

به نام خدا



دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

طراحی سیستم های نهفته مبتنی بر FPGA

آزمایش سوم

نام و نام خانوادگی	مهسا راستی - سعید شکوفا - محمد مهدی معینی منش
شماره دانشجویی	810198393 - 810198418 - 810198475
تاریخ ارسال گزارش	1401.09.29

فهرست

1. طراحی سطح بالای فیلتر FIR و سنتز آن به کد RTL 5
- مراحل ساخت فیلتر FIR در متلب 5
- تفاوت حالت Saturate با حالت Wrap در زمان تنظیم مد Overflow 6
- مزایا و معایب کوانتیزه کردن فیلتر 7
- انتقال فیلتر به شبیه ساز متلب 7
- فیلتر سریال 8
- فیلتر موازی 9
- عملکرد و مقایسه حالت موازی با حالت سری 11
2. اضافه کردن دستور اختصاصی به پردازنده ی Nios II 12
- گام 1: اضافه کردن دستور اختصاصی به صورت نرم افزاری 12
- گام 2: طراحی سخت افزاری فیلتر (ایجاد custom instruction جدید) 17
- گام 3: استفاده از custom instruction تولید شده 20
- سوال: نیازمندی های تعریف یک دستور Multicycle 22

شکل‌ها

- شکل 1-1. طراحی فیلتر درجه 63..... 5
- شکل 2-1. اندازه ضرایب..... 5
- شکل 3-1. حالت Saturate..... 6
- شکل 4-1. حالت Wrap..... 6
- شکل 5-1. ساخت simulink متلب..... 7
- شکل 6-1. فیلتر درست شده در simulink..... 7
- شکل 7-1. ساختار سریال..... 8
- شکل 8-1. فرکانس کاری حالت سریال..... 8
- شکل 9-1. نمای زوم شده حالت سریال..... 8
- شکل 10-1. نتایج سنتز حالت سریال..... 9
- شکل 11-1. ساختار موازی..... 9
- شکل 12-1. فرکانس کاری حالت موازی..... 10
- شکل 13-1. نمای زوم شده حالت موازی..... 10
- شکل 14-1. نتایج سنتز حالت موازی..... 10
- شکل 1-2. تابع make_denoise..... 12
- شکل 2-2. فیلتر طراحی شده..... 12
- شکل 3-1. ضرایب coef..... 13
- شکل 4-1. محاسبه زمان اجرای فیلتر..... 14
- شکل 5-1. تنظیمات timestamp..... 15
- شکل 6-1. تنظیمات timestamp..... 15
- شکل 7-1. تنظیمات timestamp..... 15
- شکل 8-1. زمان محاسبه شده حالت نرم افزاری..... 16
- شکل 9-1. ادیت کد وریلاگ فیلتر..... 17
- شکل 10-1. Component type..... 18
- شکل 11-1. Files..... 18
- شکل 12-1. Signals..... 19
- شکل 13-1. interfaces..... 19

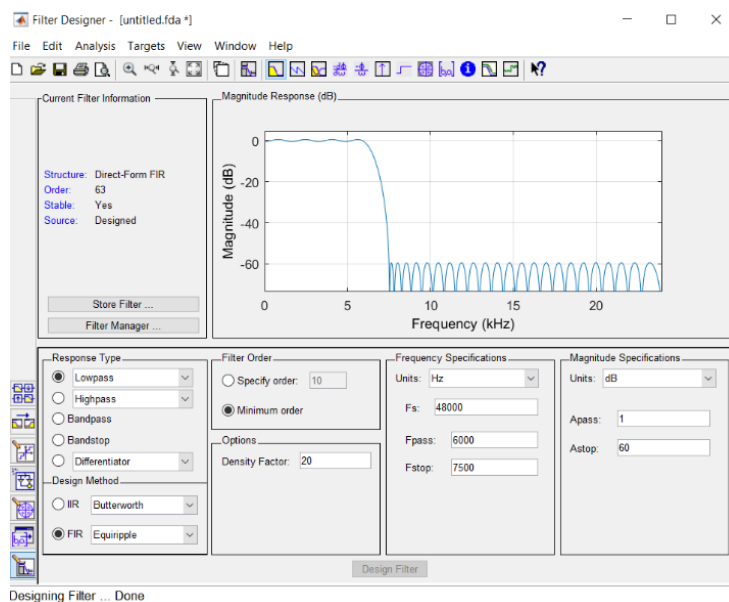
- شکل 1-14. system contents 20
- شکل 1-15. نتایج سنتز قبل از اضافه کردن custom instruction 20
- شکل 1-16. نتایج سنتز بعد از اضافه کردن custom instruction 20
- شکل 1-17. New Macro 21
- شکل 1-18. دینویز کردن با ماکروی تولید شده 21
- شکل 1-19. زمان اجرای فیلتر 22

جدول

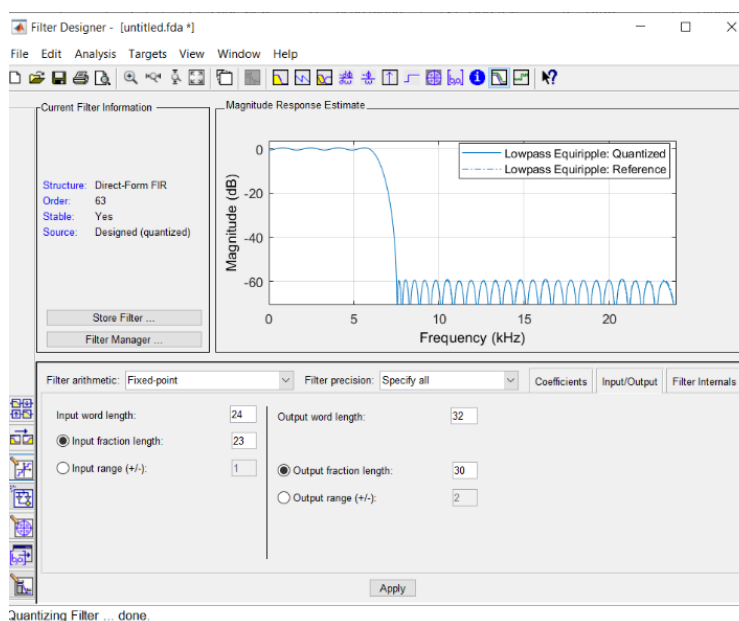
جدول 1- مقایسه میزان منابع.....21

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

مراحل ساخت فیلتر FIR در متلب



شکل 1-1. طراحی فیلتر درجه 63

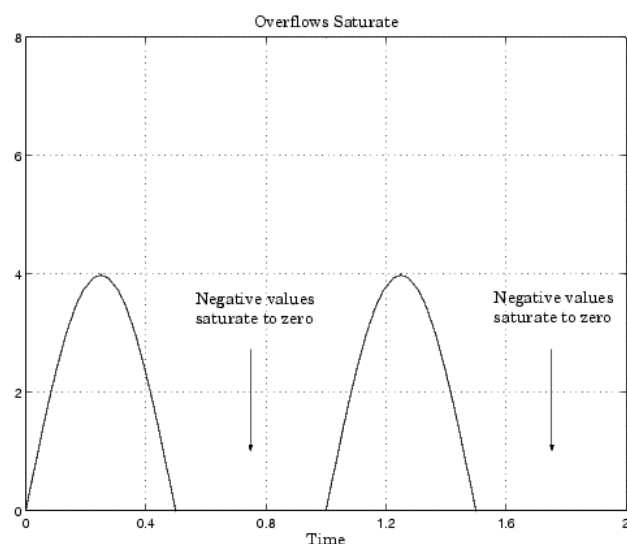


شکل 1-2. اندازه ضرایب

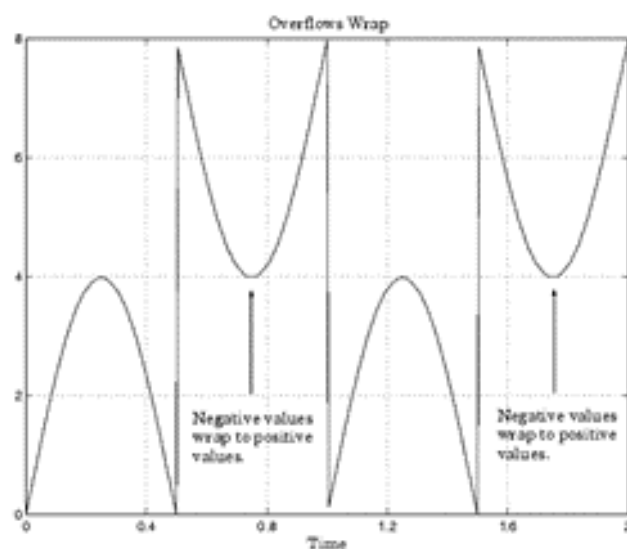
اندازه ضرایب طبق صورت گزارش (24 بیت به همراه 23 بیت اعشار برای ورودی و 32 بیت به همراه 30 بیت اعشار برای خروجی) تنظیم شده است.

تفاوت حالت Saturate با حالت Wrap در زمان تنظیم مد Overflow

پردازنده‌ها سیاست‌های متفاوتی برای برخورد با overflow دارند. حالت saturate اعداد منفی را به صفر (در واقع کوچک‌ترین عددی که می‌تواند نشان دهد) نظیر میکند و حالت wrap اعداد منفی را به اعداد مثبت به نحوی نظیر میکند که با اعداد مثبت اشتباه گرفته نشوند.



شکل 3-1. حالت Saturate

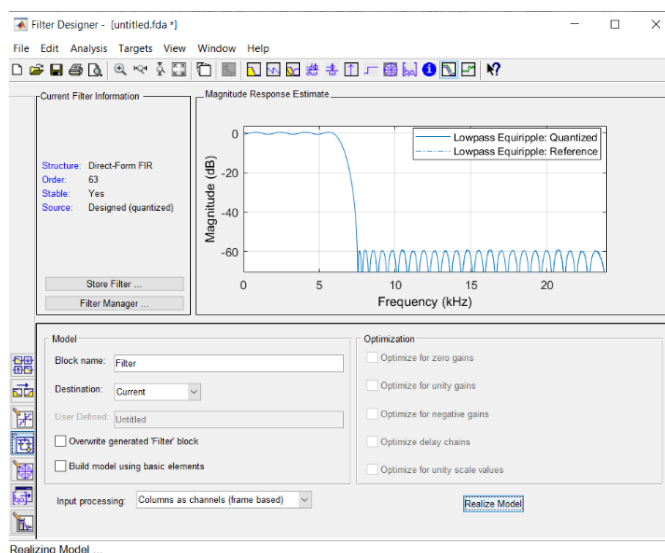


شکل 4-1. حالت Wrap

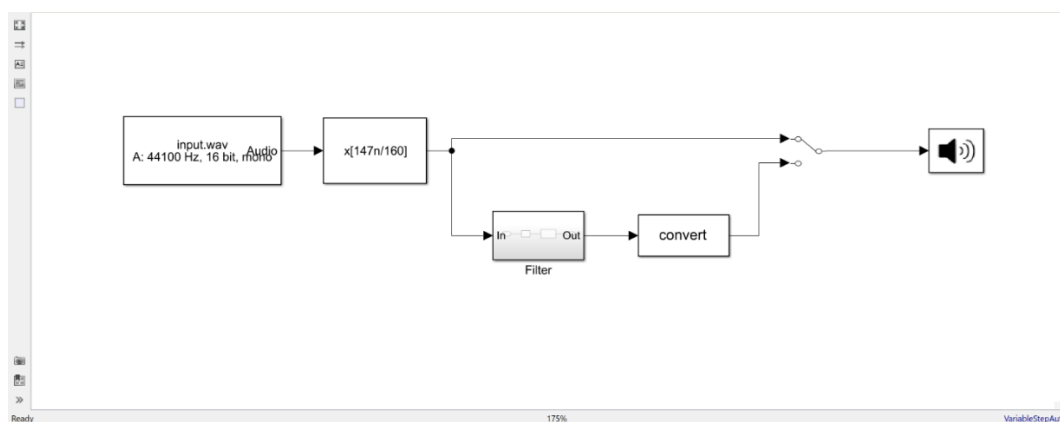
مزایا و معایب کوانتیزه کردن فیلتر

کوانتیزه کردن ضرایب این عیب را دارد که هنگامی که کوانتیزه می شوند با یک خطایی کوانتیزه می شوند. در واقع چون عدد ما 23 بیت اعشار دارد دقت ما به اندازه همان 23 بیت اعشار می شود و شامل خطا می شود. همچنین کوانتیزه کردن سبب می شود که سیگنال آنالوگ به دیجیتال تبدیل شود که خود خطایی را به همراه دارد. مزیت آن این است که اگر چه خطایی را به محاسبات اضافه میکند ولی این خطا حداقل مقدار ممکن بوده و سبب می شود که دقت ما در بالاترین حد ممکن باشد. در واقع چون ضرایب ثابت هستند تضعیف سیگنال به کمترین مقدار می رسد.

انتقال فیلتر به شبیه ساز متلب



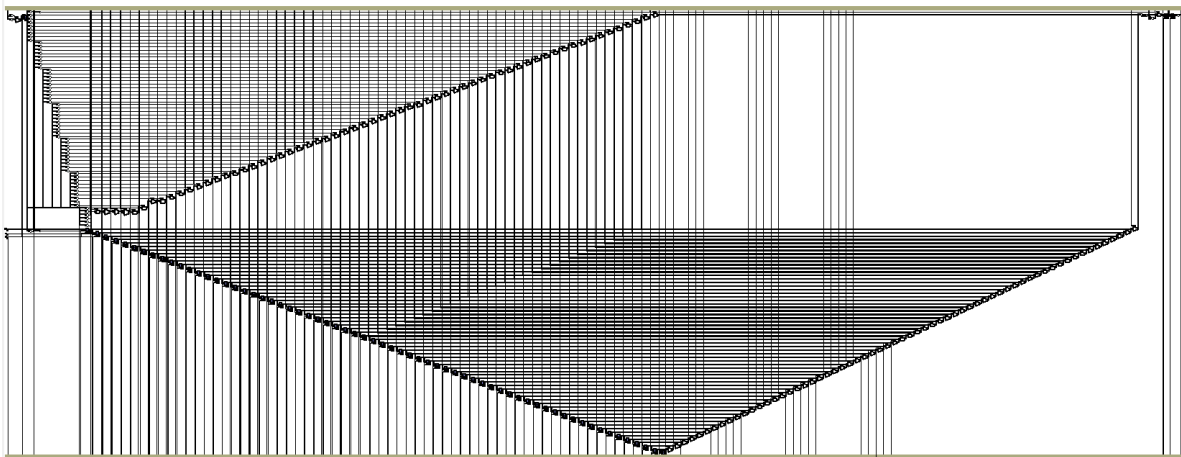
شکل 1-5. ساخت simulink متلب



شکل 1-6. فیلتر درست شده در simulink

فیلتر در متلب تست شد و مشاهده شد که به خوبی می تواند نویز را بگیرد و صدای شفاف را پخش کند.

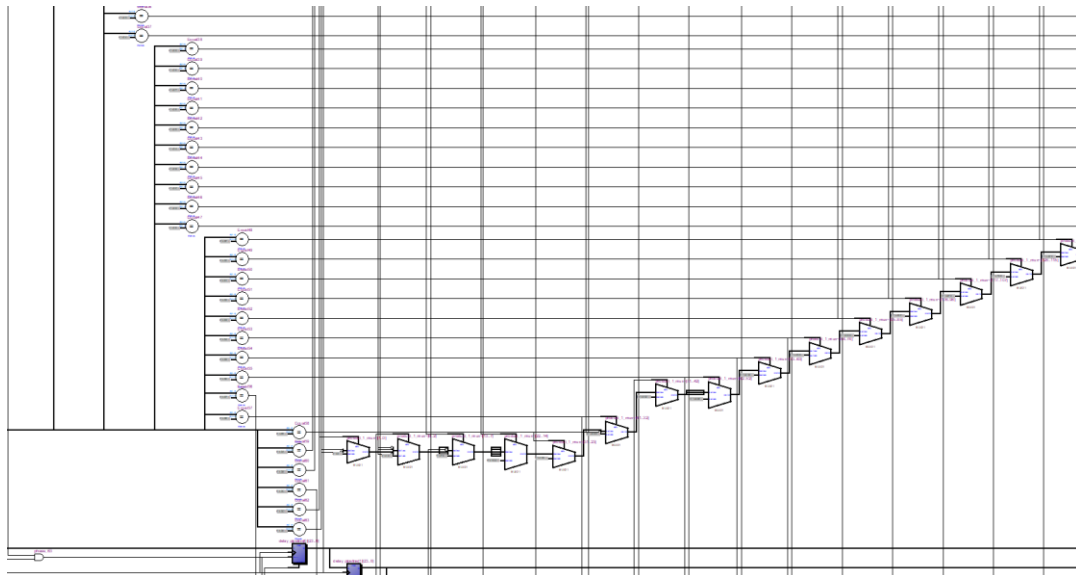
فیلتر سریال



شکل 1-7. ساختار سریال

Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	64.72 MHz	64.72 MHz	clk	

شکل 1-8. فرکانس کاری حالت سریال



شکل 1-9. نمای زوم شده حالت سریال

Table of Contents	
Flow Summary	Flow Status
Flow Settings	Quartus II 64-Bit Version
Flow Non-Default Global Settings	Revision Name
Flow Elapsed Time	Top-level Entity Name
Flow OS Summary	Family
Flow Log	Device
Analysis & Synthesis	Timing Models
Fitter	Total logic elements
Assembler	Total combinational functions
TimeQuest Timing Analyzer	Dedicated logic registers
EDA Netlist Writer	Total registers
Flow Messages	Total pins
Flow Suppressed Messages	Total virtual pins
	Total memory bits
	Embedded Multiplier 9-bit elements
	Total PLLs

شکل 10-1. نتایج سنتز حالت سریال

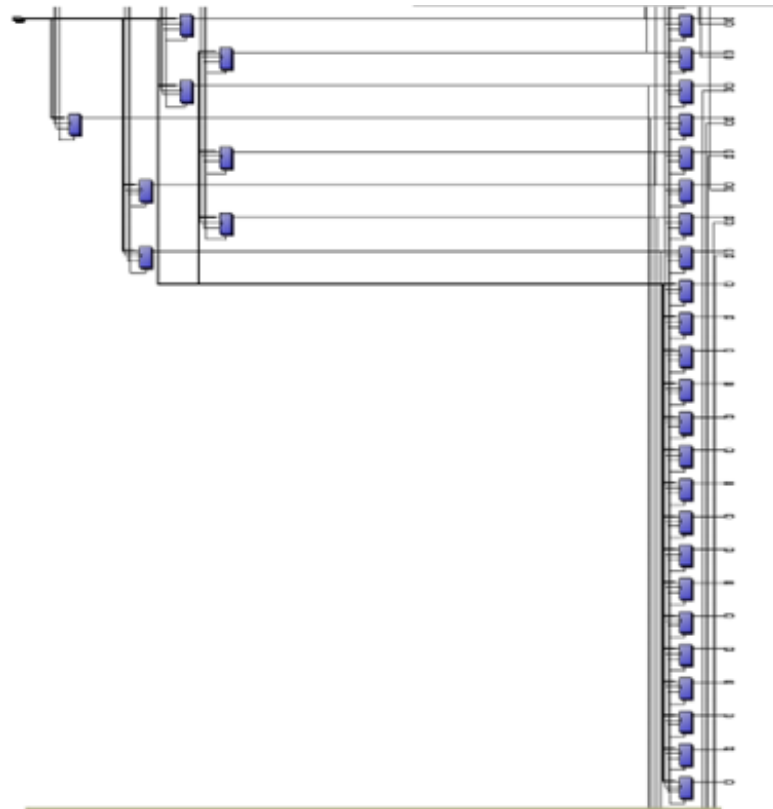
فیلتر موازی



شکل 11-1. ساختار موازی

Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	25.34 MHz	25.34 MHz	clk	

شکل 1-12. فرکانس کاری حالت موازی



شکل 1-13. نمای زوم شده حالت موازی

Flow Summary		
Flow Status	Successful - Tue Nov 29 13:06:38 2022	
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition	
Revision Name	FIR_filter	
Top-level Entity Name	FIR_filter	
Family	Cyclone II	
Device	EP2C35F672C6	
Timing Models	Final	
Total logic elements	17,032 / 33,216 (51 %)	
Total combinational functions	16,429 / 33,216 (49 %)	
Dedicated logic registers	1,568 / 33,216 (5 %)	
Total registers	1568	
Total pins	59 / 475 (12 %)	
Total virtual pins	0	
Total memory bits	0 / 483,840 (0 %)	
Embedded Multiplier 9-bit elements	70 / 70 (100 %)	
Total PLLs	0 / 4 (0 %)	

شکل 1-14. نتایج سنتز حالت موازی

عملکرد و مقایسه حالت موازی با حالت سری

در حالت سری یک ضرب کننده و یک جمع کننده داریم و 64 تا رجیستر حالت تعریف شده است. در واقع با آمدن هر ورودی باید 64 تا کلاک صبر کرد تا خروجی آماده شود و این 64 تا کلاک با همان 64 تا رجیستر حالت که تعریف شده اند مشخص می شود. همچنین 64 تا رجیستر برای ذخیره 64 تا مقدار اخیر ورودی هستند در نظر گرفته شده است که در ضرایب ضرب و تقسیم شوند و خروجی را تولید کنند. در واقع در هر کدام از این 64 تا حالت با یکی از ورودی ها ضرب شده و با مقدار قبلی (مقدار اولیه صفر است) جمع می شوند.

در حالت موازی دیگر خبری از رجیستر حالت نیست. چرا که تمامی ورودی ها به طور همزمان با ضرایب ضرب می شوند و سپس با هم جمع می شوند یعنی 64 تا ضرب کننده داریم (به همراه 64 جمع کننده) و خروجی بعد از یک کلاک تولید می شود و لازم نیست که 64 تا کلاک صبر کرد. البته در این حالت critical path بیشتر بوده و سبب می شود که فرکانس کاری کمتر از حالت قبل شود همچنین 64 تا رجیستر برای نگه داشتن 64 ورودی اخیر فیلتر در نظر گرفته شده است.

همچنین با توجه به نتایج سنتز مشاهده می شود که میزان استفاده از منابع FPGA در حالت سریال بسیار کمتر از حالت موازی بوده. چرا که فقط یک ضرب کننده و یک جمع کننده داریم؛ ولی باید 64 تا کلاک برای تولید خروجی صبر کرد. در حالت موازی میزان استفاده از منابع زیاد بوده ولی خروجی در یک کلاک تولید می شود. هر چند که فرکانس کاری آن پایین تر است.

2. اضافه کردن دستور اختصاصی به پردازنده ی Nios II

گام 1: اضافه کردن دستور اختصاصی به صورت نرم افزاری

```
void make_Denoise(){
    int i;
    for (i = 0; i < BUF_SIZE; i++){
        l_buf_Denoise[i] = filter((double)l_buf[i]);
    }
    for (i = 0; i < 64; i++){
        mem[i] = 0;
    }
    for (i = 0; i < BUF_SIZE; i++){
        r_buf_Denoise[i] = filter((double)r_buf[i]);
    }
    for (i = 0; i < 64; i++){
        mem[i] = 0;
    }
}
```

شکل 2-1. تابع make_denoise

همانطور که در تصویر بالا مشخص است، دو حلقه for جداگانه برای بافر چپ و راست برای پخش استریو صدا استفاده شده است.

تابع make_denoise را جایگزین تابع make_echo کرده و نام متغیرهای مربوط به echo را به denoise تغییر میدهیم.

```
extern float mem[64];

unsigned int filter(unsigned int in){
    in = in >> 8;
    float float_in = (float) in / 8388608;

    int i;
    for (i=63;i>0;i--){
        mem[i]=mem[i-1];
    }
    mem[0]=in;
    double res=0;
    for(i=0;i<64;i++){
        res+=(double)mem[i]*(double)coef[i];
    }
    unsigned int result=(unsigned int)(res*1073741824);
    return result;
}
```

شکل 2-2. فیلتر طراحی شده

```
#include <stdio.h>

const float coef[64]={
-0.0019989013671875,
-0.0050506591796875,
-0.008331298828125 ,
-0.0105438232421875,
-0.0092926025390625,
-0.0046539306640625,
0.0021209716796875,
0.0072174072265625,
0.0078125 ,
0.0027618408203125,
-0.004852294921875 ,
-0.0102081298828125,
-0.008819580078125 ,
-0.0006866455078125,
0.0095977783203125,
0.0146331787109375,
0.009735107421875 ,
-0.0036163330078125,
-0.0170440673828125,
-0.0205535888671875,
-0.009063720703125 ,
0.01220703125 ,
0.0296478271484375,
0.028717041015625 ,
0.00469970703125 ,
-0.031494140625 ,
-0.056121826171875 ,
-0.0446319580078125,
0.013763427734375 ,
0.106964111328125 ,
0.2028656005859375,
0.2635040283203125,
0.2635040283203125,
0.2028656005859375,
0.106964111328125 ,
```

```
0.013763427734375 ,
-0.0446319580078125,
-0.056121826171875 ,
-0.031494140625 ,
0.00469970703125 ,
0.028717041015625 ,
0.0296478271484375,
0.01220703125 ,
-0.009063720703125 ,
-0.0205535888671875,
-0.0170440673828125,
-0.0036163330078125,
0.009735107421875 ,
0.0146331787109375,
0.0095977783203125,
-0.0006866455078125,
-0.008819580078125 ,
-0.0102081298828125,
-0.004852294921875 ,
0.0027618408203125,
0.0078125 ,
0.0072174072265625,
0.0021209716796875,
-0.0046539306640625,
-0.0092926025390625,
-0.0105438232421875,
-0.008331298828125 ,
-0.0050506591796875,
-0.0019989013671875
};
```

شكل 1-3. ضرایب coef

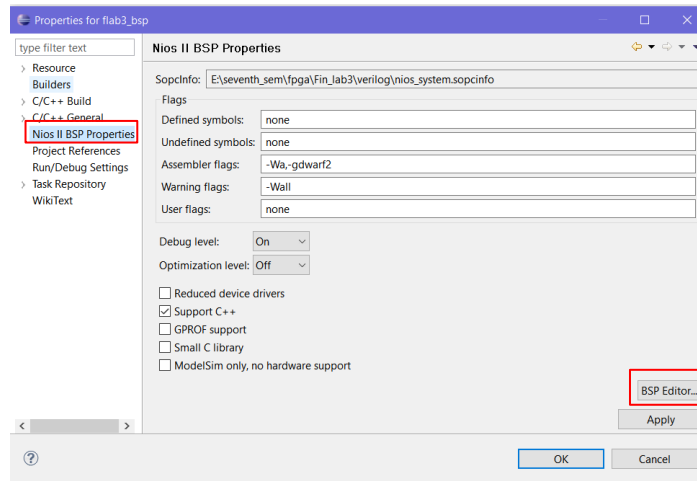
پس از اعمال فیلتر صدای پخش شده نویز صدای اولیه را نداشت ولی یک مقدار نویز دیگر به آن اضافه شده بود (حالت برفک به خود گرفته بود) که می توان به دلیل کم بودن دقت محاسبات در بخش نرم افزاری باشد.

همچنین زمان انجام محاسبات توسط توابع `alt_timestamp` و `alt_timestamp_start` محاسبه شد که این ها شمارنده کلاک می باشند و برای به دست آوردن زمان باید بر فرکانس تقسیم کرد که فرکانس را با استفاده از تابع `alt_timestamp_freq` به دست آوردیم. به این گونه که قبل از شروع کار فیلتر `alt_timestamp_start` را فراخوانی کردیم. قبل از فراخوانی فیلتر `alt_timestamp` را در یک ذخیره کردیم و پس از فراخوانی فیلتر، مقدار `alt_timestamp` را در متغیری دیگر ذخیره کردیم. در نهایت اختلاف دو مقدار بدست آمده را بر این فرکانس تقسیم کرده تا زمان به دست آید و سپس آن را چاپ کردیم.

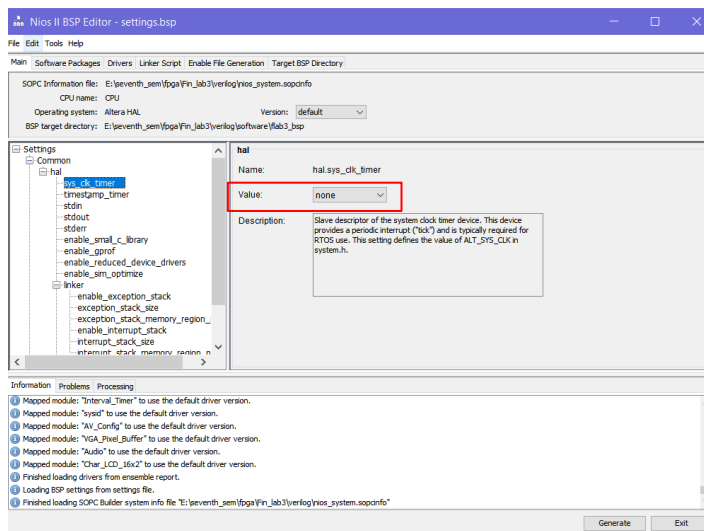
```
void run_command(struct alt_up_dev *up_dev) {
    int num_read;
    int num_written;
    if(command == 0){ //Record
        alt_u32 fr;
        fr = alt_timestamp_freq();
        // reset the buffer index for recording
        buf_index_record = 0;
        // clear audio FIFOs
        alt_up_audio_reset_audio_core (up_dev->audio_dev);
        // enable audio-in interrupts
        alt_up_audio_enable_read_interrupt (up_dev->audio_dev);
        alt_timestamp_start();
        float t_start = (float) alt_timestamp() / (float) fr;
        make_Denoise();
        float t_stop = (float) alt_timestamp() / (float) fr;
        printf("t = %f\n", t_stop - t_start);
    }
}
```

شکل 4-1. محاسبه زمان اجرای فیلتر

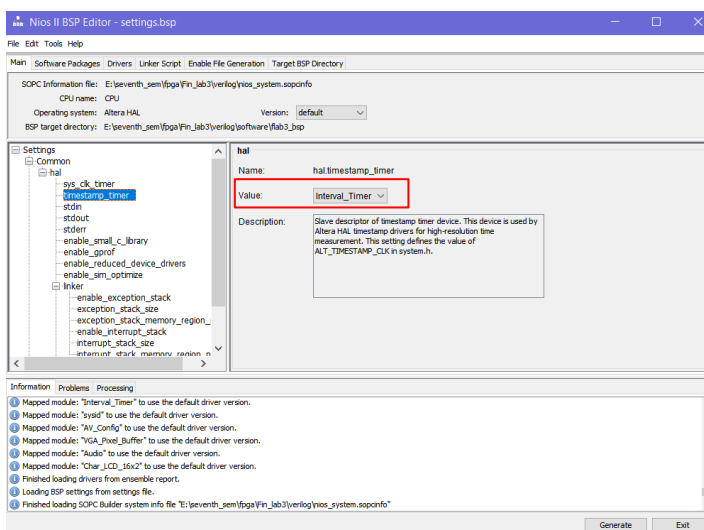
تنظیمات پروژه bsp را در eclipse مطابق مراحل دستورکار تنظیم کردیم. Timer بر روی none و timestamptimer بر روی interval_timer قرار گرفت تا بتوان از توبع زمانی که ذکر شد استفاده کرد.



شكل 5-1. تنظيمات timestamp



شكل 6-1. تنظيمات timestamp



شكل 7-1. تنظيمات timestamp

برای تست فیلتر، به دلیل تاخیر بسیار زیاد، نمونه های ضبط شده به 0.1 نمونه اولیه کاهش یافت در واقع یک ثانیه صدا ضبط شد و روی آن فیلتر اعمال شد. این کار 9 دقیقه و 50 ثانیه زمان برد که اگر 10 برابر میشد زمان ضبط شده این زمان به یک ساعت و نیم میرسید! همچنین با توجه به اینکه در این زمان شمارنده کلاک اورفلو میکند مقدار -0.000005 چاپ شد (حداکثر عدد قابل نمایش برابر $2^{32} =$ 4,294,967,296 می باشد که این مقدار در صورتی که با فرکانس 50 مگاهرتز کار کنیم برابر با 86 ثانیه می شود.) که همان گونه که گفته شد این مقدار اورفلو شده می باشد و اشتباه است.

```

# Bus 0 - FPGA_LAB3/media_internet_M4M.c - Eclipse
File Edit Source Refactor Window Help Run Project Nios II Window Help

media_internet_M4M.c 22 filter.c @ audio_ISR.c

Project Explorer 32
FPGA_LAB3
FPGA_LAB3_bsp [nios_system]

#include "globals.h"
#include "filter.h"
#include "sys/alt_timestamp.h"
#include "sys/alt_alarm.h"

/* these globals are written by interrupt service routines: we have to declare
 * them as volatile to avoid the compiler caching their values in registers */
extern volatile unsigned char valid_bytec, valid_bytec2, valid_bytec3; /* modified by PS/2 interrupt service routine */
extern volatile unsigned char cnt, flag_mouse, flag_denoise, Command;
extern volatile int i; /* need to synchronize with the timer */
extern struct alt_up_dev up_dev; /* pointer to struct that holds pointers to
                                open devices */

extern volatile int buf_index_record;
extern volatile int buf_index_play;
extern float mem[64];

extern short cursor_shape[16][8];

/* Constants */
#define X_BOUNCD 239
#define Y_BOUNCD 239

/* Global Variables */
signed short x_axis_mouse, y_axis_mouse; // Current axes of mouse
// unsigned char Command // Which Button clicked by mouse
unsigned int i_buf[BUF_SIZE]; // audio buffer
unsigned int i_buf[BUF_SIZE]; // audio buffer
extern unsigned int i_buf_Denoise[BUF_SIZE]; // audio Denoise buffer
extern unsigned int i_buf_Denoise[BUF_SIZE]; // audio Denoise buffer

short cursor_shape[16][8] = {
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
    { 0, -1, -1, -1, -1, -1, -1, -1},
};

Problems | Tasks | Console | Properties | Nios II Console |
FPGA_LAB3 Nios Hardware configuration - cable: USB-Baster on localhost [USB-2] device ID: 5 name: jtaguart_0
Opened pushbutton KEY device
Opened green LEDs device
Opened red LEDs device
Opened PS2 device
Opened audio device
Opened character LCD device
Opened pixel buffer device
Opened character buffer device
t = 0.000000

```

شکل 1-8. زمان محاسبه شده حالت نرم افزاری

گام 2: طراحی سخت افزاری فیلتر (ایجاد custom instruction جدید)

ابتدا تغییرات لازم را در کد وریلاگ تولید شده توسط متلب اعمال میکنیم:

```
module FIR_filter
(
    clk,
    clk_enable,
    reset,
    input_,
    filter_out
);

input    clk;
input    clk_enable;
input    reset;
input [31:0] input_;
wire signed [23:0] filter_in; //sfix24_En23
output signed [31:0] filter_out; //sfix32_En30

////////////////////////////////////
//Module Architecture: FIR_filter
////////////////////////////////////

// Local Functions
// Type Definitions
// Constants
assign filter_in = input_[31:8];
localparam signed [15:0] coeff1 = 16'b1111111101111101; //sfix16_En16
localparam signed [15:0] coeff2 = 16'b11111111010110101; //sfix16_En16
localparam signed [15:0] coeff3 = 16'b11111110111011110; //sfix16_En16
localparam signed [15:0] coeff4 = 16'b11111110101001101; //sfix16_En16
localparam signed [15:0] coeff5 = 16'b11111110110011111; //sfix16_En16
localparam signed [15:0] coeff6 = 16'b1111111011001111; //sfix16_En16
localparam signed [15:0] coeff7 = 16'b0000000010001011; //sfix16_En16
localparam signed [15:0] coeff8 = 16'b00000000111011001; //sfix16_En16
localparam signed [15:0] coeff9 = 16'b00000000100000000; //sfix16_En16
localparam signed [15:0] coeff10 = 16'b0000000010110101; //sfix16_En16
localparam signed [15:0] coeff11 = 16'b11111111011000010; //sfix16_En16
```

شکل 1-9. ادیت کد وریلاگ فیلتر

باید برای محاسبات فیلتر از 24 بیت پرارزش ورودی استفاده نماییم. همچنین تایپ ضرایب فیلتر را localparam signed قرار میدهیم.

مراحل زیر را برای تولید یک custom instruction برای فیلتر در qsys طی میکنیم:

File Templates

Component Type | Files | Parameters | Signals | Interfaces

► About Component Type

Name:

Display name:

Version:

Group:

Description:

Created by:

Icon:

Documentation:

Title	URL

+ -

شكل 10-1. Component type

File Templates

Component Type | Files | Parameters | Signals | Interfaces

► About Files

Synthesis Files

These files describe this component's implementation, and will be created when a Quartus II synthesis model is generated.

The parameters and signals found in the top-level module will be used for this component's parameters and signals.

Output Path	Source File	Type	Attributes
FIR_filter.v	E:/seventh_sem/fpga/Fin_lab3/FL...	Verilog HDL	Top-level File

+ - Analyze Synthesis Files Create Synthesis File from Signals

Top-level Module: (Analyze files to select module) ▾

Verilog Simulation Files

These files will be produced when a Verilog simulation model is generated.

Output Path	Source File	Type	Attributes
FIR_filter.v	E:/seventh_sem/fpga/Fin_lab3/FL...	Verilog HDL	no attributes

+ - Copy from Synthesis Files

VHDL Simulation Files

These files will be produced when a VHDL simulation model is generated.

Output Path	Source File	Type	Attributes
(No files)			

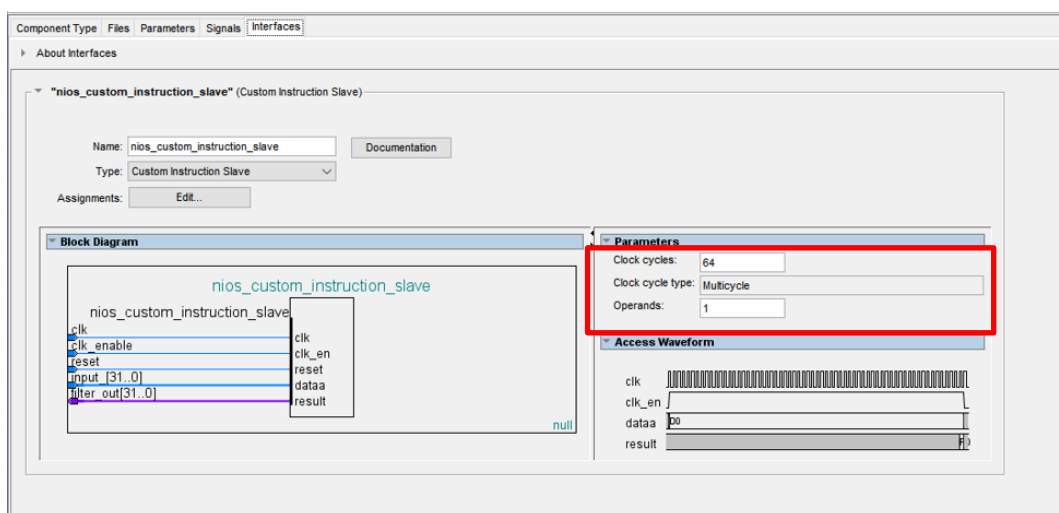
+ - Copy from Synthesis Files

شكل 11-1. Files

Component Type Files Parameters Signals Interfaces				
About Signals				
Name	Interface	Signal Type	Width	Direction
clk	nios_custom_instruction_slave	clk	1	input
clk_enable	nios_custom_instruction_slave	clk_en	1	input
reset	nios_custom_instruction_slave	reset	1	input
input_	nios_custom_instruction_slave	dataaa	32	input
filter_out	nios_custom_instruction_slave	result	32	output

شکل 1-12. Signals

در بخش interface signals همه سیگنال ها را به nios_custom_instruction_slave تغییر داده و تایپ سیگنال ها را بر اساس نوعشان تنظیم کردیم.



شکل 1-13. interfaces

در بخش interface باید مقادیری که مشخص شده اند مطابق تصویر تغییر کنند. به دلیل اینکه درجه فیلتر 64 است، تعداد سیکل مورد نیاز را 64 و باتوجه به اینکه فقط یک ورودی داریم، operands را برابر 1 قرار می‌دهیم. Clk cycle type را هم multicycle قرار می‌دهیم؛ چون عملیات فیلتر به بیشتر از یک سیکل نیاز دارد.

پس از اعمال تغییرات custom instruction جدید به system contents اضافه می‌شود.

Use	Connections	Name	Description	Export	Clock	Base	End	BIQ	Opcodes Name
		vga_clk	Clock Bridge		vga_clk	0x0000_0000	0x007e_ffff		
		SDRAM	SDRAM Controller		vga_clk	0x0080_0000	0x0087_ffff		
		SD_Card	SD Card Interface		vga_clk	0x00b0_0000	0x00b0_00ff		
		Flash	Altera IP Flash Memory IP Core		vga_clk	0x00c0_0000	0x00c0_00ff		
		Red_LEDs	Parallel Port		vga_clk	0x00d0_0000	0x00d0_00ff		
		Green_LEDs	Parallel Port		vga_clk	0x00e0_0000	0x00e0_00ff		
		HEX7_HEX4	Parallel Port		vga_clk	0x00f0_0000	0x00f0_00ff		
		Slider_Switches	Parallel Port		vga_clk	0x0100_0000	0x0100_00ff		
		PushButtons	Parallel Port		vga_clk	0x0110_0000	0x0110_00ff		
		Expansion_IP1	Parallel Port		vga_clk	0x0120_0000	0x0120_00ff		
		Expansion_IP2	Parallel Port		vga_clk	0x0130_0000	0x0130_00ff		
		PS2_Port	PS2 Controller		vga_clk	0x0140_0000	0x0140_00ff		
		USB	USB Controller		vga_clk	0x0150_0000	0x0150_00ff		
		ITAG_UART	ITAG UART		vga_clk	0x0160_0000	0x0160_00ff		
		Serial_Port	RS232 UART		vga_clk	0x0170_0000	0x0170_00ff		
		VGA_UART	VGA UART		vga_clk	0x0180_0000	0x0180_00ff		
		Ethernet	Ethernet		vga_clk	0x0190_0000	0x0190_00ff		
		Interval_Timer	Interval Timer		vga_clk	0x01a0_0000	0x01a0_00ff		
		clk	Clock Source						
		External_Clocks	Clock Signals for DE-series Board Part.		multiple	0x01b0_0000	0x01b0_00ff		
		AV_Config	Audio and Video Config		vga_clk	0x01c0_0000	0x01c0_00ff		
		VGA_Phase_Buffer	Pixel Buffer DDA Controller		vga_clk	0x01d0_0000	0x01d0_00ff		
		VGA_Phase_Buffer	RGB Resampler		vga_clk				
		VGA_Phase_Buffer	Scaler		vga_clk	multiple	multiple		
		VGA_Char_Buffer	Character Buffer for VGA Display		vga_clk				
		Alpha_Blending	Alpha Blending		vga_clk	multiple	multiple		
		VGA_Phase_Buffer	Pixel Buffer DDA Controller		vga_clk	0x01e0_0000	0x01e0_00ff		
		VGA_Controller	VGA Controller		vga_clk	0x01f0_0000	0x01f0_00ff		
		Audio	Audio		vga_clk	0x0200_0000	0x0200_00ff		
		Char_LCD_16x2	16x2 Character Display		vga_clk	0x0210_0000	0x0210_00ff		
		Video_In	Video-In Decoder		vga_clk				
		Video_In_Chroma_B	Chroma Resampler		vga_clk				
		Video_In_CSC	Color-Space Converter		vga_clk				
		Video_In_RGB_Resa...	RGB Resampler		vga_clk				
		Video_In_Clipping	Clipping		vga_clk				
		Video_In_Scaler	Scaler		vga_clk				
		Video_In_DMA_Contr...	DMA Controller		vga_clk	0x0220_0000	0x0220_00ff		
		CPU_Apiast	Floating Point Hardware						
		clk_07	Clock Bridge						
		clk_08	Clock Bridge						
		clk_09	Clock Bridge						
		clk_10	Clock Bridge						
		clk_11	Clock Bridge						
		clk_12	Clock Bridge						
		clk_13	Clock Bridge						
		clk_14	Clock Bridge						
		clk_15	Clock Bridge						
		clk_16	Clock Bridge						
		clk_17	Clock Bridge						
		clk_18	Clock Bridge						
		clk_19	Clock Bridge						
		clk_20	Clock Bridge						
		clk_21	Clock Bridge						
		clk_22	Clock Bridge						
		clk_23	Clock Bridge						
		clk_24	Clock Bridge						
		clk_25	Clock Bridge						
		clk_26	Clock Bridge						
		clk_27	Clock Bridge						
		clk_28	Clock Bridge						
		clk_29	Clock Bridge						
		clk_30	Clock Bridge						
		clk_31	Clock Bridge						
		clk_32	Clock Bridge						
		clk_33	Clock Bridge						
		clk_34	Clock Bridge						
		clk_35	Clock Bridge						
		clk_36	Clock Bridge						
		clk_37	Clock Bridge						
		clk_38	Clock Bridge						
		clk_39	Clock Bridge						
		clk_40	Clock Bridge						
		clk_41	Clock Bridge						
		clk_42	Clock Bridge						
		clk_43	Clock Bridge						
		clk_44	Clock Bridge						
		clk_45	Clock Bridge						
		clk_46	Clock Bridge						
		clk_47	Clock Bridge						
		clk_48	Clock Bridge						
		clk_49	Clock Bridge						
		clk_50	Clock Bridge						
		clk_51	Clock Bridge						
		clk_52	Clock Bridge						
		clk_53	Clock Bridge						
		clk_54	Clock Bridge						
		clk_55	Clock Bridge						
		clk_56	Clock Bridge						
		clk_57	Clock Bridge						
		clk_58	Clock Bridge						
		clk_59	Clock Bridge						
		clk_60	Clock Bridge						
		clk_61	Clock Bridge						
		clk_62	Clock Bridge						
		clk_63	Clock Bridge						
		clk_64	Clock Bridge						
		clk_65	Clock Bridge						
		clk_66	Clock Bridge						
		clk_67	Clock Bridge						
		clk_68	Clock Bridge						
		clk_69	Clock Bridge						
		clk_70	Clock Bridge						
		clk_71	Clock Bridge						
		clk_72	Clock Bridge						
		clk_73	Clock Bridge						
		clk_74	Clock Bridge						
		clk_75	Clock Bridge						
		clk_76	Clock Bridge						
		clk_77	Clock Bridge						
		clk_78	Clock Bridge						
		clk_79	Clock Bridge						
		clk_80	Clock Bridge						
		clk_81	Clock Bridge						
		clk_82	Clock Bridge						
		clk_83	Clock Bridge						
		clk_84	Clock Bridge						
		clk_85	Clock Bridge						
		clk_86	Clock Bridge						
		clk_87	Clock Bridge						
		clk_88	Clock Bridge						
		clk_89	Clock Bridge						
		clk_90	Clock Bridge						
		clk_91	Clock Bridge						
		clk_92	Clock Bridge						
		clk_93	Clock Bridge						
		clk_94	Clock Bridge						
		clk_95	Clock Bridge						
		clk_96	Clock Bridge						
		clk_97	Clock Bridge						
		clk_98	Clock Bridge						
		clk_99	Clock Bridge						
		clk_100	Clock Bridge						
		clk_101	Clock Bridge						
		clk_102	Clock Bridge						
		clk_103	Clock Bridge						
		clk_104	Clock Bridge						
		clk_105	Clock Bridge						
		clk_106	Clock Bridge						
		clk_107	Clock Bridge						
		clk_108	Clock Bridge						
		clk_109	Clock Bridge						
		clk_110	Clock Bridge						
		clk_111	Clock Bridge						
		clk_112	Clock Bridge						
		clk_113	Clock Bridge						
		clk_114	Clock Bridge						
		clk_115	Clock Bridge						
		clk_116	Clock Bridge						
		clk_117	Clock Bridge						
		clk_118	Clock Bridge						
		clk_119	Clock Bridge						
		clk_120	Clock Bridge						
		clk_121	Clock Bridge						
		clk_122	Clock Bridge						
		clk_123	Clock Bridge						
		clk_124	Clock Bridge						
		clk_125	Clock Bridge						
		clk_126	Clock Bridge						
		clk_127	Clock Bridge						
		clk_128	Clock Bridge						
		clk_129	Clock Bridge						
		clk_130	Clock Bridge						
		clk_131	Clock Bridge						
		clk_132	Clock Bridge						
		clk_133	Clock Bridge						
		clk_134	Clock Bridge						
		clk_135	Clock Bridge						
		clk_136	Clock Bridge						
		clk_137	Clock Bridge						
		clk_138	Clock Bridge						
		clk_139	Clock Bridge						
		clk_140	Clock Bridge						
		clk_141	Clock Bridge						
		clk_142	Clock Bridge						
		clk_143	Clock Bridge						
		clk_144	Clock Bridge						
		clk_145	Clock Bridge						

منابع	قبل از اضافه کردن custom instruction	بعد از اضافه کردن custom instruction
Total logic elements	17865	20022
Total registers	12119	13766
Embedded Multiplier 9-bit elements	21	25

جدول 1- مقایسه میزان منابع

پس از سنتز کردن، یک ماکرو برای custom instruction تولید شده در کد سخت افزار در قسمت system.h ایجاد میشود.

```
#define ALT_CI_FIR_SER_0(A) __builtin_custom_ini(ALT_CI_FIR_SER_0_N,(A))
#define ALT_CI_FIR_SER_0_N 0x0
```

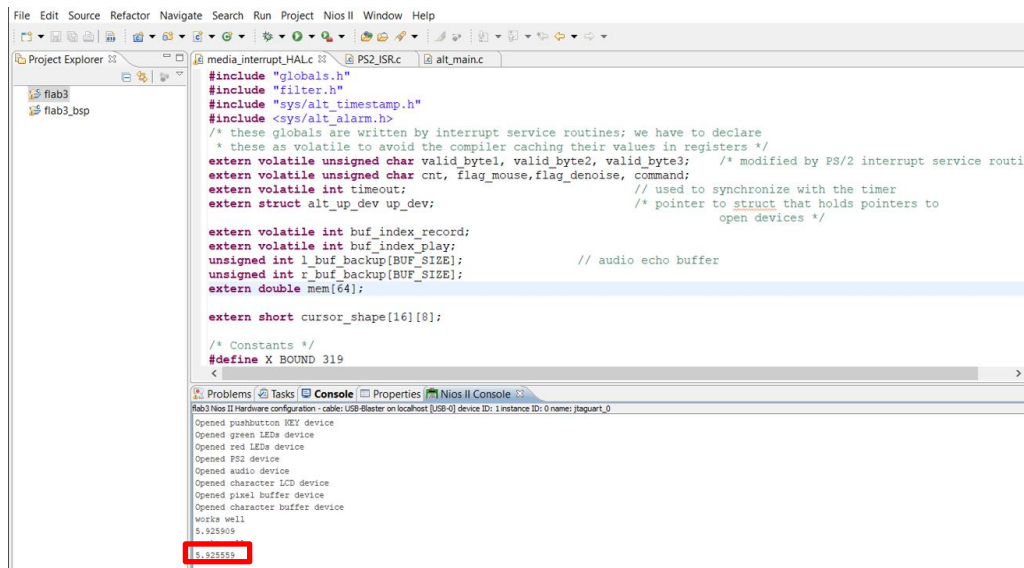
شکل 1-17. New Macro

تغییرات زیر را در کد تابع make_denoise ایجاد میکنیم:

```
void make_Denoise(){
    int i;
    for (i = 0; i < BUF_SIZE; i++){
        l_buf_Denoise[i] = ALT_CI_FIR_SER_0(l_buf[i]);
    }
    for (i = 0; i < BUF_SIZE; i++){
        r_buf_Denoise[i] = ALT_CI_FIR_SER_0(r_buf[i]);
    }
}
```

شکل 1-18. دینویز کردن با ماکروی تولید شده

در طراحی سخت افزار پس از اجرای برنامه صدا ضبط شده و به سرعت دینویز شد. صدای دینویز شده با کیفیت خوبی بود و واضح شده بود و فیلتر بسیار خوب در بخش سخت افزاری کار کرد. به طوری که نویز موجود در صدای ضبط شده را به طور کامل برطرف کرد و دیگر نویزی قابل شنیدن نبود. مطابق حالت نرم افزاری با استفاده از timestamp زمان اجرای فیلتر را محاسبه میکنیم.



شکل 1-19. زمان اجرای فیلتر

زمان اندازه گیری شده مطابق تصویر، 5.925559 ثانیه است که واضحاً از فیلتر نرم افزاری بسیار سریع تر است.

از مقایسه زمان سخت افزار و نرم افزار در میابیم که سرعت طراحی سخت افزاری خیلی بیشتر بوده و در حد چند ثانیه است. در صورتی که زمان نرم افزار به ساعت میرسد. پس طراحی سخت افزاری هر چند ممکن است سخت تر باشد ولی از نظر زمان و توان بسیار بهتر عمل میکند.

سوال: نیازمندی های تعریف یک دستور Multicycle

برای تعریف دستور multicycle باید دید آیا دستوری که می‌خواهیم اضافه کنیم در یک تعداد سیکل مشخص عملیات آن به پایان میرسد یا تعداد سیکل انجام عملیات آن متغیر است. اگر تعداد سیکل مشخص باشد در generation system تعداد سیکل مورد نیاز دستور را تعیین میکنیم. اگر تعداد سیکل های اجرای دستور متغیر باشد، با استفاده از تعریف دو پورت جدید start و done شروع و پایان انجام عملیات دستور را با استفاده از مکانیزم handshaking انجام میدهیم. در تعریف custom instruction ها، پورت هایی میتواند تعیین شود که برای multicycle custom instruction ها پورت های clk_en, rst, clk اجباری هستند. برای دستور با تعداد سیکل های متغیر پورت های start, done هم نیاز است. در شکل 1-8. Signals پورت های تعریف شده برای custom instruction مورد نیاز ما قابل مشاهده است.