

به نام او



دانشگاه صنعتی شریف

دانشکده مهندسی برق

پروژه درس ASIC/FPGA

استاد: دکتر حاج صادقی

سارا محمدی ۹۵۱۰۲۲۵۴

زینب شریفی ۹۵۱۰۱۷۵۲

تاریخ تحویل: ۱۱ تیر ۹۸

نیمسال دوم ۹۸-۱۳۹۷

## I. Voice Recorder:

### ➤ Step 1: Recorder without filter

- Wecontrol: به منظور تعیین مقدار write enable در ماژول mybram این متغیر در زمان playback و زمانی که به بیت هشتم در مود record نرسیده ایم، صفر قرار داده میشود تا نوشتنی روی مموری صورت نگیرد.
- Flag: برای تشخیص اتمام record و بازگشت به خانه صفر در مموری به هنگام record مجدد یک flag نیاز است تا به کمک آن در بازگشت دوباره به record آدرس خانه حافظه صفر قرار داده شود.
- recordCounter/playbackCounter: در مود record از هر ۸ دیتای دریافت شده از ac97 یکی داخل مموری ذخیره میشود. در مود playback هم از هر آدرس مموری ۸ بار خوانده میشود و به ac97 داده میشود.
- addressRecord/addressPlayback: پوینتر هایی هستند که به خانه ای در مموری به هنگام خواندن یا نوشتن اشاره میکنند. پوینتر هر مود در مود دیگری صفر میشود.
- endAddress: در هربار record کردن آدرس خانه آخری از مموری که دیتا در آن قرار میگیرد در این متغیر ذخیره میشود تا در مود playback از خانه صفر تا این خانه خوانده شود و دوباره به اول مموری برای خواندن مجدد باز گردیم.
- Playback Mode: با شروع برنامه ابتدا در این مود قرار میگیریم در صورتی که addressRecord صفر باشد (هنوز چیزی ضبط نشده) در خروجی سیگنال tone قرار میگیرد. در غیر اینصورت خروجی ضبط شده از مموری خوانده شده و از هر آدرس ۸ بار به ac97 فرستاده میشود. (صدای ضبط شده مرتباً تکرار میشود).
- Record Mode: در این مود از هر ۸ دیتای دریافتی یکی در مموری نوشته میشود. در این مرحله فیلتری وجود ندارد. نتیجتاً در مود playback آنچه در حافظه ذخیره شده شنیده میشود.

### ➤ Step 3: Recorder with filter

- فیلتر پایین گذر اعمال شده به طول ۳۱ در دو مود record و playback به ازای هر ۱ شدن ready یک ورودی جدید میگیرد و در بافر sample خود ذخیره میکند. در اینجا از circular buffer استفاده شده که با offset روی آن حرکت میکنیم. برای جمع زدن ۳۲ نمونه از ورودی پس از اعمال فیلتر از تغییر index از ۰ تا ۳۰ استفاده شده است. این تغییر بین ۱ شدن ۲ ready انجام و ضرب نمونه ها در ضرایب فیلتر محاسبه و جمع زده میشوند.
- در حالتی که فیلتر اعمال شود ورودی دریافتی از ac97 ابتدا به فیلتر داده میشود سپس از این ورودی فیلتر شده از هر ۸ نمونه یکی در مموری ذخیره میشود. در مود playback نیز ابتدا به ازای هر خانه حافظه ۷ تا صفر بین داده ها قرار میدهم تا طولی مشابه قبل داشته باشیم سپس این سیگنال zero-extend شده را به فیلتر میدهم و خروجی فیلتر به ورودی ac97 برای بخش میرود. در نهایت کد این دو بخش به صورت زیر است:

```
module recorder(
    input wire clock,           // 100mhz system clock
    input wire reset,           // 1 to reset to initial state
    input wire playback,        // 1 for playback, 0 for record
    input wire ready,           // 1 when AC97 data is available
    input wire filter,          // 1 when using low-pass filter
    input wire [7:0] from_ac97_data, // 8-bit PCM data from mic
    output reg [7:0] to_ac97_data // 8-bit PCM data to headphone
);
    // test: playback 750hz tone, or loopback using incoming data
    wire [19:0] tone;
    tone750hz xxx(.clock(clock),.ready(ready),.pcm_data(tone));

    wire we;
    reg wecontrol=1'b0;
    reg [2:0] recordCounter;
    reg [15:0] addressRecord=16'b0;
    reg [15:0] addressPlayback;
    wire [7:0] mem_in;
    wire [7:0] mem_out;
    reg flag=1'b0;
    reg flagPlayBack;
    wire [15:0] a;
    reg [15:0] endAddress=16'b0;
    reg [2:0] playbackCounter;

    wire FIRreset;
    //reg [4:0] FIRcounter=5'b0;
    wire filterReady;
    wire [7:0] FIRinput;
    wire [17:0] FIRoutput;

    fir31 fil(clock,FIRreset,ready,FIRinput,FIRoutput);

    mybram #(.LOGSIZE(16),.WIDTH(8)) example(.addr(a),.clk(clock),.we(we),.din(mem_in),.dout(mem_out))
    assign a[15:0]=(!playback)?addressRecord:addressPlayback;

    always @(posedge clock)begin
        if(ready)begin
            if(playback)begin
                wecontrol<=1'b0;
                flag<=1'b0;
                recordCounter<=3'b0;
                to_ac97_data<=(addressRecord!=16'b0)?(filter?FIRoutput[14:7]:mem_out):tone[19:12];
                playbackCounter<=playbackCounter+3'b1;
                if(addressPlayback<addressRecord)begin
                    addressPlayback<=(playbackCounter==3'b111)?(addressPlayback+16'b1):addressPlayback;
                end else begin
                    addressPlayback<=16'b0;
                end
            end
        end
    end
```

```

end else begin
    playbackCounter<=3'b0;
    addressPlayback<=16'b0;
    addressRecord<=(flag)?addressRecord:16'b0;
    to_ac97_data<=mem_in;
    //-----
    //to_ac97_data<=filter ? filterOutputReady ? FIRoutput[17:10]:0: from_ac97_data;
    ..
    flag<=1'b1;
    recordCounter<=recordCounter+3'b1;
    if(recordCounter==3'b111 && addressRecord<16'hFFFF)begin
        wecontrol<=1'b1;
        addressRecord<=addressRecord+16'b1;
    end else begin
        wecontrol<=1'b0;
    end //else

end //else

end

end

assign we=(playback)?1'b0:(ready & wecontrol);
assign mem_in=filter?FIRoutput[17:10]:from_ac97_data;
assign FIRinput=playback?(playbackCounter!=7?0:mem_out):from_ac97_data;
assign FIRreset=reset || !filter;
endmodule

module fir31(
    input wire clock,reset,ready,
    input wire signed [7:0] x,
    output reg signed [17:0] y
);

// for now just pass data through
coeffs31 coef(index,coeff);
reg [4:0] offset=5'b11111;
wire signed [9:0] coeff;
wire signed [17:0] multiple;
reg signed [7:0] sample [31:0];
reg [4:0] index=5'b0;
always @(posedge clock) begin
    if(reset)begin
        y<=18'b0;
        offset<=5'b11111;
        index<=5'b0;
    end else if(ready) begin
        offset<=(offset==5'b0)?5'b11111:offset-5'b1;
        sample[offset]<=x;
        y<=18'b0;
        index<=5'b0;
    end else if(index<5'b11111)begin
        index<=index+5'b1;
        y<=y+multiple;
    end

end

assign multiple=coeff*sample[offset-index];
endmodule

```

در نهایت مشاهده میشود با اعمال فیلتر پایین گذر صدا نرم تر شده و نویز کمتری در صدای خروجی مشاهده میشود.

## ➤ Step 2: Test low-pass filter

در این بخش خروجی مورد انتظار با خروجی گرفته شده از فیلتر طراحی شده مقایسه شدند.(فایل دو خروجی به همراه کد ها تحویل داده شده.) نتایج برابری میکنند.

2	47269
3	42567
4	37809
5	32526
6	26210
7	18568
8	9690
9	69
10	-9553
11	-18432
12	-26077
13	-32398
14	-37691
15	-42464
16	-47184
17	-52040
18	-56828
19	-61006
20	-63891
21	-64924
22	-63891
23	-61006
24	-56828
25	-52040
26	-47184
27	-42464
28	-37691
29	-32398
30	-26077
31	-18432
32	-9553
33	69
34	9690
35	18568
36	26210
37	32526

نمونه گرفته شده از فیلتر طراحی شده:

1	47269
2	42567
3	37809
4	32526
5	26210
6	18568
7	9690
8	69
9	-9553
10	-18432
11	-26077
12	-32398
13	-37691
14	-42464
15	-47184
16	-52040
17	-56828
18	-61006
19	-63891
20	-64924
21	-63891
22	-61006
23	-56828
24	-52040
25	-47184
26	-42464
27	-37691
28	-32398
29	-26077
30	-18432
31	-9553
32	69
33	9690
34	18568
35	26210
36	32526
37	37808

نمونه مورد انتظار :

## II. Pitch Shifter:

- ایده اصلی استفاده شده در این بخش استفاده از دو بافر به طور همزمان در هر مود switch هاست به گونه ای که در یکی از آن ها از ac97 بخوانیم و تغییرات لازم را انجام دهیم درحالی که از رجیستر دیگر به ac97 ورودی میدهیم.
- به این منظور ۳ مدل counter تعریف شده که بسته به مود switch ها از آن ها استفاده بهینه میشود. counter ۲۵۶ تایی همواره برای دادن داده به ac97 استفاده میشود چون فرض گرفته شده که داده ها در بازه های ۲۵۶ تایی process شوند. Counter ۱۲۸ تایی برای نصف کردن فرکانس و counter ۶۴ تایی برای به ۴ تقسیم کردن فرکانس استفاده شده.
- در هر مود وقتی برای بار اول وارد میشویم دیتایی برای فرستادن به ac97 در همان لحظه وجود ندارد بنابراین فقط از ورودی دیتا گرفته شده و تغییرات را اعمال میکنیم. بعد از ۲۵۶ تای اول flag مربوط به این بخش (first-time) ۱ شده و وارد حالت همزمانی خواندن و نوشتن میشویم.
- برای جابجایی بین مود های مختلف از یک case استفاده شده که هر case در ابتدا flag بخش های دیگر را صفر میکند.
- در مود هایی که قرار است فرکانس را افزایش دهیم با توجه به عدد داده شده دیتا ۲۵۶ تایی را عینا تکرار میکنیم سپس یکی در میان (برای ۲) یا ۴ تا یکی (برای ۴) نمونه برداری میکنیم و به خروجی میدهیم.
- در مودهایی که قرار است فرکانس را کاهش دهیم با توجه به عدد داده شده نصف یا یک چهارم دیتا را نگه داشته و سپس در خروجی در بین آن ها صفر قرار میدهیم (یکی یا سه تا)
- در ماژول recorder ماژول pitchshifter صدا زده میشود. در این ماژول یک reset هم برای pitchshifter تعریف شده. با هر بار وارد شدن به مود record ماژول pitchshifter یک بار reset میشود و همه counter ها برابر صفر قرار میگیرند.
- مشاهده شد که با افزایش فرکانس صدا زیر و با کاهش آن صدا بم میشود. البته برای ۴ برابر یا یک چهارم کردن فرکانس نویز بیشتر میشود.

کد این بخش به صورت زیر است :

```
566 module pitchShifter(
567     input wire ready,
568     input wire [7:0] x,
569     input wire reset,
570     input wire [2:0] switch,
571     input wire clk,
572     output reg [7:0] y
573 );
574
575 //-----
576 //wire [2:0] switch;
577 //assign switch=3'b010;
578 //-----
579 reg [7:0] sample1 [1023:0];
580 reg [7:0] sample2 [1023:0];
581 reg [7:0] sample1_d [255:0];
582 reg [7:0] sample2_d [255:0];
583 reg whichSample=1'b0;
584 reg firstTime1=1'b0;
585 reg firstTime2=1'b0;
586 reg firstTime3=1'b0;
587 reg firstTime4=1'b0;
588 reg [7:0] counter256_1=8'b0;
589 reg [7:0] counter256_2=8'b0;
590 reg [6:0] counter128_1=7'b0;
591 reg [6:0] counter128_2=7'b0;
592 reg [5:0] counter64_1=6'b0;
593 reg [5:0] counter64_2=6'b0;
594 reg [2:0] changeMode;
595
596 always @(posedge clk)begin
597     if(reset)begin
598         firstTime1<=1'b0;
599         firstTime2<=1'b0;
600         firstTime3<=1'b0;
601         firstTime4<=1'b0;
602         counter256_1<=8'b0;
603         counter256_2<=8'b0;
604         counter128_1<=7'b0;
605         counter128_2<=7'b0;
606         counter64_1<=6'b0;
607         counter64_2<=6'b0;
608         changeMode<=switch;
609     end else if (ready)begin
610         case (switch)
611             3'b000:begin
612                 y<=x;
613                 firstTime1<=1'b0;
614                 firstTime2<=1'b0;
615                 firstTime3<=1'b0;
616                 firstTime4<=1'b0;
617                 counter256_1<=8'b0;
618                 counter256_2<=8'b0;
619                 counter128_1<=7'b0;
620                 counter128_2<=7'b0;
621                 counter64_1<=6'b0;
622                 counter64_2<=6'b0;
623             end
624             3'b001:begin //2-increase
625                 firstTime3<=1'b0;
626                 firstTime2<=1'b0;
627                 firstTime4<=1'b0;
628             end
629             if (changeMode!=3'b001)begin
630                 counter256_1<=8'b0;
631                 counter256_2<=8'b0;
632                 counter128_1<=7'b0;
633                 counter128_2<=7'b0;
634                 counter64_1<=6'b0;
635                 counter64_2<=6'b0;
636                 changeMode<=3'b001;
637             end else if (firstTime1==1'b0)begin
638                 sample1[counter256_1]<=x;
639                 sample1[counter256_1+9'd256]<=x;
640                 counter256_1<=counter256_1+8'b1;
641                 firstTime1<=(counter256_1==8'd255)?1'b1:1'b0;
642             end else if (~whichSample) begin
643                 counter256_1<=8'b0;
644                 y<=sample1[2'd2*counter256_2];
645                 counter256_2<=counter256_2+8'b1;
646                 sample2[counter256_2]<=x;
647                 sample2[counter256_2+9'd256]<=x;
648                 whichSample<=(counter256_2==8'd255)?1'b1:1'b0;
649             end else begin
650                 counter256_2<=8'b0;
651                 y<=sample2[2'd2*counter256_1];
652                 counter256_1<=counter256_1+8'b1;
653                 sample1[counter256_1]<=x;
654                 sample1[counter256_1+9'd256]<=x;
655                 whichSample<=(counter256_1==8'd255)?1'b0:1'b1;
656             end
657         end
658     end //end 3'b001
659     3'b010:begin //4 increase
660         firstTime3<=1'b0;
661         firstTime1<=1'b0;
662         firstTime4<=1'b0;
663         if (changeMode!=3'b010)begin
664             counter256_1<=8'b0;
665             counter256_2<=8'b0;
666             counter128_1<=7'b0;
667             counter128_2<=7'b0;
668             counter64_1<=6'b0;
669             counter64_2<=6'b0;
670             changeMode<=3'b010;
671         end else if (firstTime2==1'b0)begin
672             sample1[counter256_1]<=x;
673             sample1[counter256_1+9'd256]<=x;
674             sample1[counter256_1+10'd512]<=x;
675             sample1[counter256_1+10'd768]<=x;
676             counter256_1<=counter256_1+8'b1;
677             firstTime2<=(counter256_1==8'd255)?1'b1:1'b0;
678         end else if (~whichSample) begin
679             counter256_1<=8'b0;
680             y<=sample1[3'd4*counter256_2];
681             counter256_2<=counter256_2+8'b1;
682             sample2[counter256_2]<=x;
683             sample2[counter256_2+9'd256]<=x;
684             sample2[counter256_2+10'd512]<=x;
685             sample2[counter256_2+10'd768]<=x;
686             whichSample<=(counter256_2==8'd255)?1'b1:1'b0;
687         end else begin
688             counter256_2<=8'b0;
689             y<=sample2[3'd4*counter256_1];
690             counter256_1<=counter256_1+8'b1;
691             sample1[counter256_1]<=x;
692             sample1[counter256_1+9'd256]<=x;
693             sample1[counter256_1+10'd512]<=x;
694             sample1[counter256_1+10'd768]<=x;
695             whichSample<=(counter256_1==8'd255)?1'b0:1'b1;
696         end
697     end
698     3'b100:begin //2 decrease
699         firstTime1<=1'b0;
700         firstTime2<=1'b0;
701         firstTime4<=1'b0;
702         if (changeMode!=3'b100)begin
703             counter256_1<=8'b0;
704             counter256_2<=8'b0;
705             counter128_1<=7'b0;
706             counter128_2<=7'b0;
707             counter64_1<=6'b0;
708             counter64_2<=6'b0;
709             changeMode<=3'b100;
710         end else if (firstTime3==1'b0)begin
711             sample1_d[2*counter128_1]<=x;
712             sample1_d[2*counter128_1+7'b1]<=8'b0;
713             counter128_1<=counter128_1+7'b1;
714             firstTime3<=(counter128_1==8'd127)?1'b1:1'b0;
715         end else if (~whichSample) begin
716             counter128_1<=7'b0;
717             counter256_2<=8'b0;
718             y<=sample1_d[counter256_1];
719             counter256_1<=counter256_1+8'b1;
720             if (counter128_2<=7'd127)begin
721                 sample2_d[2*counter128_2]<=x;
722                 sample2_d[2*counter128_2+7'b1]<=8'b0;
723             end
724             counter128_2<=counter128_2+7'b1;
725             whichSample<=(counter256_1==8'd255)?1'b1:1'b0;
726         end else begin
727             counter128_2<=7'b0;
728             y<=sample2_d[2*counter128_2];
729             counter128_2<=counter128_2+7'b1;
730             sample2[counter128_2]<=x;
731             sample2[counter128_2+9'd256]<=x;
732             whichSample<=(counter128_2==8'd255)?1'b1:1'b0;
733         end
734     end
735 end
```

```

699 3'b100:begin //2 decrease
700     firstTime1<='b0;
701     firstTime2<='b0;
702     firstTime4<='b0;
703     if(changeMode!=3'b100)begin
704         counter256_1<=3'b0;
705         counter256_2<=3'b0;
706         counter128_1<=7'b0;
707         counter128_2<=7'b0;
708         counter64_1<=6'b0;
709         counter64_2<=6'b0;
710         changeMode<=3'b100;
711     end else if(firstTime3==1'b0)begin
712         sample1_d[2*counter128_1]<=x;
713         sample1_d[2*counter128_1+7'b1]<=8'b0;
714         counter128_1<=counter128_1+7'b1;
715         firstTime3<=(counter128_1==8'd127)?1'b1:1'b0;
716     end else if(~whichSample) begin
717         counter128_1<=7'b0;
718         counter256_2<=3'b0;
719         y<=sample1_d[counter256_1];
720         counter256_1<=counter256_1+8'b1;
721         if(counter128_2<=7'd127)begin
722             sample2_d[2*counter128_2]<=x;
723             sample2_d[2*counter128_2+7'b1]<=8'b0;
724         end
725         counter128_2<=counter128_2+7'b1;
726         whichSample<=(counter256_1==8'd255)?1'b1:1'b0;
727     end else begin
728         counter128_2<=7'b0;
729         counter256_1<=3'b0;
730         y<=sample2_d[counter256_2];
731         counter256_2<=counter256_2+8'b1;
732         if(counter128_1<=7'd127)begin
733             sample1_d[2*counter128_1]<=x;
734             sample1_d[2*counter128_1+7'b1]<=8'b0;
735         end
736         counter128_1<=counter128_1+7'b1;
737         whichSample<=(counter256_2==8'd255)?1'b0:1'b1;
738     end
739 end
740 3'b101:begin //4 decrease
741     firstTime1<='b0;
742     firstTime2<='b0;
743     firstTime3<='b0;
744     if(changeMode!=3'b101)begin
745         counter256_1<=3'b0;
746         counter256_2<=3'b0;
747         counter128_1<=7'b0;
748         counter128_2<=7'b0;
749         counter64_1<=6'b0;
750         counter64_2<=6'b0;
751         changeMode<=3'b101;
752     end else if(firstTime4==1'b0)begin
753         sample1_d[4*counter64_1]<=x;
754         sample1_d[4*counter64_1+6'b1]<=8'b0;
755         sample1_d[4*counter64_1+6'd2]<=8'b0;
756         sample1_d[4*counter64_1+6'd3]<=8'b0;
757         counter64_1<=counter64_1+6'b1;
758         firstTime4<=(counter64_1==6'd63)?1'b1:1'b0;
759     end else if(~whichSample) begin
760         counter64_1<=6'b0;
761         counter256_2<=3'b0;
762         y<=sample1_d[counter256_1];
763         counter256_1<=counter256_1+8'b1;
764         if(counter64_2<=6'd63)begin
765             sample2_d[4*counter64_2]<=x;
766             sample2_d[4*counter64_2+6'd1]<=8'b0;
767             sample2_d[4*counter64_2+6'd2]<=8'b0;
768             sample2_d[4*counter64_2+6'd3]<=8'b0;
769         end
770         counter64_2<=counter64_2+6'b1;
771         whichSample<=(counter256_1==8'd255)?1'b1:1'b0;
772     end else begin
773         counter64_2<=6'b0;
774         counter256_1<=3'b0;
775         y<=sample2_d[counter256_2];
776         counter256_2<=counter256_2+8'b1;
777         if(counter64_1<=6'd63)begin
778             sample1_d[4*counter64_1]<=x;
779             sample1_d[4*counter64_1+6'd1]<=8'b0;
780             sample1_d[4*counter64_1+6'd2]<=8'b0;
781             sample1_d[4*counter64_1+6'd3]<=8'b0;
782         end
783         counter64_1<=counter64_1+6'b1;
784         whichSample<=(counter256_2==8'd255)?1'b0:1'b1;
785     end
786 end
787 default:y<=x;

```

endcase

end

end

endmodule