浙江大学

本科实验报告

课程名称： 数字系统设计实验

姓　　名： 王英杰

学　　院： 信息与电子工程学院

　　系： 信息与电子工程

专　　业： 信息工程

学　　号： 3190103370

指导教师： 屈民军 唐奕

2021 年 6 月 18 日

专业：信息工程
姓名：王英杰
学号：3190103370
日期：2021/6/18
地点：东四 223

浙江大学实验报告

课程名称：数字系统设计实验 指导老师：屈民军 唐奕 实验名称：Lab19 音乐播放实验

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、实验任务与要求

1）任务一

设计并仿真 DDS 正弦信号发生器，要求：

1）采样频率 $f_c = 48kHz$；
2）正弦信号频率范围为$20Hz$~$20kHz$；
3）正弦信号序列宽度为 16 位，包括一位符号。

2）任务二

设计一个音乐播放器，要求满足以下条件：

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）LED0 指示播放情况（播放时点亮）、LED2 和 LED3 指示当前乐曲序号。

三、实验原理

1、DDS 原理

在数字域产生正弦信号，可以用一个存储器（ROM/RAM）存储一张正弦表；然后将存于表中的正弦样品取出，经数模转换器 D/A，形成模拟量波形。DDS 通过改变寻址的步长来改变输出信号的频率，步长为对数字波形查表的相位增量，输出正弦频率与相位增量呈线性关系。

DDS 的基本原理框图如图 1 所示，由相位累加器、正弦查找表（Sine ROM）、D/A 转换器和低通滤波器组成。

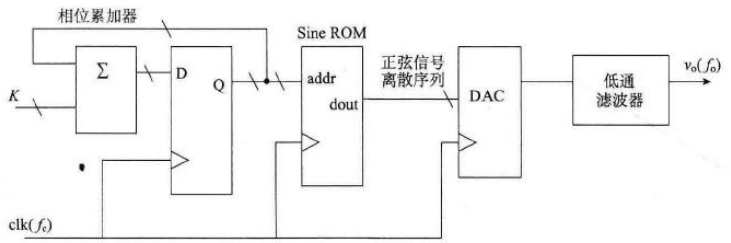


图 1　DDS 原理框图

Sine ROM 中存放一个完整的正弦信号样品，正弦信号样品根据式（1）的映射关系构成，

$$S(i) = (2^{n-1} - 1) \times sin\left(\frac{2\pi i}{2^m}\right) \qquad (1)$$

式中，m 为 Sine ROM 地址线位数；n 为 ROM 的数据线宽度，$S(i)$的数据形式为补码。

f_c为取样时钟 clk 的频率，K 为相位增量，输出正弦信号的频率f_o由f_c和 K 共同决定，

$$f_o = \frac{K \times f_c}{2^m} \qquad (2)$$

DDS 系统设计原理

根据 DDS 输出信号的最低频率要求，计算出$m = 12$，为了得到。但为了得到更准确的正弦信号频率，相位累加器位数会增加 10 位小数。所以，相位累加器为 22 位累加器，高 12 位为 Sine ROM 的地址。

$$f_o(min) = \frac{f_c}{2^m} \leq 20Hz$$

由于正弦信号的对称性，将正弦波分成 4 个区域，只要在 Sine ROM 中存储 1/4 的正弦信号样品即可。这样，Sine ROM 容量可减少为$2^{10} \times 16$bits, 存储 0-1023 共 1024 个 1/4 的正弦信号样品。四分之一周期的正弦信号样品未给出$90°$的样品值，在 ROM 地址为 1024 时可取地址为 1023 的值。

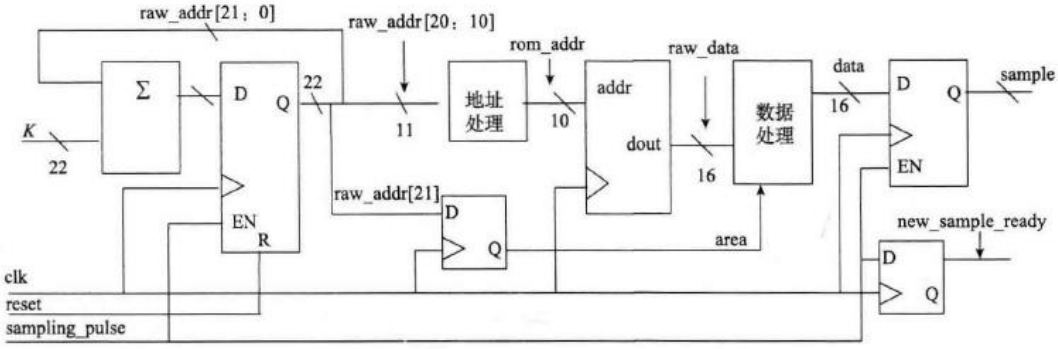


图 2　优化后的 DDS 结构

（1）很多情况下，时钟信号与采样脉冲不是同一个信号，图 2 中，sampling_pulse 为采样脉冲，应低于时钟 clk 的频率，且宽度必须为一个时钟 clk 的周期。

（2）相位累加得到 22 位原始地址 raw_addr[21:0]，整数部分 raw_addr[21:10]为完整周期正弦信号样品的地址，其中高两位地址 raw_addr[21:20]用作区分正弦的四个区域。由于 sine_rom 只保存了 1/4 周期的 1024 个样品值，因此，raw_addr[21:10]应进行相应的处理，处理方法如表 1 所示：

表 1　sine_rom 的地址和数据处理方法

正弦区域	Sine ROM 地址	正弦样品 data	备注
0	raw_addr[19:10]	raw_data[15:0]	
1	当 raw_addr[20:10]=1024 时，rom_addr 取 1023，其他情况取~raw_addr[19:10]+1	raw_data[15:0]	地址镜像翻转
2	raw_addr[19:10]	~raw_data[15:0]+1	数据取反
3	当 raw_addr[20:10]=1024 时，rom_addr 取 1023，其他情况取~raw_addr[19:10]+1	~raw_data[15:0]+1	地址镜像翻转 数据取反

2、音乐播放器原理

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。各主要模块的作用如下：

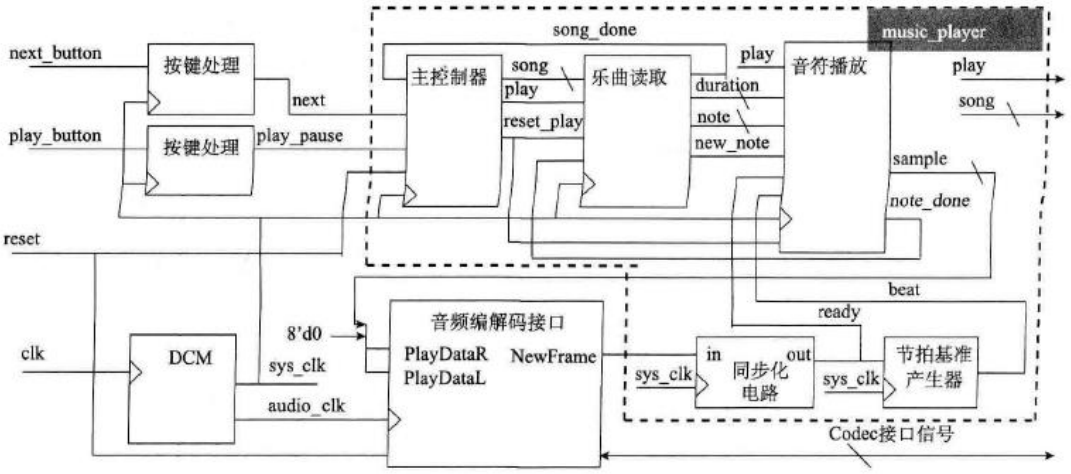


图 3　音乐播放器顶层模块

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（note_player）模块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ADAUl761。音频编解码芯片ADAU l761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。

同步化电路模块 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，节拍基准产生器即为分频比为 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。

设计原理：

1、主控制模块 mcu 的设计

主控制模块 mcu 具有响应按键信息、控制系统播放两大任务，其结构框图如图 4 所示。

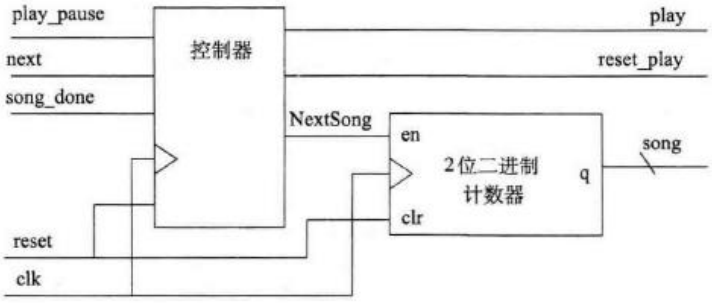


图 4　mcu 的结构框图

各端口含义如表 2 所示：

表 2　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

算法流程图如图 5 所示：

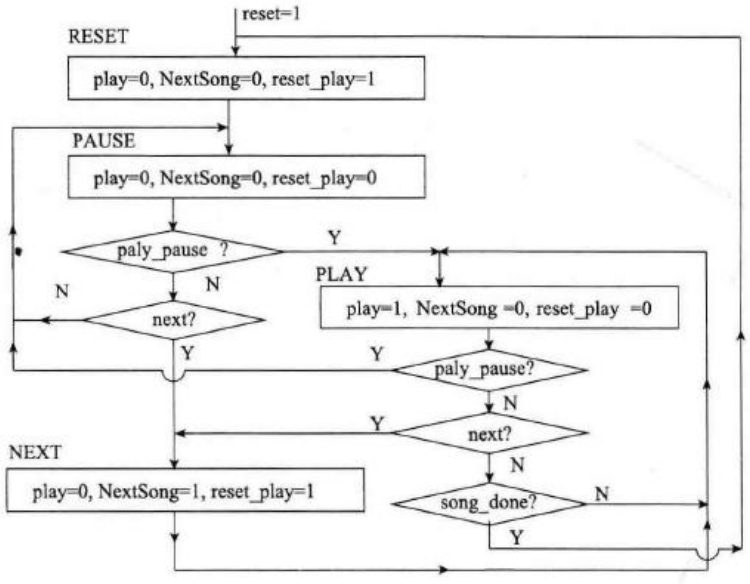


图 5　mcu 控制器的算法流程图

控制器有RESET、PLAY、PAUSE、NEXT四个状态。系统复位后，经RESET状态初始化后进入PAUSE状态，等待控制信号输入；play_pause信号控制系统在PLAY、PAUSE两个状态间互转；在PLAY、PAUSE状态下，若按下next_button按钮，则系统在进入NEXT状态且输出reset_play脉冲复位song_reader && noter_player 两个模块的同时输出NextSong信号使乐曲序号计数器加1，从而播放下一曲；在PLAY状态下，若乐曲播放结束即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。

2、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；

（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。

乐曲读取模块 song_reader 的结构框图如图 6 所示：

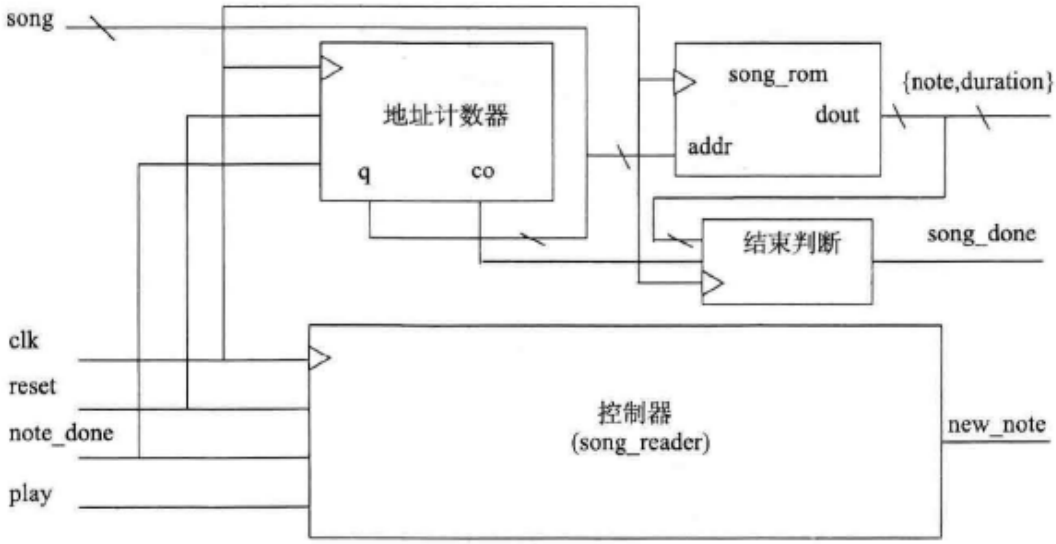
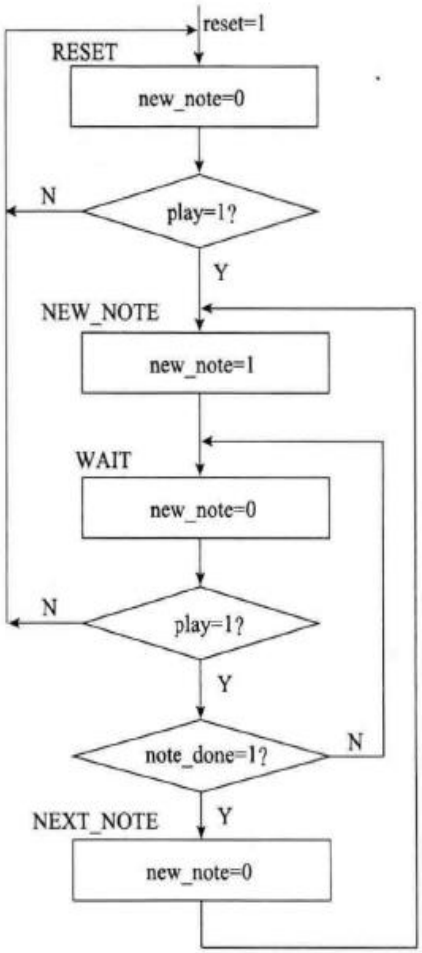


图 6　song_reader 的结构框图

其控制器的算法流程图如图 7 所示：



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，发出 note_done 脉冲信号索取下一音符，该脉冲一方面使得地址计数器递增，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 8 所示。再流程图中插入 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。

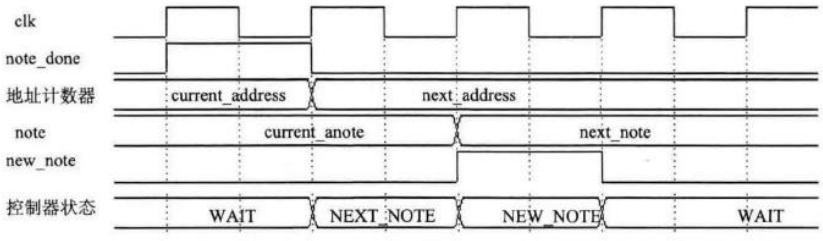


图 7 song_reader 控制器的算法流程图　图 8　song_reader 控制器、地址计数器和 ROM 的时序关系

各端口的含义如表 3 所示：

表 3　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

3、音符播放模块 note_player 的设计

音乐播放模块 note_player 是本实验的核心模块，它主要的任务包括以下方面：

（1）从 song_reader 模块接收需要播放的音符；

（2）根据 note 值找出 DDS 的相位增量 k；

（3）以 48kHz 速率从 SineROM 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 9 所示：

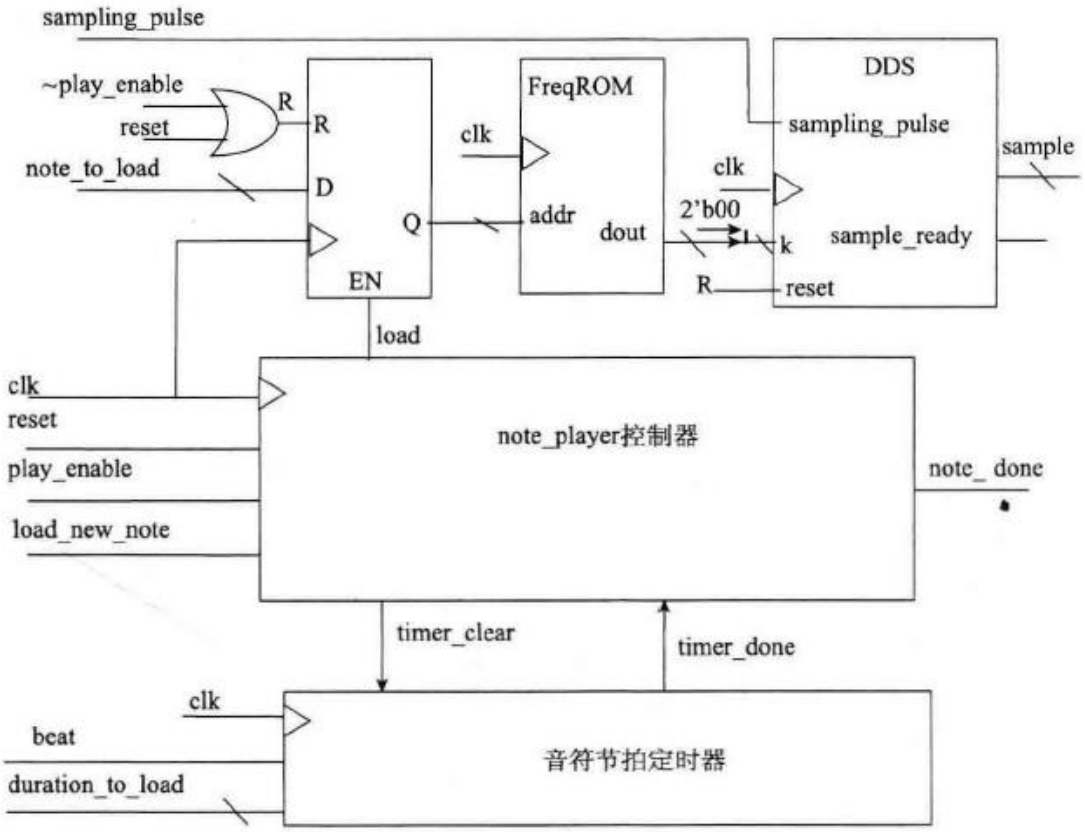


图 9　note_player 模块的结构框图

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。

各端口的含义如表 4 所示：

表 4　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

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。其控制器的算法流程图如图 10 所示：

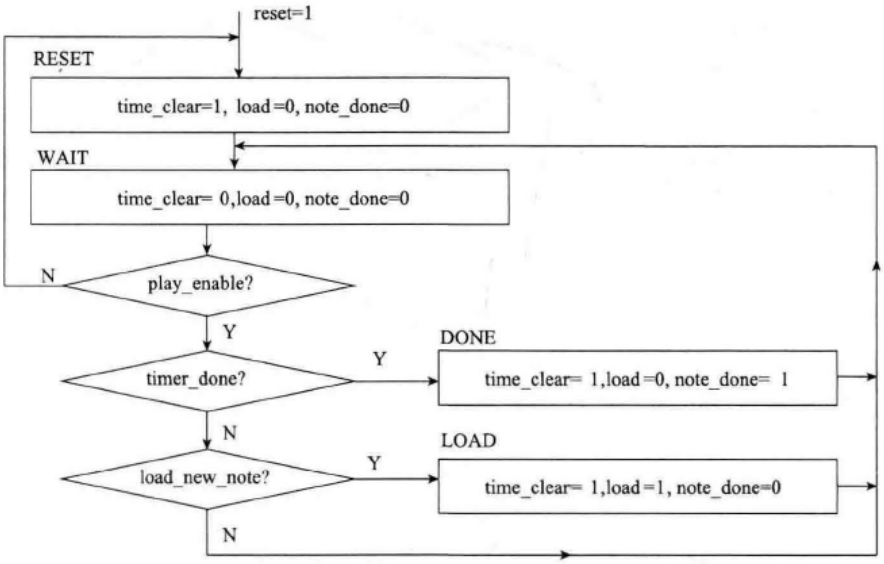


图 10　note_player 控制器的算法流程图

音符定时器为 6 位二进制计数器，beat、timer_clear 分别为使能、清零信号，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。由于 DDS 模块的输入 k 为 22 位二进制，因此需要 FreqROM 输出的 20 位相位增量高位加 2 个 0 后接入 DDS。

4、同步化电路

由同步器和脉冲宽度变换电路组成，电路如图 11 所示：

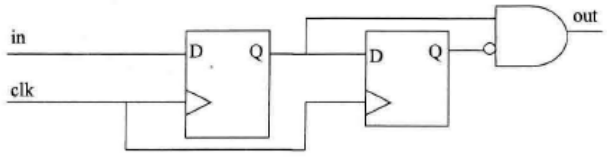


图 11　同步化电路

四、主要仪器设备

ModelsimSE-64 10.4、Vivado2017.4、Nexys Video 开发板、有源耳机

五、实验过程及仿真结果

1、DDS

1.1 Verilog HDL 代码设计

DDS 模块由顶层模块 dds.v 和子模块加法器 fulladder_n.v、D 触发器 dffre.v、正弦查找表 sine_rom.v 组成。

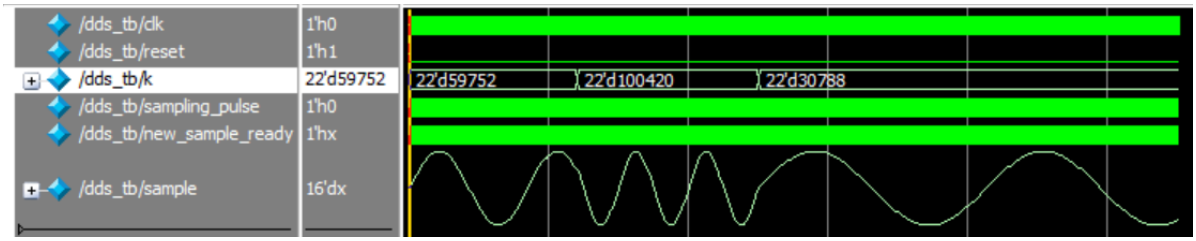
顶层模块 dds 代码如下：

```
module dds (
    clk,
    reset,
    k,                              //相位增量
    sampling_pulse,                 //采样脉冲
    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;           //地址处理输入
    wire [21:0] fulladder_out;      //加法器输出
    wire [9:0]rom_addr;             //ROM 地址
    wire [15:0]raw_data;            //ROM 数据输出
    wire [15:0]data;                //数据处理输出
    wire area;                      //sine 分区
    //加法器实现相位增量
    fulladder_n #(.n(22)) fulladder1(.a(k), .b(raw_addr), .s(fulladder_out), .
ci(0), .co());
    //D 寄存器存储得到的相位
    dffre #(.n(22))ff0(.d(fulladder_out), .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];
    //ROM 查表
    sine_rom rom1(.clk(clk), .addr(rom_addr), .dout(raw_data));
    //D 寄存器得到 area 分区
    dffre #(.n(1))ff1(.clk(clk), .d(raw_addr[21]), .en(1), .r(0), .q(area));
    //数据处理
    assign data = area?(~raw_data[15:0]+1):raw_data[15:0];
    //D 寄存器获得 sample
```

```
    dffre #(.n(16))ff2(.d(data), .en(sampling_pulse), .r(0), .clk(clk), .q(sam
ple));
    //D 寄存器得到 new_sample_ready
    dffre #(.n(1))ff3(.d(sampling_pulse), .en(1), .r(0), .clk(clk), .q(new_sam
ple_ready));
endmodule
```

1.2 仿真测试



分析：由仿真结果可见，当 k 变大时，输出正弦波的频率变大；在 k 改变时，输出正弦波频率发生突变，仿真图中结果符合设计要求。

2、mcu

2.1 Verilog HDL 代码设计

mcu 模块由顶层模块 mcu.v 和子模块控制器 mcu_ctrl.v、计数器 counter_n.v 组成。

顶层模块 mcu 代码如下：

```
module mcu (
    clk,         //100MHz 时钟信号
    reset,       //复位信号，高电平有效
    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
    next,        //来自按键处理的“下一首”控制信号，一个时钟周期宽度
    song_done,   //song_reader 的应答信号，一个时钟周期宽度的高电平脉冲,表示播放结束
    play,        //输出控制 song_reader 模块是否播放
    reset_play, //时钟周期宽度的高电平复位脉冲，同时复位 song_reader&note_player
    song         //当前播放的乐曲序号
);
    input clk, reset, play_pause, next, song_done;
    output play, reset_play;
    output [1:0] song;

    wire NextSong;  //2 位二进制计数器使能信号，next 为高电平时改变当前播放乐曲序号
    //mcu 控制器
    mcu_ctrl mcu_ctrl1(
        .clk(clk),
        .reset(reset),
        .play_pause(play_pause),
        .next(next),
        .song_done(song_done),
        .play(play),
```

```
        .reset_play(reset_play),
        .NextSong(NextSong)
    );
    //歌曲计数器
    counter_n #(.n(4), .counter_bits(2)) song_counter(
        .clk(clk),
        .en(NextSong),
        .r(reset),
        .q(song),
        .co()
    );
endmodule
```

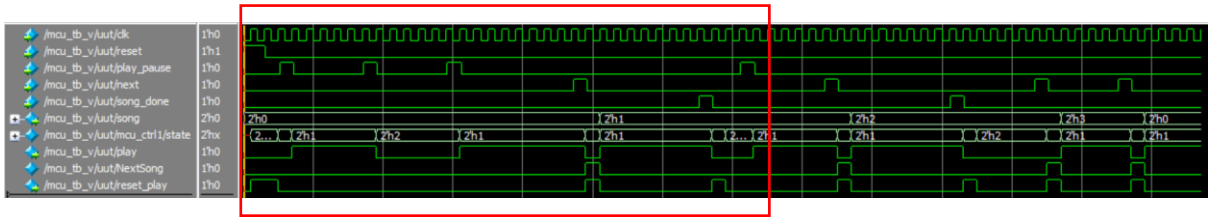
控制器模块 mcu_ctrl 代码如下：

```
module mcu_ctrl (
    clk,
    reset,
    play_pause,
    next,
    song_done,
    play,
    reset_play,
    NextSong
);
    parameter RESET = 0, PLAY = 1, PAUSE = 2, NEXT = 3;
    input clk, reset, play_pause, next, song_done;
    output reg play, reset_play, NextSong;
    reg[1:0] state, nextstate;
    //第一段-时序电路：D 寄存器
    always @(posedge clk) begin
        if(reset) state = RESET;
        else state = nextstate;
    end
    //第二段-组合电路：下一状态和输出电路
    always @(*) begin
        play = 0; reset_play = 0;NextSong = 0; //默认为 0
        case(state)
            RESET: begin
                reset_play = 1;
                nextstate = PAUSE;
            end
            PAUSE: begin
                if(play_pause) nextstate = PLAY;
                else if(next)    nextstate = NEXT;
```

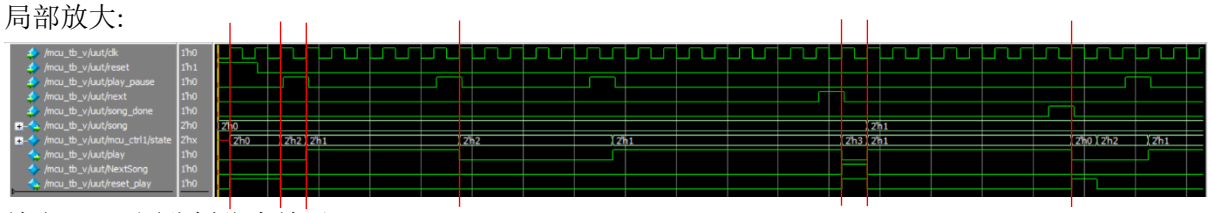
```
                    else nextstate = PAUSE;
            end
            PLAY: begin
                play = 1;
                if(play_pause) nextstate = PAUSE;
                else begin
                    if(next) nextstate = NEXT;
                    else begin
                        if(song_done) nextstate = RESET;
                        else nextstate = PLAY;
                    end
                end
            end
            NEXT: begin
                NextSong = 1; reset_play = 1;
                nextstate = PLAY;
            end
        endcase
    end
endmodule
```

2.2 仿真测试

全局：



局部放大:



结合 ASM 图分析仿真结果：

直线 1：在时钟第一个上升沿到来时，reset = 1，进入 state = RESET 状态，此时输出 play = 0，NextSong = 0，reset_play = 1，符合 ASM 图;

直线 2：时钟上升沿到来时，reset = 0，state 由 RESET 无条件转入 PAUSE，此时输出 play = 0，NextSong = 0，reset_play = 0，符合 ASM 图;

直线 3：时钟上升沿到来时，reset = 0，play_pause = 1，即 play/pause 按钮按下，state 由 PAUSE 转入 PLAY，输出 play = 1，NextSong = 0，reset_play = 0，符合 ASM 图;

直线 4：时钟上升沿到来时，reset = 0，play_pause = 1，即 play/pause 按钮按下，state 由 PLAY 转入 PAUSE，输出 play = 0，NextSong = 0，reset_play = 0，符合 ASM 图;

直线 5：时钟上升沿到来时，reset = 0，play_pause = 0，next = 1，即 next 按钮按下，state 由 PAUSE 转入 NEXT，输出 play = 0，NextSong = 1，reset_play = 1，符合 ASM 图;

直线 6：时钟上升沿到来时，reset＝0，state 由 NEXT 无条件转入 PLAY，输出 play＝1，NextSong＝0，reset_play＝0，符合 ASM 图;

直线 7：时钟上升沿到来时，reset＝0，play_pause＝0，next＝0，song_done＝1，state 由 PLAY 转入 RESET，输出 play＝0，NextSong＝0，reset_play＝1，符合 ASM 图;

综上，仿真结果符合设计要求。

3、song_reader

3.1 Verilog HDL 代码设计

song_reader 模块由顶层模块 song_reader.v 和子模块控制器 song_reader_ctrl.v、地址计数器 counter_n.v、乐曲只读存储器 song_rom.v、结束判断模块 is_over.v 组成。

顶层模块 song_reader 代码如下：

```
module song_reader (
    clk,                //100MHz 时钟信号
    reset,              //复位信号，高电平有效
    play,               //来自 mcu 的控制信号，高电平有效
    song,               //来自 mcu 的控制信号，当前播放歌曲的序号
    note_done,          //note_player 的应答信号，表示一个音符播放结束并索取新音符
    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
    note,               //音符标记
    duration,           //音符的持续时间
    new_note            //给模块 note_player 的控制信号，表示新的音符需播放
);
    input clk, reset, play, note_done;
    input [1:0] song;
    output song_done, new_note;
    output [5:0] note, duration;

    wire[4:0] q;        //歌曲音符地址（五位）
    wire co;            //歌曲结束信号
    //控制器
    song_reader_ctrl song_reader1(
        .clk(clk),
        .reset(reset),
        .note_done(note_done),
        .play(play),
        .new_note(new_note)
    );
    //地址计数器
    counter_n #(.n(32), .counter_bits(5))song_choose(
        .clk(clk),
        .en(note_done),
        .r(reset),
        .q(q),
```

```
        .co(co)
    );
    //song_rom
    song_rom song1(
        .clk(clk),
        .dout({note[5:0], duration[5:0]}),
        .addr({song[1:0],q[4:0]})
    );
    //结束判断
    is_over is_over1(
        .clk(clk),
        .duration(duration),
        .r(co),
        .out(song_done)
    );
endmodule
```

控制器模块 song_reader_ctrl 代码如下：

```
module song_reader_ctrl (
    clk,
    reset,
    note_done,
    play,
    new_note
);
    input clk, reset, note_done, play;
    output reg new_note;
    parameter RESET = 0, NEW_NOTE = 1, WAIT = 2, NEXT_NOTE = 3;
    reg [1:0] state , nextstate;
    //第一段-时序电路：D 寄存器
    always @(posedge clk) begin
        if(reset) state = RESET;
        else state = nextstate;
    end
    //第二段-组合电路：下一状态和输出电路
    always @(*) begin
        new_note = 0; //默认为 0
        case (state)
            RESET: begin
                if(play) nextstate = NEW_NOTE;
                else nextstate = RESET;
            end
            NEW_NOTE: begin
               new_note = 1;
```

```
                nextstate = WAIT;
            end
            WAIT: begin
                if(!play) nextstate = RESET;
                else if(note_done) nextstate = NEXT_NOTE;
                     else nextstate = WAIT;
            end
            NEXT_NOTE: begin
                nextstate = NEW_NOTE;
            end
        endcase
    end
endmodule
```

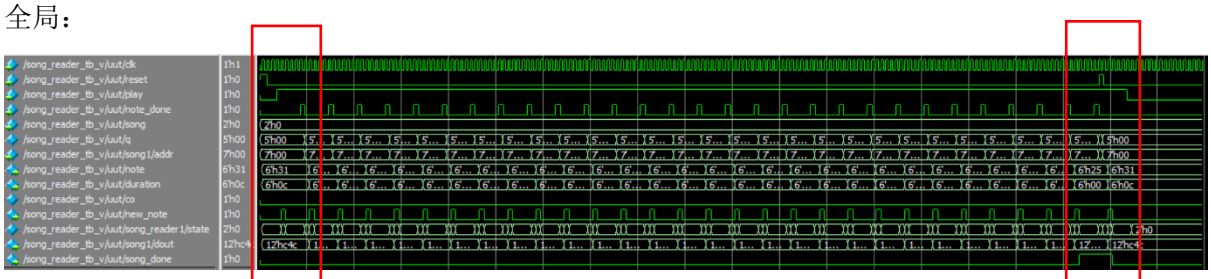
结束判断模块 is_over 代码如下：

```
//判断是否结束模块
module is_over (
    clk,
    duration,   //音符长度
    r,          //co 与此连接，32 个音符是否播放完
    out         //song_done
);
    parameter OVER = 0, PLAY = 1;
    input clk, r;
    input [5:0] duration;
    output reg out;
    reg state, nextstate;
    //第一段-时序电路：D 寄存器
    always @(posedge clk) begin
        state = nextstate;
    end
    //第二段-组合电路：下一状态和输出电路
    always @(*) begin
        out = 0;    //默认为 0
        case (state)
            OVER:begin
                out = 1;
                if(duration) nextstate = PLAY;
                else  nextstate = OVER;
            end
            PLAY:begin
                if(duration == 0 || r) begin nextstate = OVER; end
                else begin
                    nextstate = PLAY;
```

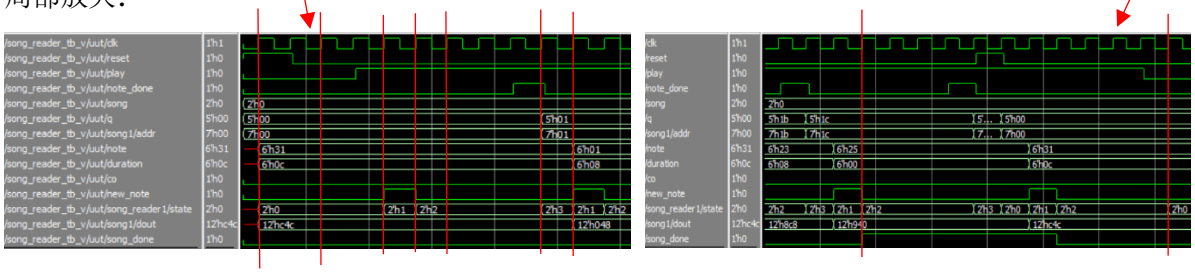
```
                end
            end
            default: nextstate = OVER;
        endcase
    end
endmodule
```

3.2 仿真测试

全局：



局部放大：



结合 ASM 图分析仿真结果：

直线 1：时钟上升沿到来时，reset = 1，进入 RESET 状态，输出 new_note = 0，符合 ASM 图;

直线 2：时钟上升沿到来时,reset=0，play=0，state 转回 RESET，输出 new_note=0,符合 ASM 图；

直线 3：时钟上升沿到来时，reset=0，play = 1，state 由 RESET 转入 NEW_NOTE，输出 new_note=1，表示新的音符需要播放，符合 ASM 图；

直线 4：时钟上升沿到来时，reset=0，state 由 NEW_NOTE 无条件转入 WAIT，输出 new_note=0，符合 ASM 图；

直线 5：时钟上升沿到来时，reset = 0，play = 1，note_done = 0，state 由 WAIT 转入 WAIT，输出 new_note = 0，符合 ASM 图；

直线 6：时钟上升沿到来时，reset = 0，play = 1，note_done = 1，表示该音符播放完毕，state 由 WAIT 转入 NEXT_NOTE,输出 new_note = 0，符合 ASM 图，地址计数器加 1，指向下一个音符；

直线 7：时钟上升沿到来时，reset = 0，state 由 NEXT_NOTE 无条件转入 NEW_NOTE，输出 new_note = 1，新音符{note，duration}成功读取，符合 ASM 图；

直线 8：时钟上升沿到来时，reset = 0，play = 1，duration = 0，输出 song_done = 1，表示歌曲播放结束（不是一个时钟周期宽度，应该是由于状态机设置的不合理，改进部分已补充在文档末尾）；

直线 9：时钟上升沿到来时，reset = 1，play = 0，state 由 WAIT 转入 RESET，输出 new_note =0，播放停止，符合 ASM 图；

4、note_player

4.1 Verilog HDL 代码设计

note_player 模块由顶层模块 note_player.v 和子模块控制器 note_player_ctrl.v、D 触发器 dffre.v、相位增量查找表 frequency_rom.v、DDS 模块 dds.v、音符节拍定时器 timer.v 组成。

顶层模块 note_player 代码如下：

```
module note_player(
    clk,                //时钟信号
    reset,              //复位信号,来自 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
    duration_to_load,   //来自 song_reader 模块的音符时长 duration
    load_new_note,      //来自 song_raeder 模块的 new_note 信号，表示音符播放完毕
    note_done,          //给 song_reader 模块的应答信号，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，表示索取新的
样品
    beat,               //定时基准信号，频率为 48kHz
    sample,             //正弦样品输出
    sample_ready        //下一个正弦信号
);
    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, timer_clr, timer_done;
    wire[5:0] q;
    wire [19:0] dout;
    //控制器
    note_player_ctrl note_player_ctrl1(
        .clk(clk),
        .reset(reset),
        .play_enable(play_enable),
        .load_new_note(load_new_note),
        .note_done(note_done),
        .load(load),
        .timer_clr(timer_clr),
        .timer_done(timer_done)
    );
    //D 寄存器
    dffre #(.n(6))ff0(
        .clk(clk),
        .d(note_to_load),
        .en(load),
        .r(~play_enable || reset),
        .q(q)
    );
    //Frequency ROM
    frequency_rom freq1(
        .clk(clk),
        .dout(dout),
```

```
        .addr(q)
    );
    //DDS
    dds dds1(
        .clk(clk),
        .reset(reset || (~play_enable)),
        .k({2'b00, dout}),
        .sampling_pulse(sampling_pulse),
        .new_sample_ready(sample_ready),
        .sample(sample)
    );
    //音符节拍计时器
    timer #(.n(64), .counter_bits(6))timer1(
        .clk(clk),
        .r(timer_clr),
        .en(beat),
        .count_number(duration_to_load),
        .done(timer_done)
    );

endmodule
```

控制器 note_player_ctrl 代码如下：

```
module note_player_ctrl (
    clk,
    reset,
    play_enable,
    load_new_note,
    note_done,
    load,
    timer_clr,
    timer_done
);
    input clk, reset, play_enable, load_new_note, timer_done;
    output reg timer_clr, note_done, load;
    parameter  RESET = 0, WAIT = 1, DONE = 2, LOAD = 3;
    reg[1:0] state, nextstate;
    //第一段-时序电路：D 寄存器
    always @(posedge clk) begin
        if(reset) state = RESET;
        else state = nextstate;
    end
    //第二段-组合电路：下一状态和输出电路
    always @(*) begin
        timer_clr = 0; load = 0; note_done = 0; //默认为 0
```

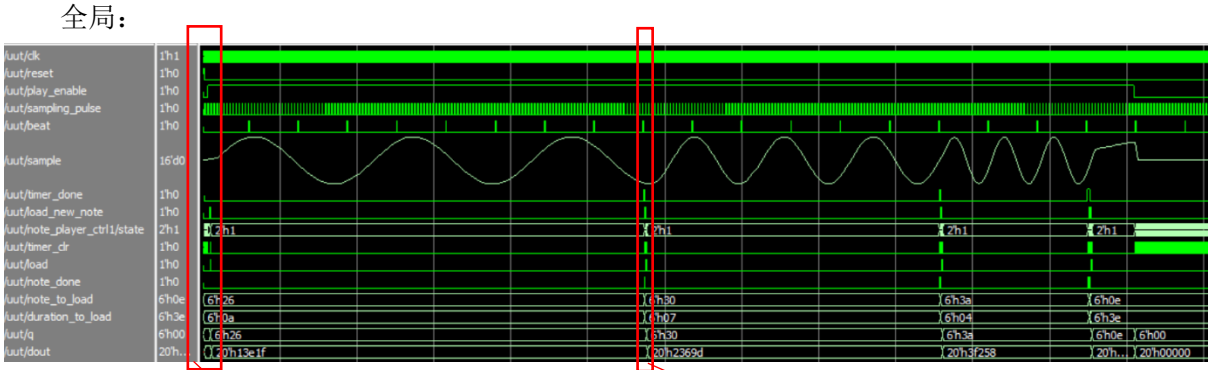
```
        case (state)
            RESET:begin
                timer_clr = 1;
                nextstate = WAIT;
            end
            WAIT:begin
                if(~play_enable) nextstate = RESET;
                else begin
                    if(timer_done) nextstate = DONE;
                    else begin
                        if(load_new_note) nextstate = LOAD;
                        else nextstate = WAIT;
                    end
                end
            end
            DONE:begin
                timer_clr = 1; note_done = 1;
                nextstate = WAIT;
            end
            LOAD:begin
                timer_clr = 1; load = 1;
                nextstate = WAIT;
            end
        endcase
    end
endmodule
```

音符节拍定时器 timer 代码如下：

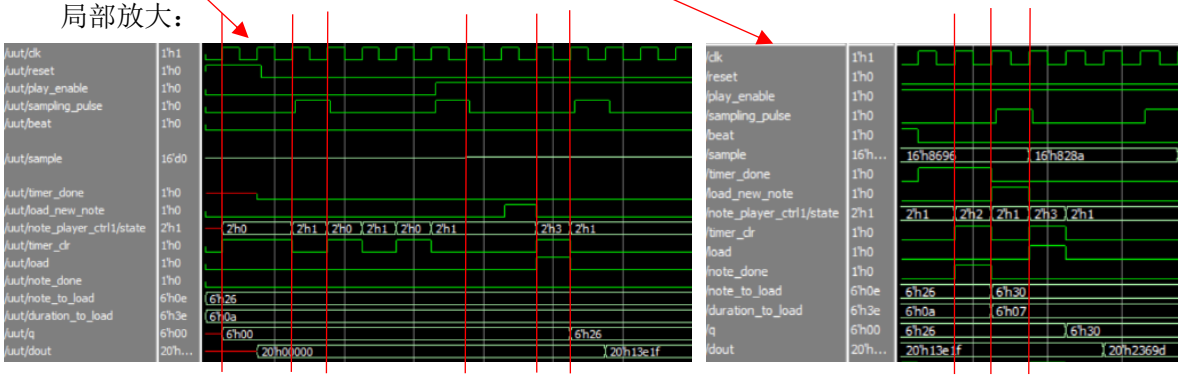
```
module timer(clk, r, en, done, count_number);
    parameter n = 2;
    parameter counter_bits=1; //参数 counter_bits 表示计数的位数
    input clk, r, en;
    input[counter_bits-1:0] count_number;
    output done;
    reg [counter_bits-1:0] q;
    assign done = (q == count_number-1);
    always @(posedge clk)
        begin
            if(r) q = 0;
            else if(en) q = q+1;
                 else q = q;
        end
endmodule
```

4.2 仿真测试

全局：



局部放大：



整体上看，输出信号 sample 的频率与输入的音符 q 成正比，当 play_enable = 0 时，sample = 0；结合 ASM 图分析：

直线 1：时钟上升沿到来时，reset = 1，进入 RESET 状态，输出 timer_clr = 1，load = 0，note_done=0，符合 ASM 图；

直线 2：时钟上升沿到来时，reset = 0，state 由 RESET 无条件转入 WAIT，输出 time_clr = 0，load = 0，note_done = 0，符合 ASM 图；

直线 3：时钟上升沿到来时，reset = 0，play_enable = 0，state 由 WAIT 转入 RESET，输出 timer_clr=1，load = 0，note_done=0，符合 ASM 图；

直线 4：时钟上升沿到来时，reset = 0，play_enable = 1，timer_done = 0，load_new_note = 0，stste 保持在 WAIT，输出 time_clr = 0，load = 0，note_done = 0，符合 ASM 图；

直线 5：时钟上升沿到来时，reset = 0，play_enable = 1,timer_done = 0,load_new_note = 1, state 由 WAIT 转入 LOAD，输出 time_clr = 1，load = 1，note_done = 0，符合 ASM 图，下一个时钟上升沿寄存器输出 q 转变为新的 note；

直线 6：时钟上升沿到来时，reset = 0，state 由 LOAD 无条件转入 WAIT，输出 time_clr = 0，load = 0，note_done = 0，符合 ASM 图，下一个时钟上升沿从 FreqROM 中读出对应的相位增量 dout；

直线 7：音符节拍定时器模块 timer 计时结束，发出 timer_done = 1 至控制器，reset=0,play_enable=0, state 由 WAIT 转入 DONE，输出 timer_clr = 1 复位音符节拍定时器模块 timer，load = 0，note_done=1；

直线 8：时钟上升沿到来时，reset = 0，state 由 DONE 无条件转入 WAIT，输出 time_clr = 0，load = 0，note_done = 0，符合 ASM 图；

直线 9：时钟上升沿到来时，reset = 0，play_enable = 1,timer_done = 0,load_new_note = 1, state 由 WAIT 转入 LOAD，输出 time_clr = 1，load = 1，note_done = 0，加载新音符符合 ASM 图；

5、synch 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。

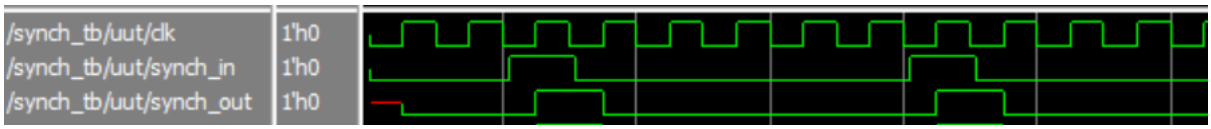
本例中音频编解码接口模块的输出信号 NewFrame 的脉冲宽度为一个 audio_clk 时钟周期，需通过同步化处理，产生与 sys_clk 同步且脉冲宽度为一个 sys_clk 时钟周期的信号 ready。

5.1 Verilog HDL 代码设计

```
//同步器
module synch(clk, synch_in, synch_out);
    input clk, synch_in;
    output synch_out;

    wire q0, q1;
    dffre ff0(.clk(clk), .en(1), .r(0), .d(synch_in), .q(q0));
    dffre ff1(.clk(clk), .en(1), .r(0), .d(q0), .q(q1));
    assign synch_out = (~q1) && q0;
endmodule
```

5.2 仿真测试



输出 synch_out 与系统时钟 clk 同步，满足设计要求。

6、music_player

6.1 Verilog HDL 代码设计

music_player 模块由子模块主控制器 mcu.v、乐曲读取 song_reader.v、音符播放 note_player.v、同步化电路 synch.v 和节拍基准产生器 counter_n.v 组成。

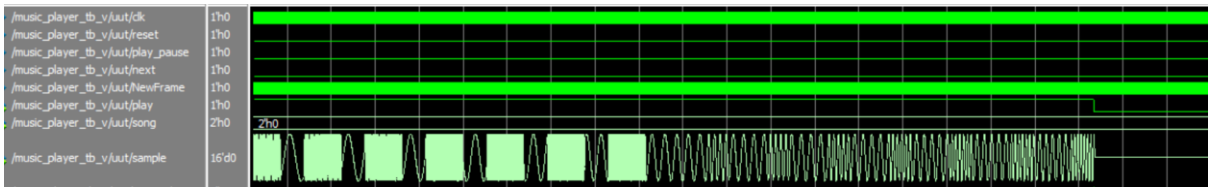
顶层模块 music_player 代码如下：

```
module music_player (
    clk,
    reset,
    play_pause,
    next,
    NewFrame,
    sample,
    play,
    song
);
    parameter sim = 0;
    input clk, reset, play_pause, next, NewFrame;
    output[15:0] sample;
    output play;
    output[1:0] song;
```

```
    wire reset_play, song_done, new_note, note_done, ready, beat, sample_ready
;
    wire [5:0] note, duration;
    //主控制器
    mcu mcu1(
        .clk(clk),
        .reset(reset),
        .play_pause(play_pause),
        .next(next),
        .play(play),
        .song(song),
        .reset_play(reset_play),
        .song_done(song_done)
    );
    //song_reader
    song_reader song_reader1(
        .clk(clk),
        .reset(reset_play),
        .play(play),
        .song(song),
        .song_done(song_done),
        .note(note),
        .duration(duration),
        .new_note(new_note),
        .note_done(note_done)
    );
    //note_player
    note_player note_player1(
        .clk(clk),
        .reset(reset_play),
        .play_enable(play),
        .note_to_load(note),
        .duration_to_load(duration),
        .note_done(note_done),
        .load_new_note(new_note),
        .beat(beat),
        .sampling_pulse(ready),
        .sample(sample),
        .sample_ready(sample_ready)
    );
    //synchronizer
    synch synch1(
        .clk(clk),
        .synch_in(NewFrame),
```

```
        .synch_out(ready)
    );
    //节拍基准产生器 分频器
    counter_n #(.n(sim?64:1000), .counter_bits(sim?6:10))div1(
        .clk(clk),
        .en(ready),
        .r(0),
        .q(),
        .co(beat)
    );
endmodule
```

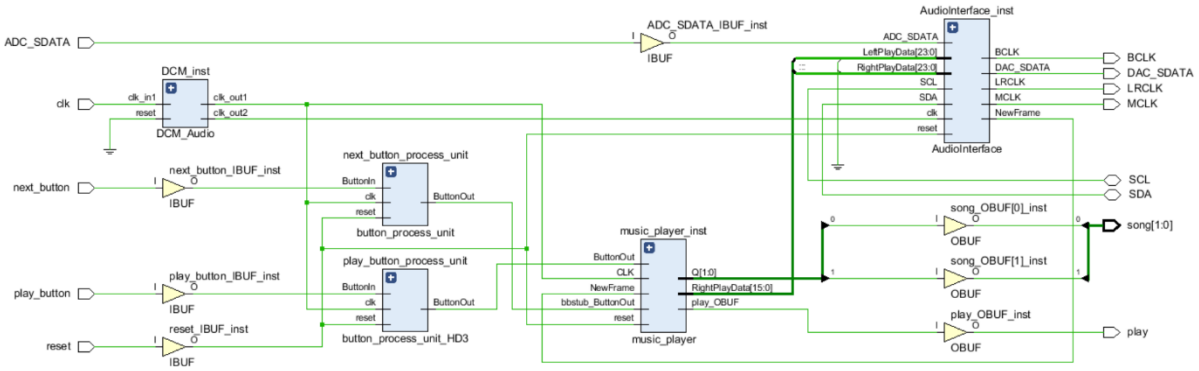
6.2 仿真测试



观察仿真结果可知，波形符合要求。

六、Vivado 工程

Vivado 综合出的原理图如下图所示：



生成比特流下载至开发板，将耳机接入开发板音频输出插座，操作 reset、play/pause、next 三个按键，试听耳机中的乐曲并观察实验板上指示灯变化情况，与预期实验结果较为符合。

七、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？

因为系统中，reset 一旦有效，系统内部会被统一复位，即 reset 的下一个状态的目的有且仅有 RESET 一个，reset 按键的颤动只会引起多次复位，其效果与一次复位效果相同；而 next_button、play_pause_button 两个按键有效时，其次态有多种情况，且会随现态改变，若产生颤动，则效果相当于 next_button、play_pause_button 多次生效，系统会在多个状态切换，例如 play_pause_button 按

钮颤动，可能会造成次态应是 PLAY 的而由于颤动多次生效造成次态可能变成 PAUSE。

（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。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。

八、实验心得

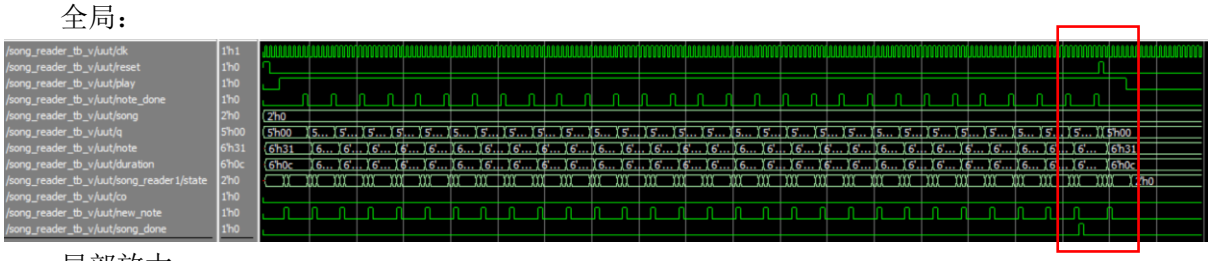
本次实验是一个比较大的项目，一开始感觉无从下手。刚开始做的时候，就一点一点的看实验 15 和 19 的教材内容，直到感觉大概知道整个项目的过程和原理了才着手做，不过由于看实验 15 和 19 花费了较长时间，关于 Verilog HDL 的内容又不怎么熟悉了，就又回去看 Verilog HDL 部分内容。在设计 mcu 控制器的时候，因为之前没有设计过关于状态机的代码，就去翻教材 2.4.2 状态机的 Verilog HDL 描述方法，跟着教材的思路设计了 mcu 控制器，一开始由于没有理解，也不懂什么是 ASM 图，波形不符合要求，具体什么问题记不太清楚了，后来仔细分析 ASM 图和代码结构，搞清楚了后，接下来模块的控制器也就很快解决了。但很多信号应该是什么样的，起到什么作用还是比较模糊，当时跟着书上原理图的连线将端口匹配上就好了，直到写实验报告时更深入的分析每一个信号的高低电平变换，才理解了整个系统的运作。整个实验完成后，还是很有成就感的。也深刻体会到自顶而下、模块化设计对数字电路的设计很有帮助，可以降低不少工作量。

九、订正补充：

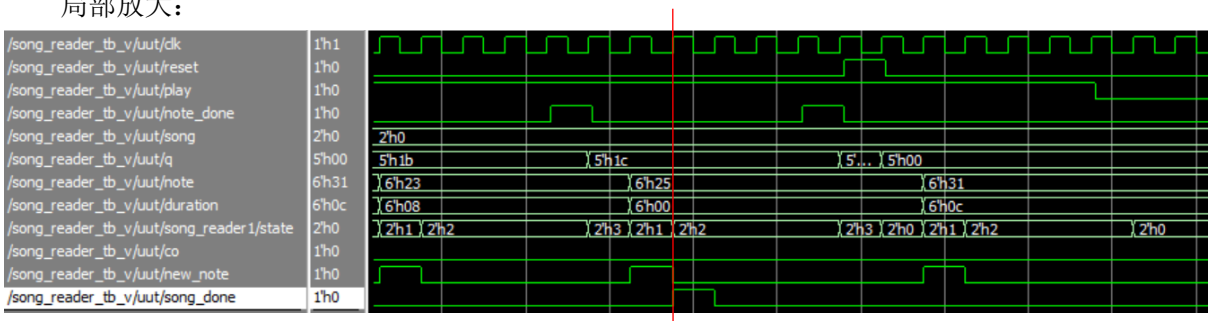
该部分为写实验报告详细分析时发现的错误之处，由于没有进行修改结果烧入 fpga 验证，不确定正确与否，故添加至此：

（1）撰写实验报告时发现 song_reader 模块的 song_done 信号与预期不符，修改原本的状态机设置，重新仿真，结果如下图所示：

全局：

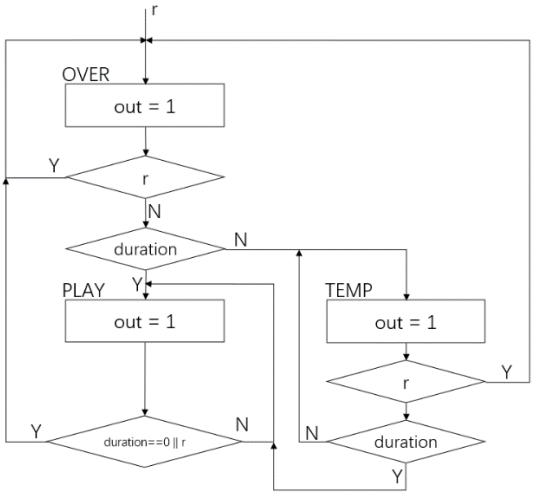


局部放大：



直线处：时钟上升沿到来时，reset＝0，play＝1，duration＝0，输出 song_done＝1，表示歌曲播放结束，song_done 脉冲宽度为一个时钟周期。

算法流程图设计如下图所示：



代码如下所示：

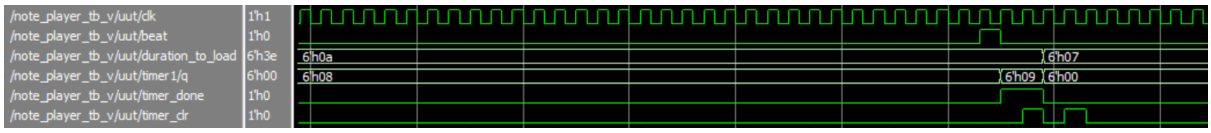
```
//判断是否结束模块
module is_over (
    clk,
    duration,   //音符长度
    r,          //co 与此连接，32 个音符是否播放完
    out         //song_done
);
    parameter OVER = 0, PLAY = 1, TEMP = 2;
    input clk, r;
    input [5:0] duration;
    output reg out;
    reg [1:0] state = PLAY, nextstate;
    //第一段-时序电路：D 寄存器
    always @(posedge clk) begin
        state = nextstate;
    end
    //第二段-组合电路：下一状态和输出电路
    always @(*) begin
        out = 0;    //默认为 0
        case (state)
            OVER:begin
                out = 1;
                if(r) nextstate = OVER;
                else begin
                    if(duration) nextstate = PLAY;
                    else  nextstate = TEMP;
                end
            end
            PLAY:begin
```

```
                if(duration == 0 || r) begin nextstate = OVER; end
                else begin
                    nextstate = PLAY;
                end
            end
            TEMP:begin
                if(r) nextstate = OVER;
                else begin
                    if(duration != 0) nextstate = PLAY;
                    else nextstate = TEMP;
                end
            end
            default: nextstate = TEMP;
        endcase
    end
endmodule
```

（2）验收时，唐奕老师说音乐节奏整体偏快一些，当时一时间也无法分析出什么原因，写实验报告分析波形时，发现了我认为的原因，duration_to_load 表示音符该持续的长度，q 为音符节拍计数器的计数值，下图中 duration_to_load＝h0A，因此 q 应当从 0 开始计数完整的 10 个数，即 9 应当完整计数，而图中当 q＝9，timer 即发出 timer_done＝1，后续控制器复位 timer，导致整个音符持续时间缩短近 1/10。基于上述分析，最简单的方法仅需将 timer.v 稍作修改，

将 assign done = (q == count_number-1);一句修改为 assign done = (q == count_number);　，修改后的结果因当是完整的音符长度加 2 个 clk 时钟周期，而时钟周期极小可以忽略。

如下所示：




```
module timer(clk, r, en, done, count_number);
    parameter n = 2;
    parameter counter_bits=1; //参数 counter_bits 表示计数的位数
    input clk, r, en;
    input[counter_bits-1:0] count_number;
    output done;
    reg [counter_bits-1:0] q;

    assign done = (q == count_number-1);
    always @(posedge clk)
        begin
            if(r) q = 0;
            else if(en) q = q+1;
                 else q = q;
        end
endmodule
```