浙江大学

本科实验报告

课程名称：	数字系统实验设计
姓　　名：	姚桂涛
学　　院：	信息与电子工程学院
专　　业：	信息工程
学　　号：	3190105597
指导老师：	屈民军、唐奕

2021 年 6 月 29 日

一、 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“自顶而下”的数字系统设计方法。

二、 实验任务

设计一个音乐播放器，要求以下条件。

(1) 可以播放四首乐曲，设置 play/pause _ button、next _ button、reset 三个按键。按 play/pause _ button 键，音乐在播放和暂停之间切换；按 next _ button 键播放下一首乐曲。

(2) LEDO 指示播放情况 (播放时点亮)、LED2 和 LED 3 指示当前乐曲序号。

三、 实验原理

根据实验任务可将系统划分为时钟管理模块 (DCM)、按键处理、主控制器、乐曲读取、音符播放 (note _ player)、同步化电路、节拍基准产生器和音频编解码接口电路等子模块。

时钟管理模块 (DCM) 产生 100MHz 的系统时钟 sys _ clk 和 12.5MHz 的音频时钟 audio _ clk。

主控制器 (mcu) 模块接收按键信息，通知 song _ reader 模块是否要播放 (play) 及播放哪首乐曲 (song)。

乐曲读取 (song _ reader) 模块根据 mcu 模块的要求，逐个取出音符信息 note，duration 送给 note _ player 模块播放, 当一首乐曲播放完毕, 回复 mcu 模块乐曲播放结束信号 (song _ done)。

音符播放接收到需播放的音符，在音符的持续时间内，以 48kHz 速率送出该音符的正弦波样品给音频编解码接口模块。当一个音符播放结束, 向 song _ reader 模块发送一个 note _ done 脉冲索取新的音符。

音频编解码接口模块负责将音符的正弦波样品转换为串行输出并发送给音频编解码芯片 ADAU1761。音频编解码芯片 ADAU1761 接收正弦波样品，再进行 AD 转换并放大，最后送至扬声器播放。注意，note _ player 模块产生的正弦波样品为 16 位二进制，需在低位加 8 个 0 后送入音频编解码接口模块。

由于音频编解码模块与系统使用不同时钟，因此需要同步化电路协调两部分电路。

节拍基准产生器产生 48Hz 的节拍定时基准脉冲信号 (beat), 而 ready 信号频率为 48kHz, 因此，节拍基准产生器为分频比为 1000 的分频器。而按键处理模块完成输入同步化、防颤动和脉宽变换等功能。

1. 具体设计

1.1 DDS 模块的设计

由于前面已经自己做过 DDS 模块的相关实验，所以实验中所用到的 DDS 模块直接采用了前面实验所用到的设计。

DDS 模块代码

```
1 module dds (
2     clk,
```

```
    reset,
    k,
    sampling_pulse,
    new_sample_ready,
    sample
);
    input clk, reset,sampling_pulse;
    input [21:0] k;
    output [15:0] sample;
    output new_sample_ready;
    wire [21:0] raw_addr;
    wire [21:0] s_d;
    wire [9:0] rom_addr;
    wire [15:0] raw_data;
    wire [15:0] data;
    wire area;
    // 实现s_d = k + raw_addr
    full_adder addr(.a(k), .b(raw_addr), .s(s_d), .co());

    //d触发器1
    dffre #(.n(22))d1(.d(s_d), .en(sampling_pulse), .r(reset), .clk(clk), .q(
        raw_addr));

    //地址处理
    assign rom_addr[9:0] = raw_addr[20]?((raw_addr [20:10] == 1024)?1023:(~raw_addr
        [19:10]+1)):raw_addr[19:10];

    //SineROM
    sine_rom rom(.clk(clk), .addr(rom_addr), .dout(raw_data));

    //d触发器得到area
    dffre #(.n(1))d2(.d(raw_addr[21]), .en(1), .r(0), .clk(clk), .q(area));

    //数据处理
    assign data[15:0] = area?(~raw_data[15:0]+1):raw_data[15:0];

    //d触发器得到sample
    dffre #(.n(16))d3(.d(data), .en(sampling_pulse), .r(0), .clk(clk), .q(sample));

    //d触发器得到
    dffre #(.n(1))d4(.d(sampling_pulse), .en(1), .r(0), .clk(clk), .q(
        new_sample_ready));

endmodule
```

1.2 主控制模块 mcu 的设计

主控制模块 mcu 有响应按键信息、控制系统播放两大任务，表 6.11 为其端口含义。

表 1: 主控制模块 mcu 的端口含义

引脚名称	I/O	引脚说明
clk	Input	100MHz 时钟信号
reset	Input	复位信号，高电平有效
play_pause	Input	来自按键处理模块的“播放/暂停”控制信号，一个时钟周期宽度的脉冲
next	Input	来自按键处理模块的“下一曲”控制信号，一个时钟周期宽度的脉冲
play	Output	输出控制信号，高电平表示播放，控制 song _reader 模块是否要播放
reset_play	Output	时钟周期宽度的高电平复位脉冲 reset play，用于同时复位模块 song_reader 和 note_ player
song_done	Input	song_ reader 模块的应答信号，一个时钟周期宽度的高电平脉冲，表示一曲播放结束
song[1:0]	Output	当前播放乐曲的序号

根据设计要求，模块 mcu 的原理框图如图所示。图中的 2 位二进制数用来计算乐曲序号 (song)。

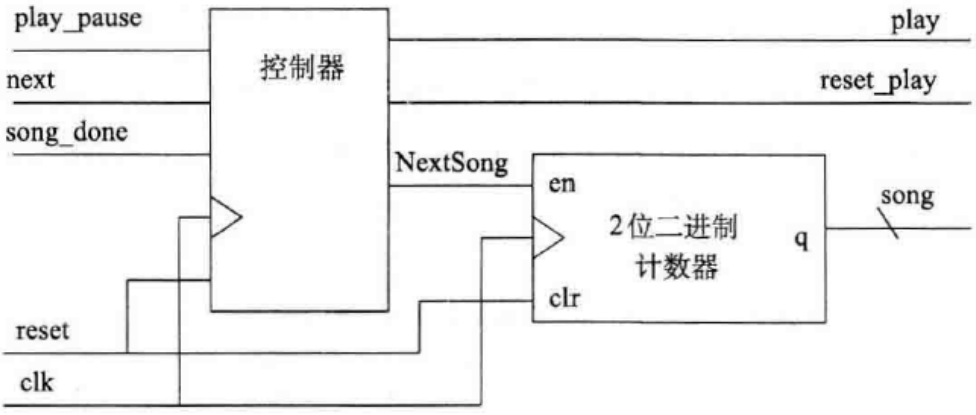

图 1: mcu 的结构框图

根据原理框图设计出 mcu 顶层代码如下：

mcu 顶层代码

```
1  module mcu (
2     clk,          //100MHz时钟信号
3     reset,        //复位信号，高电平有效
4     play_pause,   //来自按键处理模块的“播放/暂停”控制信号，一个时钟周期宽度的脉冲
5     next,         //来自按键处理模块的“下一曲”控制信号，一个时钟周期宽度的脉冲
6     play,         //输出控制信号，高电平表示播放，控制song _reader模块是否要播放
7     reset_play,   //时钟周期宽度的高电平复位脉冲reset play，用于同时复位模块song_reader和
          note_ player
8     song_done,    //song_ reader模块的应答信号，一个时钟周期宽度的高电平脉冲，表示一曲播放结束
9     song          //当前播放乐曲的序号
10 );
11    input clk, reset, play_pause, next, song_done;
12    output play, reset_play;
```

```
13      output [1:0] song;
14      wire NextSong;
15
16      //mcu控制器
17      mcu_controller m_ctrl1 (
18          .clk(clk),
19          .reset(reset),
20          .play_pause(play_pause),
21          .next(next),
22          .song_done(song_done),
23          .play(play),
24          .reset_play(reset_play),
25          .NextSong(NextSong)
26      );
27
28      //2位二进制计数器
29      counter_n #(.n(4), .counter_bits(2)) m_counter(
30          .clk(clk),
31          .en(NextSong),
32          .r(reset),
33          .q(song),
34          .co()
35      );
36
37  endmodule
```

控制器的工作流程图如图 6.46 所示，控制器设置初始复位 (RESET)、播放 (PLAY)、暂停 (PAUSE) 和下一首 (NEXT) 四种状态。系统复位后，经 RESET 状态初始化后进入 PAUSE 状态，等待各种命令输入; play _ pause 脉冲信号使系统在 PLAY、PAUSE 两状态之间互转; 在 PLAY 或 PAUSE 状态下，若按下 next _ button 按钮，则使系统在进入 NEXT 状态，输出 reset _ play 脉冲复位 song _ reader 和 note _ player 两个模块, 同时输出脉冲 NextSong 乐曲序号计数器加 1，进入下一曲播放; 另外，在 PLAY 状态时，若乐曲播放结束 (song _ done 有效) 则结束播放，经 RESET 状态复位 song _ reader 和 note _ player 两个模块，并进入 PAUSE 状态，再次等待各种命令输入。

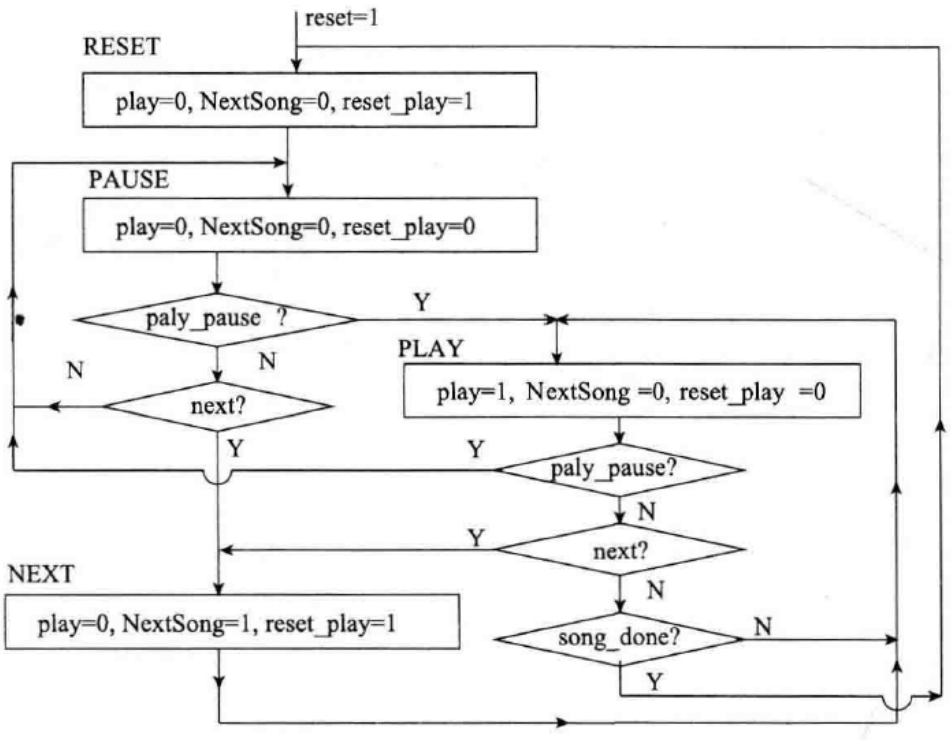

图 2: mcu 控制器的算法流程

根据原理框图设计出 mcu 控制器代码如下：

mcu 控制器代码

```
module mcu_controller (
    clk,
    reset,
    play_pause,
    next,
    song_done,
    play,
    reset_play,
    NextSong
);
    input clk, reset, play_pause, next, song_done;
    output reg play, reset_play, NextSong;
    parameter RESET = 0, PAUSE = 1, PLAY = 2, NEXT = 3;

    reg [1:0] state;    //状态信号
    reg [1:0] nextstate; //驱动信号

    //状态寄存器
    always @(posedge clk ) begin
        if(reset) state = RESET;
        else state = nextstate;
    end
```

```
23
24      //下一状态和输出
25      always @(*) begin
26          // play = 0;NextSong = 0;reset_play = 0;
27          case(state)
28              RESET:
29                  begin
30                      play = 0;
31                      NextSong = 0;
32                      reset_play = 1;
33                      nextstate = PAUSE;
34                  end
35              PAUSE:
36                  begin
37                      play = 0;
38                      NextSong = 0;
39                      reset_play = 0;
40                      if(play_pause) nextstate = PLAY;
41                      else
42                          begin
43                              if(next) nextstate = NEXT;
44                              else nextstate = PAUSE;
45                          end
46                  end
47              PLAY:
48                  begin
49                      play = 1;
50                      NextSong = 0;
51                      reset_play = 0;
52                      if(play_pause) nextstate = PAUSE;
53                      else
54                          begin
55                              if(next) nextstate = NEXT;
56                              else
57                                  begin
58                                      if(song_done) nextstate = RESET;
59                                      else nextstate = PLAY;
60                                  end
61                          end
62                  end
63              NEXT:
64                  begin
65                      play = 0;
66                      NextSong = 1;
67                      reset_play = 1;
68                      nextstate = PLAY;
69                  end
70              default: nextstate = RESET;
71          endcase
72      end
```

```
73 endmodule
```

1.3 乐曲读取模块 song _ reader 的设计

乐曲读取模块 song _ reader 的任务有 (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。根据 song _ reader 模块的任务要求，song _ reader 模块需包含表 6.12 所示的输入、输出端口。

表 2: 乐曲读取模块 song _ reader 的端口含义

引脚名称	I/O	引脚说明
clk	Input	100MHz 时钟信号
reset	Input	复位信号，高电平有效
play	Input	来自 mcu 的控制信号，高电平要求播放
song[1:0]	Input	来自 mcu 的控制信号，当前播放乐曲的序号
note_done	Input	即模块 note player 的应答信号，一个时钟周期宽度的脉冲，表示一个音符播放结束并索取新音符
song_done	Output	给 mcu 的应答信号，当乐曲播放结束，输出-一个时钟周期宽度的脉冲，表示乐曲播放结束
note[5:0]	Output	音符标记
duration[5:0]	Output	音符的持续时间
new_note	Output	给模块 note_playe 的控制信号，一个时钟周期宽度的高电平脉冲，表示新的音符需播放

song _ rom 是一个只读存储器，用来存放乐曲，容量为 2'×12bits。共存放四首乐曲，每首乐曲占用 25×12bits 空间，即每首乐曲最长由 32 个音符组成。因此，song _ rom 高 2 位地址决定哪首乐曲，而低 5 位地址决定这首乐曲的哪个音符。song _ rom 每个地址存放一个音符信息，音符信息由 12 位二进制组成，高 6 位表示音符标记 note，低 6 位表示音长 duration。

song _ rom 模块已由作者提供，前三首乐曲已填写，第 4 首乐曲空白，由读者自己填写，这里说明一下，若乐曲不足 32 个音符，多余的空间用数字 0 填补。

根据 song _ reader 模块的功能及 song _ rom 结构，可画出图 6.47 所示的结构框图，控制器主要负责接收 mcu 模块与 note _ player 模块的控制信号，并做出响应。算法流程图如图 6.48 所示。

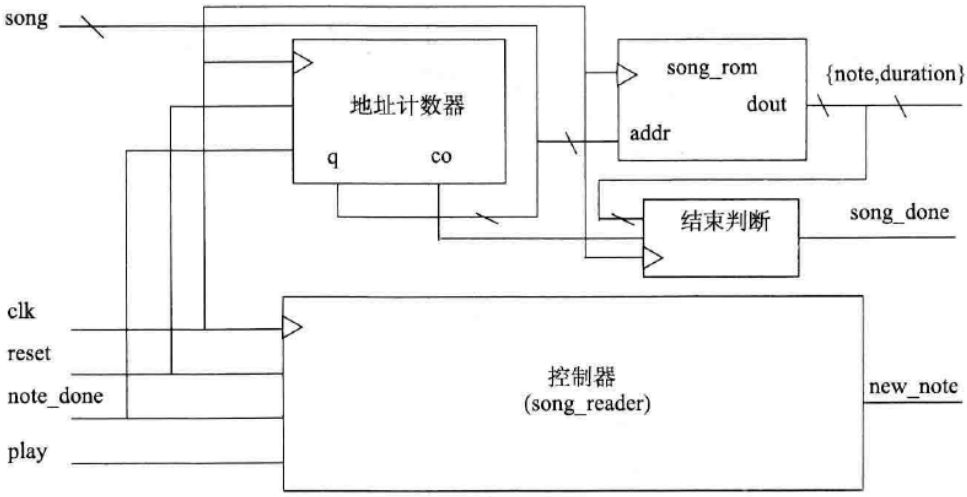

图 3: song _ reader 的结构框图

根据原理框图设计出 song _ reader 顶层代码如下：

song _ reader 顶层代码

```
module song_reader (
    song,          //来自mcu的控制信号，当前播放乐曲的序号
    clk,           //100MHz时钟信号
    reset,         //复位信号，高电平有效
    note_done,     //即模块note player的应答信号，一个时钟周期宽度的脉冲，表示一个音符播放结束并索取新音符
    play,          //来自mcu的控制信号，高电平要求播放
    song_done,     //给mcu 的应答信号，当乐曲播放结束，输出一个时钟周期宽度的脉冲，表示乐曲播放结束
    new_note,      //给模块note_playe的控制信号，一个时钟周期宽度的高电平脉冲，表示新的音符需播放
    note,          //音符标记
    duration       //音符的持续时间
);
    input clk, reset, play, note_done;
    input [1:0] song;
    output song_done, new_note;
    output [5:0] note, duration;
    wire [4:0] q; //song_rom的低5位地址
    wire co;      //地址计数器进位

    //地址计数器
    counter_n #(.n(32), .counter_bits(5)) song_counter(
        .clk(clk),
        .en(note_done),
        .r(reset),
        .q(q),
        .co(co)
    );
```

```
27
28      //控制器
29      song_reader_controller s_ctrl1(
30          .clk(clk),
31          .reset(reset),
32          .play(play),
33          .note_done(note_done),
34          .new_note(new_note)
35      );
36
37      //结束判断
38      over over1(
39          .clk(clk),
40          .duration(duration),
41          .co(co),
42          .out(song_done),
43          .reset(reset)
44      );
45
46      //song_rom
47      song_rom songs(
48          .clk(clk),
49          .dout({note,duration}),
50          .addr({song,q})
51      );
52
53
54  endmodule
```

系统复位后一直在 RESET 状态等待 mcu 模块控制信号输入，当 mcu 模块发出播放命令 (play 为高电平) 时，进入 NEW _ NOTE 状态输出 new _ note 脉冲要求 note _ player 模块播放音符; 然后进入 WAIT 状态等待，当 note _ player 模块播放完音符时，会发出 note _ done 脉冲信号索取下一音符，note _ done 脉冲信号一方面让地址计数器递增，并从 song _ rom 取出一个新的音符，另一方面让控制器进入 NEW _ NOTE 状态，输出 new _ note 脉冲通知 note _ player 模块有新的音符需要播放。当 note _ done 有效，需要两个时钟周期才能从 song _ rom 中读取下一个音符信息。因此新音符有效标记信号 new _ note 也应在新音符数据输出后有效，其时序关系如图 6.49 所示。所以在流程图中插入 NEXT _ NOTE 状态，目的是延迟一个时钟周期输出信号，以配合 song _ rom 的读取要求。

地址计数器为 5 位二进制计数器，其中 note _ done 为计数使能输入，当 note _ done 为高电平时，允许计数。计数器状态 q 为 song _ rom 的低 5 位地址,song[1:0] 为 song _ rom 高两位地址。

当地址计数器出现进位或 duration 为 0 时，表示乐曲结束，应输出一个时钟周期宽度的高电平脉冲信号 song _ done。

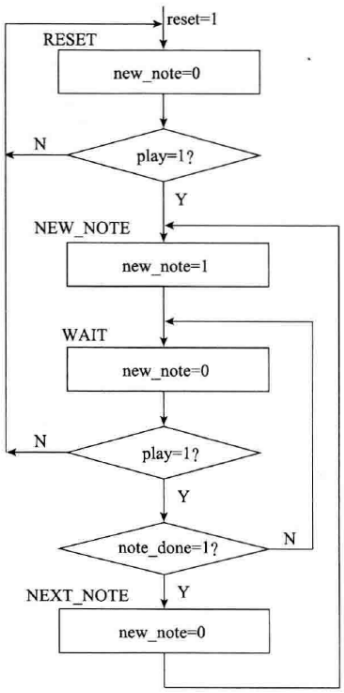

图 4: song _ reader 控制器的算法流程

根据原理框图设计出 song _ reader 控制器代码如下：

song _ reader 控制器代码

```
module song_reader_controller (
    clk,
    reset,
    play,
    note_done,
    new_note
);
    input clk, reset, play, note_done;
    output reg new_note;
    parameter RESET = 0, NEW_NOTE = 1, WAIT = 2, NEXT_NOTE = 3;
    reg [1:0] state, nextstate;

    //状态寄存器
    always @(posedge clk ) begin
        if(reset) state = RESET;
        else state = nextstate;
    end

    //下一状态和输出
    always @(*) begin
        new_note = 0;
        case(state)
            RESET:
```

```
24              begin
25                  if(play) nextstate = NEW_NOTE;
26                  else nextstate = RESET;
27              end
28          NEW_NOTE:
29              begin
30                 new_note = 1;
31                 nextstate = WAIT;
32              end
33          WAIT:
34              begin
35                  if(play)
36                      begin
37                          if(note_done) nextstate = NEXT_NOTE;
38                          else nextstate = WAIT;
39                      end
40                  else nextstate = RESET;
41              end
42          NEXT_NOTE:
43              begin
44                  nextstate = NEW_NOTE;
45              end
46          default: nextstate = RESET;
47      endcase
48   end
49 endmodule
```

1.4 音符播放模块 note _ player 的设计

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) 当一个音符播放完毕，向 song _ reader 模块索取新的音符。

根据 note _ player 模块的任务，进一步划分功能单元，如图 6.50 所示，图中 FreqROM 为只读存储器，完成音符标记 note 与 DDS 模块的相位增量 k 查找表关系。表 6.13 所示为 note _ player 模块的端口含义。

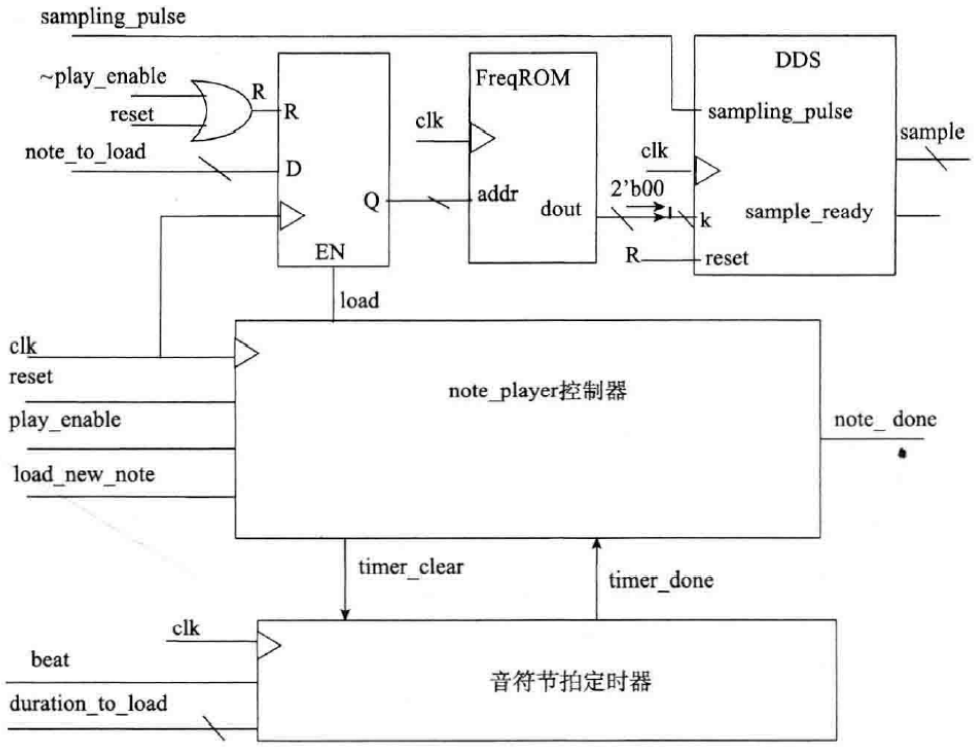

图 5: note _ player 的结构框图

表 3: note _ player 模块的端口含义

引脚名称	I/O	引脚说明
clk	Input	系统时钟信号，外接 sys_clk
reset	Input	复位信号，高电平有效，外接 mcu 模块的 reset_ play
play_enable	Input	来自 mcu 模块的 play 信号，高电平表示播放
note_to_load[5:0]	Input	来自 song_reader 模块的音符标记 note，表示需播放的音符
duration_to_load[5:0]	Input	来自 song _reader 模块的音符持续时间 duration，表示需播放音符的音长
load_new_note	Input	来自 song_reader 模块的 new_note 信号，一个时钟周期宽度的高电平脉冲，表示新的音符需播放
note_done	Output	给 song_reader 模块的应答信号，一个时钟周期宽度的高电平脉冲，表示音符播放完毕
sampling_pulse	Input	来自同步化电路模块的 ready 信号，频率 48kHz，一个时钟周期宽度的高电平脉冲，表示索取新的正弦样品
beat	Input	定时基准信号，频率为 48Hz 脉冲，一个时钟周期宽度的高电平脉冲
sample[15:0]	Output	正弦样品输出

根据原理框图设计出 note _ player 顶层代码如下：

note _ player 顶层代码

```
module note_player (
    clk,              //系统时钟信号，外接sys_clk
    reset,            //复位信号，高电平有效，外接mcu模块的reset_ play
    play_enable,      //来自mcu模块的play信号，高电平表示播放
    note_to_load,     //来自song_reader模块的音符标记note，表示需播放的音符
    duration_to_load, //来自 song _reader模块的音符持续时间duration，表示需播放音符的音长
    load_new_note,    //来自 song_reader模块的new_note信号，一个时钟周期宽度的高电平脉冲，表
        示新的音符需播放
    note_done,        //给song_reader模块的应答信号，一个时钟周期宽度的高电平脉冲，表示音符播放
        完毕
    sampling_pulse,  //来自同步化电路模块的ready信号，频率48kHz，一个时钟周期宽度的高电平脉
        冲，表示索取新的正弦样品
    beat,             //定时基准信号，频率为48Hz脉冲，一个时钟周期宽度的高电平脉冲
    sample,           //正弦样品输出
    sample_ready      //正弦成功输出信号
);
    input clk, reset, play_enable, load_new_note,sampling_pulse,beat;
    input [5:0] note_to_load, duration_to_load;
    output note_done, sample_ready;
    output [15:0] sample;
    wire load;                  //读取新的音符
    wire [5:0] q;               //FreqROM地址输入
    wire [19:0] dout;           //FreqROM读取的相位增量
    wire timer_clear,timer_done; //清零信号与定时结束标志

    //D触发器
    dffre #(.n(6)) note_dffre(
        .d(note_to_load),
        .en(load),
        .r(~play_enable||reset),
        .clk(clk),
        .q(q)
    );

    //FreqROM
    frequency_rom FreqROM(
        .clk(clk),
        .dout(dout),
        .addr(q)
    );

    //DDS
    dds note_dds(
        .clk(clk),
        .reset(~play_enable||reset),
        .k({2'b00,dout}),
        .sampling_pulse(sampling_pulse),
        .new_sample_ready(sample_ready),
        .sample(sample)
```

```
47      );
48
49      //控制器
50      note_player_controller note_ctrl(
51          .clk(clk),
52          .reset(reset),
53          .play_enable(play_enable),
54          .load_new_note(load_new_note),
55          .load(load),
56          .note_done(note_done),
57          .timer_clear(timer_clear),
58          .timer_done(timer_done)
59      );
60
61      //音符节拍定时器
62      timer note_counter(
63          .clk(clk),
64          .r(timer_clear),
65          .en(beat),
66          .n(duration_to_load),
67          .done(timer_done)
68      );
69  endmodule
```

note _ player 控制器负责与 song _ reader 模块接口，读取音符信息，并根据音符信息从 Frequency ROM 中读取相应相位增量 k 送给 DDS 子模块。另外，note _ player 控制器还需要控制音符播放时间。note _ player 控制器的算法流程如图 6.51 所示。在复位或未播放时，控制器处于 RESET 状态或 PLAY 状态，由于此时高电平 reset 或低电平 play _ enable 都使图 6.35 中的 D 型寄存器清 0，进而使 k 为 0，不会输出正弦样品。当 play _ enable 为高电平，系统进入音符播放 PLAY 状态，当一个音符播放结束时，控制器进入 DONE 状态，置位 done _ with _ note，向 song _ reader 模块索取新的音符，此时 song _ reader 模块输出一个 new _ note 脉冲信号使控制器进入 LOAD 状态，读取新的音符，然后进入 PLAY 状态播放下一个音符。

音符定时器为 6 位二进制计数器，beat、timer _ clear 分别为使能、清　信号，均为高电平有效。定时时间由音长信号 duration _ to _ load 决定, 即 duration _ to _ load 个 beat 周期, timer _ done 为定时结束标志。子模块 DDS 的功能就是利用 DDS 技术产生正弦样品，其工作原理已在实验 15 中介绍了，注意:DDS 模块的输入 k 为 22 位二进制，因此需 FreqROM 输出的 20 位相位增量高位加 2 个 0 后接入 DDS。

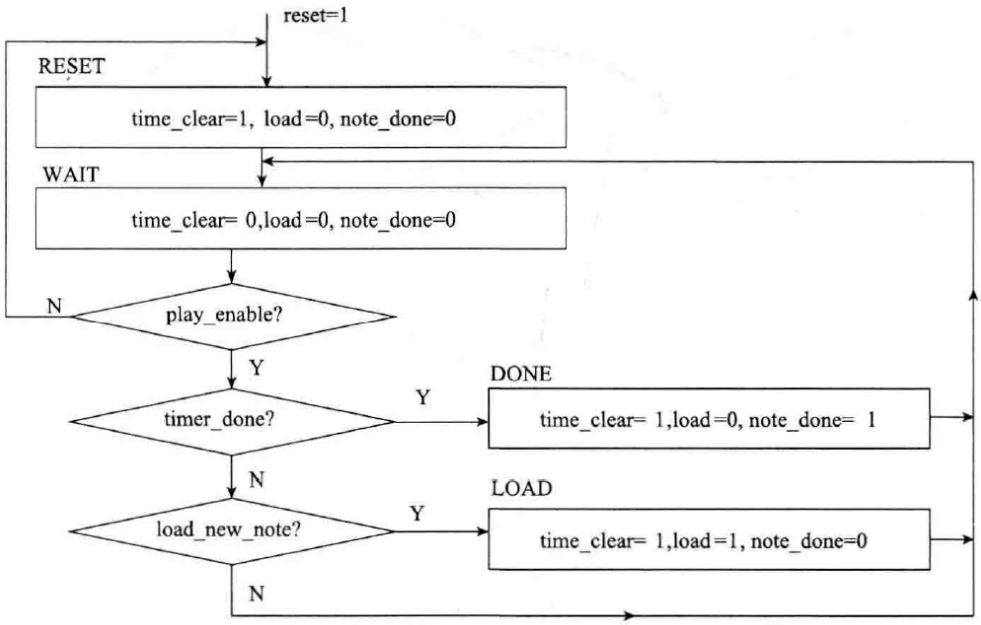

图 6: note _ player 控制器的算法流程

根据原理框图设计出 note _ player 控制器代码如下：

note _ player 控制器代码

```
module note_player_controller (
    clk,
    reset,
    play_enable,
    load_new_note,
    load,
    note_done,
    timer_clear,
    timer_done
);
    input clk, reset, play_enable, timer_done, load_new_note;
    output reg load,timer_clear,note_done;
    parameter RESET = 0, WAIT = 1, DONE = 2, LOAD = 3;
    reg [1:0] state, nextstate;
    //状态
    always @(posedge clk ) begin
        if(reset) state = RESET;
        else state = nextstate;
    end

    //下一状态和输出
    always @(*) begin
        load = 0; timer_clear = 0; note_done = 0;
        case(state)
            RESET:
```

```
26              begin
27                  timer_clear = 1;
28                  load = 0;
29                  note_done = 0;
30                  nextstate = WAIT;
31              end
32          WAIT:
33              begin
34                  timer_clear = 0;
35                  load = 0;
36                  note_done = 0;
37                  if(play_enable)
38                      begin
39                          if(timer_done) nextstate = DONE;
40                          else
41                              begin
42                                  if(load_new_note) nextstate = LOAD;
43                                  else nextstate = WAIT;
44                              end
45                      end
46                  else nextstate = RESET;
47              end
48          DONE:
49              begin
50                  timer_clear = 1;
51                  load = 0;
52                  note_done = 1;
53                  nextstate = WAIT;
54              end
55          LOAD:
56              begin
57                  timer_clear = 1;
58                  load = 1;
59                  note_done = 0;
60                  nextstate = WAIT;
61              end
62          default: nextstate = RESET;
63      endcase
64   end
65 endmodule
```

1.5　同步化电路

由于音频编解码接口模块和其他模块采用不同的时钟，因此两者之间的控制及应答信号须进行同步化处理。本例中音频编解码接口模块的输出信号 NewFrame 的脉冲宽度为一个 audio _ clk 时钟周期, 需通过同步化处理, 产生与 sys _ clk 同步且脉冲宽度为一个 sys _ clk 时钟周期的信号 ready。电路如图 6.52 所示，由同步器和脉冲宽度变换电路组成。

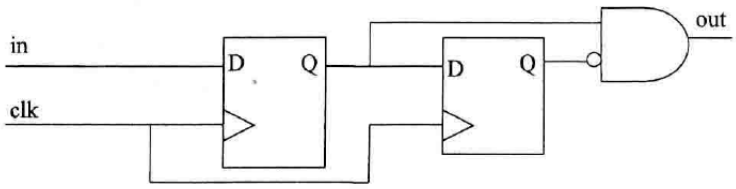

图 7: 同步化电路

同步化电路代码如下：

同步化电路代码

```
module synchro (
    clk,
    in,
    out
);
    input clk, in;
    output out;

    reg q1,q2;

    //非阻塞赋值
    always @(posedge clk ) begin
        q1 <= in;
        q2 <= q1;
    end

    assign out = q1 && (~q2);
endmodule
```

1.6 时钟管理模块 (DCM)

IP 内核时钟管理模块的输入时钟 clk 频率为 100MHz，产生 100MHz 的系统时钟和 12.50MHz 的音频时钟。

另外，音频编解码接口模块与按键处理模块按实验 18 和实验 11 介绍设计。也可调用作者提供的设计。不过，作者是以 BlackBox 方式提供，即音频编解码接口模块提供综合网表文件 AudioInterface.edf 和端口文件 AudioInterface.v; 而按键处理模块提供综合网表文件 button press _ unit.edf 和端口文件 button _ press _ unit.v。

四、 主要仪器设备

Modelsim SE、Vivado、Nexys Video Artix-7 FPGA 多媒体音视频智能互联开发系统、有源音响或耳机。

五、 实验过程

1. 仿真测试

1.1 主控制器 mcu 模块

仿真图：

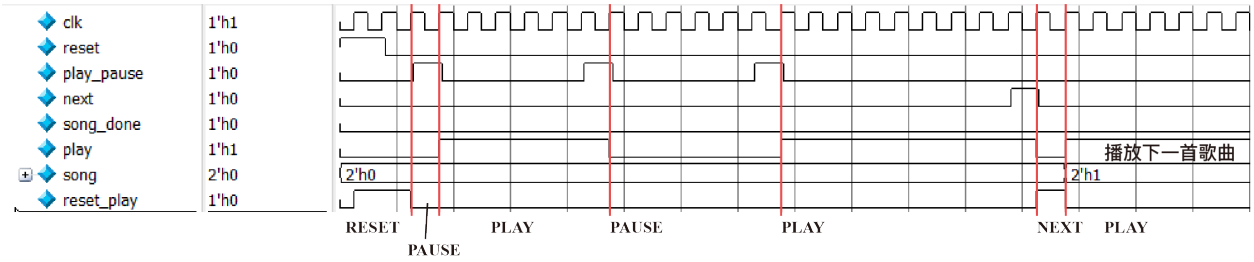

图 8: mcu 仿真图

仿真分析：

(1) 开始 reset = 1，clk 上沿到来时，电路处于 RESET 状态，输出 play = 0，nextsong = 0，reset_ play = 1。

(2) 第一条红线时，reset = 0，play _ pause = 0，电路进入 PAUSE 状态。

(3) 第二条红线时，reset = 0, play _ pause = 1，相当于按下 paly 按钮，电路进入 PLAY 状态，play = 1。

(4) 第三条红线时，reset = 0，play_ pause = 1，电路进入 PAUSE 状态，输出 play = 0。

(5) 第四条红线时，reset = 0, play_ pause = 0，next = 1，相当于按下 next 按钮，电路进入 NEXT 状态。

1.2 乐曲读取 song _ reader 模块

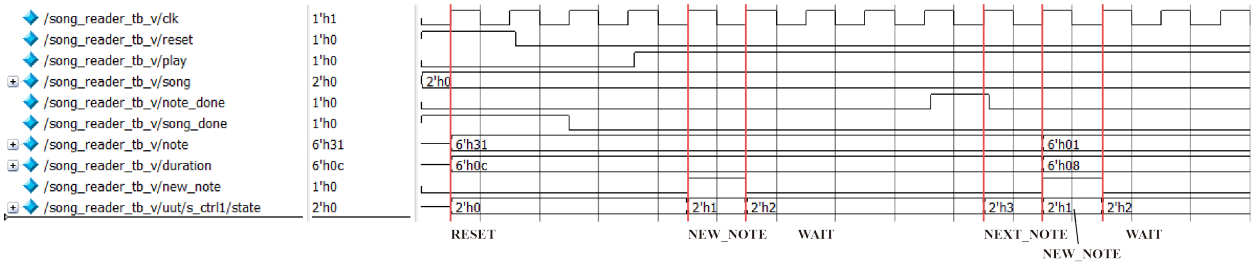

图 9: song _ reader 仿真图

仿真分析：

(1) 第一条红线时，reset = 1, 电路进入 RESET 状态。

(2) 第二条红线时，reset = 0, play = 0，电路保持 RESET 状态。

(3) 第三条红线时，reset = 0, play = 1，电路接收到 mcu 发来的播放命令，进入 NEW_ NOTE 状态，new _ note = 1，要求 note_ player 模块播放音符。

(4) 第四条红线时，电路进入 WAIT 状态，等待播放停止或者音符播放完毕。

(5) 第五条红线时，note_ done = 1，音符播放完毕，电路进入 NEXT_ NOTE 状态，以延迟一个时钟周期输出信号，配合 song_ rom 的读取要求。

(6) 第六条红线时，电路进入 NEW_ NOTE 状态，播放新的音符。

1.3 音符播放 note _ player 模块

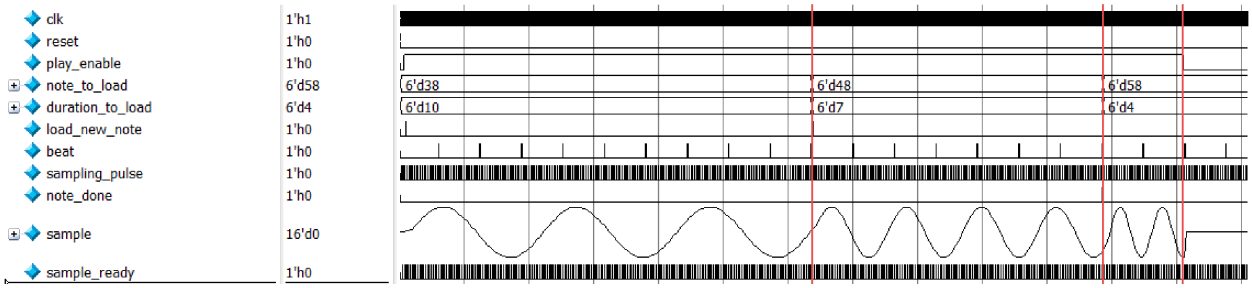

图 10: note _ player 仿真图

仿真分析：

由仿真图可知，正弦波形的频率与 note _ to_ load 的值呈正相关。同时只有当 play _ enabe = 1 时才会输出波形。

1.4 次顶层 music _ player

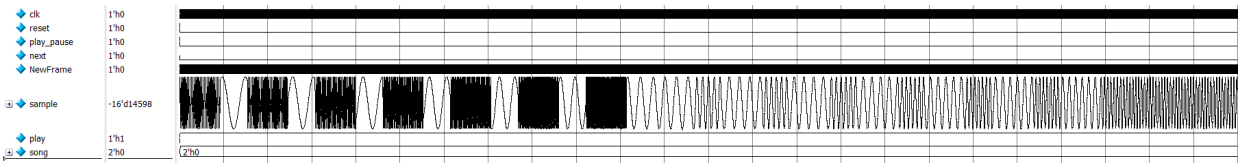

图 11: 次顶层 music_ player 仿真图

根据仿真图，波形基本符合要求。

2. vivado 实现

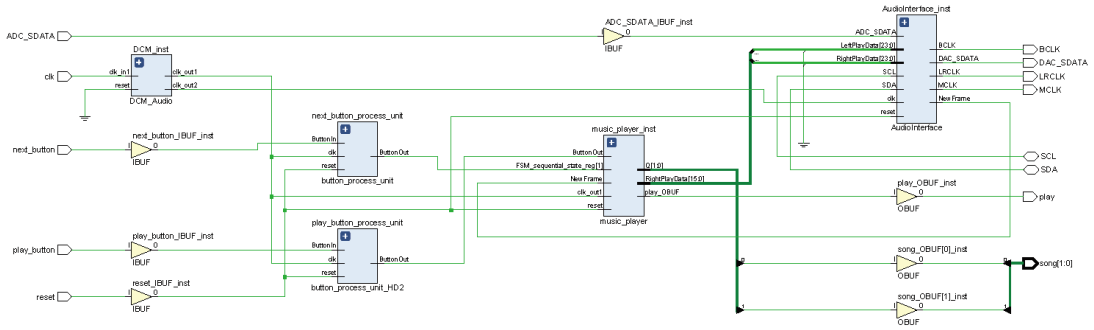

图 12: vivado 实现

3. 实验中遇到的问题和解决方法

(1) 在最终上板测试阶段，发现播放歌曲的时候，当播放完毕时，无法自动停止。在老师的指导下，发现是我 song _ reader 模块的结束判断代码出了问题。具体表现在我将判断条件 duration = 0 在代码中写成了 ~duration，但是由于 duration 并非 1 位数据，所以不能用简单的取反判断，应该用按位取反或 duration == 0 作为判断条件。在修改判断条件为 duration ==0 后，问题得到了解决。

六、 思考题

1. 在实验中，为什么 next_button、play_pause_button 两个按键需要消颤动及同步化处理，而 reset 按键不需要消颤动及同步化处理?

因为 reset 是置零信号，当 reset=1 时，系统置零，之后 reset 为 0 或为 1 系统内部该置零的信号都为 0，reset 的颤动对系统无影响。而 next_button 和 play _pause_button 都会影响系统内部状态的转换，如果 next 输入不稳定，则不能达到播放下一首的目的，可能会跳到后面几首歌，也就是会使播放的下一首歌不确定; 如果 play_pause 输入不确定，则系统会不断地在播放和暂停之间切换，这不仅会增加系统的损耗，而且会输出断断续续的音乐，达不到预期效果。

2. 在主控制器（mcu）设计中，是否存在接收不到按键信息？若存在，概率多大？有没必要修改设计?

存在。在 RESET 到 PAUSE 状态转换过程中和 NEXT 到 PLAY 状态转换过程中没有判断 next 和 play_pause，这期间按下按钮接收不到按键信息。概率很小，因为只有一个时钟周期，因此没有必要修改设计。

七、 实验心得

本次实验一开始我其实是无从下手的。所以我就按照老师的建议，去做了前的涉及到的 DDS 子实验。在花了半天的时间后，我完成了 DDS 实验。然后我去看了老师提供的视频，学会了二段式描述中

状态机。

之后我便开始了最后的实验，一开始面对复杂的次顶层，我不知道从何开始，于是我便一个小模块一个小模块的设计。并且每个小模块的顶层文件和调用的子模块分开写，保证文件管理的清晰明确。

在设计完一个一个的小模块之后，一开始觉得复杂的次顶层也慢慢变得清晰简单起来。最后也顺利完成了实验。

在调试的过程中，遇到了切换下一首后不能直接播放的问题，但是在仔细观察了仿真波形后还是没能发现问题所在。