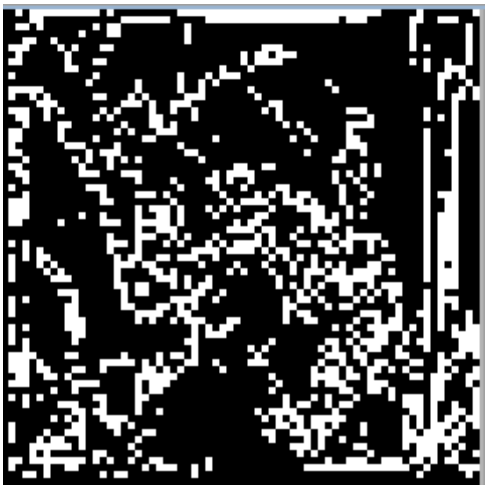


.5

FIGURE 11.10: Original Image

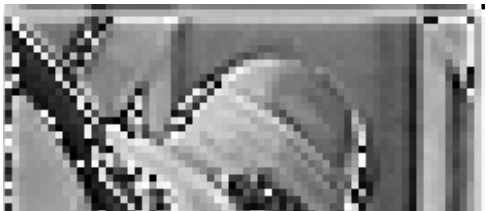
.5

FIGURE 11.11: Sharpened image



.5

FIGURE 11.12: Outline



Observations

Better sharpening and outlines are obtained when gaussian filtering was done prior to sharpening and outline filters.

Results

Performed sharpening and obtained outline of the image.

II.Morphological Operations

Read the given binary image, perform following operations and display using CCS properties:

1. Dilation
2. Erosion
3. Opening
4. Closing
5. Boudary Extraction using (a) Dilation and Erosion (b) Opening and Closing 6. Hit or Miss

Theory

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image.

1.Dilation:

The dilation of an image A by a structuring element B (denoted by $A \oplus B$) produces a new binary image C with ones in all locations (x,y) of a structuring element's origin at which that structuring element B hits the the input image A.

2.Erosion:

The erosion of a binary image A by a structuring element B (denoted by $A \ominus B$) produces a new binary image C with ones in all locations (x,y) of a structuring

element's origin at which that structuring element B fits the input image.

3.Opening:

This operation is performed by erosion followed by dilation and is denoted as $A \circ B$.

4.Closing:

This operation is performed by dilation followed by erosion and is denoted as $A \bullet B$.

5.Boundary Extraction:

Boundary can be extracted by subtracting the dilated image from the original image or subtracting a closed image from an opened image,i.e.$boundary(A) = A - (A \oplus B)$ or $boundary(A) = (A \circ B) - (A \bullet B)$.

6.Hit or miss:

This operation allows to derive information on how objects in a binary image are related to their surroundings. The operation requires a matched pair of structuring elements, $B = B1, B2$, that probe the inside and outside, respectively, of objects in the image. $A * B = (A \ominus B1) \cap (A^c \ominus B2)$

1.Dilation

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring element 'kernel'

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and kernel

Step 6: If the correlation value is not equal to zero, give it the value 255

Step 7: Stop

Program

```
# include "qn2.h"
```

```
# include "stdio.h"

short kernel[3][3] = {0,1,0,1,1,1,0,1,0};
short image[64][64] , dil[64][64];
unsigned char im[64][64] ,dilated[64][64];
short i,j,k=0,l;

int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{image[i][j] = h[k];
im[i][j] = (unsigned char) image[i][j];
k++;}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
dil[i][j] = 0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
dil[i][j] = image[i+k][j+l]*kernel[k][l] + dil[i][j] ;}}
if(dil[i][j] != 0)
dil[i][j] = 255;
dilated[i][j] = (unsigned char) dil[i][j];}}

return 0;
}
```

2.Erosion

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring element 'kernel'

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and kernel

Step 6: If the correlation value is not equal to (5*255), give it the value 0

Step 7: Stop

Program

```
# include "qn2.h"
# include "stdio.h"

short kernel[3][3] = {0,1,0,1,1,1,0,1,0};
short image[64][64] , dil[64][64] ,er[64][64];
unsigned char im[64][64] ,dilated[64][64] ,eroded[64][64];
short i,j,k=0,l;

int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{image[i][j] = h[k];
im[i][j] = (unsigned char) image[i][j];
k++;}}
```

```
for(i=0;i<64;i++){
for(j=0;j<64;j++){
er[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
er[i][j] = er[i][j] + image[i+k][j+l]*kernel[k][l] ;}}
if(er[i][j] != 1275)
er[i][j]=0;
eroded[i][j] = (unsigned char) er[i][j];}}
return 0;
}
```

3.Opening

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring element 'kernel'

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and kernel

Step 6: If the correlation value is not equal to (5*255), give it the value 0 and store it in matrix er[64][64]

Step 7: Do correlation between er and kernel

Step 8: If the correlation value is not equal to zero, give it the value 255 and store it in dil[64][64]

Step 9: Stop

Program

```
# include "qn2.h"
```

```
# include "stdio.h"

short kernel[3][3] = {0,1,0,1,1,1,0,1,0};
short image[64][64] , dil[64][64] ,er[64][64];
unsigned char im[64][64] ,dilated[64][64] ,eroded[64][64];
short i,j,k=0,l;

int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{image[i][j] = h[k];
im[i][j] = (unsigned char) image[i][j];
k++;}}



for(i=0;i<64;i++){
for(j=0;j<64;j++){
er[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
er[i][j] = er[i][j] + image[i+k][j+l]*kernel[k][l] ;}}
if(er[i][j] != 1275)
er[i][j]=0;
eroded[i][j] = (unsigned char) er[i][j];}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
dil[i][j] = 0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
```

```
dil[i][j] = eroded[i+k][j+l]*kernel[k][l] + dil[i][j] ;}}
if(dil[i][j] != 0)
dil[i][j] = 255;
dilated[i][j] = (unsigned char) dil[i][j];}}

return 0;
}
```

4.Closing

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring element 'kernel'

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and kernel

Step 6: If the correlation value is not equal to zero, give it the value 255 and store it in dil[64][64]

Step 7: Do correlation between dil and kernel

Step 8: If the correlation value is not equal to (5*255), give it the value 0 and store it in matrix er[64][64]

Step 9: Stop

Program

```
# include "qn2.h"
# include "stdio.h"

short kernel[3][3] = {0,1,0,1,1,1,0,1,0};
short image[64][64] , dil[64][64] ,er[64][64];
```

```
unsigned char im[64][64] ,dilated[64][64] ,eroded[64][64];
short i,j,k=0,l;

int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{image[i][j] = h[k];
im[i][j] = (unsigned char) image[i][j];
k++;}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
dil[i][j] = 0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
dil[i][j] = image[i+k][j+l]*kernel[k][l] + dil[i][j] ;}}
if(dil[i][j] != 0)
dil[i][j] = 255;
dilated[i][j] = (unsigned char) dil[i][j];}}
for(i=0;i<64;i++){
for(j=0;j<64;j++){
er[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
er[i][j] = er[i][j] + dil[i+k][j+l]*kernel[k][l] ;}}
if(er[i][j] != 1275)
er[i][j]=0;
eroded[i][j] = (unsigned char) er[i][j];}}
return 0;
}
```

5(a).Boundary Extraction using Dilation and Erosion

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring element 'kernel'

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and kernel

Step 6: If the correlation value is not equal to (5*255) give it the value 0 and store it in matrix er[64][64]

Step 7: Do correlation between image and kernel

Step 8: If the correlation value is not equal to zero, give it the value 255 and store it in dil[64][64]

Step 9: Calculate boundary1[i][j] = dil[i][j] - er[i][j] for all (i,j)s

Step10: Calculate boundary[i][j] = image[i][j] - dil[i][j] for all (i,j)s

Step11: Stop

Program

```
# include "qn2.h"
# include "stdio.h"

short kernel[3][3] = {0,1,0,1,1,1,0,1,0};
short image[64][64] , dil[64][64] ,er[64][64];
unsigned char im[64][64] ,dilated[64][64] ,eroded[64][64] , b1[64][64];
short i,j,k=0,l;
```

```
int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{image[i][j] = h[k];
im[i][j] = (unsigned char) image[i][j];
k++;}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
dil[i][j] = 0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
dil[i][j] = image[i+k][j+l]*kernel[k][l] + dil[i][j] ;}}
if(dil[i][j] != 0)
dil[i][j] = 255;
dilated[i][j] = (unsigned char) dil[i][j];
}}
for(i=0;i<64;i++){
for(j=0;j<64;j++){
er[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
er[i][j] = er[i][j] + image[i+k][j+l]*kernel[k][l] ;}}
if(er[i][j] != 1275)
er[i][j]=0;
eroded[i][j] = (unsigned char) er[i][j];}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
```

```
b1[i][j] = (unsigned char) (dilated[i][j] - eroded[i][j]);}}
return 0;
}
```

5(b). Boundary extraction using Opening and Closing

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring element 'kernel'

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and kernel

Step 6: If the correlation value is not equal to zero, give it the value 255 and store it in dil[64][64]

Step 7: Do correlation between dil and kernel

Step 8: If the correlation value is not equal to (5*255), give it the value 0 and store it in matrix close[64][64]

Step 9: Do correlation between image and kernel

Step10: If the correlation value is not equal to (5*255), give it the value 0 and store it in matrix er[64][64]

Step11: Do correlation between er and kernel

Step12: If the correlation value is not equal to zero, give it the value 255 and store it in open[64][64]

Step13: Calculate boundary[i][j] = open[i][j] - close[i][j] for all (i,j)s

Step14: Stop

Program

```
# include "qn2.h"
# include "stdio.h"

short kernel[3][3] = {0,1,0,1,1,1,0,1,0};
short image[64][64] , dil[64][64] ,er[64][64],cl[64][64],op[64][64];
unsigned char im[64][64] ,dilated[64][64] ,eroded[64][64] , b1[64][64],close[64
short i,j,k=0,l;

int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{image[i][j] = h[k];
im[i][j] = (unsigned char) image[i][j];
k++;}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
dil[i][j] = 0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
dil[i][j] = image[i+k][j+l]*kernel[k][l] + dil[i][j] ;}}
if(dil[i][j] != 0)
dil[i][j] = 255;
dilated[i][j] = (unsigned char) dil[i][j];}}
for(i=0;i<64;i++){
for(j=0;j<64;j++){
er[i][j] = 0;
for(k=0;k<3;k++){
```

```
for(l=0;l<3;l++){
er[i][j] = er[i][j] + dilated[i+k][j+l]*kernel[k][l] ;}}
if(er[i][j] != 1275)
er[i][j]=0;
close[i][j] = (unsigned char) er[i][j];}}

//-------------------------------------------------------------------
for(i=0;i<64;i++){
for(j=0;j<64;j++){
er[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
er[i][j] = er[i][j] + image[i+k][j+l]*kernel[k][l] ;}}
if(er[i][j] != 1275)
er[i][j]=0;
eroded[i][j] = (unsigned char) er[i][j];}}

for(i=0;i<64;i++){
for(j=0;j<64;j++){
dil[i][j] = 0;

for(k=0;k<3;k++){
for(l=0;l<3;l++){
dil[i][j] = eroded[i+k][j+l]*kernel[k][l] + dil[i][j] ;}}
if(dil[i][j] != 0)
dil[i][j] = 255;
open[i][j] = (unsigned char) dil[i][j];}}


for(i=0;i<64;i++){
for(j=0;j<64;j++){
```

```
if(open[i][j]==0)
open[i][j]=255;
else
open[i][j]=0;
if(close[i][j]==0)
close[i][j]=255;
else
close[i][j]=0;

b1[i][j] =  (unsigned char) (open[i][j] - close[i][j]);}}
return 0;
}
```

6.Hit or miss operation

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Define the structuring elements B1 and B2

Step 4: Read the image data,convert it into a matrix image[64][64]

Step 5: Do correlation between image and B1

Step 6: If the correlation value is less than 250, give it the value 0 and store it in e1

Step 7: Find the complement of image, imagecomp Step 8: Do correlation between imagecomp and B2

Step 9: If the correlation value is less than 250, give it the value 0 and store it in e2

Step10: Calculate the intersection between e1 and e2 and store it in hm

Step11: Stop

Program

```
# include "hm.h"
# include "stdio.h"

short B1[3][3] = {0,1,0,1,1,1,0,1,0};
short B2[3][3] = {1,0,1,0,0,0,1,0,1};
unsigned char b1[3][3],b2[3][3];

short image[64][64],imagecomp[64][64],e1[64][64] ,e2[64][64],hm[64][64];
unsigned char im1[64][64] ,imcomp1[64][64] ,e11[64][64] , e22[64][64],hitmiss[64
short i,j,k=0,l;

int main(void) {
for(i=0;i<64;i++)
{for(j=0;j<64;j++)
{
image[i][j] = h[k];
imagecomp[i][j] = 255 - h[k];
im1[i][j] = (unsigned char) image[i][j];
imcomp1[i][j] = (unsigned char) imagecomp[i][j];
k++;}}
for(i=0;i<64;i++){
for(j=0;j<64;j++){
e1[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
e1[i][j] = e1[i][j] + image[i+k][j+l]*B1[k][l] ;}}
if(e1[i][j] < 250)
e1[i][j]=0;
e11[i][j] = (unsigned char) e1[i][j];}}
```

```
//-------------------------------------------------------------
for(i=0;i<64;i++){
for(j=0;j<64;j++){
e2[i][j] = 0;
for(k=0;k<3;k++){
for(l=0;l<3;l++){
e2[i][j] = e2[i][j] + imagecomp[i+k][j+l]*B2[k][l] ;}}
if(e2[i][j]>300)
e2[i][j]=0;
else
e2[i][j]=255;
e22[i][j] = (unsigned char) e2[i][j];}}


for(i=0;i<64;i++){
for(j=0;j<64;j++){
hm[i][j] = e11[i][j]*e22[i][j];
if(hm[i][j]!=0)
hm[i][j] = 255;
hitmiss[i][j] =  (unsigned char) hm[i][j];}}
for(i=0;i<3;i++){
for(j=0;j<3;j++){
if(B1[i][j]==1)
B1[i][j]=255;
if(B2[i][j]==1)
B2[i][j]=255;
b1[i][j] = (unsigned char) B1[i][j];
b2[i][j] = (unsigned char) B2[i][j];
}
}
```

```
return 0;
}
```

Output

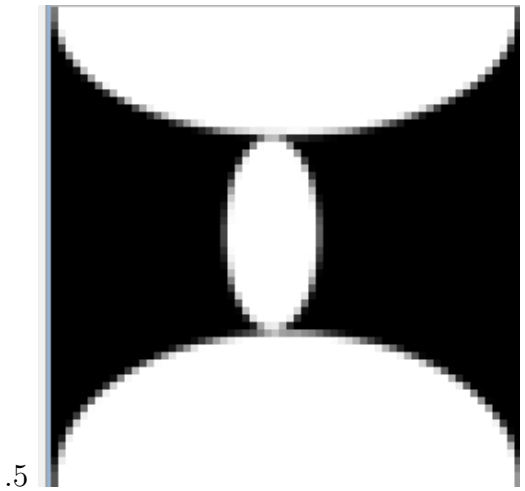


FIGURE 11.15: Original image

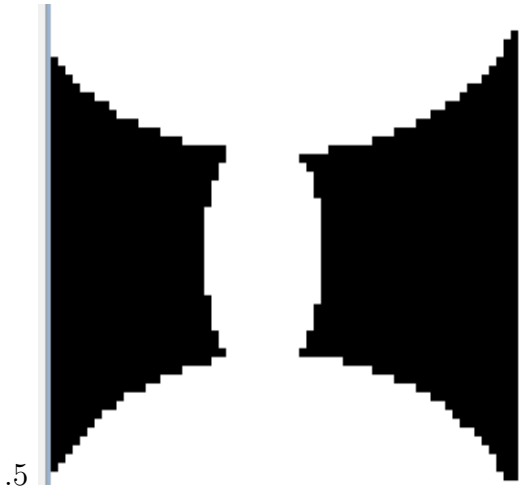


FIGURE 11.16: Dilated image

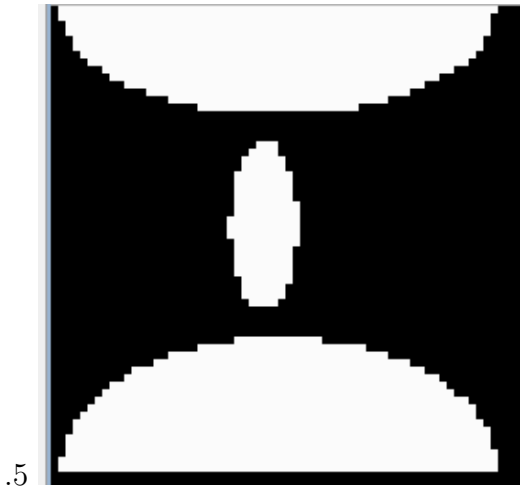


FIGURE 11.17: Eroded image

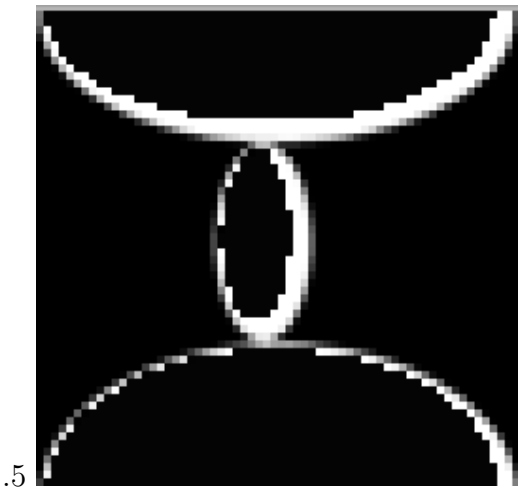


FIGURE 11.20: Boundary extracted:$(A - A \oplus B)$

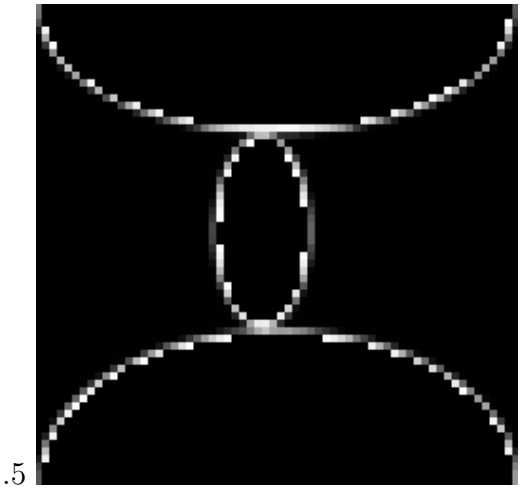


FIGURE 11.21: Boundary extracted:$(A - A \ominus B)$

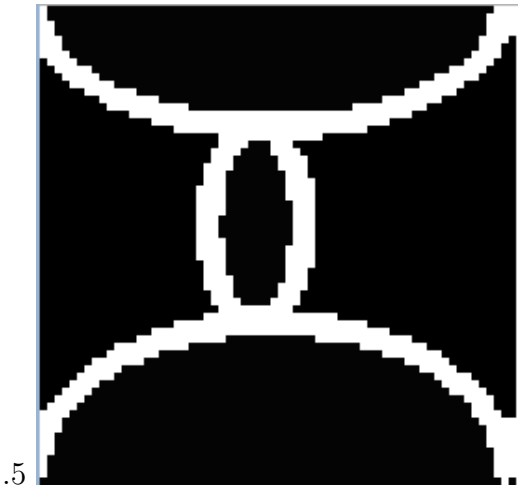
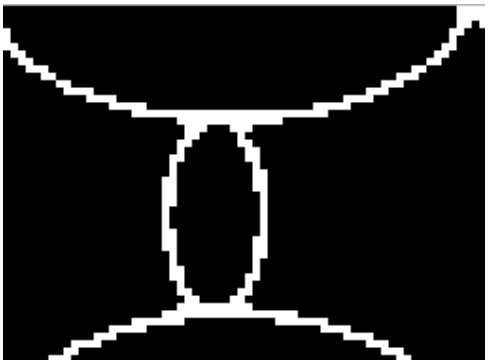


FIGURE 11.22: Boundary extracted:(Dilation - Erosion)



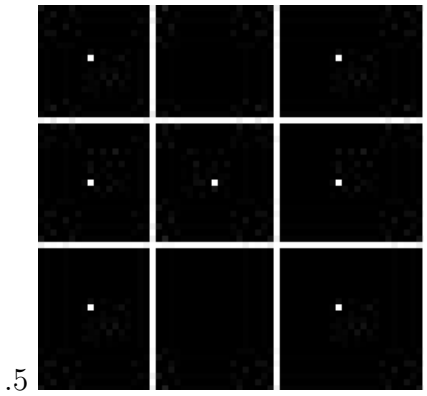


FIGURE 11.25: Image A

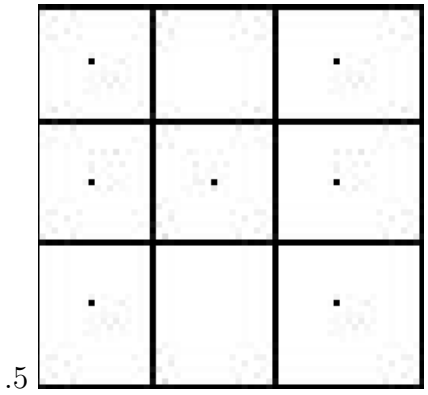


FIGURE 11.26: A^c

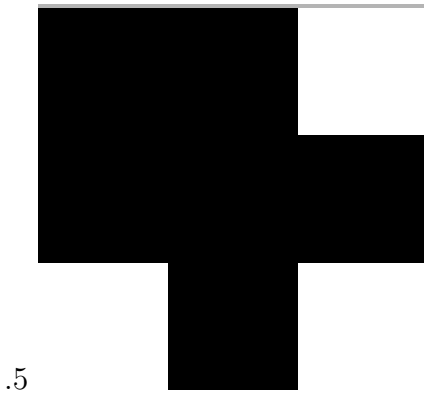


FIGURE 11.27: Structural element B1

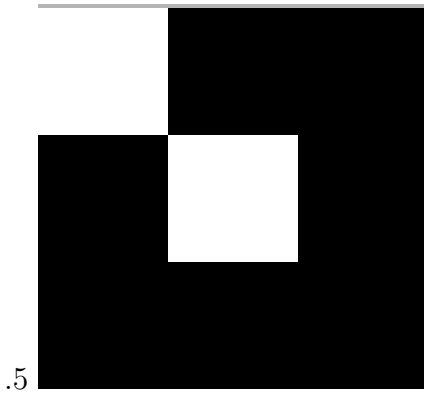


FIGURE 11.28: Structural element B2

Observations

Erosion shrinks the image while Dilation expands it,even adds extra layers to the image. Closing fills the holes present in the image while opening opens up gaps between objects connected by a thin bridge of pixels.

Results

Performed different morphological operations on the binary image.

III.Contrast Stretching and Histogram Equalization

1. Compute the normalized histogram and CDF of "Image 1".
2. Grey Level Slicing: Try to segment out the whitish tumour from the MRI image "Image 2", using gray level slicing specified by the graph in Fig 1.
3. Contrast Stretching: Improve the contrast of "Image 3".
4. Histogram Equalization: Equalize the histogram of low contrast image. Visualize and compare the input and output images and their histograms using image analysis tool and graph properties in CCS.

Theory

Histogram equalization is a technique for adjusting image intensities to enhance contrast. If L is the number of possible grey values in image f , let p denote the normalized histogram of f with a bin for each possible intensity.

$$p_n = \frac{numberofpixelswithintensityn}{totalnumberofpixels}$$

Then the histogram equalized image g is defined as

$$g_{i,j} = floor((L-1)\sum_{n=0}^{f_{i,j}} p_n)$$

1.Histogram and CDF

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Read the image intensity values and create histogram

Step 4: Divide the values by the total number of pixels to get normalized histogram

Step 5: Starting at i=0 sum up the cumulative intensities from the histogram to create the CDF

Step 6: Stop

Program

```
# include "lenabw.h"
# include"stdio.h"
short i;
float hist[256];

float hist_norm[256],cdf[256],p;
int main(void) {
for(i=0;i<256;i++){
hist[i]=0;
cdf[i]=0;
hist_norm[i]=0;
}
for(i=0;i<N;i++){
hist[h[i]]=hist[h[i]]+1;
    }
```

```
p=0;
for(i=0;i<256;i++){
hist_norm[i] = hist[i] *0.000244;
cdf[i]=p+hist_norm[i];
p=cdf[i];
}
/*for(i=0;i<256;i++){
hist[i]=hist[i]/4096;
cdf[i]=p+hist[i];
p=cdf[i];
}*/
return 0;
}
```

Output

FIGURE 11.33: Original Image

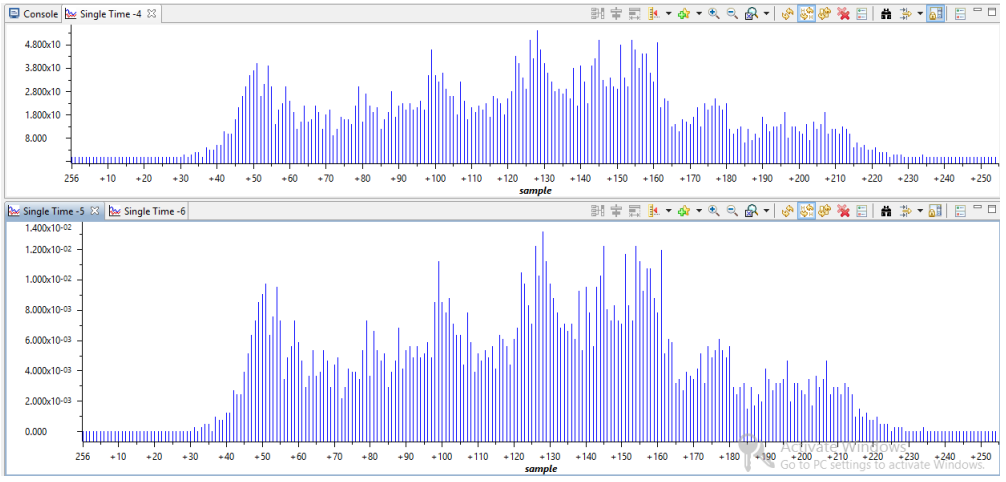


FIGURE 11.34: Histogram(top) and Normalized histogram(bottom)

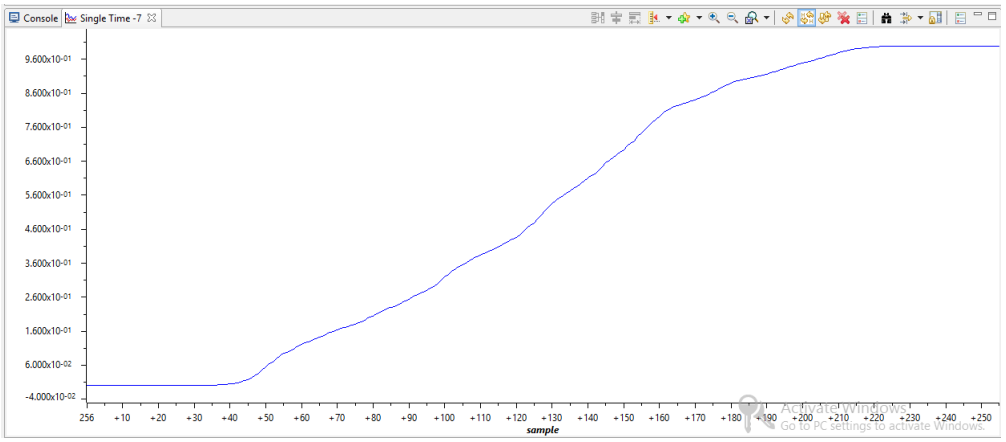


FIGURE 11.35: CDF of the given image

Result

Computed the normalized histogram and CDF of the given image.

2.Grey level slicing

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Read the image intensity values

Step 4: If 150 <= *intensity* <= 230 do step 5, else step6

Step 5: $newintensity = oldintensity$

Step 6: $newintensity = 0$

Step 7: Stop

Program

```
# include "tumor.h"
# include"stdio.h"
short i;
unsigned char tum[4096];
int main(void) {
for(i=0;i<N;i++){
if(h[i]>=150 & h[i]<=230)
tum[i]=h[i];
else
tum[i]=0;

}
return 0;
}
```

Output

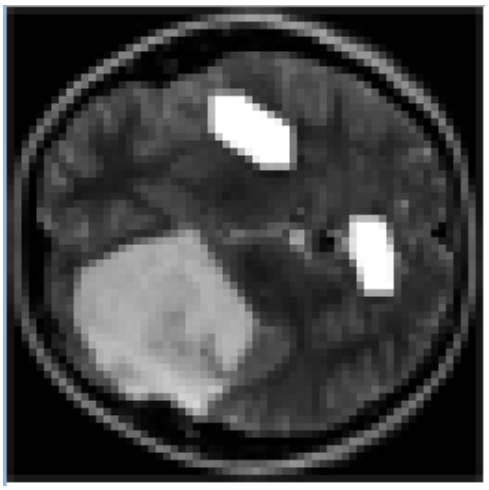


FIGURE 11.36: Original MRI image

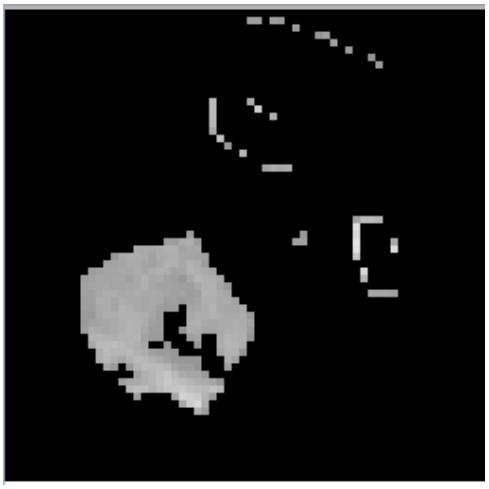


FIGURE 11.37: Grey level sliced image

FIGURE 11.38: Grey level slicing

Result

The intensities between 150 and 230 were segmented from the given image.

3.Contrast Stretching

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Read the intensity values h[i]

Step 4: If $h[i] <= 40$ then $tum[i] = 5 * h[i]$

Step 5: else $tum[i] = 0.266 * h[i]$

Step 6: Stop

Program

```
# include "lena3.h"
# include"stdio.h"
short i;
unsigned char tum[4096];
int main(void) {

for(i=0;i<N;i++){
if(h[i]<=40)
tum[i]=(unsigned char) 5*h[i];
else
tum[i]=(unsigned char) 0.266*h[i];

return 0;
}
```

Output

.5

FIGURE 11.39: Original low contrast image

.5

FIGURE 11.40: After contrast stretching

Result

Performed contrast stretching based on the given specifications.

4.Histogram Equalization

Algorithm

Step 1: Start

Step 2: Initialise the required headerfiles and variables

Step 3: Read the intensity values and create the CDF

Step 4: Multiply the CDF by 255 to get histeqcdf[i]

Step 5: Do $out[i] = histeqcdf[h[i]]$ for i from 0 to N

Step 6: Find the histogram of the equalised image out

Step 7: Stop

Program

```
# include "lena3.h"
# include"stdio.h"
short i;
float hist[256],hist_eq[256];
unsigned char hist_eq_cdf[256],out[4096];
float hist_norm[256],cdf[256],p,cdf_eq[256];
int main(void) {
for(i=0;i<256;i++){
hist[i]=0;
cdf[i]=0;
hist_eq[i]=0;
hist_norm[i]=0;
}
for(i=0;i<N;i++){
hist[h[i]]=hist[h[i]]+1;
    }
```

```
p=0;
for(i=0;i<256;i++){
hist_norm[i] = hist[i] *0.000244;
cdf[i]=p+hist_norm[i];
p=cdf[i];
}

for(i=0;i<256;i++){
cdf[i]=cdf[i]*255;
hist_eq_cdf[i] = (unsigned char) cdf[i];
}
for(i=0;i<N;i++){
out[i] = hist_eq_cdf[h[i]];
}

for(i=0;i<N;i++){
hist_eq[out[i]]=hist_eq[out[i]]+1;
    }
p=0;
for(i=0;i<256;i++){
hist_norm[i] = hist[i] *0.000244;
cdf_eq[i]=p+hist_norm[i];
p=cdf_eq[i];
}
return 0;
}
```

Output

.5

FIGURE 11.41: Original Low contrast image

.5

FIGURE 11.42: Image after histogram equalization

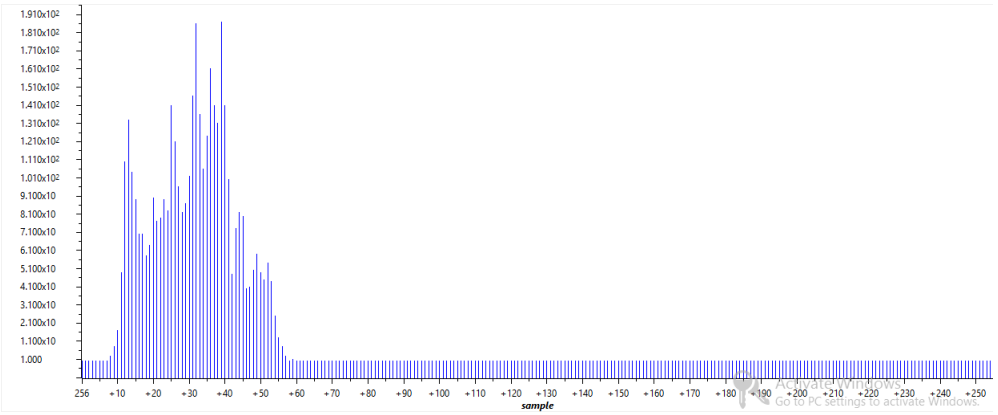


FIGURE 11.43: Histogram of the original image

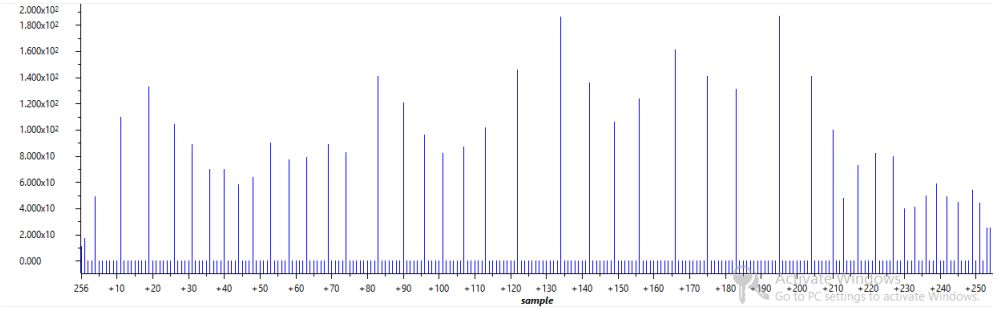


FIGURE 11.44: Equalized histogram

Result

Performed histogram equalization on the given low contrast image and obtained the output.

Chapter 12

Video Processing

Introduction

The DSP STAR TFT LCD Video Daughtercard (VM3224K2) is a video in/out hardware module, which provides developers with an easy-to-use, cost-effective way to evaluate and develop video processing algorithm based on TMS320C6000TM DSP.

The VM3224K2 acquires NTSC/PAL analog video signal and displays digital video data on TFT LCD display screen. NTSC(National Television Selection Committee) is the video system or standard used in North America and most of South America. In NTSC, 30 frames are transmitted each second. Each frame is made up of 525 individual scan lines.PAL(Phase Alternating Line) is the predominant video system or standard mostly used overseas. In PAL, 25 frames are transmitted each second. Each frame is made up of 625 individual scan lines.The luminance Y in PAL is given by $Y = 0.299R + 0.587G + 0.114B$ where R, G and B are the different color channels.

It uses an RGB565 pixel expression that is 320x240in size. The RGB565 color formatuses 5 bits for red value, 6 bits for the green value instead of 5 and 5 for the blue value. Therefore, all 16 bits are in use.

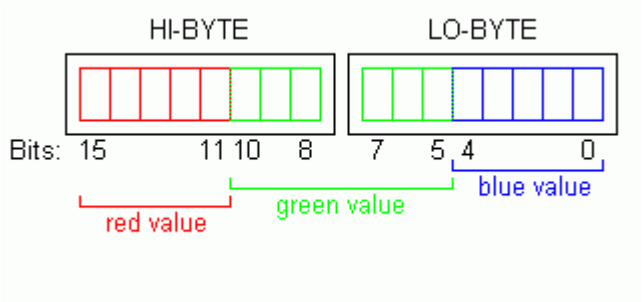


FIGURE 12.1: RGB565 format

The LCD panel must provide pixel data periodically according to the pixel array pattern.The DSP stores image data in the RGB565 format, in order to display it on the TFT LCD. When the DSP stores desired image data in the memory of the video module, the memory location determines a pixel location in the LCD panel.

There are four CPLD registers for control and data processing of the video module which are Control/Status register, Data register,Low and High address registers.There are two DRAMs in the VM3224K2. Each of memory has 512 KB in size. One memory is used as screen buffer and the other is used as capture buffer. The images obtained from NTSC camera are stored in capture buffer. The DSP processor accesses the screen buffer through High address register(ADDH) and Low address register(ADDL). Therefore, the screen buffer data can be read by putting the data address in ADDH and ADDL registers and then reading it from the DATA register.

1.Create a video of a ball moving around the screen display it on LCD video daughtercard using example codes provided.

Output

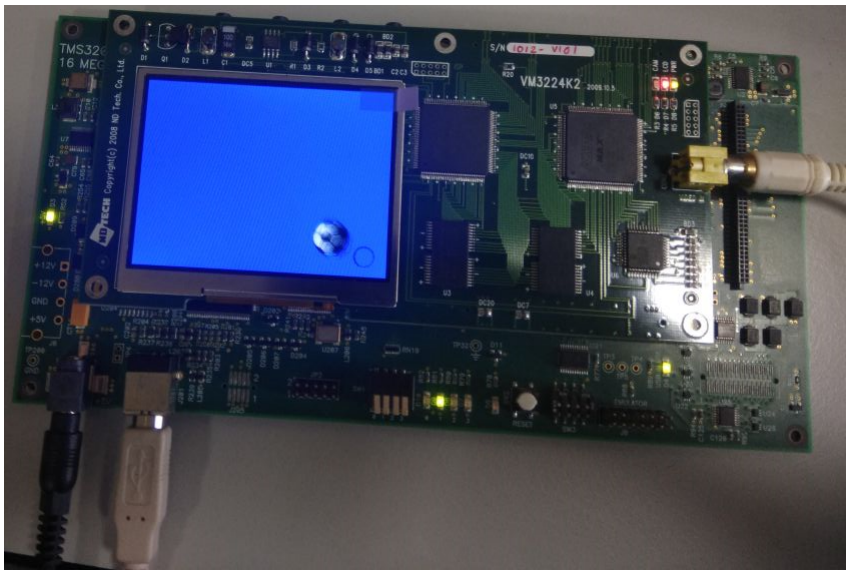


FIGURE 12.2: Ball moving

Observations

As in code, the ball moved bouncing around in the screen.

Results

Created a video of a ball moving around the screen using the example code provided.

2.Capture live video using the CCD camera and LCD Video Daughtercard.

Output

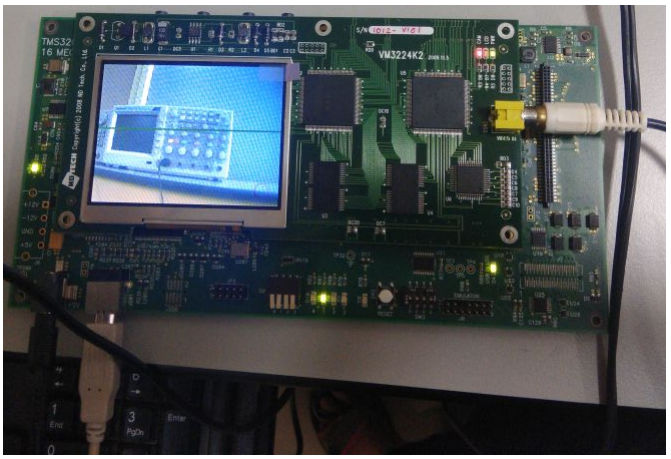


FIGURE 12.3: Vide captured using CCD camera

Observations

Live video could be captured and seen on the LCD screen with very less or no delay.

Results

Implemented live video capturing using CCD camera and LCD video daughtercard on DSK6713.

3.Create a video of different colour bars and display it on LCD video daughtercard using the example programs given.

Output

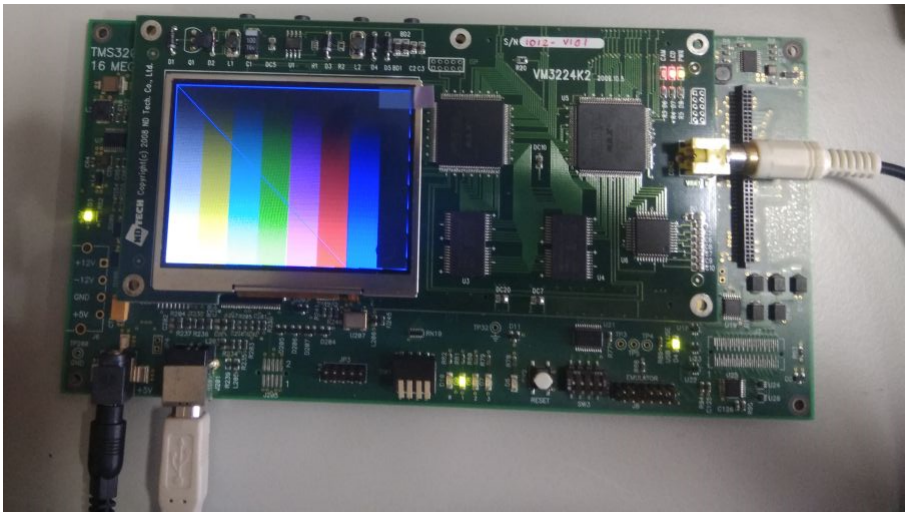


FIGURE 12.4: Video of different colour bands

Observations

Different colour bands could be seen on the LCD screen.

Results

Created a video of different colour bars and displayed it on the LCD screen of video card.

4.Display the lena image on the LCD screen of video daughtercard.

Output

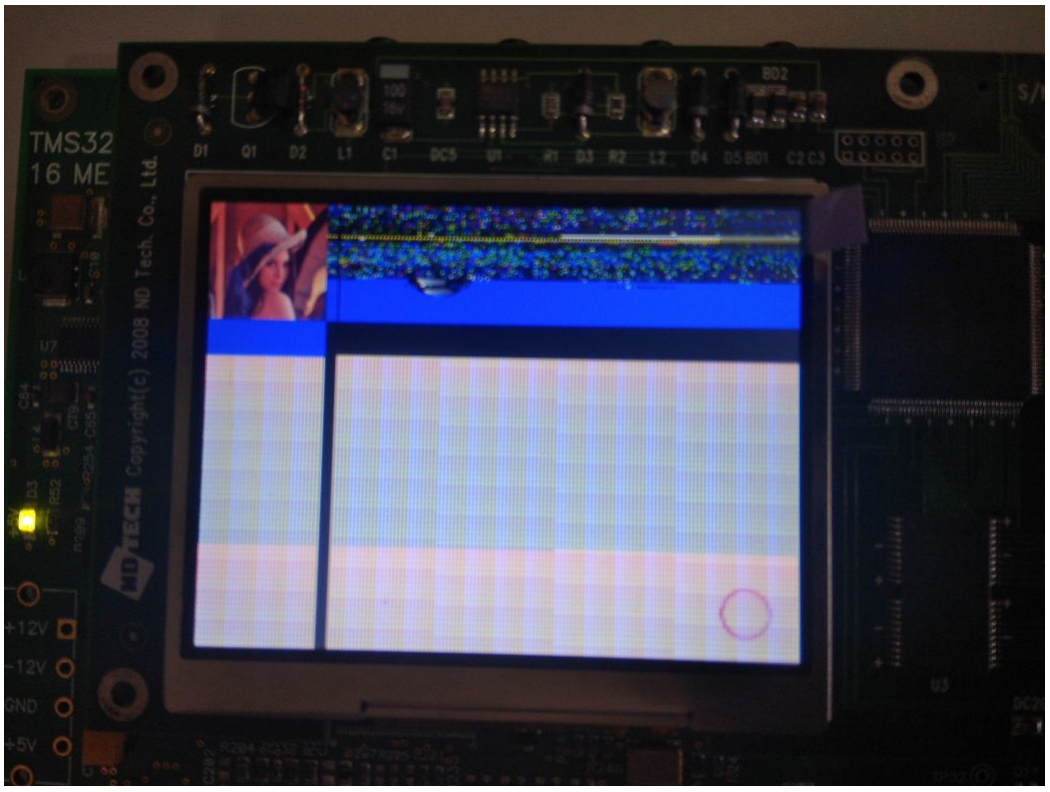


FIGURE 12.5: Lena image displayed on the LCD screen

Observations

The Lena image appeared shifted by 90 degrees at first and after changing the x and y ranges in code, it appeared without any shift but transposed and the rest of the screen had stripes of different colour.

Results

Displayed the Lena colour image on the LCD screen of video daughtercard.

Chapter 13

Appendix

Codes to generate header files 1. FIR header file

```
function dsk_fir67(coeff)
coefflen=length(coeff);
fname = input('enter filename for coefficients: ','s');
fid = fopen(fname,'wt');
fprintf(fid,'// %s\n',fname);
fprintf(fid,'// this file was generated automatically using function dsk_fir67.m
fprintf(fid,'\n#define N %d\n',coefflen);
fprintf(fid,'\nfloat h[N] = { \n');
j=0;
% i is used to count through coefficients
for i=1:coefflen
% if six coeffs have been written to current line
% then start new line
  if j>5
    j=0;
    fprintf(fid,'\n');
  end
% if this is the last coefficient then simply write
% its value to the current line
```

```
% else write coefficient value, followed by comma
  if i==coefflen
   fprintf(fid,'%2.4E',coeff(i));
  else
    fprintf(fid,'%2.4E,',coeff(i));
    j=j+1;
  end
end
fprintf(fid,'\n};\n');
fclose(fid);
```

2.IIR Header file

```
% DSK_SOS_IIR67.M
% MATLAB function to write SOS IIR filter coefficients
% in format suitable for use in C6713 DSK programs
% iirsos.c, iirsosprn.c and iirsosdelta.c
% assumes that coefficients have been exported from
% fdatool as two matrices
% first matrix has format
% [ b10 b11 b12 a10 a11 a12
%   b20 b21 b22 a20 a21 a22
%   ...
% ]
% where bij is the bj coefficient in the ith stage
% second matrix contains gains for each stage
%
function dsk_sos_iir67(coeff,gain)
%
num_sections=length(gain)-1;
fname = input('enter filename for coefficients ','s');
fid = fopen(fname,'wt');
fprintf(fid,'// %s\n',fname);
```

```
fprintf(fid,'// this file was generated automatically using function dsk_sos_iir
fprintf(fid,'\n#define NUM_SECTIONS %d\n',num_sections);
% first write the numerator coefficients b
% i is used to count through sections
fprintf(fid,'\nfloat b[NUM_SECTIONS][3] = { \n');
for i=1:num_sections
  if i==num_sections
      fprintf(fid,'{%2.8E, %2.8E, %2.8E} };\n',coeff(i,1)*gain(i),coeff(i,2)*gai
  else
      fprintf(fid,'{%2.8E, %2.8E, %2.8E},\n',coeff(i,1)*gain(i),coeff(i,2)*gain(
  end
end
% then write the denominator coefficients a
% i is used to count through sections
fprintf(fid,'\nfloat a[NUM_SECTIONS][2] = { \n');
for i=1:num_sections
  if i==num_sections
      fprintf(fid,'{%2.8E, %2.8E} };\n',coeff(i,5),coeff(i,6));
  else
      fprintf(fid,'{%2.8E, %2.8E},\n',coeff(i,5),coeff(i,6));
  end
end
fclose(fid);
```

3. Image header file

```
% DSK_FIR67.M
% MATLAB function to write FIR filter coefficients
% in format suitable for use in C6713 DSK programs
% fir.c and firprn.c
% written by Donald Reay
% %
% function dsk_image67(inp_image)
```

```
%
image = imread('Lena.jpg');
channel = 1;
img = image(:,:,channel);
imshow(image);
img = imresize(img,[64 64]);
 inp_image = img;%temp1;
[r,c]=size(inp_image);
fname = input('enter filename for image: ','s');
fid = fopen(fname,'wt');
fprintf(fid,'// %s\n',fname);
fprintf(fid,'// this file was generated automatically using function dsk_fir67.m
fprintf(fid,'\n#define R %d\n',r);
fprintf(fid,'\n#define C %d\n',c);
fprintf(fid,'\n#define N %d\n',r*c);
fprintf(fid,'\n unsigned char h[N] = { \n');
coeff = inp_image(:);
% j is used to count coefficients written to current line
% in output file
j=0;
% i is used to count through coefficients
for i=1:r*c
% if six coeffs have been written to current line
% then start new line
  if j>50
    j=0;
    fprintf(fid,'\n');
  end
% if this is the last coefficient then simply write
% its value to the current line
% else write coefficient value, followed by comma
```

```
  if i==r*c
   fprintf(fid,'%d',coeff(i));
  else
    fprintf(fid,'%d,',coeff(i));
    j=j+1;
  end
end
fprintf(fid,'\n};\n');
fclose(fid);
```

4. TFT LCD Video Daughter Card (VM3224K2) codes

a. Ball1

```
#include "vm3224k.h"
#include "ball.dat"

#define CE2CTL *(volatile int *)(0x01800010)
// Definitions for async access(change as you wish)
#define WSU (2<<28)  // Write Setup : 0-15
#define WST (8<<22)  // Write Strobe: 0-63
#define WHD (2<<20)  // Write Hold  : 0-3
#define RSU (2<<16)  // Read Setup  : 0-15
#define TA (3<<14)  // Turn Around : 0-3
#define RST (8<<8)  // Read Strobe : 0-63
#define RHD (2<<0)  // Read Hold   : 0-3
#define MTYPEA (2<<4)

short buf[240][320];

void PLL6713()
{
```

```
int i;
// CPU Clock Input : 50MHz

*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) & 0xfffffffe;
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) | 0x08;
*(volatile int *)(0x01b7c114) = 0x08001;  // 50MHz/2 = 25MHz
  *(volatile int *)(0x01b7c110) = 0x0c;     // 25MHz * 12 = 300MHz
*(volatile int *)(0x01b7c118) = 0x08000;  // SYSCLK1 = 300MHz/1 = 300MHz
*(volatile int *)(0x01b7c11c) = 0x08001;  // SYSCLK2 = 300MHz/2 = 150MHz // Peri
*(volatile int *)(0x01b7c120) = 0x08003;  // SYSCLK3 = 300MHz/4 =  75MHz // SDRA
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) & 0xfffffff7;
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) | 0x01;
}

void mem2lcd(short *x)
{
    int i,j;
    VM3224ADDH = 0; // Select LCD screen buffer and Set address (0<<15|0x00000)
    for (j=0;j<240;j++)
    for (i=0;i<320;i++) VM3224DATA=x[j*320+i];
}

void main()
{
int  i,j,m,n,x,y;
PLL6713();
CE2CTL = (WSU|WST|WHD|RSU|RST|RHD|MTYPEA);
vm3224init();       // Initialize vm3224k2
```

```
vm3224rate(3); // Set frame rate
vm3224bl(15);       // Set backlight

for (j=0;j<240;j++) {
for(i=0;i<320;i++) {
buf[j][i] = BG_COLOR;
}
}

m=n=0;
x=y=2;
while (1) {
for (j=0;j<BALL_SIZE;j++) {
for(i=0;i<BALL_SIZE;i++) {
buf[m+i][n+j] = ball[j][i];
}
}
if ((m+x>240-BALL_SIZE)||(m+x<0)) x = -x;
if ((n+y>320-BALL_SIZE)||(n+y<0)) y = -y;
m += x;
n += y;
mem2lcd((short *)buf);
}
}
```

b. Cam2lcd2

```
 #include "vm3224k.h"

#define CE2CTL *(volatile int *)(0x01800010)
// Definitions for async access(change as you wish)
```

```
#define WSU (2<<28)  // Write Setup : 0-15
#define WST (8<<22)  // Write Strobe: 0-63
#define WHD (2<<20)  // Write Hold  : 0-3
#define RSU (2<<16)  // Read Setup  : 0-15
#define TA (3<<14)  // Turn Around : 0-3
#define RST (8<<8)  // Read Strobe : 0-63
#define RHD (2<<0)  // Read Hold   : 0-3
#define MTYPEA (2<<4)

#pragma  DATA_SECTION ( lcd,".sdram" )
#pragma  DATA_SECTION ( cam,".sdram" )
short lcd[240][320];
short cam[240][320];
short rgb[64][32][32];

void PLL6713()
{
int i;
// CPU Clock Input : 50MHz

*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) & 0xfffffffe;
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) | 0x08;
*(volatile int *)(0x01b7c114) = 0x08001;  // 50MHz/2 = 25MHz
  *(volatile int *)(0x01b7c110) = 0x0c;     // 25MHz * 12 = 300MHz
*(volatile int *)(0x01b7c118) = 0x08000;  // SYSCLK1 = 300MHz/1 = 300MHz
*(volatile int *)(0x01b7c11c) = 0x08001;  // SYSCLK2 = 300MHz/2 = 150MHz // Peri
*(volatile int *)(0x01b7c120) = 0x08003;  // SYSCLK3 = 300MHz/4 =  75MHz // SDRA
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) & 0xfffffff7;
for(i=0;i<4;i++);
```

```
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) | 0x01;
}

unsigned short ybr_565(short y,short u,short v)
{
int r,g,b;

  b = y + 1.772*(u-128);
if (b<0) b=0;
if (b>255) b=255;
g = y - 0.344*(u-128) - 0.714*(v-128);
if (g<0) g=0;
if (g>255) g=255;
r = y + 1.402*(v-128);
if (r<0) r=0;
if (r>255) r=255;
return ((r&0x0f8)<<8)|((g&0x0fc)<<3)|((b&0x0f8)>>3);
}

void main()
{
int  i,j,k,y0,y1,v0,u0;
PLL6713(); // Initialize C6713 PLL
CE2CTL = (WSU|WST|WHD|RSU|RST|RHD|MTYPEA);
vm3224init();       // Initialize vm3224k2
vm3224rate(3);     // Set frame rate
vm3224bl(15);       // Set backlight

    for (k=0;k<64;k++)  // Create RGB565 lookup table
    for (i=0;i<32;i++)
    for (j=0;j<32;j++) rgb[k][i][j] = ybr_565(k<<2,i<<3,j<<3);
```

```
while (1) {
VM3224ADDH = 0x08000; // Select Cam screen buffer and Set address (1<<15|0x00000
    for (j=0;j<240;j++)
    for (i=0;i<320;i++) cam[j][i]=VM3224DATA;
for (j=0;j<240;j++)
for (i=0;i<320;i+=2) { // Conversion yuv to RGB565
y0 = (cam[j][i]>>8) & 0x0ff;
u0 = cam[j][i] & 0x0ff;
y1 = (cam[j][i+1]>>8) & 0x0ff;
v0 = cam[j][i+1] & 0x0ff;
y0 = y0>>2;
y1 = y1>>2;
u0 = u0>>3;
v0 = v0>>3;
lcd[j][i]=rgb[y0][u0][v0];
lcd[j][i+1]=rgb[y1][u0][v0];
}
VM3224ADDH = 0;    // Select LCD screen buffer and Set address (0<<15|0x00000)
for (j=0;j<240;j++)
for (i=0;i<320;i++) VM3224DATA=lcd[j][i];
}
}
```

c. Colorbar5

```
#include "vm3224k.h"

#define CE2CTL *(volatile int *)(0x01800010)
// Definitions for async access(change as you wish)
#define WSU (2<<28)  // Write Setup : 0-15
```

```
#define WST (8<<22)  // Write Strobe: 0-63
#define WHD (2<<20)  // Write Hold  : 0-3
#define RSU (2<<16)  // Read Setup  : 0-15
#define TA (3<<14)  // Turn Around : 0-3
#define RST (8<<8)  // Read Strobe : 0-63
#define RHD (2<<0)  // Read Hold   : 0-3
#define MTYPEA (2<<4)

// color       :   W   Y   Cy  G   Mg  R   B   Bk
int R[8]={255,255,  0,  0,255,255,  0,  0};
int G[8]={255,255,255,255,  0,  0,  0,  0};
int B[8]={255,  0,255,  0,255,  0,255,  0};
short buf[240][320];

void PLL6713()
{
int i;
// CPU Clock Input : 50MHz

*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) & 0xfffffffe;
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) | 0x08;
*(volatile int *)(0x01b7c114) = 0x08001;  // 50MHz/2 = 25MHz
  *(volatile int *)(0x01b7c110) = 0x0c;     // 25MHz * 12 = 300MHz
*(volatile int *)(0x01b7c118) = 0x08000;  // SYSCLK1 = 300MHz/1 = 300MHz
*(volatile int *)(0x01b7c11c) = 0x08001;  // SYSCLK2 = 300MHz/2 = 150MHz // Peri
*(volatile int *)(0x01b7c120) = 0x08003;  // SYSCLK3 = 300MHz/4 =  75MHz // SDRA
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) & 0xfffffff7;
for(i=0;i<4;i++);
*(volatile int *)(0x01b7c100) = *(volatile int *)(0x01b7c100) | 0x01;
```

```
}

void main()
{
int  i,j,k;
PLL6713();                  // Initialize C6713 PLL
CE2CTL = (WSU|WST|WHD|RSU|RST|RHD|MTYPEA);
vm3224init();               // Initialize vm3224k2
vm3224rate(3);              // Set frame rate
vm3224bl(15);               // Set backlight

for (j=0;j<240;j++) {
for(i=0;i<320;i++) {  // Conversion RGB to RGB565
k = i/40;
buf[j][i] = (((R[k]*j/240)&0x0f8)<<8)
      | (((G[k]*j/240)&0x0fc)<<3)
      | (((B[k]*j/240)&0x0f8)>>3);
}
}
                            // Draw border lines and a diagonal line
for (i=0;i<320;i++) buf[0][i] = 0x001f;
for (j=0;j<240;j++) buf[j][0] = 0x001f;
for (j=0;j<240;j++) buf[j][j] = 0x001f;
for (i=0;i<320;i++) buf[239][i] = i;
for (j=0;j<240;j++) buf[j][319] = j;

while (1) {
VM3224ADDH = 0;       // Select LCD screen buffer and Set address (0<<15|0x00000
for (j=0;j<240;j++)
for (i=0;i<320;i++) VM3224DATA=buf[j][i];
}
```

```
}
```