

浙江大学

本科实验报告

实验设计报告

课程名称： 数字系统设计实验

姓 名： 周灿松

学 院： 信息与工程学院

系：

专 业： 信息工程

学 号： 3190105055

指导老师： 屈民军、唐奕

2021 年 6 月 21 日

浙江大学实验报告

专业： 信息工程
姓名： 周灿松
学号： 3190105055
日期： 2021 年 6 月 21 日
地点： 东 4-223

课程名称： 数字系统设计实验 指导老师： 屈民军、唐奕 成绩： _____
实验名称： 实验设计报告 实验类型： 设计实验 同组学生姓名： _____

一、 实验目的和要求

1. 实验目的

- (1) 掌握音符产生的方法，了解 DDS 技术的应用
- (2) 了解音频编解码的应用
- (3) 掌握系统“自顶向下”的数字系统设计方法

2. 实验要求

设计出一个能够播放四首乐曲的音乐播放器，满足如下两个要求：

- (1) 设置 play/pause_button、next_button、reset 三个按键，play/pause_button 按键实现乐曲在暂停与播放之间切换，按下 next_button 可以播放下一首乐曲；
- (2) 设置 3 个 LED 灯，LED0 显示目前的播放状态 (亮为播放，灭为暂停)，LED1 和 LED2 显示目前乐曲号

二、 实验内容和原理

1. 主控制器模块

1.1 设计说明

主控制器模块主要作用为响应用户按键信息、控制系统播放两大任务，算法流程图和书上类似，此处不表。

1.2 主控制器代码

Listing 1: 主控制器模块代码

```
1 // 主控制器
  module mcu (
3     input clk , reset , play_pause , next , song_done ,
      output reg play, reset_play,
```

```
5     output [1:0] song
    );
7     reg NextSong;

9     //状态编码
    parameter RESET = 0 , PAUSE = 1 , PLAY = 2 , NEXT = 3;
11    reg [1:0] state , nextstate;

13    //时序逻辑，控制状态切换
    always @(posedge clk) begin
15        if(reset) state = RESET;
        else state = nextstate;
17    end

19

    //组合部分：控制次态以及控制器部分输出
21    always @(*) begin
        //初始状态
23        play = 0 ; NextSong = 0 ; reset_play = 1;
        case(state)
25            RESET:begin
                play = 0 ; NextSong = 0 ; reset_play = 1;
27                nextstate = PAUSE;
            end
29            PAUSE:begin
                play = 0 ; NextSong = 0 ; reset_play = 0;
31                if(play_pause)begin
                    nextstate = PLAY;
33                end
                else if(next)begin
35                    nextstate = NEXT;
                end
37                else nextstate = PAUSE;
            end
39            PLAY:begin
                play = 1 ; NextSong = 0 ; reset_play = 0;
41                if(play_pause) nextstate = PAUSE;
                else if(next) nextstate = NEXT;
43                else if(song_done) nextstate = RESET;
                else nextstate = PLAY;
45            end
            NEXT:begin
47                play = 0 ; NextSong = 1 ; reset_play = 1;
```

```

        nextstate = PLAY;
49         end
        endcase
51     end

53     //实例化两位二进制计数器得到song
        counter_n #(.n(4) , .counter_bits(2)) counter(.en(NextSong) , .r(reset
        ) , .clk(clk) , .q(song) , .co());
55
endmodule

```

2. song_reader 模块

2.1 设计说明

song_reader 模块任务如下：

- (1) 根据 mcu 模块的要求, 选择播放乐曲。
- (2) 响应 note_player 模块请求, 从 song_rom 中逐个取出音符 note, duration 送给 note_player 模块播放
- (3) 判断乐曲是否播放完毕, 若播放完毕, 则回复 mcu 模块应答信号。

我们采用了书上给出的电路结构实现了上述功能, 其中 song_rom 模块是一个只读存储器, 用于存放乐曲; 地址计数器计算播放的音符数, 它的进位输出作为; 结束判断模块利用地址计数器进位输出和从 song_rom 中取出的 duration 判断是否输出 song_done 信号。

2.2 结束判断模块设计

结束判断模块采用了老师建议的状态机方法实现：

一、地址计数器进位输出 co 作为 reset 信号, duration 单独输入, 共四个状态实现, 代码如下：

Listing 2: 判断结束模块一

```

//判断乐曲是否播放结束模块
2 module is_over (
    input [5:0] duration ,
4     input reset , clk ,
    output reg done
6 );
    parameter RESET = 0 , PAUSE = 1 , PLAY = 2 , OUT = 3;
8     reg [1:0] state=0 , nextstate;

10    always @(posedge clk) begin
        if(reset) state = RESET;
12        else state = nextstate;

```

```
14     end
15
16     always @(*) begin
17         done = 0;
18         case (state)
19             RESET:begin
20                 done = 1;
21                 nextstate = PAUSE;
22             end
23             PAUSE:begin
24                 done = 0;
25                 if(duration)begin
26                     nextstate = PLAY;
27                 end
28                 else nextstate = PAUSE;
29             end
30             PLAY:begin
31                 done = 0;
32                 if(duration) nextstate = PLAY;
33                 else nextstate = OUT;
34             end
35             OUT:begin
36                 done = 1;
37                 nextstate = PAUSE;
38             end
39         endcase
40     end
41 endmodule
```

二、听了老师的讲解之后，意识到将两个信号按“duration == 0 || co”进行输入状态机会更加简单，如是将代码简化至如下：

Listing 3: 判断结束模块二

```
//判断乐曲是否播放结束模块
2 module is_over (
3     input [5:0] duration ,
4     input reset , clk ,
5     output reg done
6 );
7     parameter RESET = 0 , PAUSE = 1 , PLAY = 2 , OUT = 3;
8     reg [1:0] state=0 , nextstate;
9
10    always @(posedge clk) begin
```

```
12     if(reset) state = RESET;
13     else state = nextstate;
14 end
15
16 always @(*) begin
17     done = 0;
18     case (state)
19         RESET:begin
20             done = 1;
21             nextstate = PAUSE;
22         end
23         PAUSE:begin
24             done = 0;
25             if(duration)begin
26                 nextstate = PLAY;
27             end
28             else nextstate = PAUSE;
29         end
30         PLAY:begin
31             done = 0;
32             if(duration) nextstate = PLAY;
33             else nextstate = OUT;
34         end
35         OUT:begin
36             done = 1;
37             nextstate = PAUSE;
38         end
39     endcase
40 end
41 endmodule
```

2.3 song_reader 模块代码

Listing 4: song_reader 模块

```
module song_reader (
2     input clk,reset,play,note_done,
3     input [1:0] song,
4     output reg new_note,
5     output song_done,
6     output [5:0] note , duration
7 );
8     // 状态编码
```

```
parameter RESET = 0 , NEW_NOTE = 1 , WAIT = 2 , NEXT_NOTE = 3;
10 reg [1:0] state , nextstate;

12 wire [4:0] lowaddr;//song_rom的低五位地址
wire judge;//歌曲结束标志一

14 //控制器时序部分
16 always @(posedge clk) begin
    if(reset) state = RESET;
18     else state = nextstate;
end

20 //控制器组合部分
22 always @(*) begin
    //默认输出
24     new_note = 0;
    case (state)
26         RESET:begin
            new_note = 0;
28             if(play) nextstate = NEW_NOTE;
            else nextstate = RESET;
30         end
        NEW_NOTE:begin
32             new_note = 1;
            nextstate = WAIT;
34         end
        WAIT:begin
36             new_note = 0;
            if(play == 0) nextstate = RESET;
38             else if(note_done) nextstate = NEXT_NOTE;
            else nextstate = WAIT;
40         end
        NEXT_NOTE:begin
42             new_note = 0;
            nextstate = NEW_NOTE;
44         end
    endcase
46 end

48 //实例化地址计数器
counter_n #(.n(32) , .counter_bits(5)) addrCounter(.clk(clk) , .r(
    reset) , .en(note_done) , .q(lowaddr) , .co(judge));
50
```

```

52      //实例化 song_rom, 取出音符
      song_rom song_rom(.clk(clk) , .dout({note,duration}) , .addr({song
          ,lowaddr}));

54      //实例化判断模块
      over_is_over(.signal((duration==0)||co) , .reset(reset) , .clk(clk
          ) , .done(song_done));

56
      endmodule

```

3. note_player 模块

3.1 设计说明

音符播放模块 note_player 是本实验的核心模块, 它主要任务包括以下几方面。

- (1) 从 song_reader 模块接收需播放的音符 note, duration
- (2) 根据 note 值找出 DDS 的相位增量 k
- (3) 以 48kHz 速率从 Sine_rom 取出正弦样品送给音频编解码器接口模块
- (4) 当一个音符播放完成, 向 song_reader 模块索取新的音符

进一步划分模块可将 note_player 划分为一下各个模块: 作为控制单元的控制器、记录音符标记 note 和 DD 模块相位增量 k 查找表关系的 FreqROM、DDS 模块、音符节拍计时器。

下面给出 DDS 模块、节拍计时器、note_player 代码

3.2 代码

Listing 5: DDS 模块

```

1  module dds (
      K , clk , reset , sampling_pulse , sample , new_sample_ready
3  );
      input [21:0] K;
5      input clk , reset , sampling_pulse;
      output [15:0] sample;
7      output new_sample_ready;

9      wire [21:0] temp ;//作为加法器输出
      wire [21:0] raw_addr; //待处理地址
11     wire [9:0] rom_addr; //ROM地址
      wire area;//区域
13     wire [15:0] raw_data;//原始数据
      wire [15:0] data;//处理后的数据

```



```
15      //实例化加法器
17      adder_n #(.n(22)) adder(.adder1(K) , .adder2(raw_addr) , .result(temp)
      , .co());

19      //实例化D型寄存器
      dffre #(.n(22)) D1(.d(temp) , .en(sampling_pulse) , .r(reset) , .clk(
      clk) , .q(raw_addr));

21

23      //处理地址
      //rom_addr应该可以不指定位数
      assign rom_addr = raw_addr[20]?((raw_addr[20:10]==1024)?1023:(~
      raw_addr[19:10]+1)):raw_addr[19:10];

25

27      //实例化D触发器得到area
      dffre #(.n(1)) D2(.d(raw_addr[21]) , .en(1) , .r(0) , .clk(clk) , .q(
      area));

29      //取出原始数据
      sine_rom rom(.clk(clk) , .addr(rom_addr) , .dout(raw_data));

31

33      //处理原始数据
      assign data = area?(~raw_data+1):raw_data;

35      //实例化D寄存器得到取样值sample
      dffre #(.n(16)) D3(.d(data) , .en(sampling_pulse) , .r(0) , .clk(clk)
      , .q(sample));

37

39      //实例化D触发器得到new_sample_ready信号
      dffre #(.n(1)) D4(.d(sampling_pulse) , .en(1) , .r(0) , .clk(clk) , .q
      (new_sample_ready));

41 endmodule
```

Listing 6: 节拍计时器

```
1 module timer (
      r , en , clk , done , n
3 );
      parameter counter_bits = 6;
5 input [counter_bits-1:0] n;
      input r , en , clk;
7 output done;
```

```
reg [counter_bits:1]q;
9 assign done = en && (q==n-1);
always @(posedge clk) begin
11     if(r) q = 0;
    else begin
13         if(en)
            begin
15                 if(q==n-1) q=0;
                    else q = 1+q;
17             end
        else begin
19             q=q;
        end
21     end
end
23 endmodule
```

Listing 7: note_player 模块

```
1 module note_player (
    input clk , reset , play_enable ,
3    input [5:0] note_to_load,duration_to_load,
    input load_new_note,sampling_pulse,beat,
5    output reg note_done,
    output [15:0] sample,
7    output sample_ready);
    //状态编码
9    parameter RESET = 0 , WAIT = 1 , DONE = 2 , LOAD = 3;
    reg [1:0] state , nextstate;
11
    reg timer_clear , load;
13    wire timer_done;
    wire [5:0] addr;
15    wire [19:0] klow;

17    always @(posedge clk) begin
        if(reset) state = RESET;
19        else state = nextstate;
    end
21

23    always @(*) begin
        //默认状态
        timer_clear = 1 ; load = 0 ; note_done = 0;
```

```
25
    case (state)
27         RESET:begin
            timer_clear = 1 ; load = 0 ; note_done = 0;
29             nextstate = WAIT;
        end
31         WAIT:begin
            timer_clear = 0 ; load = 0 ; note_done = 0;
33             if(play_enable == 0) nextstate = RESET;
            else if(timer_done) nextstate = DONE;
35             else if(load_new_note) nextstate = LOAD;
            else nextstate = WAIT;
37         end
        DONE:begin
39             timer_clear = 1 ; load = 0 ; note_done = 1;
            nextstate = WAIT;
41         end
        LOAD:begin
43             timer_clear = 1 ; load = 1 ; note_done = 0;
            nextstate = WAIT;
45         end
    endcase
47 end

49 //实例化音符节拍定时器
    timer #(.counter_bits(6)) timer1(.clk(clk) , .en(beat) , .r(
        timer_clear) , .done(timer_done) , .n(duration_to_load));
51
53 //实例一个D寄存器
    dffre #(.n(6)) D(.d(note_to_load) , .en(load) , .r(~play_enable||reset
        ) , .clk(clk) , .q(addr));
55
57 //实例化FreqROM
    frequency_rom FreqROM(.clk(clk) , .dout(klow) , .addr(addr));
59
61 //实例化DDS
    dds DDS(.K({2'b00,klow}) , .clk(clk) , .reset(~play_enable||reset) , .
        sampling_pulse(sampling_pulse) , .sample(sample) , .
        new_sample_ready(sample_ready));
61 endmodule
```

4. 同步化电路

4.1 设计说明

因为音频解码接口模块和其他模块采用了不同的时钟，我们需要将二者进行同步化

4.2 代码

Listing 8: 同步化电路

```
1 //利用两个D触发器作为同步器使用
module synch (
3     asynch_in , clk , asynch_out
);
5     input asynch_in , clk;
    output asynch_out;
7     wire Q1,Q2;
    dffre #(.n(1)) d1(.en(1) , .r(0) , .clk(clk) , .d(asynch_in) , .q(Q1))
        ;
9     dffre #(.n(1)) d2(.en(1) , .r(0) , .clk(clk) , .d(Q1) , .q(Q2));
    assign asynch_out = Q1 & (~Q2);
11 endmodule
```

三、 主要仪器设备

- (1) 装有 Vivado 和 ModelSim SE 软件的计算机。
- (2) Nexys Video Artix-7 FPGA 多媒体音视频智能互联开发系统。
- (3) 有源音箱或耳机。

四、 操作方法和实验步骤

- (1) 按照书中提供的框图，将音乐播放器次顶层划分为一下几个模块：主控制器、乐曲读取、音符播放、同步化电路以及节拍基准产生器
- (2) 参考实验 15 的资料，学习 DDS 技术相关知识，编写 DDS 模块并进行仿真
- (3) 根据书上的指导，依次编写剩下模块并进行仿真验证，测试其是否符合要求
- (4) 编写次顶层模块，并在其中设置参数 sim 方便仿真
- (5) 新建 Vivado 工程，生成符合要求的 DCM 模块，将自己编写的模块以及老师提供的网表文件及接口文件加入工程。对工程进行综合、约束、实现，并下载到开发板中进行验证