

FPGA – based Embedded System Design

LAB #4

Group Members

Mohammad Taghizadeh Givari	810198373
Zeinab Saeedi	810198411
Amin Aroufzad	810198538

محتوا

۱. پیاده سازی سیستم Audio plot به صورت نرم افزاری.....۲
۲. پیاده سازی سیستم Audio plot به صورت شتاب دهنده سخت افزاری.....۳

پیاده سازی سیستم Audio plot به صورت نرم افزاری

رسم میانگین داده های ضبط شده پس از ضبط صدا:

ابتدا مقادیر موقعیت عمودی هر مستطیل، پهنای ناحیه ای که مستطیل ها در آن محدوده رسم می شوند، طول هر بازه، پهنای هر مستطیل رسم شده، موقعیت افقی اولین مستطیلی که قرار است رسم شود و بیشینه ارتفاع هر مستطیل را معین می کنیم:

```
void plot_audio(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev, unsigned int N)
{
    int i;
    int y_of_each_plot = 48;
    int width_of_screen = 62;
    unsigned long long length_of_each_period = (BUF_SIZE/N);
    int width_of_each_plot = (width_of_screen / N);
    int x_of_current_drawing_plot = 14; // 14 is x position of first plot
    int max_heigh_of_plots = 20;
```



ابتدا یک آرایه ای از اعداد صحیح به نام average به تعداد بازه هایی که داریم (N) تعریف می کنیم تا میانگین هر بازه را پس از محاسبه، در این آرایه ذخیره کنیم.

برای رسم میانگین داده های ضبط شده، ابتدا طی دو حلقه تو در تو:

- در هر بار اجرا حلقه بیرونی، میانگین یک بازه محاسبه می شود.
- پس از اجرا حلقه داخلی نیز مقدار مجموع داده های آن بازه، محاسبه می شود.

(از جایی که داده ی ضبط شده در ۲۴ بیت پر ارزش record_r_buf هست، آن را به اندازه ۸ بیت به سمت راست شیفت می دهیم تا داده ضبط شده حاصل شود)

- با تقسیم مجموع محاسبه شده بر طول هر بازه (length_of_each_period)، میانگین بازه محاسبه در آرایه average ذخیره می شود.

```

Nios II - FPGA_Lab4_Software/media_interrupt_HALc - Eclipse
File Edit Source Refactor Navigate Search Run Project Nios II Window Help

media_interrupt_HALc.c PS2_ISR.c audio_ISR.c

int width_of_each_plot = (width_of_screen / N);
int x_of_current_drawing_plot = 14; // 14 is x position of first plot
int max_height_of_plots = 20;

int average[N];
int max_average = record_r_buf[0];
int min_average = 67000000;

int color = 0xF800;

int n = 0;
unsigned long long sum;
for(i = 0; i < BUF_SIZE; i = i + length_of_each_period)
{
    int j;
    sum = 0;
    for(j = i; j < (i + length_of_each_period); j++)
    {
        sum = sum + (record_r_buf[j] >> 8);
        //printf("%llu \n", sum);
    }
    sum = sum / length_of_each_period;
    average[n] = sum;
    //printf("%llu, %u\n", sum, average[n]);
    if(average[n] > max_average)
    {
        max_average = average[n];
    }
    if(average[n] < min_average)
    {
        min_average = average[n];
    }
    n++;
}

```

در طی این دو حلقه تو در تو، کمینه و بیشینه مقدار میانگین هم محاسبه می کنیم تا از آن برای نرمال سازی مقادیر میانگین استفاده شود. از جایی که مقدار میانگین ارتفاع مستطیل های رسم شده را تعیین می کنند و ارتفاع مستطیل رسم شده نباید از یه حدی بیش تر شود (اگر ارتفاع آن زیاد باشد مستطیل در ناحیه آیکون ها رسم می شود و ممکن است از صفحه نیز بیرون زده شود که باعث کاهش زیبایی خواهد شد)

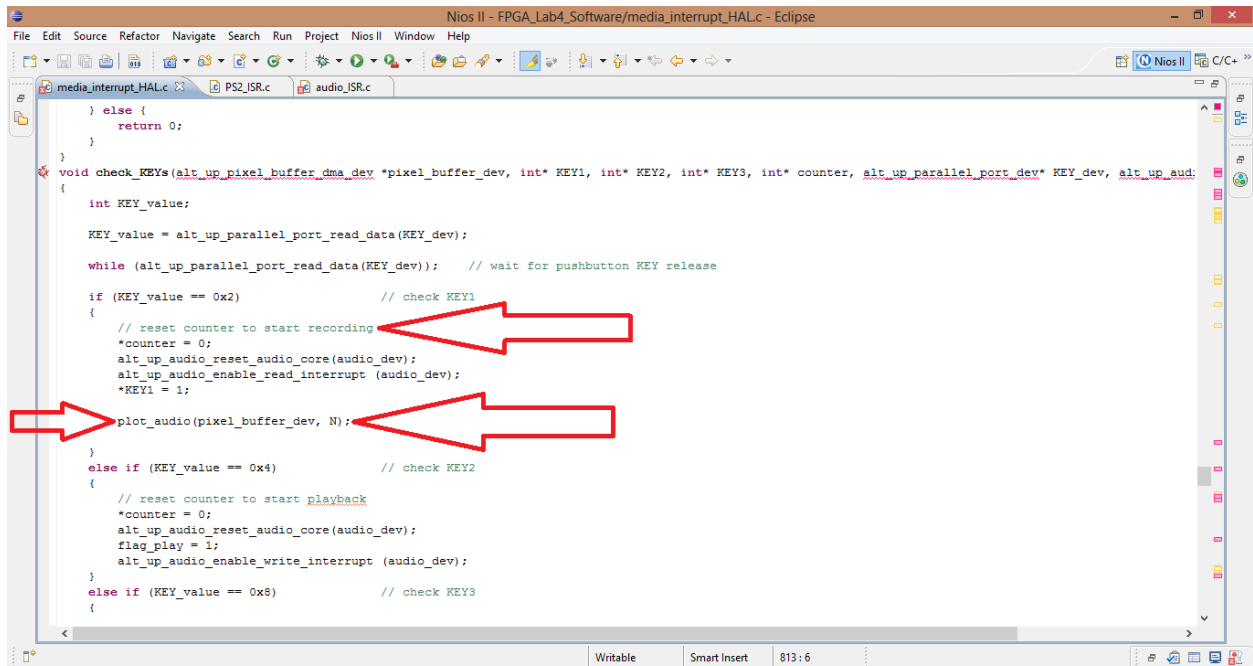
در نتیجه با نرمال سازی مقادیر میانگین کاری می کنیم که حداکثر ارتفاع مستطیل برابر با حداکثر ارتفاع مستطیل ها `max_height_of_plots` و کمینه ارتفاع آن نیز 0 شود.

در آخر طی یک حلقه، مقادیر میانگین را ابتدا نرمال سازی کرده و سپس مستطیلی به ارتفاع میانگین نرمال شده، رسم می شود. سپس موقعیت افقی به اندازه پهنا هر مستطیل اضافه می شود تا مستطیل بعدی در کنار مستطیل قبلی رسم شود و بر روی آن رسم نشود.

```

//printf("%llu \n", sum);
sum = sum / length_of_each_period;
average[n] = sum;
//printf("%llu, %u\n", sum, average[n]);
if(average[n] > max_average)
{
    max_average = average[n];
}
if(average[n] < min_average)
{
    min_average = average[n];
}
n++;
}
n = 0;
int k;
for(k = 0; k < N; k++)
{
    average[n] = ((average[n] - min_average) * max_height_of_plots) / (max_average - min_average);
    //printf("%u, ", (average[n] - min_average));
    //printf("%u\n", min_average);
    //average[n] = (average[n] > 0) ? average[n] : 0;
    alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, x_of_current_drawing_plot * 4, (y_of_each_plot - average[n]) * 4, (x_of_current_drawing_plot + 1
        y_of_each_plot * 4, color, 0);
    x_of_current_drawing_plot += width_of_each_plot;
    n++;
}
    
```

در آخر پس از اجرا عملیات ضبط صوت، با فراخوانی تابع `plot_audio`، میانگین داده های ضبط شده به صورت مستطیل های قرمز رنگ ظاهر می شود:



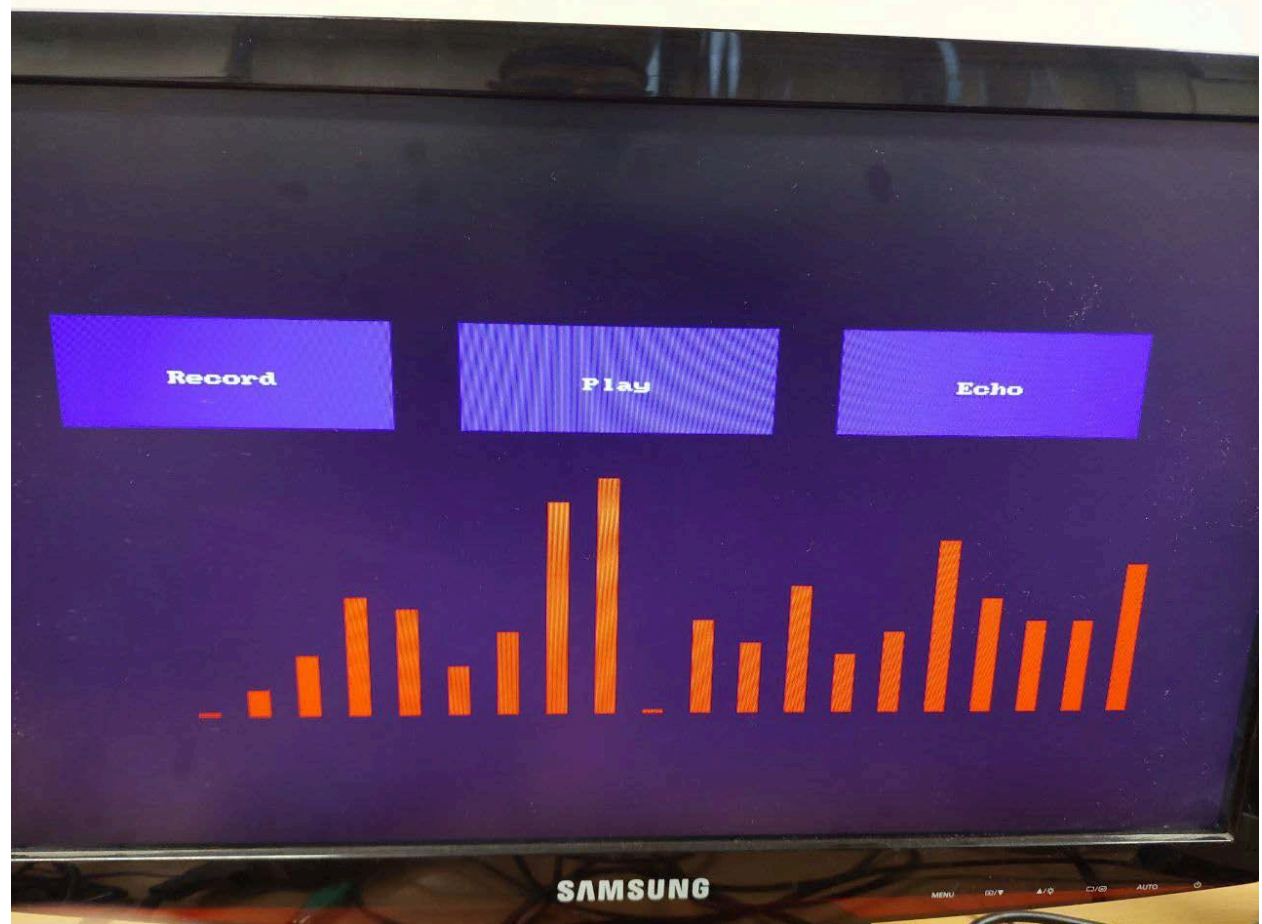
```
void check_KEYS(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev, int* KEY1, int* KEY2, int* KEY3, int* counter, alt_up_parallel_port_dev* KEY_dev, alt_up_audio_dev* audio_dev)
{
    int KEY_value;

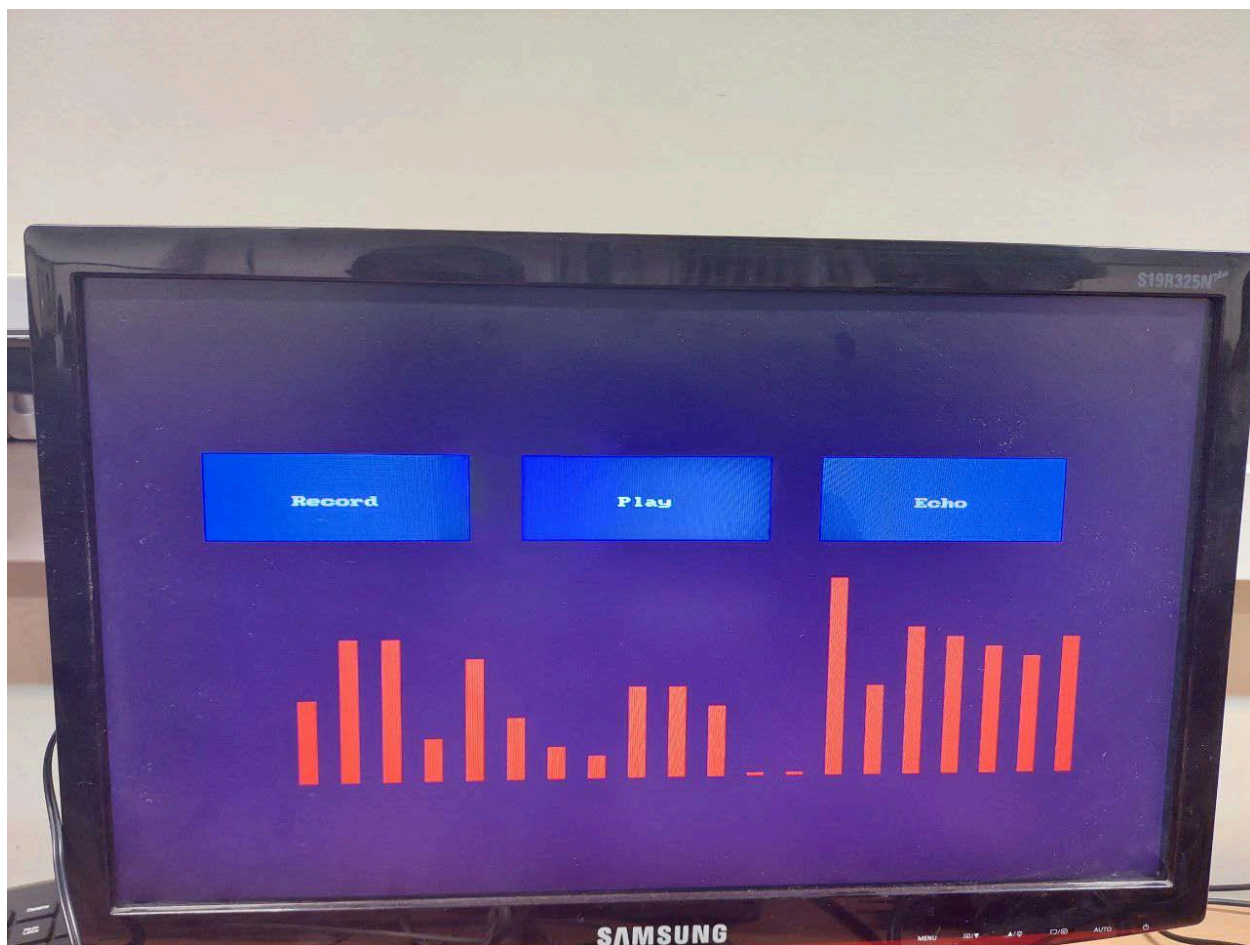
    KEY_value = alt_up_parallel_port_read_data(KEY_dev);

    while (alt_up_parallel_port_read_data(KEY_dev)); // wait for pushbutton KEY release

    if (KEY_value == 0x2) // check KEY1
    {
        // reset counter to start recording
        *counter = 0;
        alt_up_audio_reset_audio_core(audio_dev);
        alt_up_audio_enable_read_interrupt (audio_dev);
        *KEY1 = 1;

        plot_audio(pixel_buffer_dev, N);
    }
    else if (KEY_value == 0x4) // check KEY2
    {
        // reset counter to start playback
        *counter = 0;
        alt_up_audio_reset_audio_core(audio_dev);
        flag_play = 1;
        alt_up_audio_enable_write_interrupt (audio_dev);
    }
    else if (KEY_value == 0x8) // check KEY3
    {
    }
```





رسم مکان فعلی پخش صدا، هنگام پخش کردن صدا ضبط شده:

در وقفه مربوط به صوت Audio_ISR، در بخش مربوط به play:

ابتدا با توجه به موقعیت فعلی پخش صوت که با `buf_index_play` مشخص می شود، تعداد بازه ها و مستطیل هایی که باید رسم شود، محاسبه شده و در متغیر `n` ذخیره می شود.

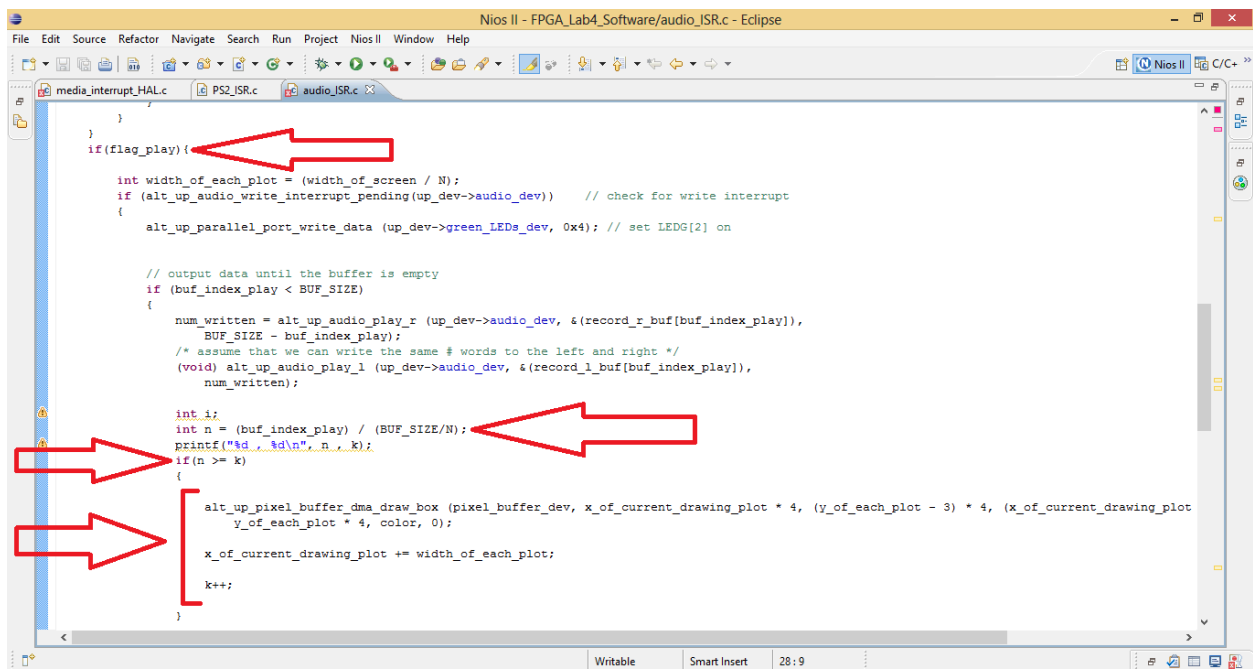
متغیر `k` با مقدار اولیه 0، بیانگر تعداد مستطیل های رسم شده تا این لحظه از پخش صوت است.

بنابراین هرگاه تعداد مستطیل هایی که باید رسم شود بیش تر از تعداد مستطیل های رسم شده باشد:

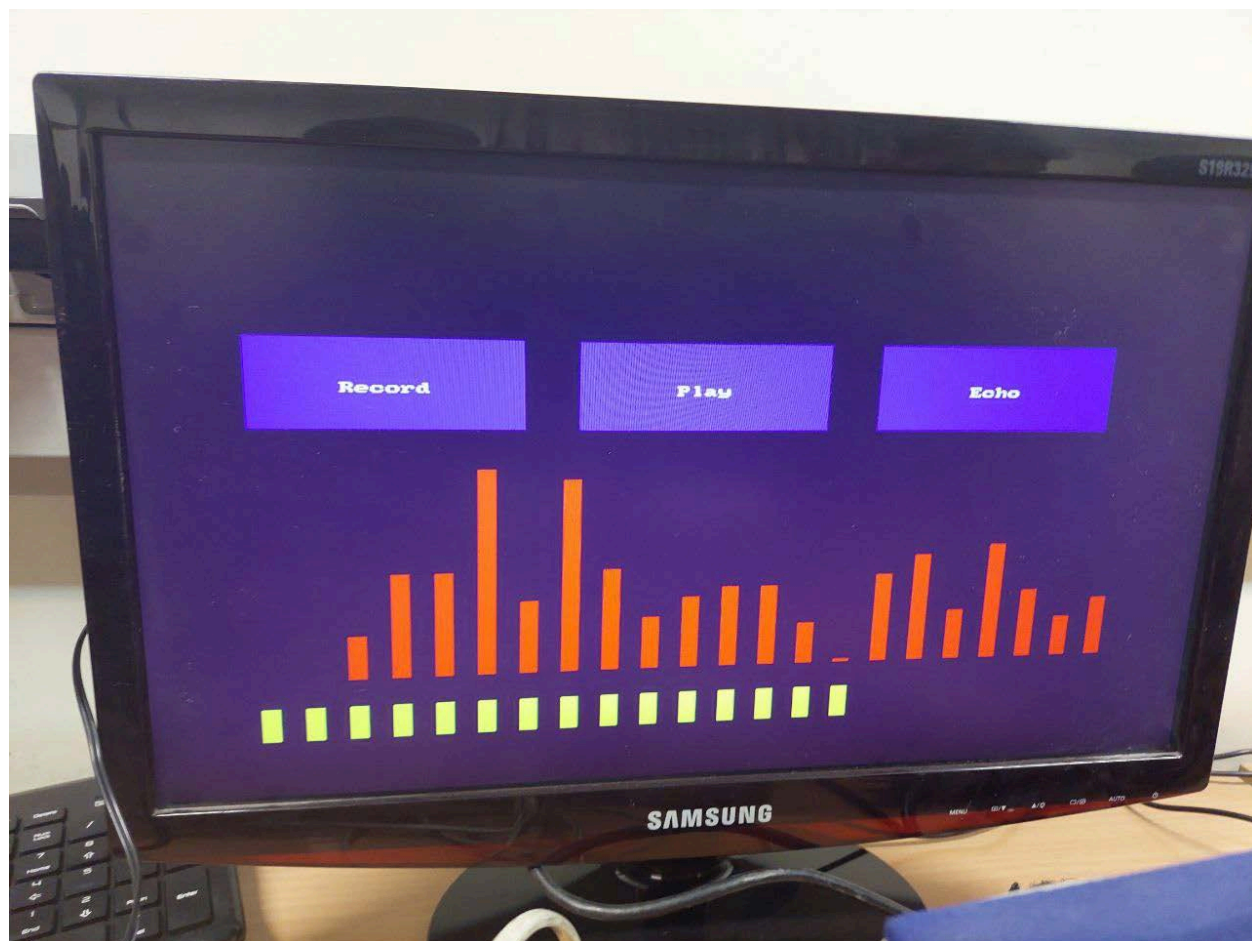
- یک مستطیل رسم شده و به تعداد مستطیل های رسم شده (`k`) یک واحد اضافه می شود.

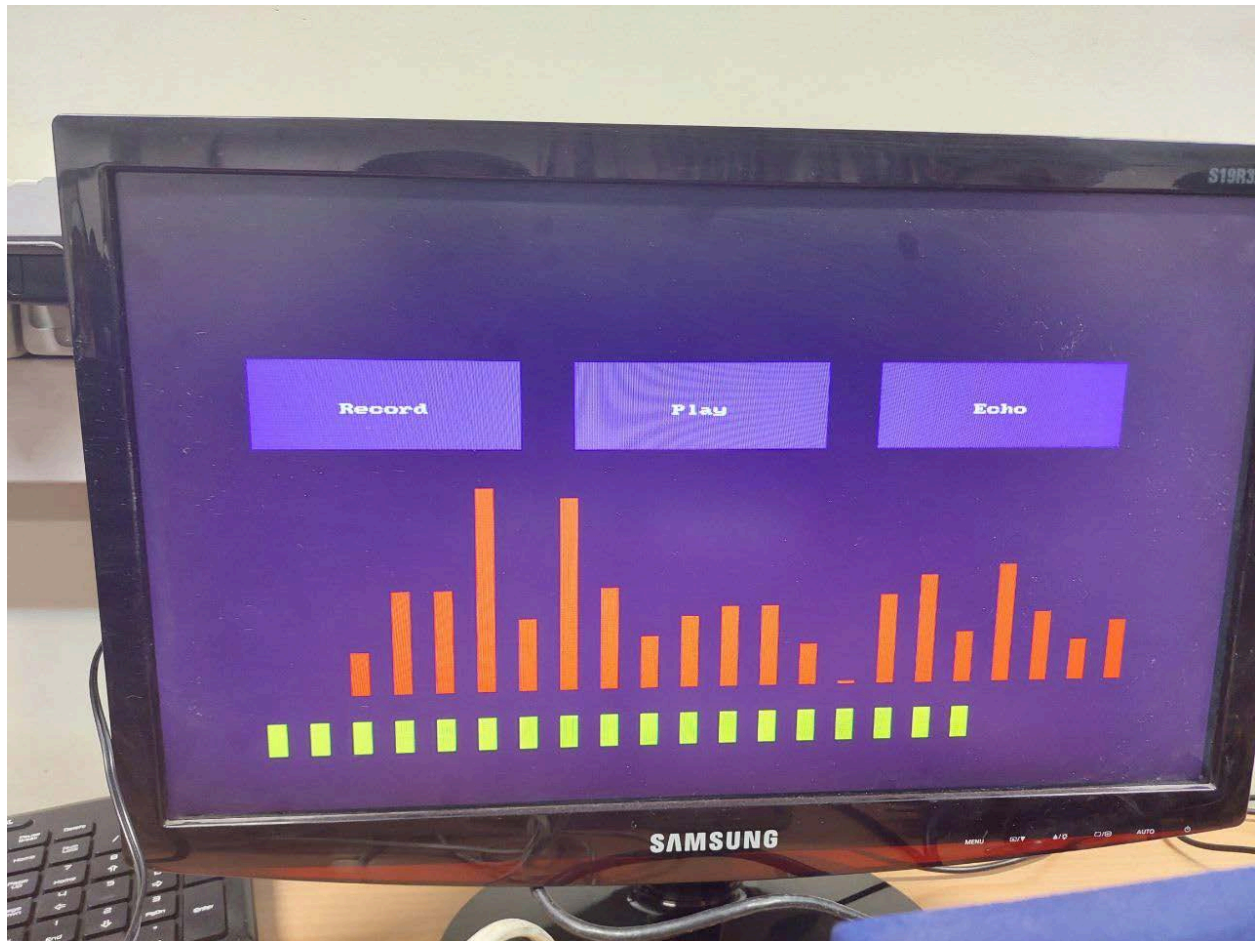
- سپس موقعیت افقی به اندازه پهنای هر مستطیل اضافه می شود تا مستطیل بعدی در کنار مستطیل قبلی

رسم شود و بر روی آن رسم نشود.



```
1 }
2 if(flag_play){
3     int width_of_each_plot = (width_of_screen / N);
4     if (alt_up_audio_write_interrupt_pending(up_dev->audio_dev)) // check for write interrupt
5     {
6         alt_up_parallel_port_write_data (up_dev->green_LEDs_dev, 0x9); // set LEDG[2] on
7
8         // output data until the buffer is empty
9         if (buf_index_play < BUF_SIZE)
10         {
11             num_written = alt_up_audio_play_r (up_dev->audio_dev, &(record_r_buf[buf_index_play]),
12             BUF_SIZE - buf_index_play);
13             /* assume that we can write the same # words to the left and right */
14             (void) alt_up_audio_play_l (up_dev->audio_dev, &(record_l_buf[buf_index_play]),
15             num_written);
16
17             int i;
18             int n = (buf_index_play) / (BUF_SIZE/N);
19             printf("%d, %d\n", n, k);
20             if(n >= k)
21             {
22                 alt_up_pixel_buffer_dma_draw_box (pixel_buffer_dev, x_of_current_drawing_plot * 4, (y_of_each_plot - 3) * 4, (x_of_current_drawing_plot
23                 y_of_each_plot * 4, color, 0);
24                 x_of_current_drawing_plot += width_of_each_plot;
25                 k++;
26             }
27         }
28     }
29 }
```





همان طور که در ۳ تصویر فوق میبینید، با پخش شدن صدا ضبط شده، به تدریج مستطیل های زرد رنگ بیش تری رسم می شوند تا اینکه در آخر مطابق تصویر فوق، به انتهای صفحه نمایش، می رسم که نشان دهنده اتمام پخش صدا ضبط شده هست.

پیاده سازی سیستم Audio plot به صورت شتاب دهنده سخت افزاری

طراحی رابطه Avalon Memory –Mapped Slave:

ابتدا ۴ رجیستر برای رجیستر های Config Reg, Right Addr, Left Addr , Out Addr تعریف می کنیم.

یک رجیستر به نام Read_data نیز تعریف می کنیم که مقداری که از این رجیستر ها خوانده می شود در آن قرار بگیرد:

```
34 reg [AVS_AVALONSLAVE_DATA_WIDTH - 1:0] read_data;
35 // these are slave registers. they MUST be here!
36 reg [AVS_AVALONSLAVE_DATA_WIDTH - 1:0] slv_reg0;
37 reg [AVS_AVALONSLAVE_DATA_WIDTH - 1:0] slv_reg1;
38 reg [AVS_AVALONSLAVE_DATA_WIDTH - 1:0] slv_reg2;
39 reg [AVS_AVALONSLAVE_DATA_WIDTH - 1:0] slv_reg3;
```



ابتدا اگر سیگنال CSI_CLOCK_RESET_N، فعال باشد (0 باشد)، مدار را ریست کرده و محتوای

رجیستر های Slave، را 0 می کنیم:

```
49 always @(posedge CSI_CLOCK_CLK)
50 begin
51 // usually resets are active low but you can change its trigger type
52 if(CSI_CLOCK_RESET_N == 0)
53 begin
54 slv_reg0 <= 0;
55 slv_reg1 <= 0;
56 slv_reg2 <= 0;
57 slv_reg3 <= 0;
58 end
end
```



برای خواندن از Slave، اگر سیگنال AVS_AVALONSLAVE_READ، فعال باشد (1 شود)، متناسب با

سیگنال AVS_AVALONSLAVE_ADDRESS، محتوای یکی از رجیستر های Slave در خروجی

AVS_AVALONSLAVE_READDATA قرار می گیرد:

```
83 assign AVS_AVALONSLAVE_READDATA = -AVS_AVALONSLAVE_READ ? AVS_AVALONSLAVE_READDATA :
84 (AVS_AVALONSLAVE_ADDRESS == 0) ? slv_reg0 :
85 (AVS_AVALONSLAVE_ADDRESS == 1) ? slv_reg1 :
86 (AVS_AVALONSLAVE_ADDRESS == 2) ? slv_reg2 :
87 (AVS_AVALONSLAVE_ADDRESS == 3) ? slv_reg3 :
88 AVS_AVALONSLAVE_READDATA;
```



برای نوشتن به Slave، اگر سیگنال AVS_AVALONSLAVE_WRITE، فعال باشد (1 شود)، متناسب با سیگنال AVS_AVALONSLAVE_ADDRESS، در یکی از رجیستر های Slave، محتوای AVS_AVALONSLAVE_WRITEDATA قرار می گیرد:

```

59   else if (AVS_AVALONSLAVE_WRITE)
60   begin
61       // address is always byte-wise so must divide it by 4 for 32bit word
62       case (AVS_AVALONSLAVE_ADDRESS)
63       0: slv_reg0 <= AVS_AVALONSLAVE_WRITEDATA;
64       1: slv_reg1 <= AVS_AVALONSLAVE_WRITEDATA;
65       2: slv_reg2 <= AVS_AVALONSLAVE_WRITEDATA;
66       3: slv_reg3 <= AVS_AVALONSLAVE_WRITEDATA;
67       default:
68       begin
69           slv_reg0 <= slv_reg0;
70           slv_reg1 <= slv_reg1;
71           slv_reg2 <= slv_reg2;
72           slv_reg3 <= slv_reg3;
73       end
74       endcase
75   end

```

اگر کار مدار محاسبه کننده میانگین دامنه، خاتمه یابد، (به عبارتی DONE فعال می شود) بیت ۳۲ ام رجیستر Config Reg(slv_reg0)، که بیانگر اتمام کار مدار هست، باید ۱ شود. پس محتوای رجیستر Config Reg(slv_reg0) را با 32'b10000000000000000000000000000000، یا به عبارتی 32'h80000000، OR می کنیم تا بیت ۳۲ ام رجیستر برابر ۱ شود.

```

77   else if (DONE)
78   begin
79       slv_reg0 <= (slv_reg0 | 32'h80000000);
80   end
81   end

```

بیت start، که بیانگر شروع کار مدار محاسبه کننده میانگین دامنه است، در اولین بیت رجیستر Config Reg(slv_reg0)، می باشد پس خروجی start را برابر با slv_reg[0] قرار دادیم:

```

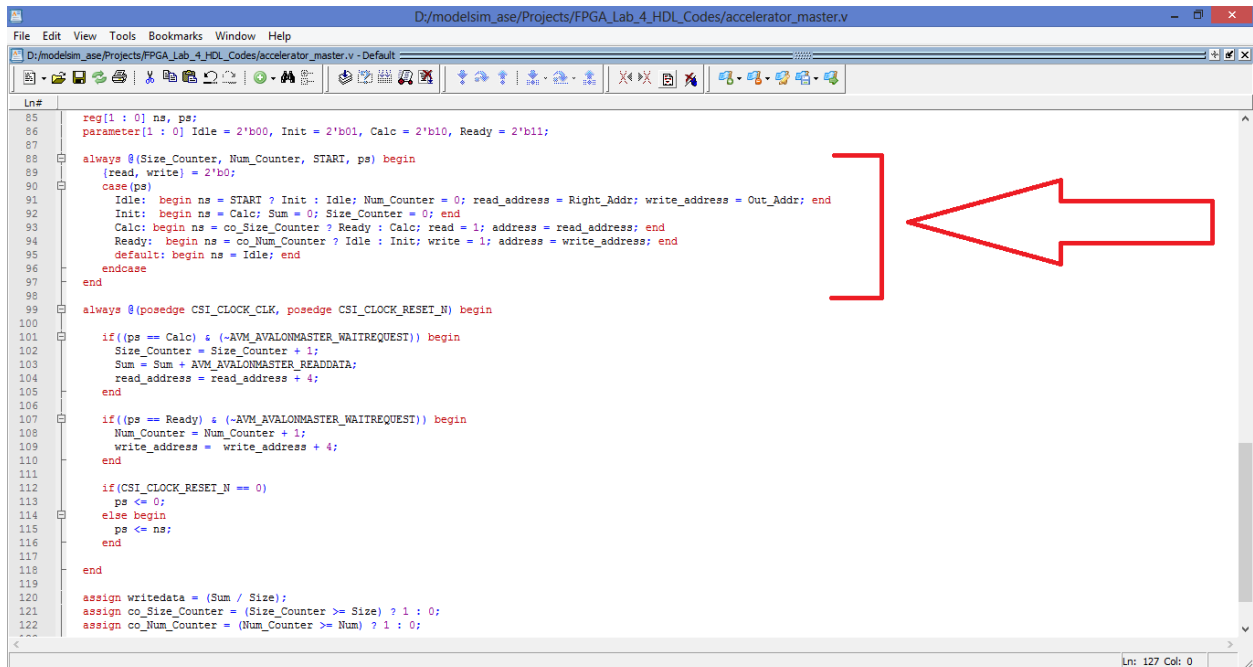
89   assign start = slv_reg0[0];
90
91

```

طراحی رابطه Avalon Memory –Mapped Master و مدار محاسبه کننده میانگین دامنه:

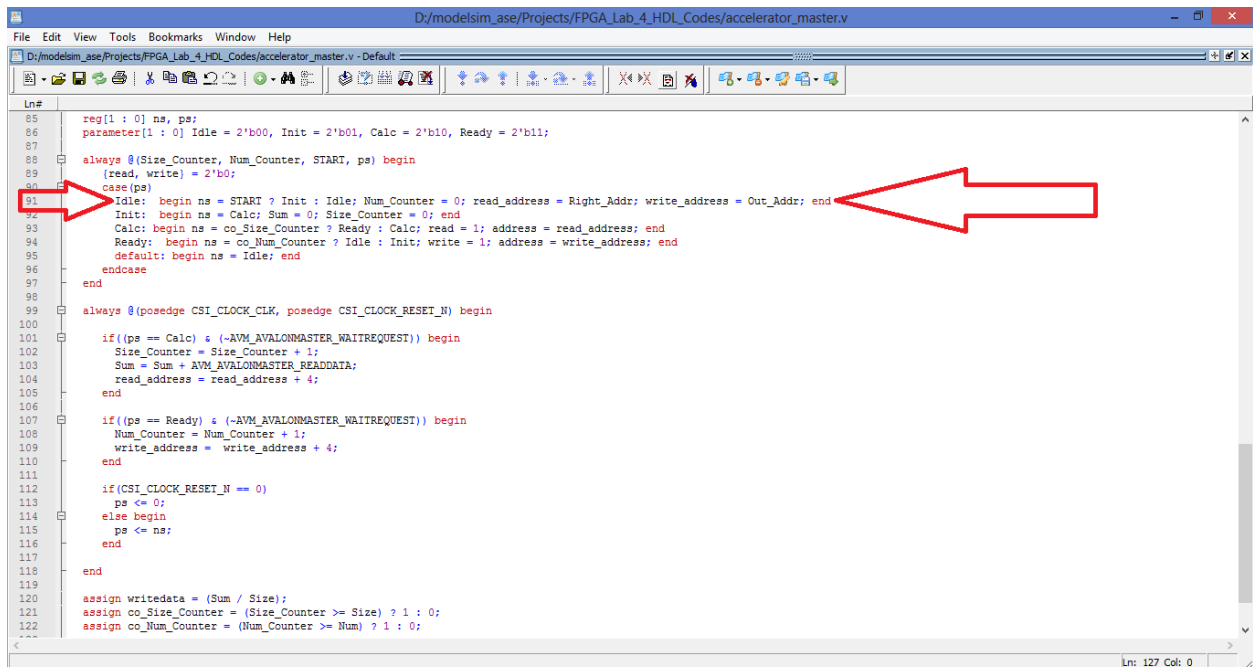
در ابتدا صبر می کنیم تا بیت start فعال شود. سپس مشابه کاری که در بخش نرم افزاری کردیم، ابتدا با استفاده از یک شمارنده به نام Size_Counter به اندازه طول یک بازه دامنه ها را با هم جمع میکنیم سپس با یک تقسیم میانگین محاسبه می شود. این کار را با یک شمارنده دیگر به نام Num_Counter به اندازه تعداد بازه ها تکرار می کنیم تا در آخر میانگین تمام بازه ها محاسبه شود سپس بیت done را فعال می کنیم.

با توجه به state machine زیر:



```
Ln#
85 reg[1 : 0] ns, ps;
86 parameter[1 : 0] Idle = 2'b00, Init = 2'b01, Calc = 2'b10, Ready = 2'b11;
87
88 always @(Size_Counter, Num_Counter, START, ps) begin
89     (read, write) = 2'b0;
90     case (ps)
91     Idle: begin ns = START ? Init : Idle; Num_Counter = 0; read_address = Right_Addr; write_address = Out_Addr; end
92     Init: begin ns = Calc; Sum = 0; Size_Counter = 0; end
93     Calc: begin ns = co_Size_Counter ? Ready : Calc; read = 1; address = read_address; end
94     Ready: begin ns = co_Num_Counter ? Idle : Init; write = 1; address = write_address; end
95     default: begin ns = Idle; end
96     endcase
97 end
98
99 always @(posedge CSI_CLOCK_CLK, posedge CSI_CLOCK_RESET_N) begin
100
101     if ((ps == Calc) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
102         Size_Counter = Size_Counter + 1;
103         Sum = Sum + AVM_AVALONMASTER_READDATA;
104         read_address = read_address + 4;
105     end
106
107     if ((ps == Ready) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
108         Num_Counter = Num_Counter + 1;
109         write_address = write_address + 4;
110     end
111
112     if (CSI_CLOCK_RESET_N == 0)
113         ps <= 0;
114     else begin
115         ps <= ns;
116     end
117
118 end
119
120 assign writedata = (Sum / Size);
121 assign co_Size_Counter = (Size_Counter >= Size) ? 1 : 0;
122 assign co_Num_Counter = (Num_Counter >= Num) ? 1 : 0;
```

تا زمانی که بیت start فعال نشده در state Idle می مانیم به عبارتی صبر می کنیم تا بیت start فعال شده و بعد شروع به کار کنیم.



```
Ln#
85 reg[1 : 0] ns, ps;
86 parameter[1 : 0] Idle = 2'b00, Init = 2'b01, Calc = 2'b10, Ready = 2'b11;
87
88 always @(Size_Counter, Num_Counter, START, ps) begin
89     (read, write) = 2'b0;
90     case (ps)
91     Idle: begin ns = START ? Init : Idle; Num_Counter = 0; read_address = Right_Addr; write_address = Out_Addr; end
92     Init: begin ns = Calc; Sum = 0; Size_Counter = 0; end
93     Calc: begin ns = co_Size_Counter ? Ready : Calc; read = 1; address = read_address; end
94     Ready: begin ns = co_Num_Counter ? Idle : Init; write = 1; address = write_address; end
95     default: begin ns = Idle; end
96     endcase
97 end
98
99 always @(posedge CSI_CLOCK_CLK, posedge CSI_CLOCK_RESET_N) begin
100
101     if ((ps == Calc) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
102         Size_Counter = Size_Counter + 1;
103         Sum = Sum + AVM_AVALONMASTER_READDATA;
104         read_address = read_address + 4;
105     end
106
107     if ((ps == Ready) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
108         Num_Counter = Num_Counter + 1;
109         write_address = write_address + 4;
110     end
111
112     if (CSI_CLOCK_RESET_N == 0)
113         ps <= 0;
114     else begin
115         ps <= ns;
116     end
117
118 end
119
120 assign writedata = (Sum / Size);
121 assign co_Size_Counter = (Size_Counter >= Size) ? 1 : 0;
122 assign co_Num_Counter = (Num_Counter >= Num) ? 1 : 0;
```

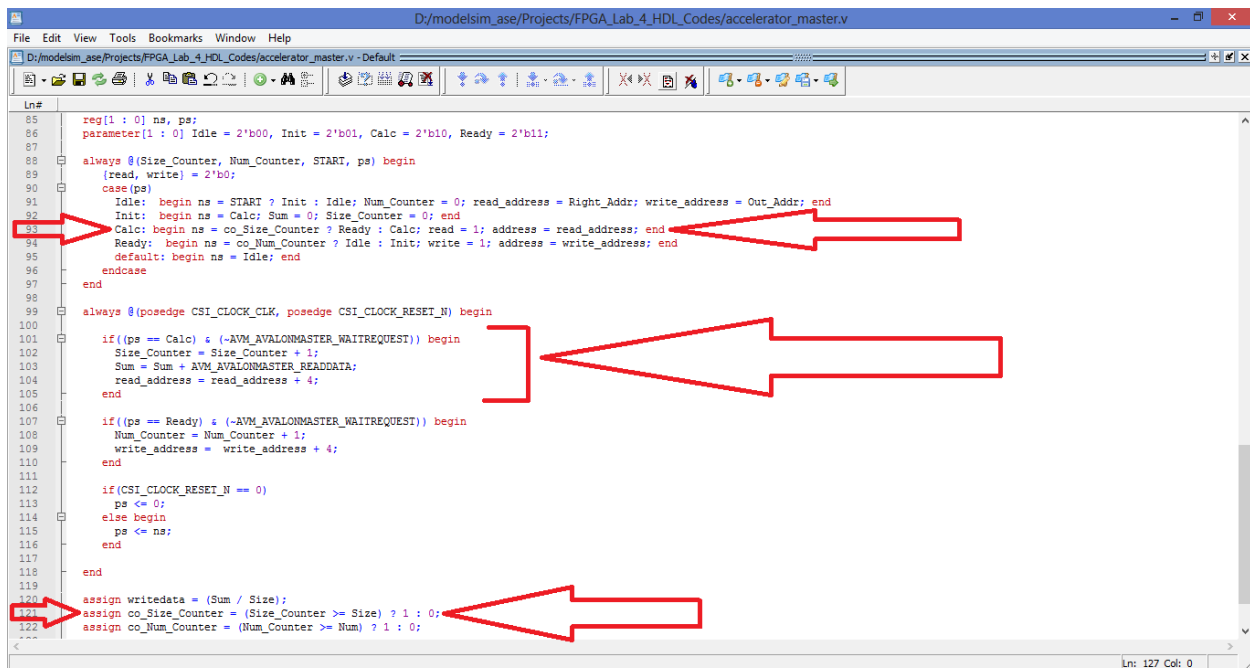

سپس در state init، sum که بیانگر مجموع دامنه های یک بازه هست را 0 میکنیم و Size_counter که شمارنده مربوط به شمارش یک بازه هست را 0 می کنیم.

```
File Edit View Tools Bookmarks Window Help
D:/modelsim_ase/Projects/FPGA_Lab_4_HDL_Codes/accelerator_master.v

Ln#
85 reg[1 : 0] ns, ps;
86 parameter[1 : 0] Idle = 2'b00, Init = 2'b01, Calc = 2'b10, Ready = 2'b11;
87
88 always @(Size_Counter, Num_Counter, START, ps) begin
89     (read, write) = 2'b0;
90     case(ps)
91         Idle: begin ns = START ? Init : Idle; Num_Counter = 0; read_address = Right_Addr; write_address = Out_Addr; end
92         Init: begin ns = Calc; Sum = 0; Size_Counter = 0; end
93         Calc: begin ns = co_Size_Counter ? Ready : Calc; read = 1; address = read_address; end
94         Ready: begin ns = co_Num_Counter ? Idle : Init; write = 1; address = write_address; end
95         default: begin ns = Idle; end
96     endcase
97 end
98
99 always @(posedge CSI_CLOCK_CLK, posedge CSI_CLOCK_RESET_N) begin
100
101     if((ps == Calc) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
102         Size_Counter = Size_Counter + 1;
103         Sum = Sum + AVM_AVALONMASTER_READDATA;
104         read_address = read_address + 4;
105     end
106
107     if((ps == Ready) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
108         Num_Counter = Num_Counter + 1;
109         write_address = write_address + 4;
110     end
111
112     if(CSI_CLOCK_RESET_N == 0)
113         ps <= 0;
114     else begin
115         ps <= ns;
116     end
117 end
118
119
120 assign writedata = (Sum / Size);
121 assign co_Size_Counter = (Size_Counter >= Size) ? 1 : 0;
122 assign co_Num_Counter = (Num_Counter >= Num) ? 1 : 0;
123
```


به اندازه طول یک بازه مدام از حافظه SRAM، می خوانیم ($read = 1$)، سپس مقدار خوانده شده را با sum جمع می کنیم تا در آخر هنگامی که به تعداد یک بازه شماردیم ($co_Size_Counter = 1$)، مجموع دامنه های یک بازه در sum قرار گیرد.

برای خواندن دامنه ها کافیهست آدرس خواندن ($read_address$) را هر دفعه ۴ واحد زیاد کنیم تا عملاً ۳۲ بیت در حافظه جلو رفته و به این ترتیب مقدار خوانده شده، محتوای خانه ی بعدی حافظه باشد.



```
Ln#
85 reg[1 : 0] ns, ps;
86 parameter[1 : 0] Idle = 2'b00, Init = 2'b01, Calc = 2'b10, Ready = 2'b11;
87
88 always @(Size_Counter, Num_Counter, START, ps) begin
89     (read, write) = 2'b0;
90     case (ps)
91         Idle: begin ns = START ? Init : Idle; Num_Counter = 0; read_address = Right_Addr; write_address = Out_Addr; end
92         Init: begin ns = Calc; Sum = 0; Size_Counter = 0; end
93         Calc: begin ns = co_Size_Counter ? Ready : Calc; read = 1; address = read_address; end
94         Ready: begin ns = co_Num_Counter ? Idle : Init; write = 1; address = write_address; end
95         default: begin ns = Idle; end
96     endcase
97 end
98
99 always @(posedge CSI_CLOCK_CLK, posedge CSI_CLOCK_RESET_N) begin
100
101     if ((ps == Calc) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
102         Size_Counter = Size_Counter + 1;
103         Sum = Sum + AVM_AVALONMASTER_READDATA;
104         read_address = read_address + 4;
105     end
106
107     if ((ps == Ready) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
108         Num_Counter = Num_Counter + 1;
109         write_address = write_address + 4;
110     end
111
112     if (CSI_CLOCK_RESET_N == 0)
113         ps <= 0;
114     else begin
115         ps <= ns;
116     end
117 end
118
119
120 assign writedata = (Sum / Size);
121 assign co_Size_Counter = (Size_Counter >= Size) ? 1 : 0;
122 assign co_Num_Counter = (Num_Counter >= Num) ? 1 : 0;
```

پس از هر بار محاسبه مجموع دامنه های یک بازه، به حافظه SRAM، مقدار (sum / len_period) که بیانگر همان میانگین یک بازه هست، را می نویسیم ($write = 1$). اگر تعداد بازه هایی که شماردیم کمتر از تعداد بازه ها باشد (به عبارتی $co_Num_Counter$ صفر باشد) دوباره به $state\ init$ می رویم تا میانگین دامنه بازه های بعدی را محاسبه کنیم. در غیر این صورت ($co_Num_Counter = 1$)، کار مدار به اتمام رسیده و سیگنال $done$ باید فعال شود، پس از جایی که در این لحظه مقدار $co_Num_Counter$ ، ۱ است، با برابر قرار دادن ($done = co_Num_Counter$) سیگنال $done$ فعال می شود.

برای نوشتن میانگین ها به حافظه، کافیت آدرس نوشتن ($write_address$) را ۴ واحد اضافه کنیم تا عملاً ۳۲ بیت در حافظه جلو رفته و به این ترتیب مقداری که می نویسیم به خانه ی بعدی حافظه نوشته شود.

```

Ln#
85 reg[1 : 0] ns, ps;
86 parameter[1 : 0] Idle = 2'b00, Init = 2'b01, Calc = 2'b10, Ready = 2'b11;
87
88 always @(Size_Counter, Num_Counter, START, ps) begin
89     (read, write) = 2'b0;
90     case(ps)
91         Idle: begin ns = START ? Init : Idle; Num_Counter = 0; read_address = Right_Addr; write_address = Out_Addr; end
92         Init: begin ns = Calc; Sum = 0; Size_Counter = 0; end
93         Calc: begin ns = co_Size_Counter ? Ready : Calc; read = 1; address = read_address; end
94         Ready: begin ns = co_Num_Counter ? Idle : Init; write = 1; address = write_address; end
95         default: begin ns = Idle; end
96     endcase
97 end
98
99 always @(posedge CSI_CLOCK_CLK, posedge CSI_CLOCK_RESET_N) begin
100
101     if((ps == Calc) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
102         Size_Counter = Size_Counter + 1;
103         Sum = Sum + AVM_AVALONMASTER_READDATA;
104         read_address = read_address + 4;
105     end
106
107     if((ps == Ready) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
108         Num_Counter = Num_Counter + 1;
109         write_address = write_address + 4;
110     end
111
112     if(CSI_CLOCK_RESET_N == 0)
113         ps <= 0;
114     else begin
115         ps <= ns;
116     end
117
118 end
119
120 assign writedata = (Sum / Size);
121 assign co_Size_Counter = (Size_Counter >= Size) ? 1 : 0;
122 assign co_Num_Counter = (Num_Counter >= Num) ? 1 : 0;

```

```

73 always @(posedge CSI_CLOCK_CLK)
74 begin
75     if(CSI_CLOCK_RESET_N == 0)
76     begin
77         done <= 0;
78     end
79     else
80     begin
81         done <= co_Num_Counter;
82     end
83 end

```

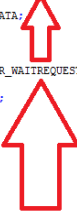
لازم به ذکر است چون فرایند خواندن و نوشتن به حافظه ممکن است بیش از یک کلاک طول بکشد،

هرگاه سیگنال AVM_AVALONMASTER_WAITREQUEST غیر فعال بود یا به عبارتی

منتظر درخواستی نبودیم میتوانیم از حافظه بخوانیم یا به حافظه بنویسیم. به همین دلیل در

شرط if، شرط (~AVM_AVALONMASTER_WAITREQUEST) دیده می شود.

```
98
99
100 always @(posedge CSI_CLOCK_CLK, posedge CSI_CLOCK_RESET_N) begin
101
102     if((ps == Calc) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
103         Size_Counter = Size_Counter + 1;
104         Sum = Sum + AVM_AVALONMASTER_READDATA;
105         read_address = read_address + 4;
106     end
107
108     if((ps == Ready) & (~AVM_AVALONMASTER_WAITREQUEST)) begin
109         Num_Counter = Num_Counter + 1;
110         write_address = write_address + 4;
111     end
112
113     if(CSI_CLOCK_RESET_N == 0)
114         ps <= 0;
115     else begin
116         ps <= ns;
117     end
118
119 end
```



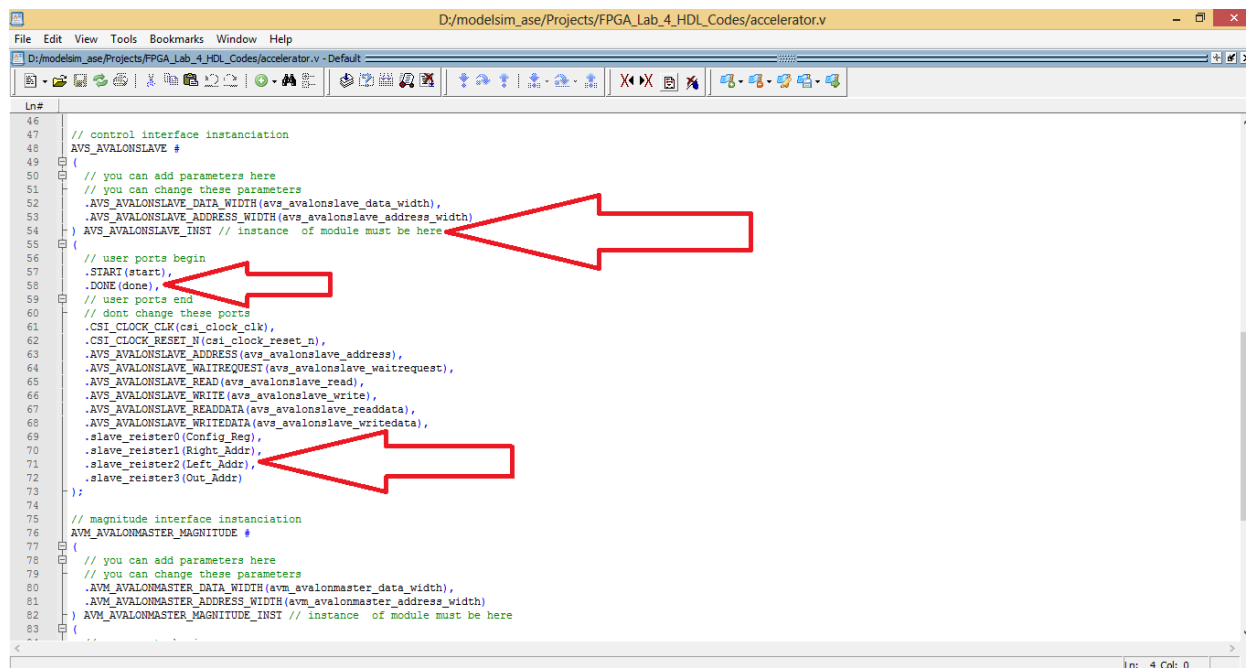
در آخر در accelerator.v یک نمونه از Slave و یک نمونه از Master، تعریف می کنیم و خروجی و ورودی های مرتبط با هر کدامشان را مقداردهی می کنیم.

لازم به ذکر است که:

سیگنال start خروجی Slave و ورودی Master است.

سیگنال done خروجی Master و ورودی Slave است.

۴ رجیستر موجود در Slave، خروجی های Slave هستند که به عنوان ورودی به Master داده میشوند تا مدار محاسبه کننده میانگین دامنه، به مقادیر رجیستر های Slave، به آدرس ها و بیت start دسترسی داشته باشد.



```
Ln#
46
47 // control interface instantiation
48 AVS_AVALONSLAVE #
49 (
50 // you can add parameters here
51 // you can change these parameters
52 .AVS_AVALONSLAVE_DATA_WIDTH(avs_avalonslave_data_width),
53 .AVS_AVALONSLAVE_ADDRESS_WIDTH(avs_avalonslave_address_width)
54 ) AVS_AVALONSLAVE_INST // instance of module must be here
55 (
56 // user ports begin
57 .START(start),
58 .DONE(done),
59 // user ports end
60 // dont change these ports
61 .CSI_CLOCK_CLK(csi_clock_clk),
62 .CSI_CLOCK_RESET_N(csi_clock_reset_n),
63 .AVS_AVALONSLAVE_ADDRESS(avs_avalonslave_address),
64 .AVS_AVALONSLAVE_WAITREQUEST(avs_avalonslave_waitrequest),
65 .AVS_AVALONSLAVE_READ(avs_avalonslave_read),
66 .AVS_AVALONSLAVE_WRITE(avs_avalonslave_write),
67 .AVS_AVALONSLAVE_READDATA(avs_avalonslave_readdata),
68 .AVS_AVALONSLAVE_WRITEDATA(avs_avalonslave_writedata),
69 .slave_reister0(Config_Reg),
70 .slave_reister1(Right_Addr),
71 .slave_reister2(Left_Addr),
72 .slave_reister3(Out_Addr)
73 );
74
75 // magnitude interface instantiation
76 AVM_AVALONMASTER_MAGNITUDE #
77 (
78 // you can add parameters here
79 // you can change these parameters
80 .AVM_AVALONMASTER_DATA_WIDTH(avm_avalonmaster_data_width),
81 .AVM_AVALONMASTER_ADDRESS_WIDTH(avm_avalonmaster_address_width)
82 ) AVM_AVALONMASTER_MAGNITUDE_INST // instance of module must be here
83 (
```

```
D:/modelsim_ase/Projects/FPGA_Lab_4_HDL_Codes/accelerator.v
File Edit View Tools Bookmarks Window Help
D:/modelsim_ase/Projects/FPGA_Lab_4_HDL_Codes/accelerator.v - Default

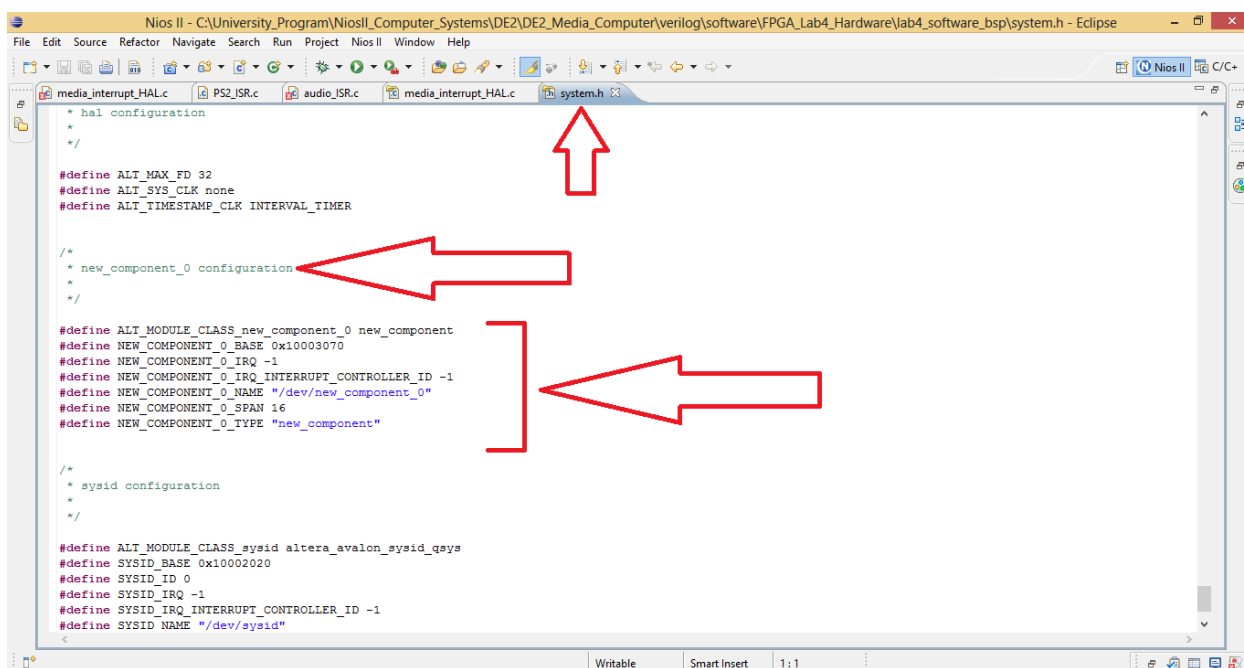
Ln#
69 .slave_reister0(Config_Reg),
70 .slave_reister1(Right_Addr),
71 .slave_reister2(Left_Addr),
72 .slave_reister3(Out_Addr),
73 );
74
75 // magnitude interface instantiation
76 AVM_AVALONMASTER_MAGNITUDE #
77 (
78 // you can add parameters here
79 // you can change these parameters
80 .AVM_AVALONMASTER_DATA_WIDTH(avm_avalonmaster_data_width),
81 .AVM_AVALONMASTER_ADDRESS_WIDTH(avm_avalonmaster_address_width)
82 AVM_AVALONMASTER_MAGNITUDE_INST // Instance of module must be here
83 )
84 // user ports begin
85
86 .slave_reister0(Config_Reg),
87 .slave_reister1(Right_Addr),
88 .slave_reister2(Left_Addr),
89 .slave_reister3(Out_Addr),
90
91 .START(start),
92 .DONE(done),
93 // user ports end
94 // dont change these ports
95 .CSI_CLOCK_CLK(csi_clock_clk),
96 .CSI_CLOCK_RESET_N(csi_clock_reset_n),
97 .AVM_AVALONMASTER_ADDRESS(avm_avalonmaster_address),
98 .AVM_AVALONMASTER_WAITREQUEST(avm_avalonmaster_waitrequest),
99 .AVM_AVALONMASTER_READ(avm_avalonmaster_read),
100 .AVM_AVALONMASTER_WRITE(avm_avalonmaster_write),
101 .AVM_AVALONMASTER_READDATA(avm_avalonmaster_readdata),
102 .AVM_AVALONMASTER_WRITEDATA(avm_avalonmaster_writedata)
103 );
104
105 endmodule
106
```

Ln: 4 Col: 0

اضافه کردن شتاب دهنده سخت افزاری به FPGA با استفاده از ابزار Qsys:

پس از نوشتن کد سخت افزاری مربوط به Slave، Master و مدار محاسبه کننده میانگین دامنه ها، فایل کد های سخت افزاری آنها (Verilog) را به ابزار Qsys می دهیم.

پس از generate کردن component جدید توسط Qsys، در فایل system.h مطابق تصویر زیر میبینیم که component جدید (که در واقع همان شتاب دهنده سخت افزاری ای هست که پیش تر کد وریلگش را توضیح دادیم) به FPGA اضافه شده است:



```
* hal configuration
*
*/

#define ALT_MAX_FD 32
#define ALT_SYS_CLK none
#define ALT_TIMESTAMP_CLK INTERVAL_TIMER

/*
 * new_component_0 configuration
 */

#define ALT_MODULE_CLASS_new_component_0 new_component
#define NEW_COMPONENT_0_BASE 0x10003070
#define NEW_COMPONENT_0_IRQ -1
#define NEW_COMPONENT_0_IRQ_INTERRUPT_CONTROLLER_ID -1
#define NEW_COMPONENT_0_NAME "/dev/new_component_0"
#define NEW_COMPONENT_0_SPAN 16
#define NEW_COMPONENT_0_TYPE "new_component"

/*
 * sysid configuration
 */

#define ALT_MODULE_CLASS_sysid altera_avalon_sysid_qsys
#define SYSID_BASE 0x10002020
#define SYSID_ID 0
#define SYSID_IRQ -1
#define SYSID_IRQ_INTERRUPT_CONTROLLER_ID -1
#define SYSID_NAME "/dev/sysid"
```

اضافه کردن کد HAL به منظور استفاده از شتاب دهنده سخت افزاری

پس از طراحی شتاب دهنده سخت افزاری و اضافه کردن آن به FPGA با استفاده از ابزار Qsys، نیاز به یک سری تابع داریم تا بتوانیم با شتاب دهنده سخت افزاری طراحی شده ارتباط برقرار کرده و مقادیر طول هر بازه، تعداد بازه ها، آدرس شروع بافر صدای راست، آدرس شروع بافر صدای چپ و آدرس مکان ذخیره سازی جواب را مقداردهی کنیم:

Amplitude_circute_set_size(unsigned int size)

برای اینکه طول هر بازه (Size) را مقداردهی کنیم، نیاز است تا بیت های ۱۲ تا ۳۰ اولین رجیستر یعنی Base_Addr، را برابر با size، که ورودی تابع است قرار دهیم. به این منظور محتوای رجیستر را با 32'b1000000000000000000000001111111111 AND کنیم تا بیت های غیر از ۱۲ تا ۳۰ تغییر نکند و بیت های ۱۲ تا ۳۰ برابر 0 شوند. در آخر اگر مقدار AND شده را با size که ۱۲ بیت به سمت راست شیفت یافته، OR کنیم محتوای size در بیت ۱۲ تا ۳۰ اولین رجیستر که Config Reg می باشد، قرار می گیرد.

```
void amplitude_circute_set_size(unsigned int size)
{
    *(Base_Addr) = ((*Base_Addr) & 0x80000FFF) | (size << 12);
}
```

Amplitude_circute_set_num(unsigned int num)

برای اینکه تعداد بازه ها (Num) را مقداردهی کنیم، نیاز است تا بیت های ۱ تا ۱۱ اولین رجیستر یعنی Base_Addr، را برابر با num، که ورودی تابع است قرار دهیم. به این منظور محتوای رجیستر را با 32'b111111111111111111111111000000000001 AND کنیم تا بیت های

غیر از ۱ تا ۱۱ تغییر نکند و بیت های ۱ تا ۱۱ برابر ۰ شوند. در آخر اگر مقدار AND شده را با num که ۱ بیت به سمت راست شیفت یافته، OR کنیم محتوای num در بیت ۱ تا ۱۱ اولین رجیستر که Config Reg می باشد، قرار می گیرد.

```
void amplitude_circute_set_num(unsigned int num)
{
    *(Base_Addr) = ((*Base_Addr) & 0xFFFFF001) | (num << 1);
}
```

برای مشخص کردن آدرس ها کافیسیت محتویات Base_Addr + 1، Base_Addr + 2 و Base_Addr + 3 را به ترتیب برابر با آدرس شروع بافر سمت راست، بافر سمت چپ و آدرس مکان ذخیره سازی جواب قرار دهیم:

```
void amplitude_circute_set_rbuff_addr(int *rbuff_addr)
{
    *(Base_Addr + 1) = rbuff_addr;
}
void amplitude_circute_set_lbuff_addr(int *lbuff_addr)
{
    *(Base_Addr + 2) = lbuff_addr;
}
void amplitude_circute_set_dest_addr(int *dest_addr)
{
    *(Base_Addr + 3) = dest_addr;
}
```

Amplitude_circute_start()

برای ۱ کردن بیت start در اولین رجیستر Slave، کافیسیت محتویات اولین رجیستر یعنی Base_Addr را با 0x00000001، OR کنیم تا اولین بیت آن، ۱ شده و مدار شروع به کار کند.

int Amplitude_circute_get_status()

برای پی بردن به اتمام کار مدار لازم است ۳۲ امین بیت اولین رجیستر یعنی Base_Addr، را بخوانیم پس برای این کار کافیسیت محتویات این رجیستر را به اندازه ۳۱ بیت به سمت راست شیفت دهیم تا ۳۲ امین بیت آن در اولین بیت ظاهر شود.

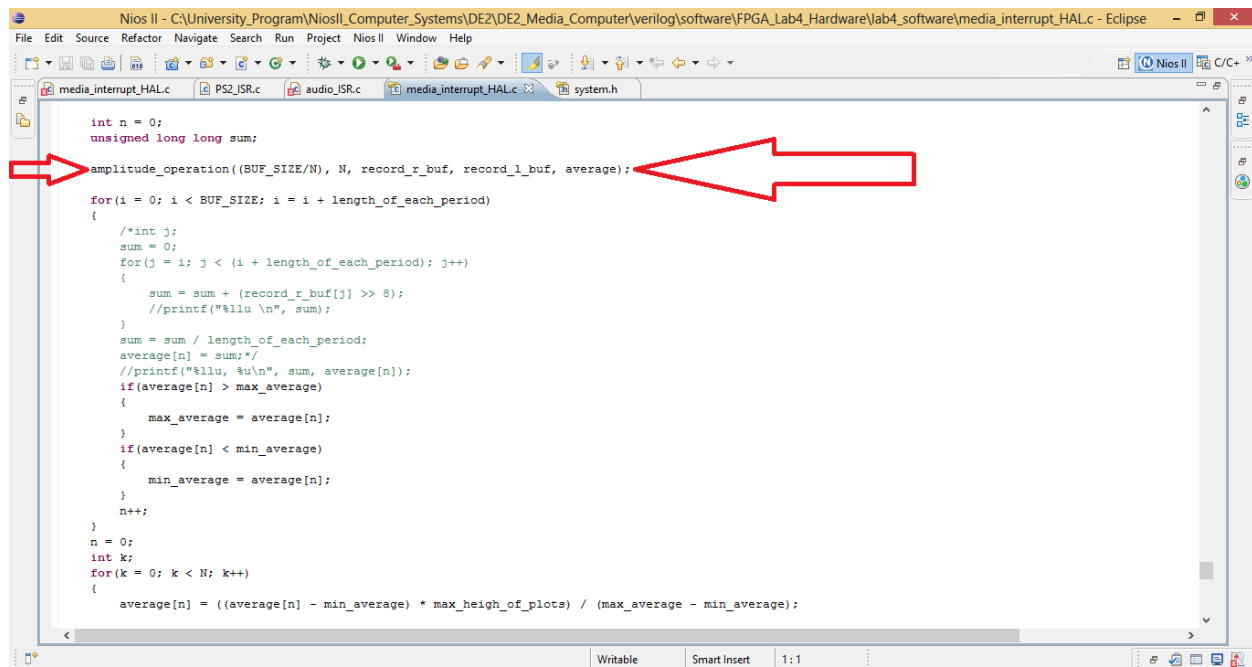
در آخر در تابع `amplitude_operation` ابتدا مقادیر مربوط به طول هر بازه، تعداد بازه ها و آدرس ها را مقداردهی کرده و در یک حلقه تا زمانی که ۳۲ امین بیت اولین رجیستر (که نشان دهنده اتمام کار مدار است) ۱ شود و کار مدار تمام شود، صبر می کنیم:

```

{
    *(Base_Addr + 2) = lbuff_addr;
}
void amplitude_circuite_set_dest_addr(int *dest_addr)
{
    *(Base_Addr + 3) = dest_addr;
}
void amplitude_circuite_start()
{
    *(Base_Addr) = (*(Base_Addr)) | 0x00000001;
}
int amplitude_circuite_get_status()
{
    return (*(Base_Addr)) >> 31;
}
void amplitude_operation(int size, int num, int rbuff_addr, int lbuff_addr, int dest_addr)
{
    amplitude_circuite_stop();
    amplitude_circuite_set_size(size);
    // also for your debugging make int amplitude_circuite_get_size(); (optional)
    amplitude_circuite_set_num(num);
    // also for your debugging make int amplitude_circuite_get_num(); (optional)
    amplitude_circuite_set_rbuff_addr(rbuff_addr);
    // also for your debugging make int amplitude_circuite_get_lbuff_addr(); (optional)
    amplitude_circuite_set_lbuff_addr(lbuff_addr);
    // also for your debugging make int amplitude_circuite_get_rbuff_addr(); (optional)
    amplitude_circuite_set_dest_addr(dest_addr);
    // also for your debugging make int amplitude_circuite_get_dest_addr(); (optional)
    amplitude_circuite_start();
    while(amplitude_circuite_get_status() == 0);
    return;
}

```

در آخر با فراخوانی تابع amplitude_operation، میانگین داده های ضبط شده محاسبه می شود:

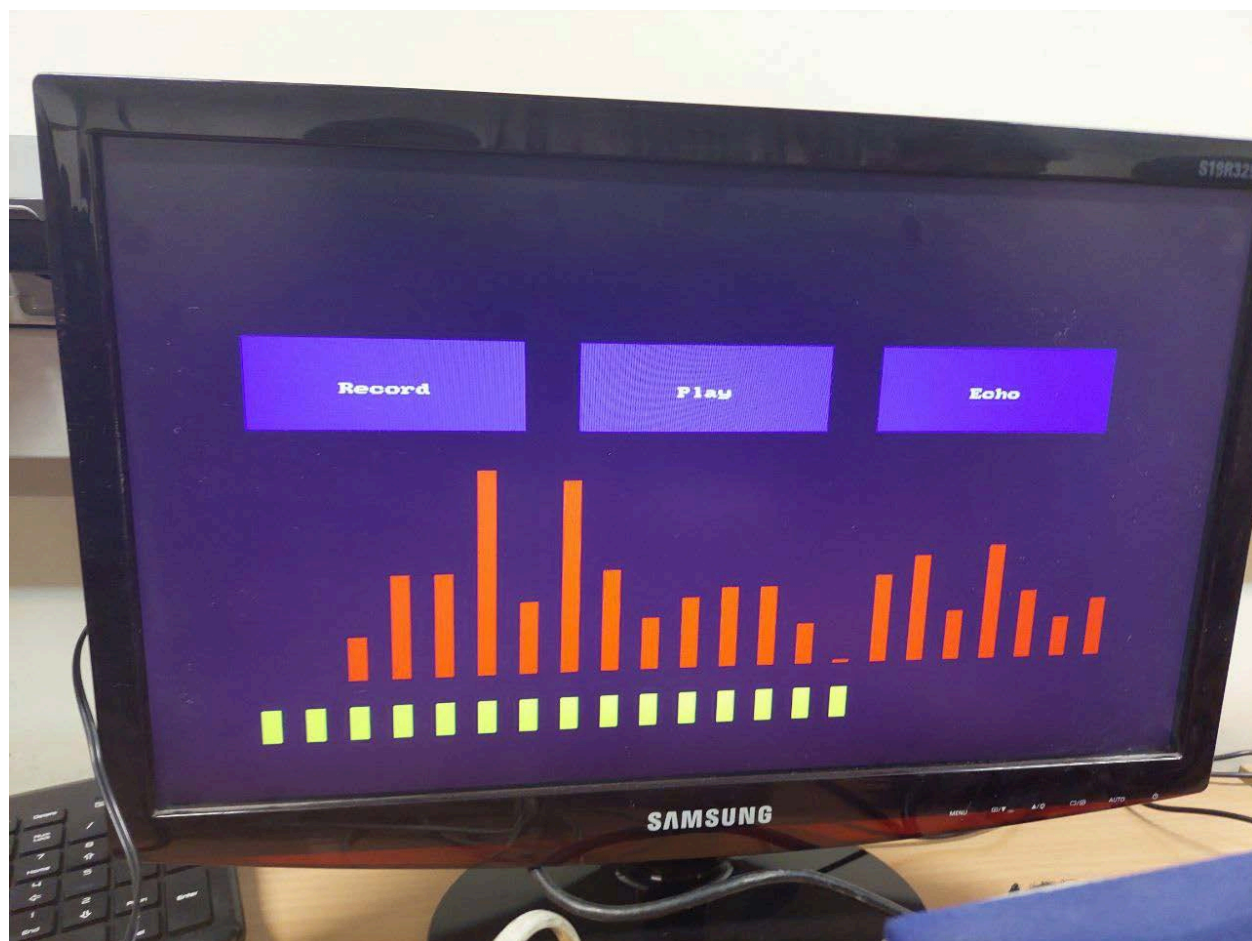


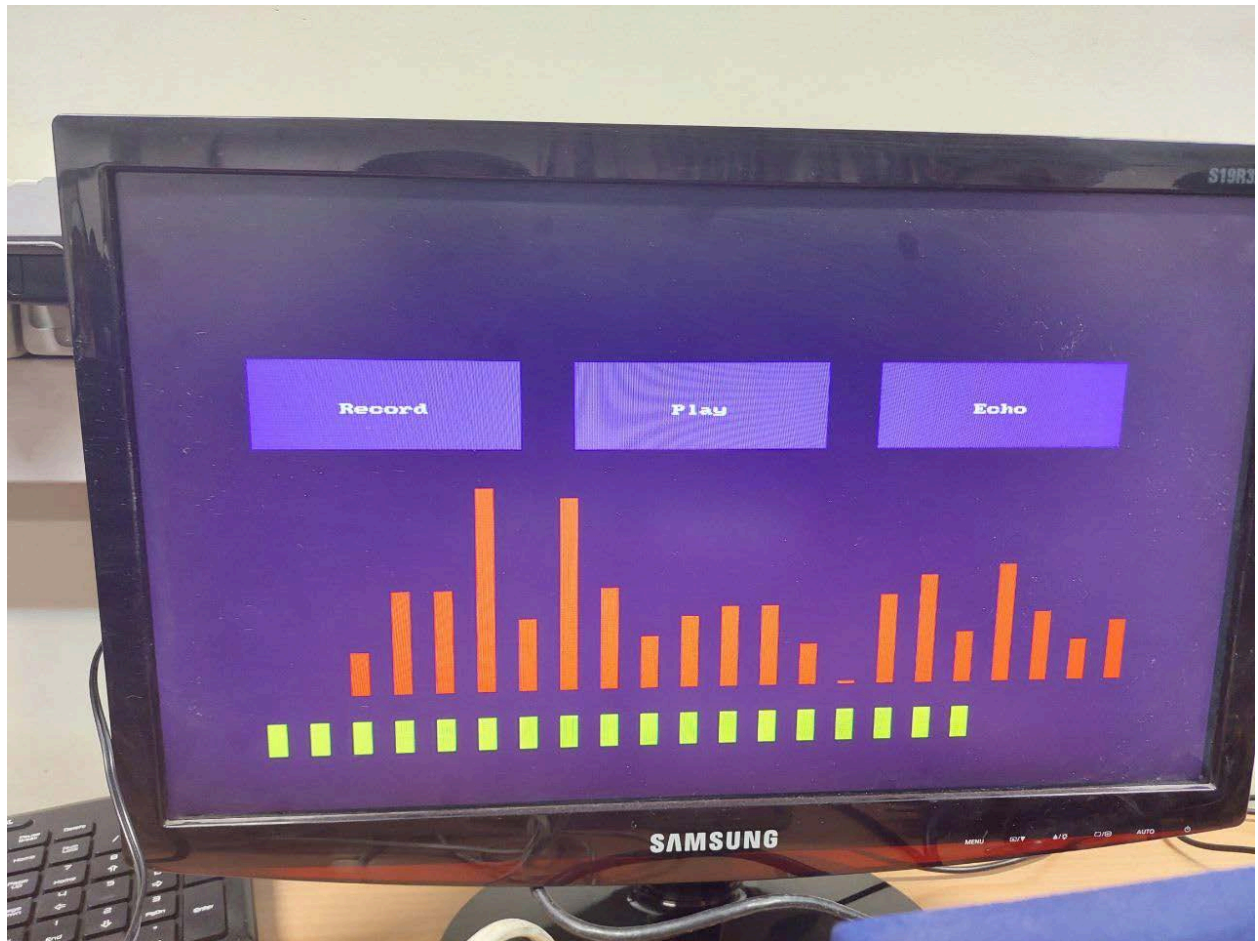
```
int n = 0;
unsigned long long sum;

amplitude_operation((BUF_SIZE/N), N, record_r_buf, record_i_buf, average);

for(i = 0; i < BUF_SIZE; i = i + length_of_each_period)
{
    /*int j;
    sum = 0;
    for(j = i; j < (i + length_of_each_period); j++)
    {
        sum = sum + (record_r_buf[j] >> 8);
        //printf("%llu \n", sum);
    }
    sum = sum / length_of_each_period;
    average[n] = sum;*/
    //printf("%llu, %u\n", sum, average[n]);
    if(average[n] > max_average)
    {
        max_average = average[n];
    }
    if(average[n] < min_average)
    {
        min_average = average[n];
    }
    n++;
}
n = 0;
int k;
for(k = 0; k < N; k++)
{
    average[n] = ((average[n] - min_average) * max_heigh_of_plots) / (max_average - min_average);
```

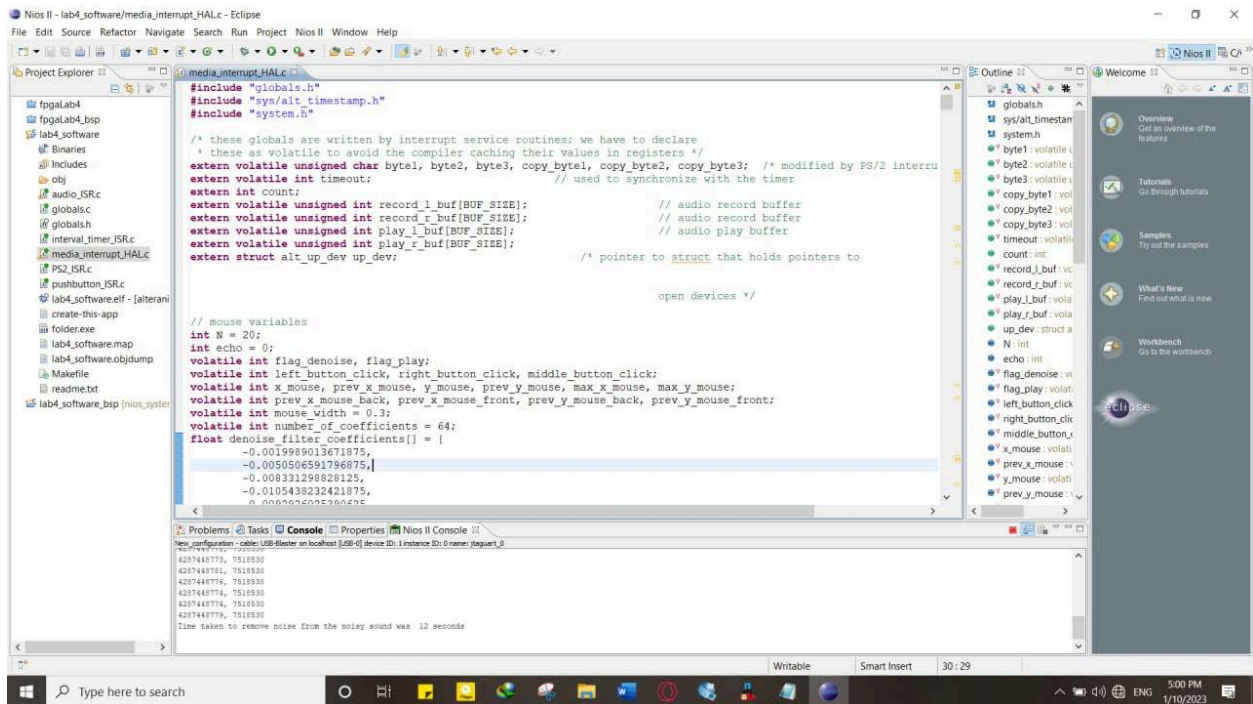
خروجی شتاب دهنده سخت افزاری بر روی برد DE2 - FPGA







زمان اجرا پیاده سازی نرم افزاری



زمان اجرا پیاده سازی با شتاب دهنده سخت افزاری

