به نام خدا

گزارش آزمایش سوم

طراحی سیستم نهفته مبتنی بر پردازنده NIOS II و اضافه کردن دستورات اختصاصی به این پردازنده

اعضای گروه :

سید عرفان حسینی –علی پرچکانی

1-طراحی سطح بالای فیلتر FIR و سنتز آن به کد RTL

با استفاده از ToolBox ، fdatool در Matlab و با انجام تنظیمات زیر فیلتر موردنظر را طراحی میکنیم .

Design Method = FIR (Equiripple)

Filter Order = Minimum order

Density Factor = 20

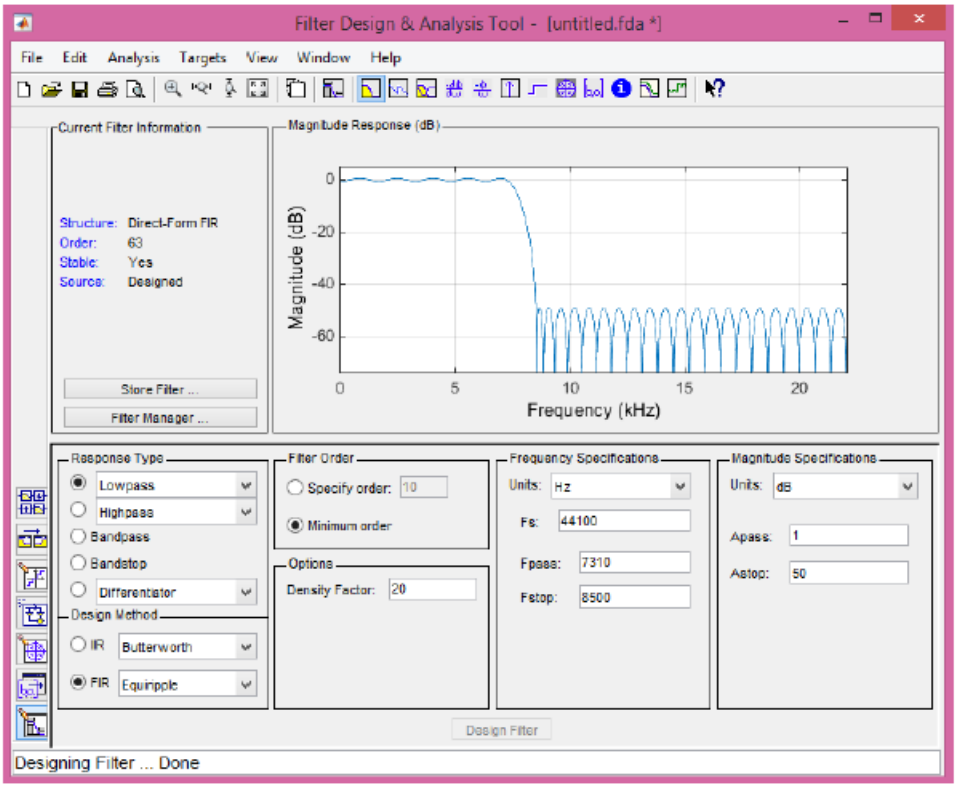
Fs = 44100 Hz

Fpass = 7310 Hz

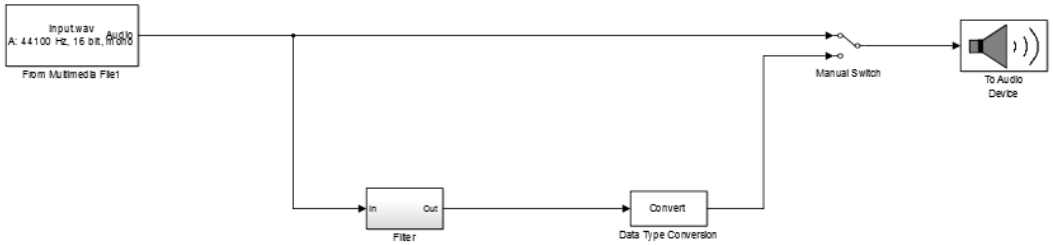
Fstop = 8500 Hz

Apass = 1 dB

Astop = 50 dB



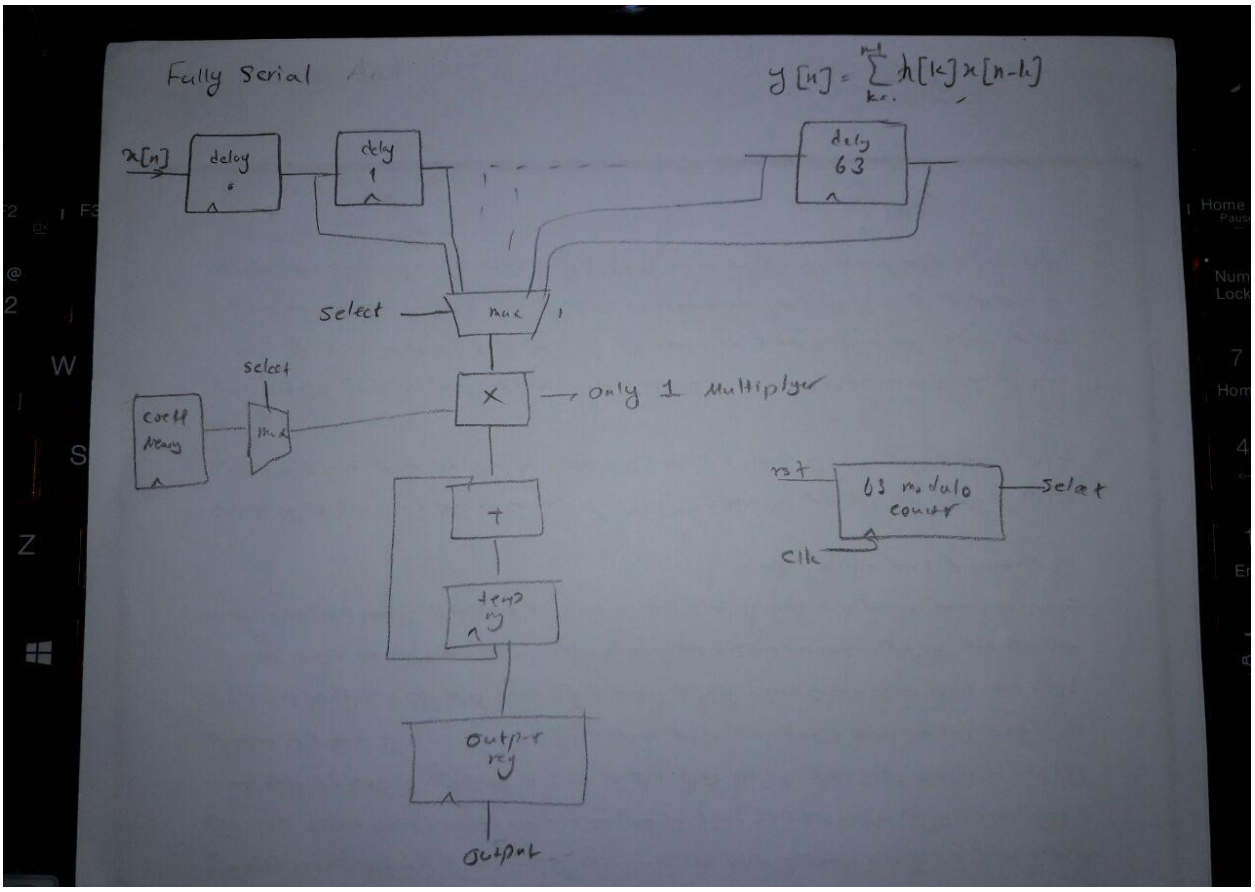
سپس فیلتر را در حیط Simulink تست میکنیم بدین منظور از بلوک دیاگرام شکل زیر استفاده میکنیم :



با استفاده از سیستم فوق میتوانیم به صدای فیلتر شده و فیلتر نشده گوش کنیم و مشاهده میشود که صدای فیلتر شده بسیار کیفیت بالاتری نسبت به صدای اولیه دارد و نویز فرکانس بالا که در این صدا موجود است توسط فیلتر حذف میشود.

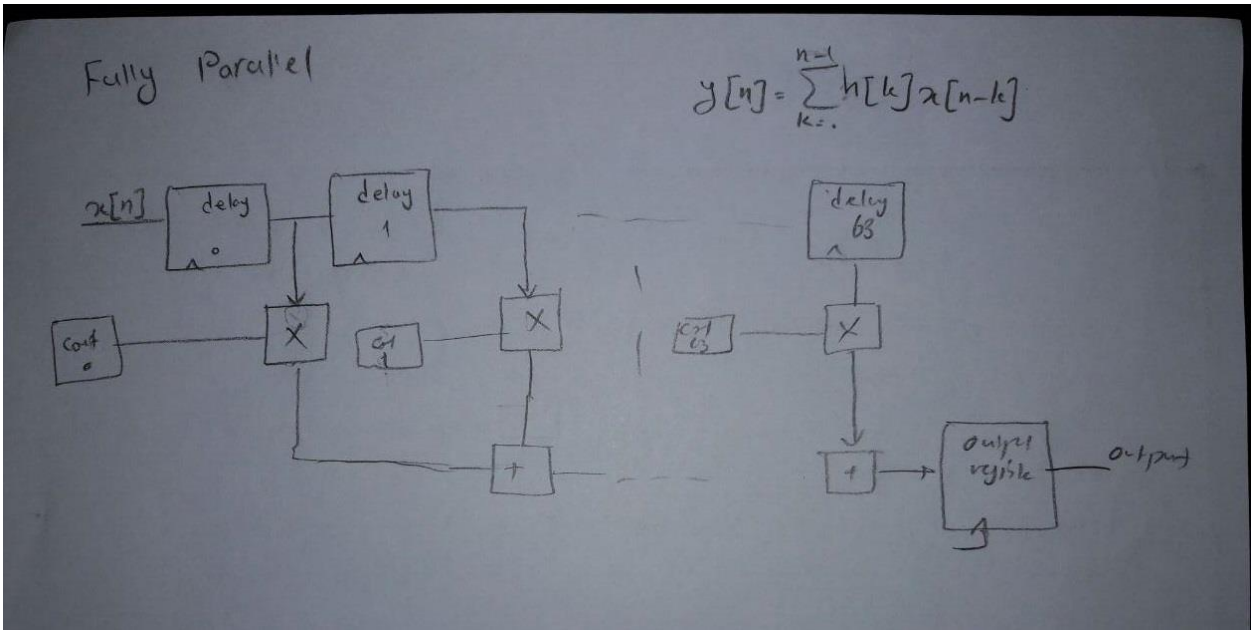
در این مرحله با استفاده از Matlab که یک ابزار سنتز سطح بالا محسوب میشود (HLS Tool) کد Verilog فیلتر را generate میکنیم . فیلتر را در دو حالت کاملا سریال (استفاده از تنها یک ضرب کننده) و حالت کاملا موازی (استفاده از 64 ضرب کننده) به کد RTL سنتز میشود که معماری سخت افزاری این دو نوع فیلتر در زیر امده است :

1- حالت کاملا سریال :



در این حالت تنها از یک ضرب کننده استفاده شده است و داده ها باید در 64 کلاک پشت سر هم ضرب شده و با هم جمع شوند تا خروجی به یک ورودی ساخته شود.

2- حالت کاملا موازی:



در این حالت از 64 ضرب کننده استفاده شده است و در یک سیکل خروجی به یک ورودی ساخته میشود .

برای مقایسه میزان سخت افزار استفاده شده در هر یک از حالت ها هر کدام ار کد ها را جداگانه سنتز میکنیم . نتایج سنتز به صورت زیر است :

برای حالت کاملا سریال :

Flow Summary	
Flow Status	Successful - Sun Jan 22 01:39:37 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	serial
Top-level Entity Name	serial
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	1,417 / 33,216 (4 %)
Total combinational functions	814 / 33,216 (2 %)
Dedicated logic registers	1,132 / 33,216 (3 %)
Total registers	1132
Total pins	53 / 475 (11 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	2 / 70 (3 %)
Total PLLs	0 / 4 (0 %)

حالت کاملا موازی :

Flow Summary	
Flow Status	Successful - Sun Jan 22 01:00:58 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	parallel
Top-level Entity Name	parallel
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	7,436 / 33,216 (22 %)
Total combinational functions	7,145 / 33,216 (22 %)
Dedicated logic registers	1,058 / 33,216 (3 %)
Total registers	1058
Total pins	53 / 475 (11 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

همانطور که مشاهده میشود در حالت کاملا موازی سخت افزار بسیار بیشتری نسبت به حالت کاملا سریال استفاده میشود.(به دلیل استفاده از تعداد زیاد ضرب کننده) .

ر طراحی کاملا موازی یک خروجی در یک سیکل کلاک اماده میشود در صورتی که در طراحی کاملا سریال یک خروجی در زمان 64 سیکل کلاک اماده میشود بنابراین بین سرعا انجام کار و میزان استفاده از منابع سخت افزاری یک trade-off وجود دارد و بر حسب کاربرد میتوان از هر یک از این دو ویا ترکیبی از آن ها استفاده کرد .

برای تست کردن فیلتر به عنواان ورودی یک صدای دارای نویز (فایل input.wav) را به فیلتر میدهیم برای شبیه سازی در محیط modelsim کد و testbench را شبیه سازی میکنیم .

در testbench ورودی های مختلف که از فایل (input) خوانده میشودو با خروجی مورد انتظار مقایسه میشود.

. بخشی از شکل موج خروجی فیلتر به صورت زیر است :

ابتدای شکل موج :

انتهای شکل موج:

2-اضافه کردن دستور اختصاصی به پردازنده Nios II :

در این بخش یک بار فیلتر را به صورت کاملا نرم افزاری پیاده سازی کردیم و یک بار به صورت سخت افزاری محاسبات مربوط به فیلتر را انجام دادیم (custom instruction) .

در فیلتر نرم افزاری بر روی تک تک نمونه های صدا عملیات کانولوشن انجام شد و خروجی به دست آمد. کد بخش فیلتر نرم افزاری و انجام کانولوشن در زیر آمده است:

```
if(flag==1){
printf("1");
flag=0;
for(i=0;i<BUF_SIZE;i++){
r_temp=0;
l_temp=0;

for(j=0;j<64;j++){
k=i-j;
if(k>=0){
r_temp+=((float)r_buf[i-j])*B[j];
l_temp+=((float)l_buf[i-j])*B[j];
{
{
rbufout[i]=r_temp;
lbufout[i]=l_temp;

{
for(j=0;j<BUF_SIZE;j++){
le_buf[j]=(int)lbufout[j];
re_buf[j]=(int)rbufout[j];
{
```

```
printf("done");
```

{

در این کد یک flag پس از ضبط شدن صدا فعال میشود ، با فعال شدن ان عملیات فیلترینگ اغاز میشود .

فیلتر نرم افزاری مدت زمان زیادی (چیزی در حدود 15-10 دقیقه) به طول می انجامد و خروجی مناسبی دارد.

درفیلتر سخت افزاری کد Verilog تولید شده توسط matlab را در Qsys برای ساختن custom instruction اضافه میکنیم و یک Multi-cycle custom instruction ایجاد میکنیم .برای این که سیگنال های موجود در کد با فرمت استاندارد تعریف شده در Qsysبرای multi-cycle custom instruction همخوانی داشته باشد سیگنال done را به کد Verilog تولیذ شده توسط Matlab اضافه میکنیم.

پس از طراحی سیستم در Qsys برای طراحی نرم افزار در کد نرم افزاری در یک حلقه با استفاده از custom instruction خروجی فیلتر به هر ورودی را محاسبه میکنیم. کد نرم افزاری این بخش به صورت زیر است :

```
if (flag==1){
move the ALTERA text around on the VGA screen */ */
printf("sharm");
for (i=0;i<BUF_SIZE;i++)
rbufe[i]=ALT_CI_FIR_0(r_buf[i]);

for (i=0;i<BUF_SIZE;i++)
lbufe[i]=ALT_CI_FIR_0(l_buf[i]);
for(i=0;i<BUF_SIZE;i++){
re_buf[i]=rbufe[i];
le_buf[i]=lbufe[i];
{
printf("done");
flag=0;}
```

{

فیلتر سخت افزاری با سرعت بسیار بیشتری انجام شده و عملیات انجام فیلتر در حد چند ثانیه طول میکشد .

نتیجه گیری:

برای انجام محاسبات ریاضی سنگین که به تعداد زیاد انجام میشوند به جای انجام تمامی عملیات به صورت نرم افزاری بهتر است با اضافه کردن یک custom instruction (این محاسبات چون در نزدیکی ALUانجام میشود سرعت زیادی دارد) به پردازنده سرعت محاسبات را به شدت افزایش میدهد.