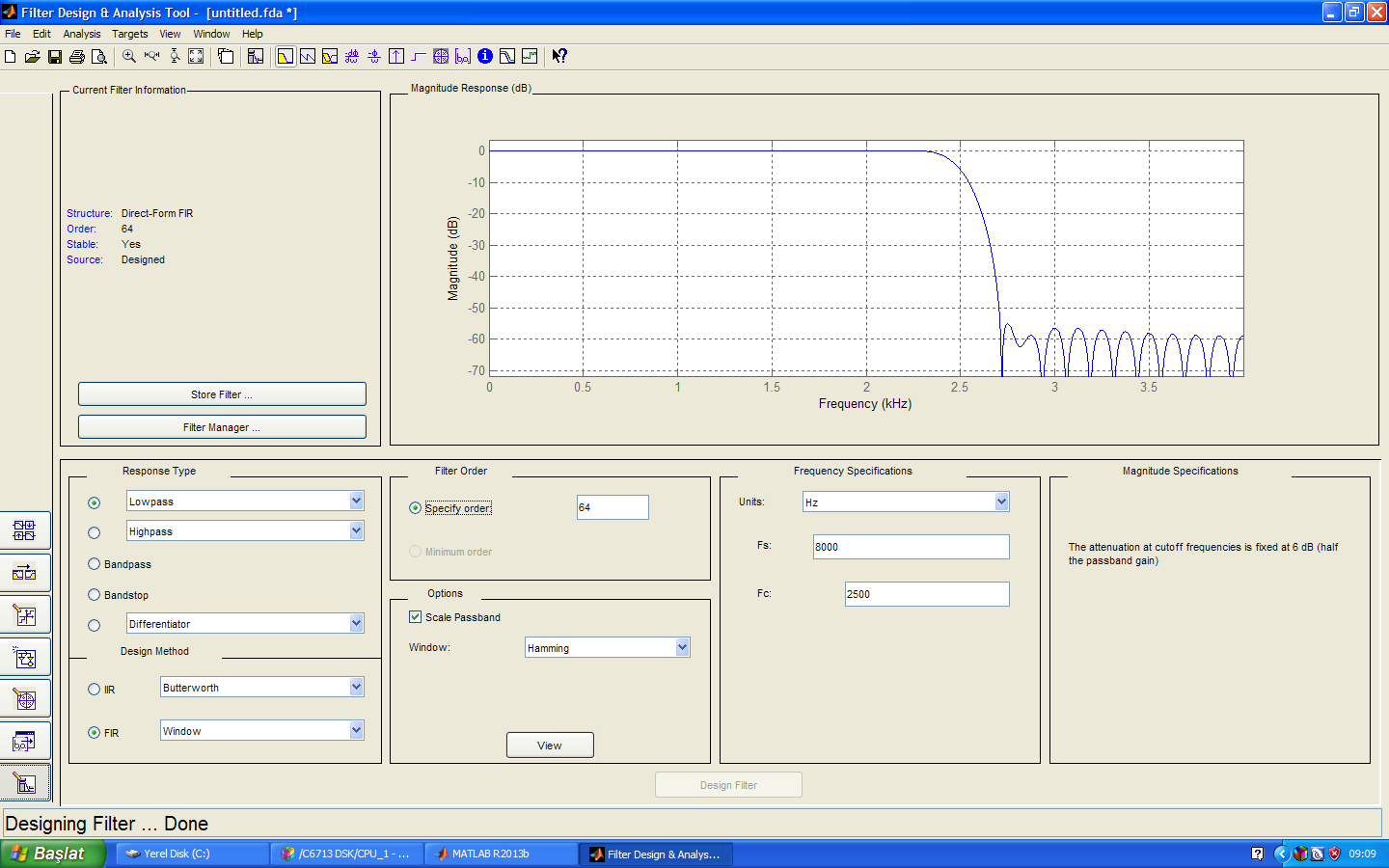
**SAYISAL İŞARET İŞLEME LABARATUARI ÖDEV 2**

**SORU 1)a.**

Hanning penceresi kullanılarak 2500 Hz kesim frekanslı 8000 Hz örnekleme frekanslı 65 katsayılı filtreyi matlab da tasarladık.

****

firprnbuf.c dosyasını kullanarak gerçekleştirdik ve osiloskop görüntüsünü aldık.

// program firprnbuf.c

#include "DSK6713\_AIC23.h" // codec support

Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_LINE; // select line in

#include "Hamming.cof" //filter coefficient file

#include "noise\_gen.h" //support file for noise sequence generation

int fb; //feedback variable

shift\_reg sreg; //shift register

#define NOISELEVEL 8000 //scale factor for +/- 1 noise sequence

float x[N]; //filter delay line

#define YNBUFLENGTH 1024

float yn\_buffer[YNBUFLENGTH];

short ynbufindex = 0;

int prand(void) //pseudo-random noise generation

{

int prnseq;

if(sreg.bt.b0)

prnseq = -NOISELEVEL; //scaled negative noise level

else

prnseq = NOISELEVEL; //scaled positive noise level

fb =(sreg.bt.b0)^(sreg.bt.b1); //XOR bits 0,1

fb^=(sreg.bt.b11)^(sreg.bt.b13); //with bits 11,13 -> fb

sreg.regval<<=1; //shift register 1 bit to left

sreg.bt.b0=fb; //close feedback path

return prnseq;

}

void resetreg(void) //reset shift register to nominal values

{

sreg.regval=0xFFFF; //initial seed value

fb = 1; //initial feedback value

}

interrupt void c\_int11() //interrupt service routine

{

short i; //declare index variable

float yn = 0.0;

x[0] = (float)(prand()); //get new input into delay line

for (i=0 ; i<N ; i++) //calculate filter output

yn += h[i]\*x[i];

for (i=(N-1) ; i>0 ; i--) //shuffle delay line contents

x[i] = x[i-1];

output\_left\_sample((short)(yn)); //output to codec

yn\_buffer[ynbufindex++] = yn;

if(ynbufindex >= YNBUFLENGTH) ynbufindex = 0;

return; //return from interrupt

}

void main()

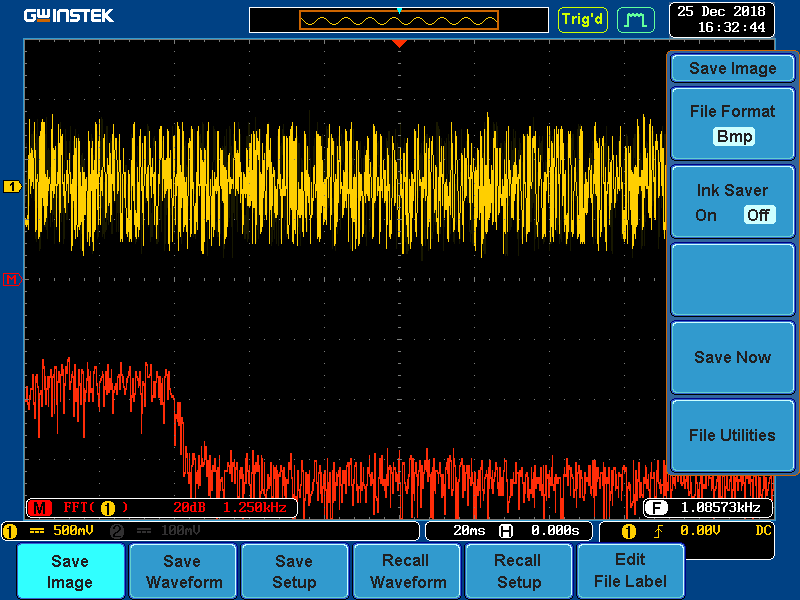
{

resetreg(); //reset shift register to nominal values

comm\_intr(); //initialise McBSP and AD535

while (1); //infinite loop

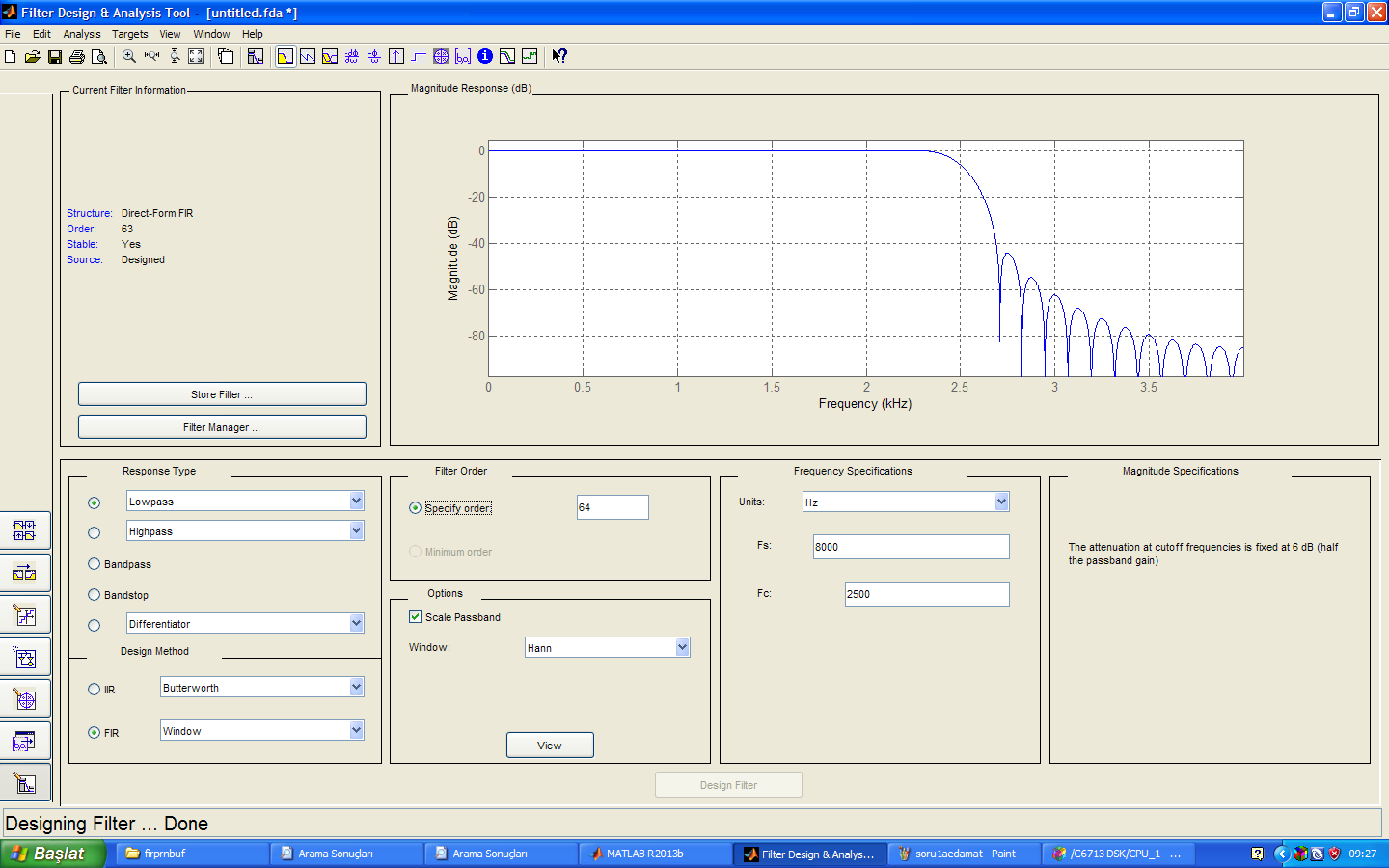
}

****

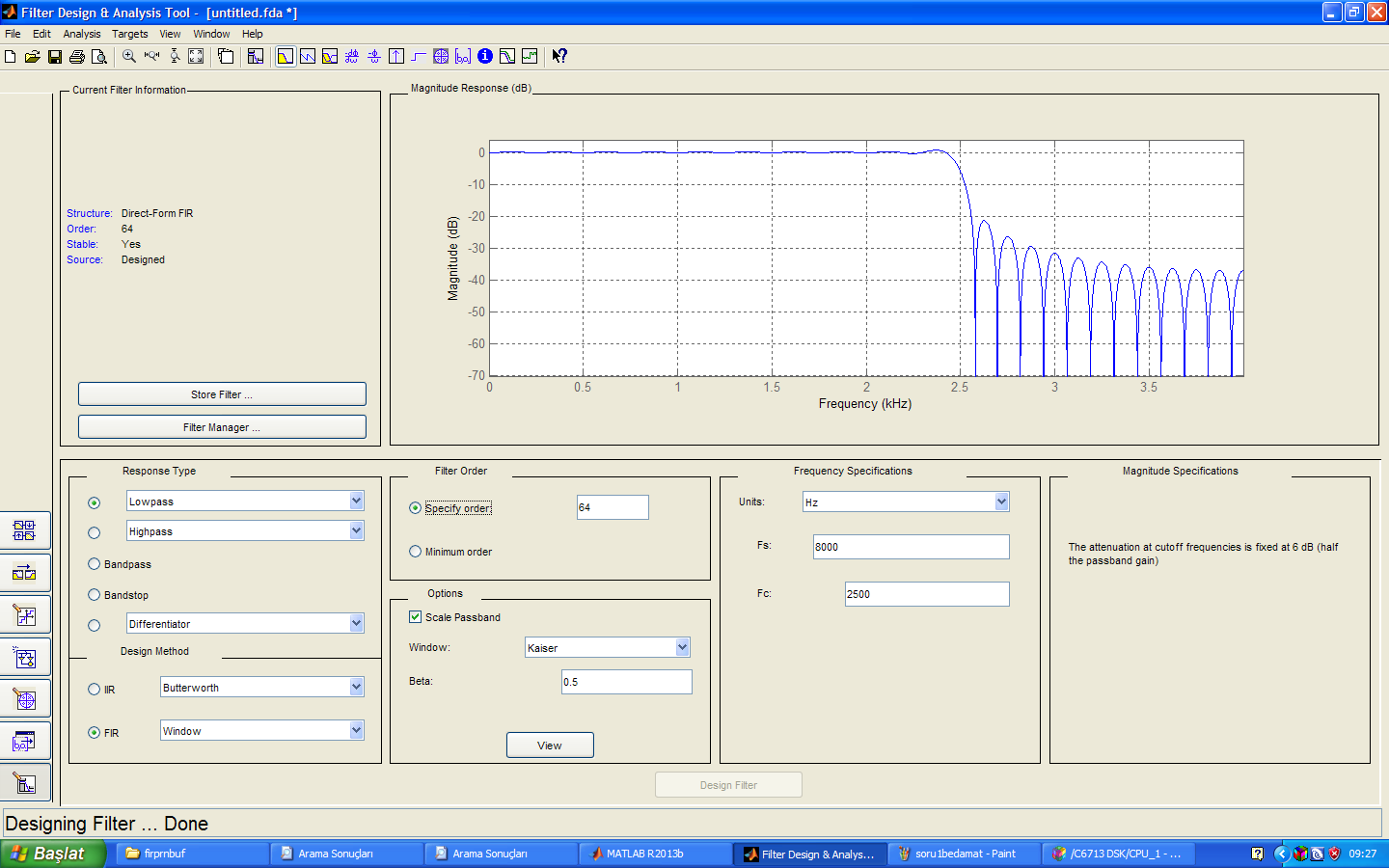
**Soru 1)b.**

A şıkkında hanning penceresi kullanarak yaptığımız filtre karakteristiğini Hann ve kaiser penceresi kullanarak tasarladık.

HANN

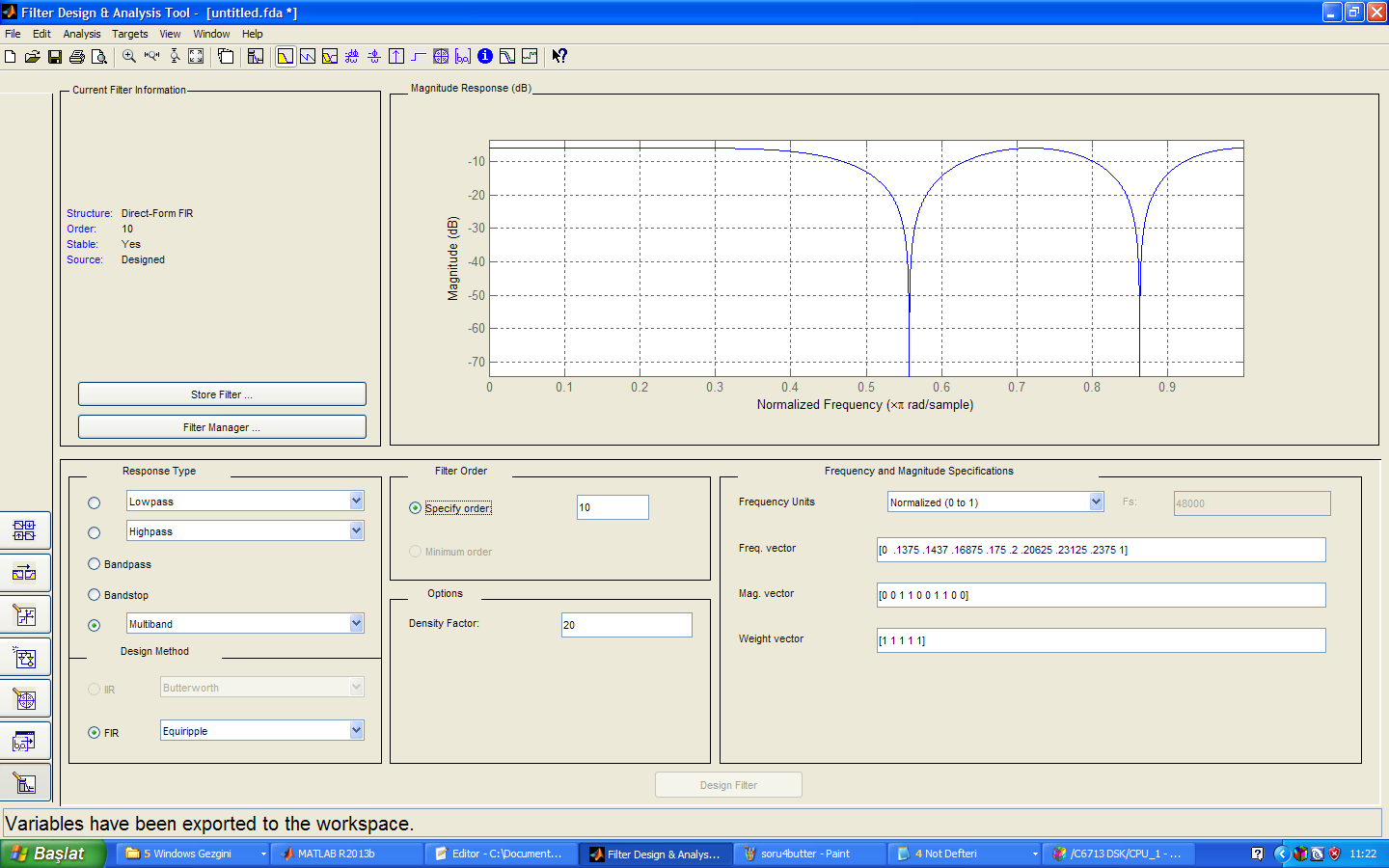


KAİSER



**Soru 2)**

2500 Hz ve 3500 Hz iki geçirme bandına sahip fir filtreyi Matlab da tasarladık.

****

**Soru 3)**

Firprn.c programında C kodlu üretilen gürültü yerine ASM kodlu gürültüyü üreten kodu yazdık.

// program fir.c

#include "DSK6713\_AIC23.h" // codec support

Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_LINE; // select line in

#include "bs2700f.cof" //filter coefficient file

float x[N]; //filter delay line

int previous\_seed;

short pos=16000, neg=-16000;

interrupt void c\_int11() //ISR - AIC23 codec interrupts at 8kHz

{

short i;

float yn = 0.0;

previous\_seed = noisefunc(previous\_seed); //call ASM function

if (previous\_seed & 0x01) x[0] = (short)(pos);//positive scaling

else x[0] = (short)(neg);

//x[0] = (float)(input\_left\_sample()); //get new input into delay line

for (i=0 ; i<N ; i++) //calculate filter output

yn += h[i]\*x[i];

for (i=(N-1) ; i>0 ; i--) //shuffle delay line contents

x[i] = x[i-1];

output\_left\_sample((short)(yn)); //output to codec

return;

}

void main() //main body of program does nothing

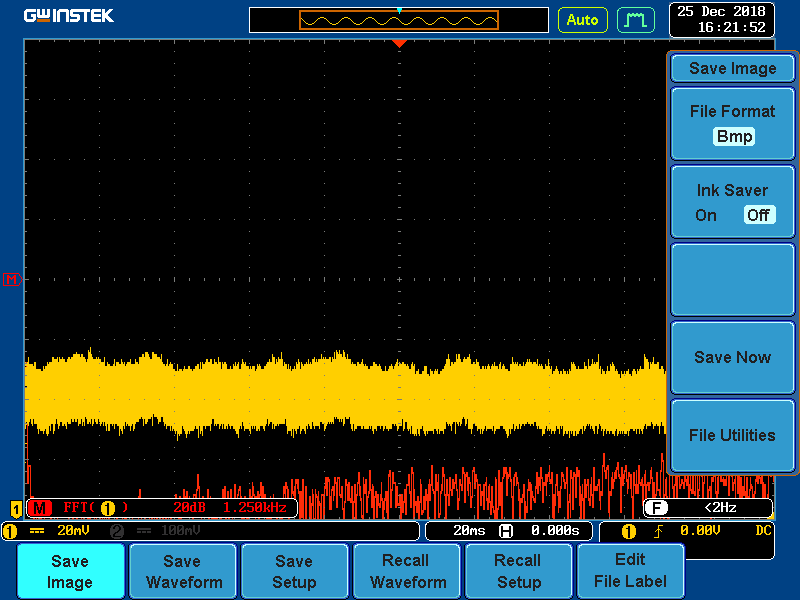
{

comm\_intr(); //initialise DSK

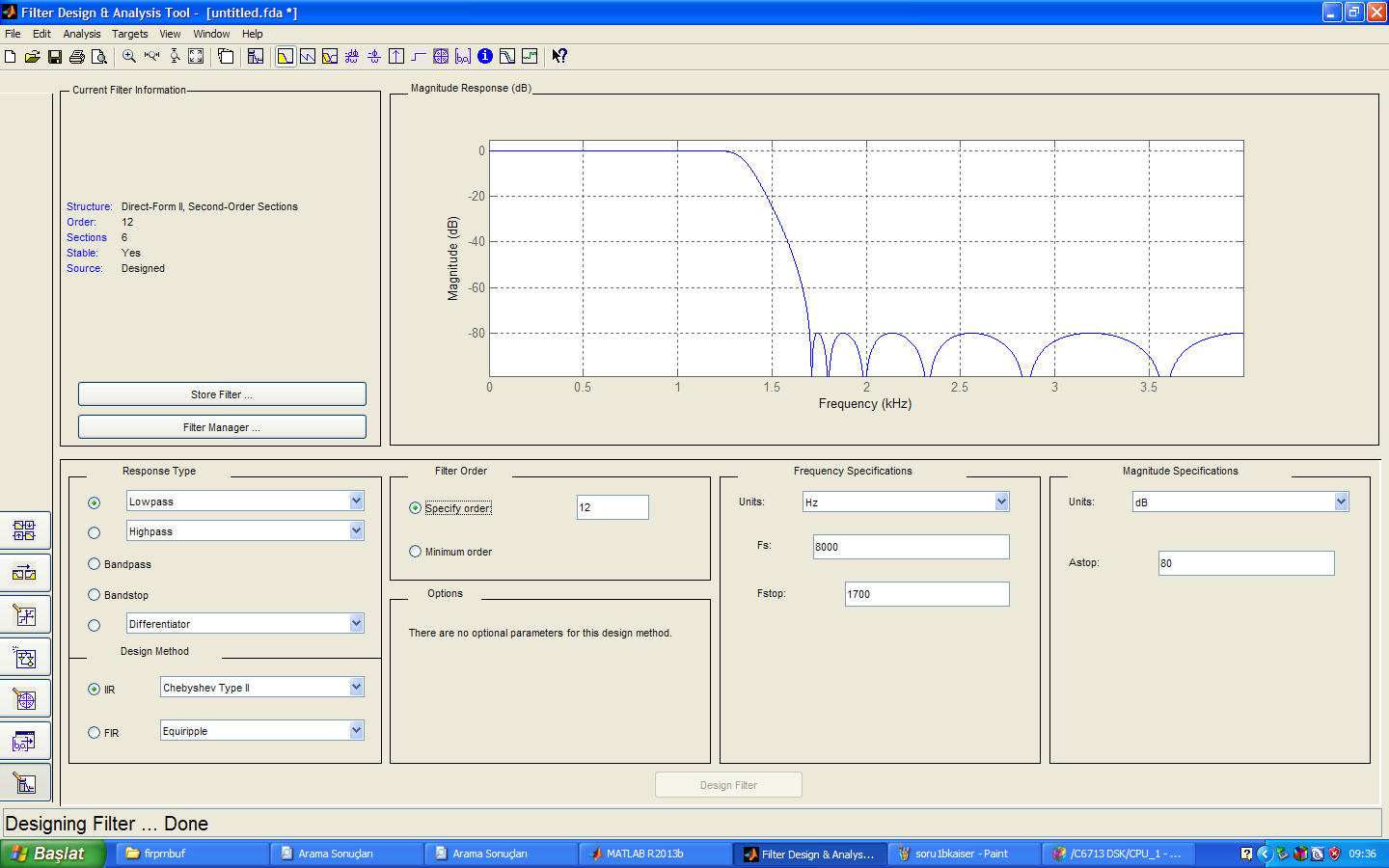
previous\_seed = noisefunc(0x7E521603);

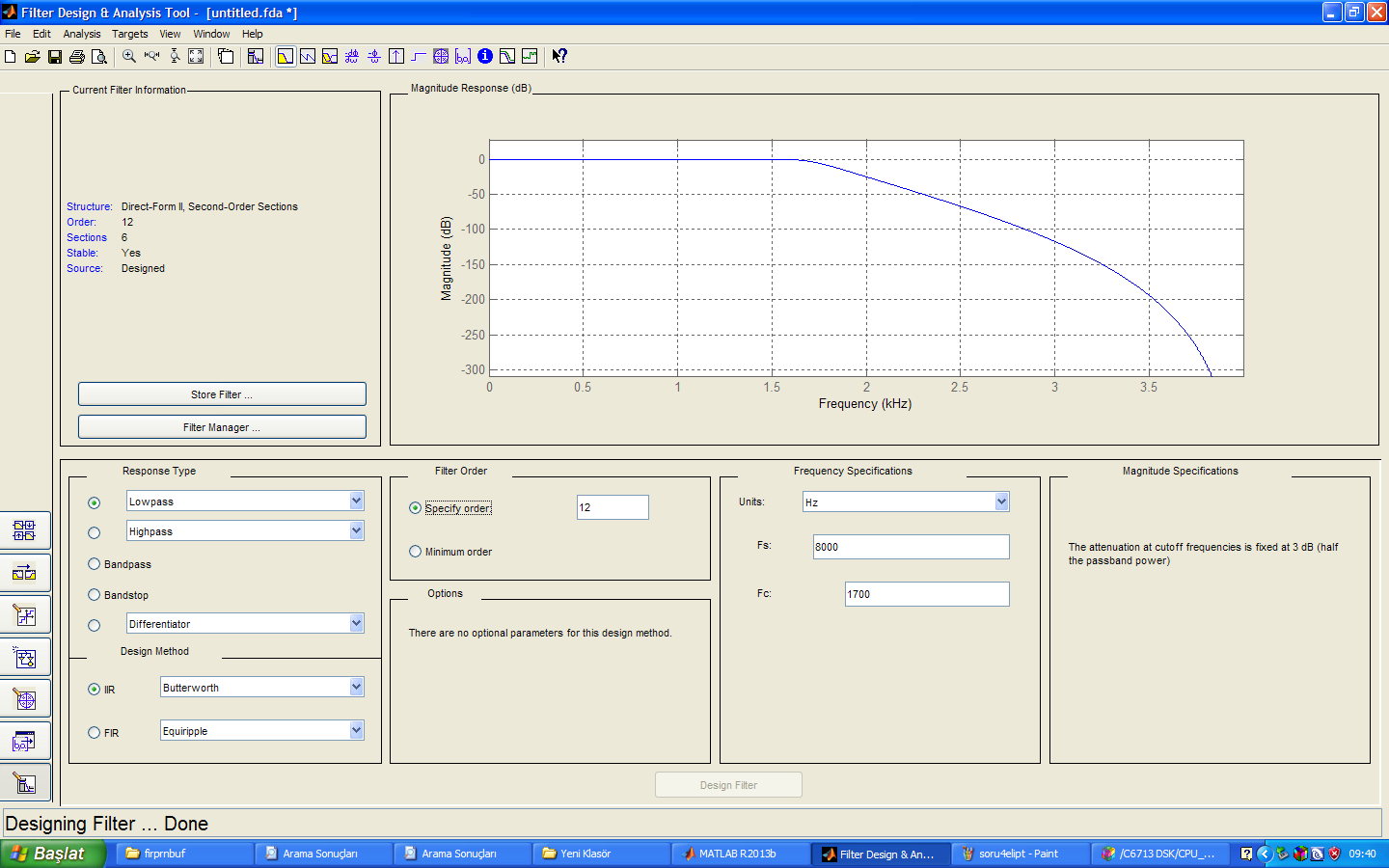
while(1); //infinite loop

}



**Soru 4)**

8000 Hz örnekleme frekanslı 1700 Hz kesim frekanslı 12. Dereceden Chebyshev 2. Alçak geçiren filtre tasarladık. 

Aynı karakteristiğe sahip Butterworth filtresi tasarladık.Butterworth filtenin davranışı oldukça kötü oldu . 

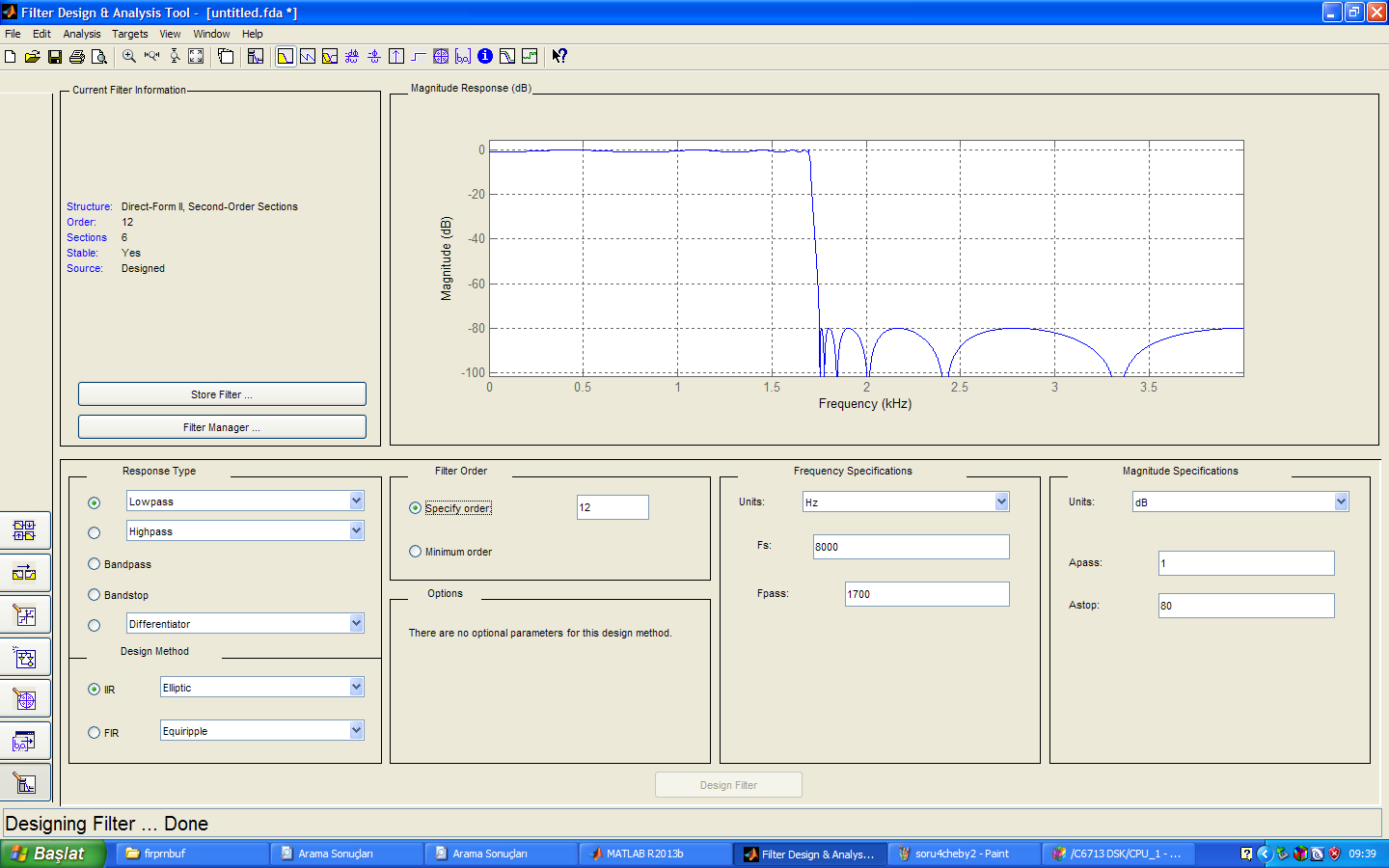
AA

EElEE

Elipti

a

Aynı karakteristikte Eliptik filtresi Chebyshev 2 ye göre geçiş bölgesi daha dik oldu . Durdurma bandı dalgalanması ise birbirine yakın sonuçlar verdi.

****

**Soru 5)**

3200 Hz den başlayıp 400 Hz de sıfırlayan bir fekansla azalan sinüs işareti üreten kodu yazdık.

#include "DSK6713\_AIC23.h" //codec support

Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_LINE; // select mic in

#include <math.h>

#define MIN\_FREQ 400

#define MAX\_FREQ 3200

#define STEP\_FREQ 20

#define SWEEP\_PERIOD 400

#define SAMPLING\_FREQ 8000

#define AMPLITUDE 5000

#define PI 3.14159265358979

float y[3] = {0.0, 0.0, 0.0};

float a1;

float freq =MAX\_FREQ ;

short sweep\_count = 0;

void coeff\_gen(float freq)

{

a1 = 2.0\*cos(2.0\*PI\*freq/SAMPLING\_FREQ);

y[0] = 0.0;

y[1] = sin(2.0\*PI\*freq/SAMPLING\_FREQ);

y[2] = 0.0;

return;

}

interrupt void c\_int11() //ISR

{

sweep\_count++;

if (sweep\_count >= SWEEP\_PERIOD)

{

if (freq <= MIN\_FREQ) freq = MAX\_FREQ;

else freq -= STEP\_FREQ;

coeff\_gen(freq);

sweep\_count = 0;

}

y[0] =(y[1]\*a1)-y[2];

y[2] = y[1]; //update y1(n-2)

y[1] = y[0]; //update y1(n-1)

output\_left\_sample((short)(y[0]\*AMPLITUDE)); //output result

return; //return to main

}

void main()

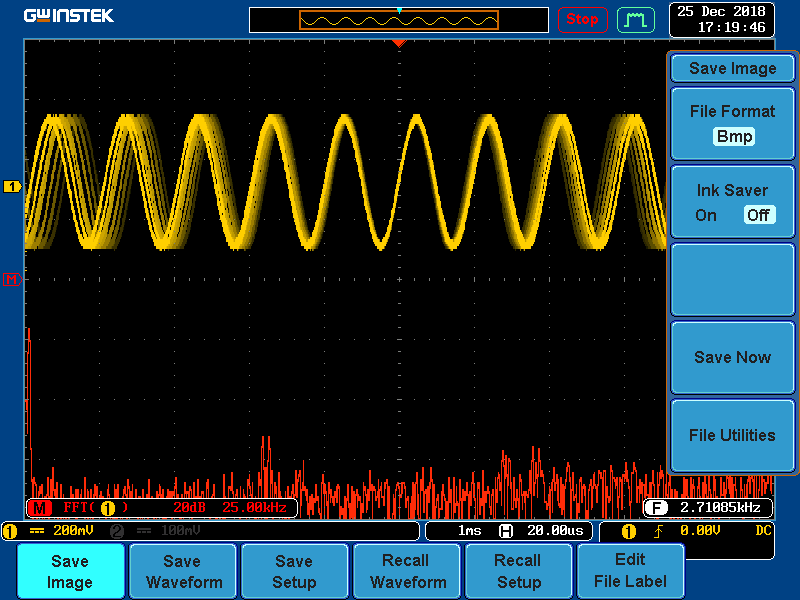
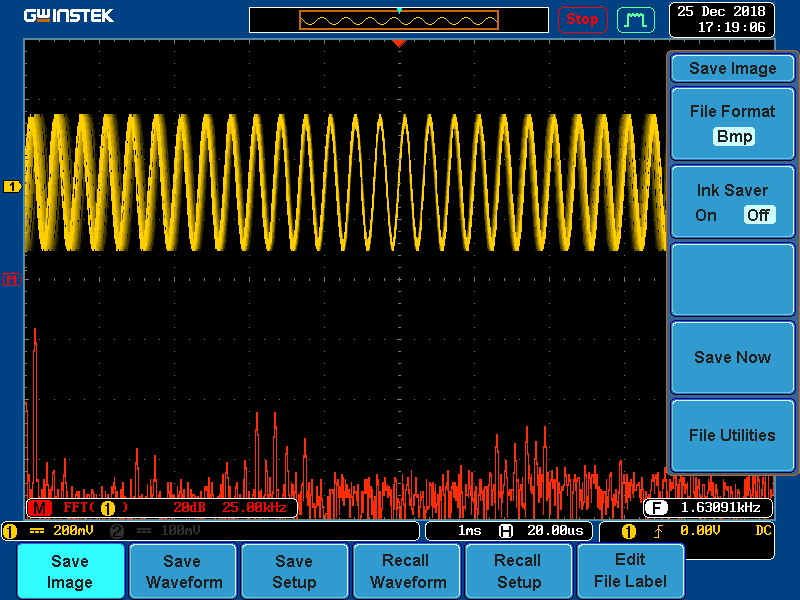
{

y[1] = sin(2.0\*PI\*freq/SAMPLING\_FREQ);

a1 = 2.0\*cos(2.0\*PI\*freq/SAMPLING\_FREQ);

comm\_intr(); //init DSK, codec, McBSP

while(1); //infinite loop

}

**Soru 6)**

2. dereceden 2 filte katsayılarını seri bağlayarak 4. Derceden IIR filtreyi elde ettik.

// iirsosprn.c generic iir filter using cascaded second order sections

// input from PRBS generator function, output to line out

// float coefficients read from included .cof file

#include "DSK6713\_AIC23.h" //codec-DSK interface support

Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_LINE;

#include "son.cof"

float w[NUM\_SECTIONS][2] = {0};

float w1[NUM\_SECTIONS][2] = {0};

#include "noise\_gen.h" //support file for noise sequence generation

int fb; //feedback variable

shift\_reg sreg; //shift register

#define NOISELEVEL 8000 //scale factor for +/- 1 noise sequence

int prand(void) //pseudo-random noise generation

{

int prnseq;

if(sreg.bt.b0)

prnseq = -NOISELEVEL; //scaled negative noise level

else

prnseq = NOISELEVEL; //scaled positive noise level

fb =(sreg.bt.b0)^(sreg.bt.b1); //XOR bits 0,1

fb^=(sreg.bt.b11)^(sreg.bt.b13); //with bits 11,13 -> fb

sreg.regval<<=1; //shift register 1 bit to left

sreg.bt.b0=fb; //close feedback path

return prnseq;

}

void resetreg(void)

{

sreg.regval = 0xFFFF; //shift register to nominal values

fb = 1; //initial feedback value

return;

}

interrupt void c\_int11() //interrupt service routine

{

int section; // index for section number

float input; // input to each section

float wn,yn; // intermediate and output values in each stage

int section1; // index for section number

float input1; // input to each section

float wn1,yn1; // intermediate and output values in each stage

// intermediate and output values in each stage

// intermediate and output values in each stage

input =((float)prand()); // input to first section read from codec

for (section=0 ; section< NUM\_SECTIONS ; section++)

{

wn = input - a[section][0]\*w[section][0] - a[section][1]\*w[section][1];

yn = b[section][0]\*wn + b[section][1]\*w[section][0] + b[section][2]\*w[section][1];

w[section][1] = w[section][0];

w[section][0] = wn;

input = yn; // output of current section will be input to next

}

//output\_left\_sample((short)yn);

input1 =(yn); // input to first section read from codec

for (section1=0 ; section1< NUM\_SECTIONS ; section1++)

{

wn1 = input1 - a1[section1][0]\*w1[section1][0] - a1[section1][1]\*w1[section1][1];

yn1 = b1[section1][0]\*wn1 + b1[section1][1]\*w1[section1][0] + b1[section1][2]\*w1[section1][1];

w1[section1][1] = w1[section1][0];

w1[section1][0] = wn1;

input1 = yn1; // output of current section will be input to next

}

output\_left\_sample((short)yn1); // before writing to codec

return; //return from ISR

}

void main()

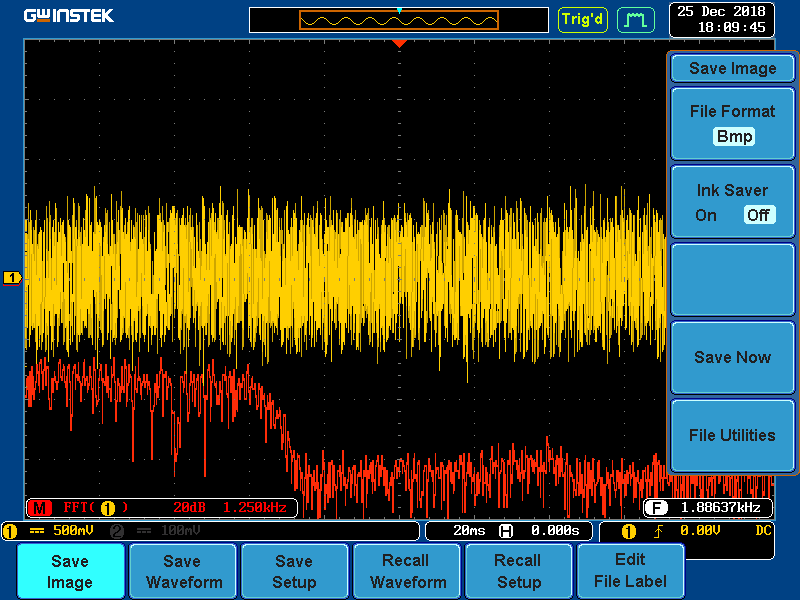
{

resetreg();

comm\_intr(); //init DSK, codec, McBSP

while(1); //infinite loop

}

^

2.dereceden 3 tane filtre katsayılarını seri bağlayarak 6.derecen filtre tasarladık.

// iirsosprn.c generic iir filter using cascaded second order sections

// input from PRBS generator function, output to line out

// float coefficients read from included .cof file

#include "DSK6713\_AIC23.h" //codec-DSK interface support

Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_LINE;

#include "son.cof"

float w[NUM\_SECTIONS][2] = {0};

float w1[NUM\_SECTIONS][2] = {0};

float w2[NUM\_SECTIONS][2] = {0};

#include "noise\_gen.h" //support file for noise sequence generation

int fb; //feedback variable

shift\_reg sreg; //shift register

#define NOISELEVEL 8000 //scale factor for +/- 1 noise sequence

int prand(void) //pseudo-random noise generation

{

int prnseq;

if(sreg.bt.b0)

prnseq = -NOISELEVEL; //scaled negative noise level

else

prnseq = NOISELEVEL; //scaled positive noise level

fb =(sreg.bt.b0)^(sreg.bt.b1); //XOR bits 0,1

fb^=(sreg.bt.b11)^(sreg.bt.b13); //with bits 11,13 -> fb

sreg.regval<<=1; //shift register 1 bit to left

sreg.bt.b0=fb; //close feedback path

return prnseq;

}

void resetreg(void)

{

sreg.regval = 0xFFFF; //shift register to nominal values

fb = 1; //initial feedback value

return;

}

interrupt void c\_int11() //interrupt service routine

{

int section; // index for section number

float input; // input to each section

float wn,yn; // intermediate and output values in each stage

int section1; // index for section number

float input1; // input to each section

float wn1,yn1; // intermediate and output values in each stage

int section2; // index for section number

float input2; // input to each section

float wn2,yn2; // intermediate and output values in each stage

// intermediate and output values in each stage

input =((float)prand()); // input to first section read from codec

for (section=0 ; section< NUM\_SECTIONS ; section++)

{

wn = input - a[section][0]\*w[section][0] - a[section][1]\*w[section][1];

yn = b[section][0]\*wn + b[section][1]\*w[section][0] + b[section][2]\*w[section][1];

w[section][1] = w[section][0];

w[section][0] = wn;

input = yn; // output of current section will be input to next

}

//output\_left\_sample((short)yn);

input1 =(yn); // input to first section read from codec

for (section1=0 ; section1< NUM\_SECTIONS ; section1++)

{

wn1 = input1 - a1[section1][0]\*w1[section1][0] - a1[section1][1]\*w1[section1][1];

yn1 = b1[section1][0]\*wn1 + b1[section1][1]\*w1[section1][0] + b1[section1][2]\*w1[section1][1];

w1[section1][1] = w1[section1][0];

w1[section1][0] = wn1;

input1 = yn1; // output of current section will be input to next

}

input2 =(yn1); // input to first section read from codec

for (section2=0 ; section2< NUM\_SECTIONS ; section2++)

{

wn2 = input2 - a2[section2][0]\*w2[section2][0] - a2[section2][1]\*w2[section2][1];

yn2 = b2[section2][0]\*wn2 + b2[section2][1]\*w2[section2][0] + b2[section2][2]\*w2[section2][1];

w2[section2][1] = w2[section2][0];

w2[section2][0] = wn2;

input2 = yn2; // output of current section will be input to next

}

output\_left\_sample((short)yn2); // before writing to codec

return; //return from ISR

}

void main()

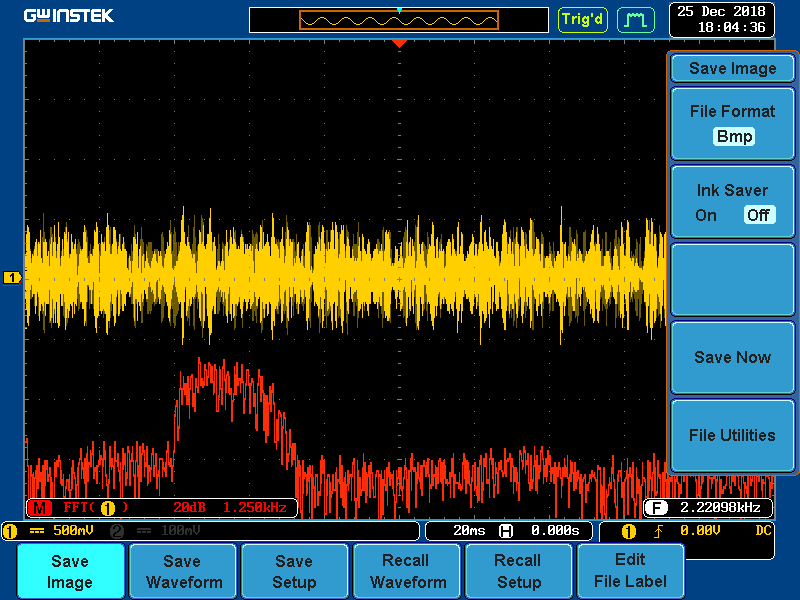
{

resetreg();

comm\_intr(); //init DSK, codec, McBSP

while(1); //infinite loop

}



2.dereceden 4 tane filtre katsayılarını seri bağlayarak 8. Dereceden filtre elde ettik. // iirsosprn.c generic iir filter using cascaded second order sections

// input from PRBS generator function, output to line out

// float coefficients read from included .cof file

#include "DSK6713\_AIC23.h" //codec-DSK interface support

Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_LINE;

#include "impinv.cof"

float w[NUM\_SECTIONS][2] = {0};

float w1[NUM\_SECTIONS][2] = {0};

float w2[NUM\_SECTIONS][2] = {0};

float w3[NUM\_SECTIONS][2] = {0};

#include "noise\_gen.h" //support file for noise sequence generation

int fb; //feedback variable

shift\_reg sreg; //shift register

#define NOISELEVEL 8000 //scale factor for +/- 1 noise sequence

int prand(void) //pseudo-random noise generation

{

int prnseq;

if(sreg.bt.b0)

prnseq = -NOISELEVEL; //scaled negative noise level

else

prnseq = NOISELEVEL; //scaled positive noise level

fb =(sreg.bt.b0)^(sreg.bt.b1); //XOR bits 0,1

fb^=(sreg.bt.b11)^(sreg.bt.b13); //with bits 11,13 -> fb

sreg.regval<<=1; //shift register 1 bit to left

sreg.bt.b0=fb; //close feedback path

return prnseq;

}

void resetreg(void)

{

sreg.regval = 0xFFFF; //shift register to nominal values

fb = 1; //initial feedback value

return;

}

interrupt void c\_int11() //interrupt service routine

{

int section; // index for section number

float input; // input to each section

float wn,yn; // intermediate and output values in each stage

int section1; // index for section number

float input1; // input to each section

float wn1,yn1; // intermediate and output values in each stage

int section2; // index for section number

float input2; // input to each section

float wn2,yn2; // intermediate and output values in each stage

int section3; // index for section number

float input3; // input to each section

float wn3,yn3; // intermediate and output values in each stage

input =((float)prand()); // input to first section read from codec

for (section=0 ; section< NUM\_SECTIONS ; section++)

{

wn = input - a[section][0]\*w[section][0] - a[section][1]\*w[section][1];

yn = b[section][0]\*wn + b[section][1]\*w[section][0] + b[section][2]\*w[section][1];

w[section][1] = w[section][0];

w[section][0] = wn;

input = yn; // output of current section will be input to next

}

//output\_left\_sample((short)yn);

input1 =(yn); // input to first section read from codec

for (section1=0 ; section1< NUM\_SECTIONS ; section1++)

{

wn1 = input1 - a1[section1][0]\*w1[section1][0] - a1[section1][1]\*w1[section1][1];

yn1 = b1[section1][0]\*wn1 + b1[section1][1]\*w1[section1][0] + b1[section1][2]\*w1[section1][1];

w1[section1][1] = w1[section1][0];

w1[section1][0] = wn1;

input1 = yn1; // output of current section will be input to next

}

input2 =(yn1); // input to first section read from codec

for (section2=0 ; section2< NUM\_SECTIONS ; section2++)

{

wn2 = input2 - a2[section2][0]\*w2[section2][0] - a2[section2][1]\*w2[section2][1];

yn2 = b2[section2][0]\*wn2 + b2[section2][1]\*w2[section2][0] + b2[section2][2]\*w2[section2][1];

w2[section2][1] = w2[section2][0];

w2[section2][0] = wn2;

input2 = yn2; // output of current section will be input to next

}

input3 =(yn2); // input to first section read from codec

for (section3=0 ; section3< NUM\_SECTIONS ; section3++)

{

wn3 = input3 - a3[section3][0]\*w3[section3][0] - a3[section3][1]\*w3[section3][1];

yn3 = b3[section3][0]\*wn3 + b3[section3][1]\*w3[section3][0] + b3[section3][2]\*w3[section3][1];

w3[section3][1] = w3[section3][0];

w3[section3][0] = wn3;

input3 = yn3; // output of current section will be input to next

}

output\_left\_sample((short)yn3); // before writing to codec

return; //return from ISR

}

void main()

{

resetreg();

comm\_intr(); //init DSK, codec, McBSP

while(1); //infinite loop

}

