浙江大学

本科实验报告

课程名称：　　数字系统设计实验（周三 9，10 节）

姓　　名：

学　　院：　　信息与电子工程学院

专　　业：　　电子科学与技术

学　　号：

指导教师：　　屈民军、唐奕

2020 年　 6 月　 16　日

浙江大学实验报告

专业：电子科学与技术
姓名：
学号：

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：音乐播放实验

一、实验目的　二、实验任务与要求
三、实验原理
四、主要仪器设备　五、实验步骤与过程
六、实验调试、实验数据记录　七、实验结果和分析处理
八、讨论、心得

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、设计并仿真一个 DDS 正弦信号发生器，要求：

（1）采样频率 fc = 48kHz；

（2）正弦信号频率范围为 20Hz～20kHz；

（3）正弦信号序列宽度 16 位，包括一位符号；

2、设计一个音乐播放器，要求：

（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 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

要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 技术通过改变寻址的步长来改变输出信号的频率，步长即为对数字波形查表的相位增量，输出正弦频率与相位增量成线性关系。

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

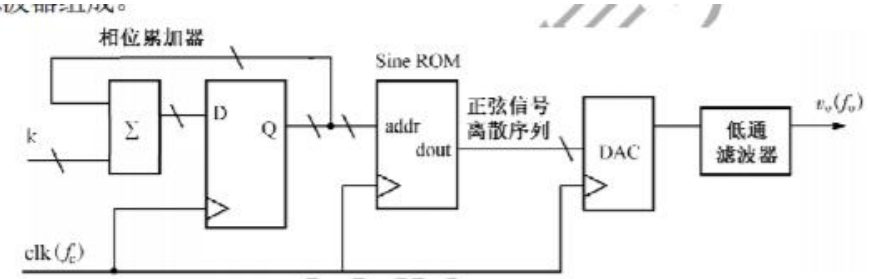


图 6.12　DDS 原理框图

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(6.14) 的映射关系构成，即

$$S(i)=(2^{n-1}-1)\times\sin\left(\frac{2\pi i}{2^m}\right)\qquad (i=0,1,2\cdots 2^{m-1}) \qquad (6.14)$$

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。

fc 为取样时钟 clk 的频率，k 为相位增量，输出正弦信号频率 fo 由 fc 和 k 共同决定，即：

$$f_o=\frac{k\times f_c}{2^m} \qquad (6.15)$$

由式(6.15)可看出，正弦信号的频率 fo 与相位增量 K 成正比关系。相位累加器的位数是由 m 位整数和 p 位小数组成。相位累加器的高 m 位整数部分作为 Sine ROM 的地址。

设计原理：

根据 DDS 输出信号的最低频率要求，可计算出 m＝12。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。

因为 m＝12，所以存储一个完整周期的正弦信号样品就需要 2^{12} × 16bits 的 ROM。但由于正弦波形的对称性，如图 6.15 所示，将正弦波形分为四个区域，只需要在 Sine ROM 中存储四分之一的正弦信号样品（0 区）即可。这样，Sine ROM 容量可减少为 2^{10} × 16bits，即 10 位地址，存储四分之一的正弦信号样品，共 1024 个。四分之一周期的正弦信号样品未给出 90° 的样品值，因此在 ROM 地址为 1024（即 90° ）时可取地址为 1023 的值（实际上地址为 1023 时，正弦信号样品已达最大值）。

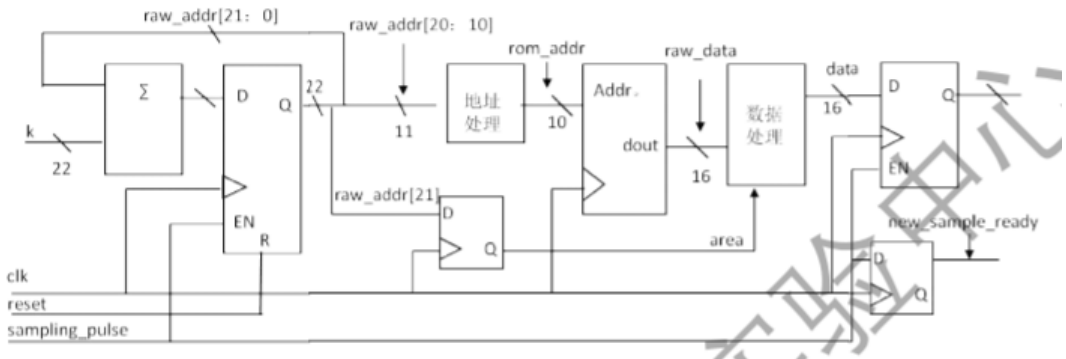


图 6.16　优化后的 DDS 结构

相位累加得到 22 位原始地址 raw_addr[21:0]，整数部分 raw_addr[21:10]即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]可区分正弦的四个区域。由于 sine_rom 只保存了四分之一周期的 1024 个样品，所以 raw_addr[21:10]不能直接作为 sine_rom 地址，必须进行必要处理，处理方法如表 6.4 所示。

表 6.4　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，如图 6.44 所示。各主要子模块作用如下。

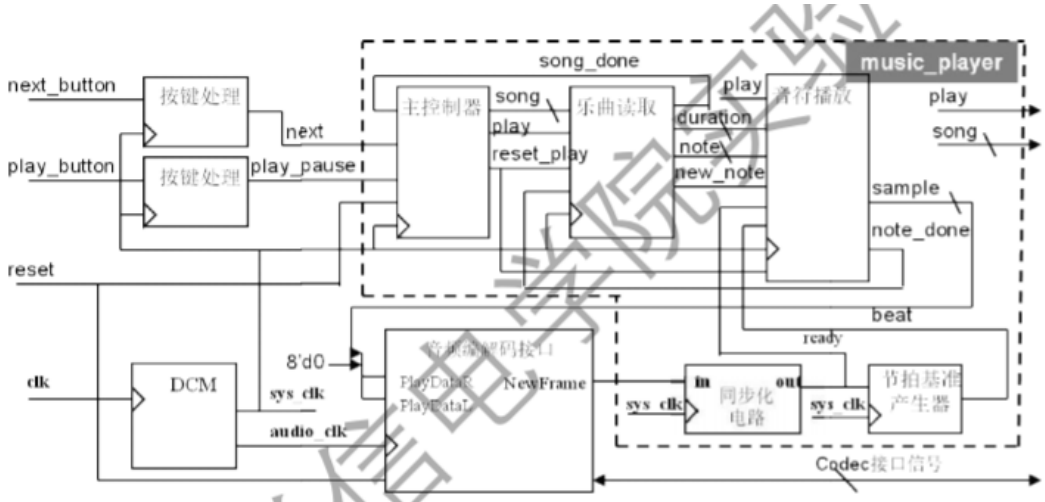


图 6.44　系统的顶层框图

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 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 6.11 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 6.45 所示。图中的 2 位二进制计数器用来计算乐曲序号（song）。

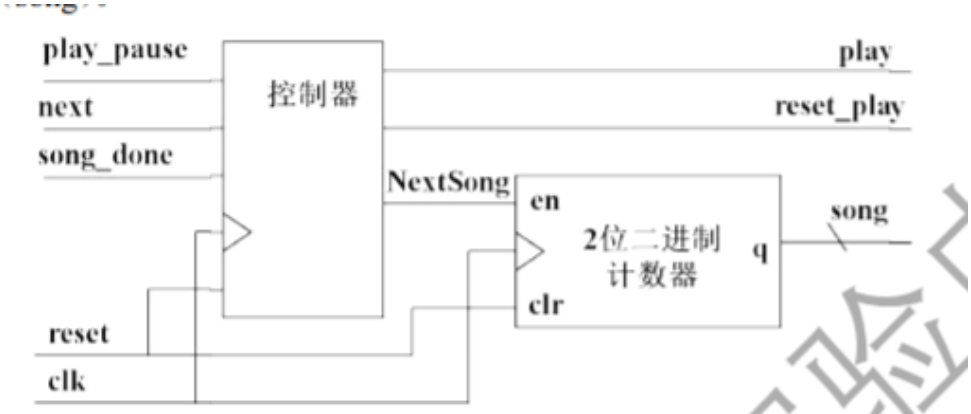


图 6.45 mcu 的结构框图

控制器的工作流程图：

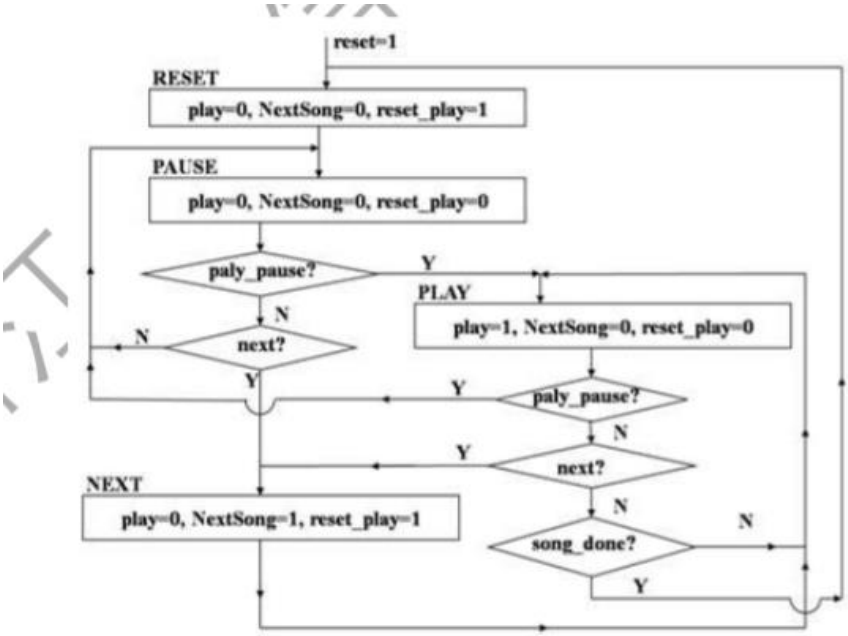


图 6.46　mcu 控制器的算法流程图

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 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 6.12　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 2^7 ×12bits。共存放四首乐曲，每首乐曲占用 2^7 ×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。

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，并作出响应。算法流程图如图 6.48 所示。

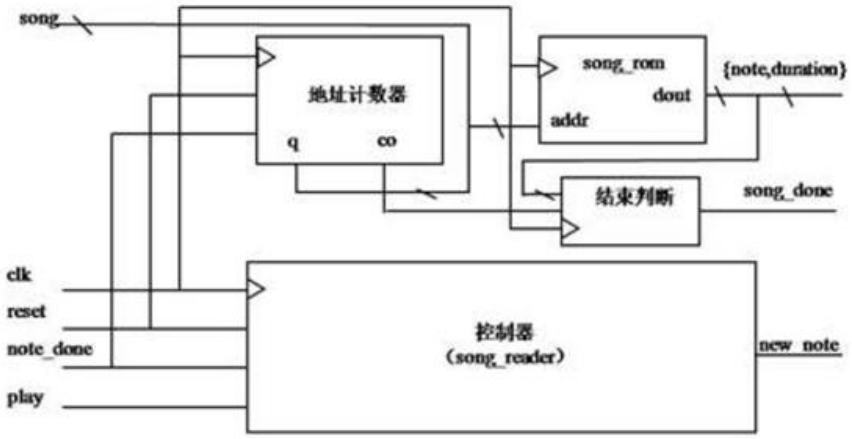


图 6.47 song_reader 的结构框图

算法流程图如下所示：

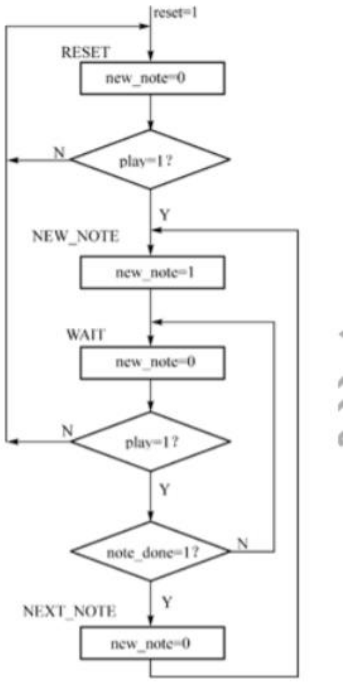


图 6.48 song_reader 控制器的算法流程

3、音乐播放模块 note_reader 设计

音符播放模块 note_player 是本实验的核心模块，它主要任务有：

（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。

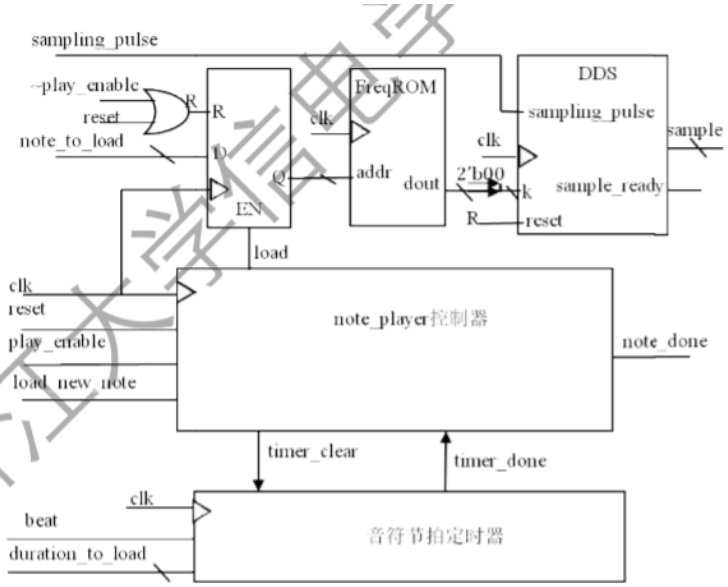


图 6.50　note_player 模块的结构框图

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

续表

引 脚 名 称	I/O	引 脚 说 明
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。note_player 控制器的算法流程如图 6.51 所示：

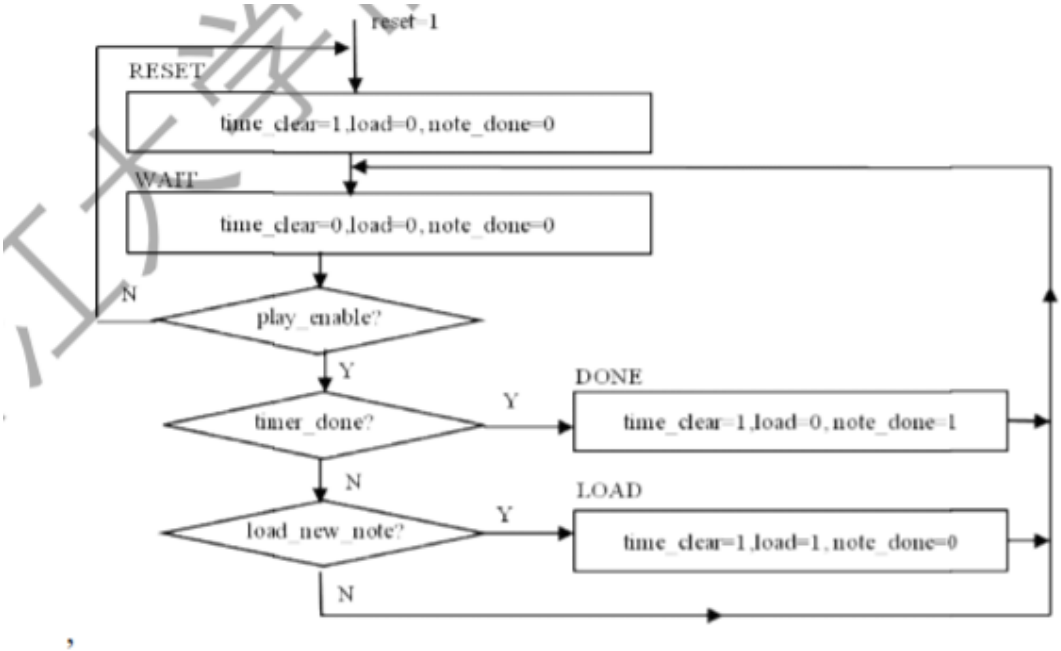


图 6.51　note_player 控制器的算法流程图

音符定时器为 6 位二进制计数器，beat、timer_clear 分别为使能、清 0 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-1 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。

四、主要仪器设备

ModelsimSE-64 10.4、Vivado2017.4

五、实验过程及仿真结果

1、DDS

1.1　Verilog HDL 代码设计

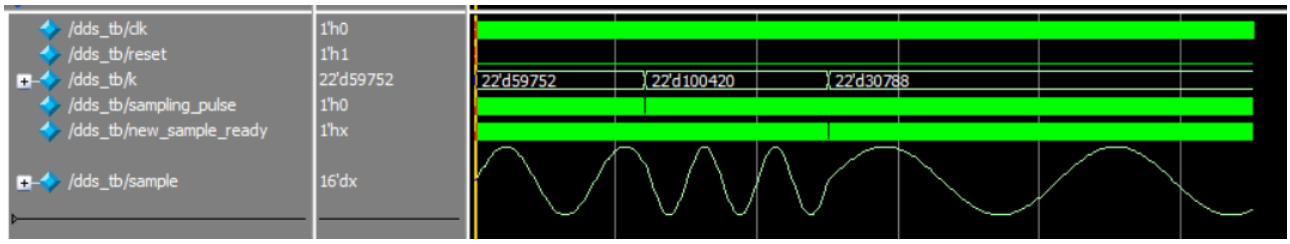
DDS 模块由顶层模块 dds.v 和子模块加法器 full_adder.v、D 触发器 dffre.v 组成。

顶层模块如下：

```
module dds(
    input clk, reset,
```

```
    input [21:0] k,           //相位增量
    input sampling_pulse,
    output new_sample_ready,
    output [15:0] sample      //正弦信号
);
wire [21:0] raw_addr;         //地址处理输入
wire [21:0] temp;             //加法器输出
wire [9:0] rom_addr;          //ROM 地址
wire [15:0] raw_data;         //ROM 输出数据
wire [15:0] data;             //数据处理输出
wire area;                    //区域
//加法器实现 temp = k + raw_addr
full_adder adder1(.a(k), .b(raw_addr), .s(temp), .co());
//D 触发器得到 row_addr
dffre#(.n(22)) dff1(.d(temp), .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 得到原始数据
sine_rom rom1(.clk(clk), .addr(rom_addr), .dout(raw_data));
//D 触发器得到 data
dffre#(.n(1)) dff2(.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 值
dffre#(.n(16)) dff3(.d(data), .en(sampling_pulse), .r(0), .clk(clk), .q(sample));
//D 触发器得到 new_sample_ready
dffre#(.n(1)) dff4(.d(sampling_pulse), .en(1), .r(0), .clk(clk), .q(new_sample_ready));
endmodule //
```

1.2 仿真测试



分析：由仿真结果可见，当 k 较大时，正弦波频率较大，k 值变化的转折点即频率变化转折点，图中体现出来是波形变化转折点。设计符合要求。

2、mcu

2.1 Verilog HDL 设计代码

该模块由顶层模块 mcu.v 和子模块控制器 mcu_ctrl.v、D 触发器 dffre.v 组成。

顶层模块如下：

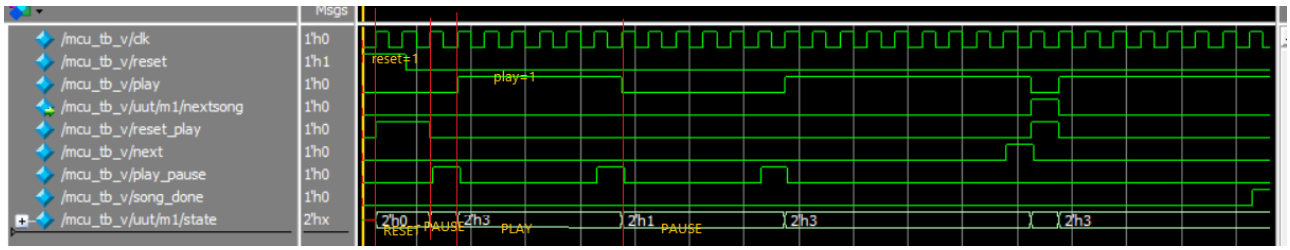
```
module mcu(
    clk,
    reset,
    play_pause,                 //play/pause 按钮
    next,                       //next 按钮
    play,                       //歌曲状态
    song,                       //歌曲序号
    reset_play,                 //脉冲复位
    song_done                   //播放完毕信号
);
input clk, reset, play_pause, next, song_done;
output play, reset_play;
output [1:0] song;
wire nextsong;
//控制器
mcu_ctrl m1(.clk(clk), .reset(reset), .play_pause(play_pause), .next(next), .play(play), .nextson
g(nextsong), .reset_play(reset_play), .song_done(song_done));
//二进制计数器
counter_n#(.n(4), .counter_bits(2)) song_count(.clk(nextsong), .r(0), .en(1), .q(song), .co());
endmodule // mcu
```

控制器代码如下：

```
module mcu_ctrl(clk, reset, play_pause, next,  play, nextsong, reset_play, song_done);
input clk, reset, play_pause, next, song_done;
output reg play, reset_play, nextsong;
parameter RESET=0, PAUSE=1, NEXT=2, PLAY=3;//状态编码
reg [1:0] state, nextstate;
//D 寄存器
always @(posedge clk) begin
    if(reset) state = RESET;
    else state = nextstate;
end
//下一状态和输出电路
always @(*)begin
    play = 0; nextsong = 0; reset_play = 0;//默认值设置为 0
    case(state)
        RESET: begin nextstate = PAUSE; reset_play = 1; end
        PAUSE: begin
            if(play_pause) nextstate = PLAY;
            else begin if(next) nextstate = NEXT;
```

```
                    else nextstate = PAUSE; end
            end
        NEXT: begin nextstate = PLAY; nextsong = 1; reset_play = 1; 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
        default: nextstate = RESET;
    endcase
end
endmodule
```

2.2 仿真测试



结合 ASM 图，分析上面四条红线：

直线 1：时钟上升沿到来前 reset = 1，当时钟上升沿到来时，state = RESET，此时输出 play = 0，nextsong = 0，reset_play = 1，符合 ASM 图。

直线 2：时钟上升沿到来前 reset = 0，当时钟上升沿到来时，state 由 RESET 自然进入 PAUSE，此时输出 play = 0，nextsong = 0，reset_play = 0，符合 ASM 图。

直线 3：时钟上升沿到来前 reset = 0，play_pause = 1，即 play 按钮按下，当时钟上升沿到来时，state = PLAY，此时输出 play = 1，nextsong = 0，reset_play = 0，符合 ASM 图。

直线 4：时钟上升沿到来前 reset = 0，play_pause = 1，play = 1，即 pause 按钮按下，当时钟上升沿到来时，state = PAUSE，此时输出 play = 0，nextsong = 0，reset_play = 0，符合 ASM 图。

综上分析，该模块设计符合 ASM 图。

3、song_reader

3.1 Verilog HDL 设计代码

该模块由顶层模块 song_reader.v 和子模块控制器 song_reader_ctrl.v、地址计数器模块 counter_n.v、歌曲选择模块 song_rom.v 和结束判断模块 is_over.v 组成。其中地址计数器模块为模 n 位数为 counter_bits 的计数器，结束判断模块采用状态机方法实现。

顶层模块代码：

```
module song_reader(
    clk,
    reset,
    play,
    song,              //歌曲序号
    song_done,         //歌曲播放完成标志
    note,              //二进制 6 位，表示音符
    duration,          //二进制 6 位，表示音长
    new_note,          //新音符标志
    note_done          //音符播放完成标志
);
input clk, reset, play, note_done;
input [1:0] song;
output song_done, new_note;
output [5:0] note, duration;
wire co;                //地址计数器进位输出，表示 32 个音符播放完毕
wire [4:0] q;           //音符地址后 5 位
//控制器
song_reader_ctrl c1(.clk(clk), .reset(reset), .play(play), .note_done(note_done), .new_note(new_n
ote));
//地址计数器
counter_n#(.n(32), .counter_bits(5)) song_choose(.clk(clk), .r(reset), .en(note_done), .q(q), .co
(co));
//歌曲选择
song_rom song_read(.clk(clk), .dout({note, duration}), .addr({song, q}));
//结束判断
is_over i1(.clk(clk), .r(co), .din(duration), .dout(song_done));
endmodule
```

控制器 song_reader_ctrl 代码：

```
module song_reader_ctrl(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;//状态编码
```

```
reg [1:0] state, nextstate;
//D 寄存器
always @(posedge clk) begin
    if(reset) state = RESET;
    else state = nextstate;
end
//下一状态和输出电路
always @(*) begin
    new_note = 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) begin
                if(note_done) nextstate = NEXT_NOTE;
                else nextstate = WAIT; end
            else nextstate = RESET;
            end
        NEXT_NOTE: nextstate = NEW_NOTE;
        default: nextstate = RESET;
    endcase
end
endmodule // song_reader_ctrl
```

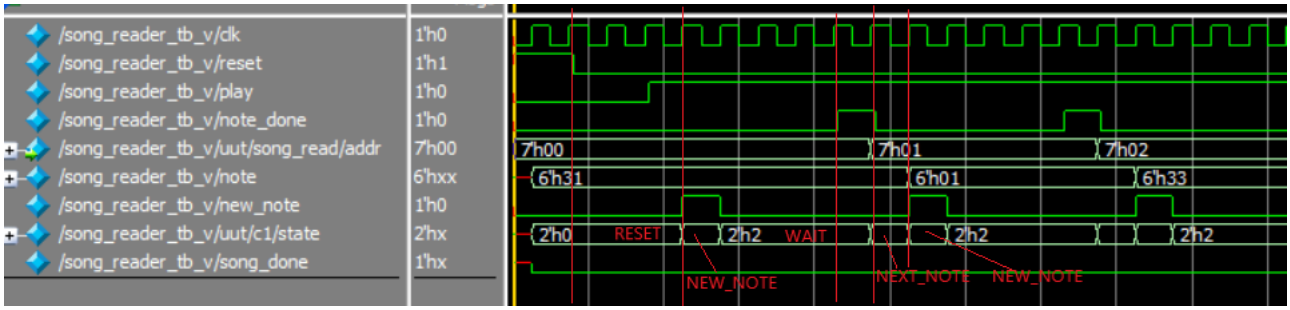
结束判断代码：

```
module is_over(clk, r, din, dout);
input clk, r;                  //r 为清零信号，即地址计数器输出进位 co，32 个音符全部播放完毕
input [5:0] din;               //din 即 duration
output reg dout;               //dout 即 song_done
parameter PAUSE = 0, PLAY = 1; //状态编码，PAUSE 表示未播放状态，PLAY 表示正在播放
reg state, nextstate;               //状态
always @(posedge clk) begin
    if(r) begin dout = 1; state =  PAUSE; end //当地址计数器输出 co 即 r=0 时，表示歌曲播放完毕，输出
dout=1，状态变为 PAUSE
    else state = nextstate;
end
always @(*)begin
    dout = 0;
    case (state)
    PAUSE: begin
        if(din) nextstate = PLAY;    //当 din 不为 0 时表示歌曲播放，进入 PLAY 状态
        else nextstate = PAUSE; end //否则继续 PAUSE 状态
    PLAY: begin
```

```
        if(din==0) begin dout = 1; nextstate = PAUSE; end       //音长为 0 进入 OVER 状态同时输出
dout=1
        else nextstate = PLAY;                  //否则继续 PLAY 状态
    end     //否则输出 0
    default: nextstate = PAUSE;
    endcase
end
endmodule // is_over
```

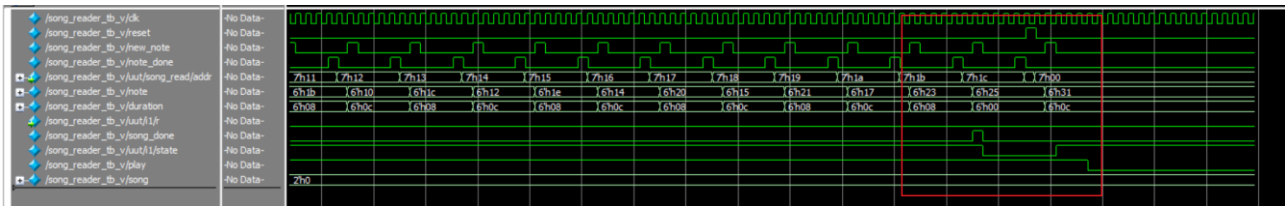
3.2 仿真测试

顶层模块测试:

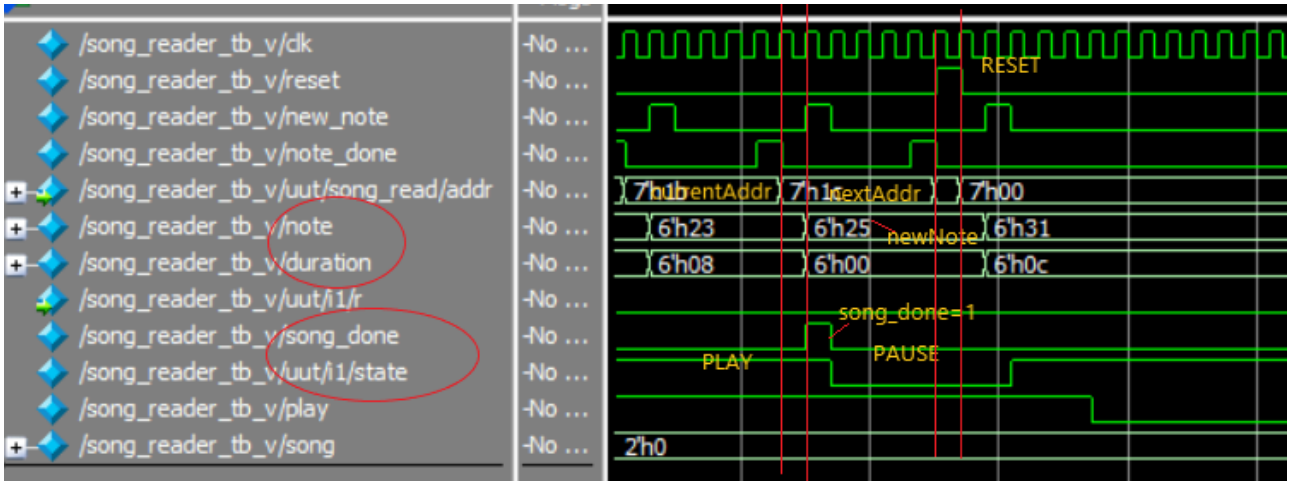


分析：波形图可见，当 reset = 1 时，进入 RESET 状态；当 play = 1 时，进入 NEW_NOTE 状态并输出 new_note = 1；然后进入 WAIT 状态，输出 new_note = 0；当 note_done = 1 时，下一个时钟上升沿到来时，进入 NEXT_NOTE 状态，输出 new_note = 0，并读取下一个音符的地址；然后下一个时钟上升沿到来时，进入 NEW_NOTE 状态，输出 new_note = 1，下一个音符播放。波形图与 ASM 图相符合。

结束判断模块测试：



局部放大图：



分析：波形图可见，当 duration=0 时，song_done=1，下一个时钟周期到来时，结束判断模块进入 PAUSE 状态。当 reset=0 时， song_reader 模块进入 RESET 状态。

4、note_player

4.1 Verilog HDL 代码设计

note_reader 模块由顶层模块 note_player.v 和子模块控制器 note_player_ctrl.v、音符节拍定时器 timer.v、D 触发器 dffre.v、只读存储器 FreqROM.v、DDS 模块 dds.v 组成。

顶层设计：

```
module note_player(
    clk,           //系统时钟信号，外接 sys_clk
    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
    note_done,     //给 song_reader 模块的应答信号，表示音符播放完毕
    load_new_note, //来自 song_raeder 模块的 new_note 信号，表示音符播放完毕
    beat,          //定时基准信号，频率为 48kHz
    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
    sample,         //正弦样品输出
    sample_ready    //下一个正弦信号
);
input clk, reset, play_enable, load_new_note, beat, sampling_pulse;
input [5:0] note_to_load, duration_to_load;
output note_done, sample_ready;
output [15:0] sample;
wire [5:0] q;       //FreqROM 的地址输入
wire [19:0] dout;
```

```
wire timer_clear, timer_done;  //计时清空和计时完成
//控制器
note_player_ctrl i1(
    .clk(clk),
    .reset(reset),
    .play_enable(play_enable),
    .load_new_note(load_new_note),
    .load(load),   //D 触发器的使能输入
    .timer_clear(timer_clear),  //音符节拍定时器计时清空
    .timer_done(timer_done),    //音符节拍定时器计时完成
    .note_done(note_done)
);
//D 触发器
dffre#(.n(6)) i2(
    .d(note_to_load),
    .en(load),
    .r(~play_enable || reset),  //清零信号
    .clk(clk),
    .q(q)      //FreqROM 地址输入
);
//FreqROM
frequency_rom i3(
    .clk(clk),
    .dout(dout),  //DDS 模块 k 的后 20 位
    .addr(q)
);
//DDS
dds i4(
    .clk(clk),
    .reset(~play_enable || reset),
    .k({2'b00, dout}),
    .sampling_pulse(sampling_pulse),
    .new_sample_ready(sample_ready),
    .sample(sample)
);
//计时器
timer i5(
    .clk(clk),
    .beat(beat),  //使能端
    .q(duration_to_load), //计时长度
    .din(timer_clear),
    .dout(timer_done)
);
endmodule // note_player
```

控制器 note_player_ctrl：

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

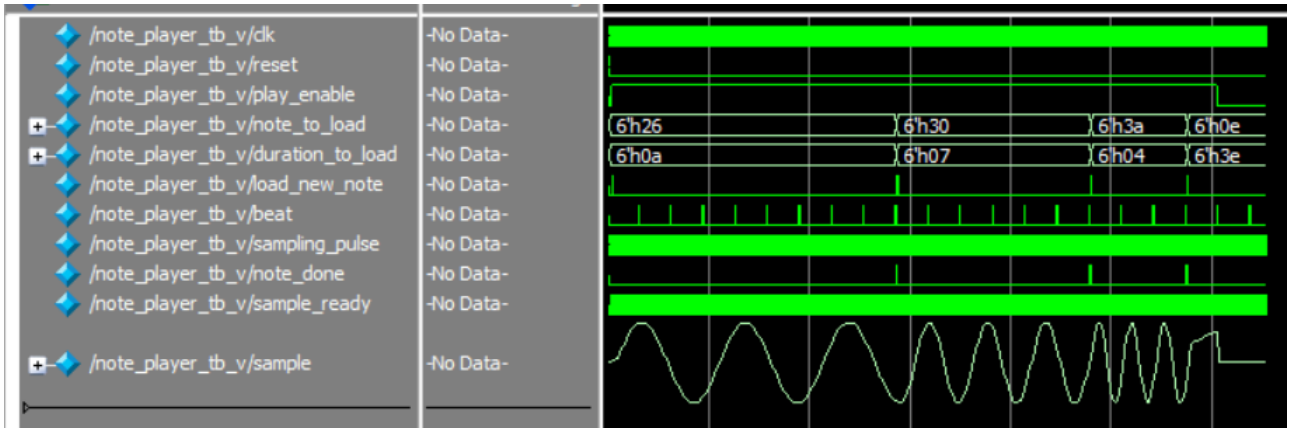
音符节拍定时器 timer.v：

```
module timer(clk, beat, q, din, dout);
input clk, beat, din;
```

```
input [5:0] q;     //计时长度
output dout;       //计时结束输出
reg [5:0] cnt = 0;
assign dout = (cnt==q-1);    //计时结束
always @(posedge clk) begin
    if(din) cnt = 0;       //reset 信号
    else begin if(beat) cnt = cnt+1;  //beat 高电平则计时+1
                else cnt = cnt; end
end
endmodule // timer
```

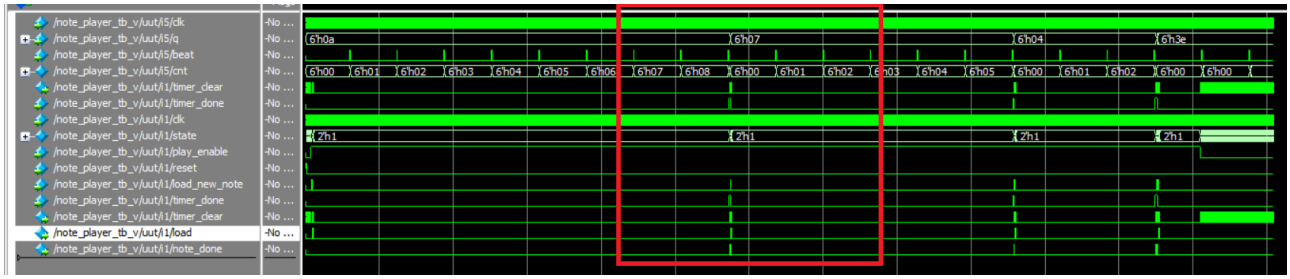
4.2 仿真测试

顶层测试：

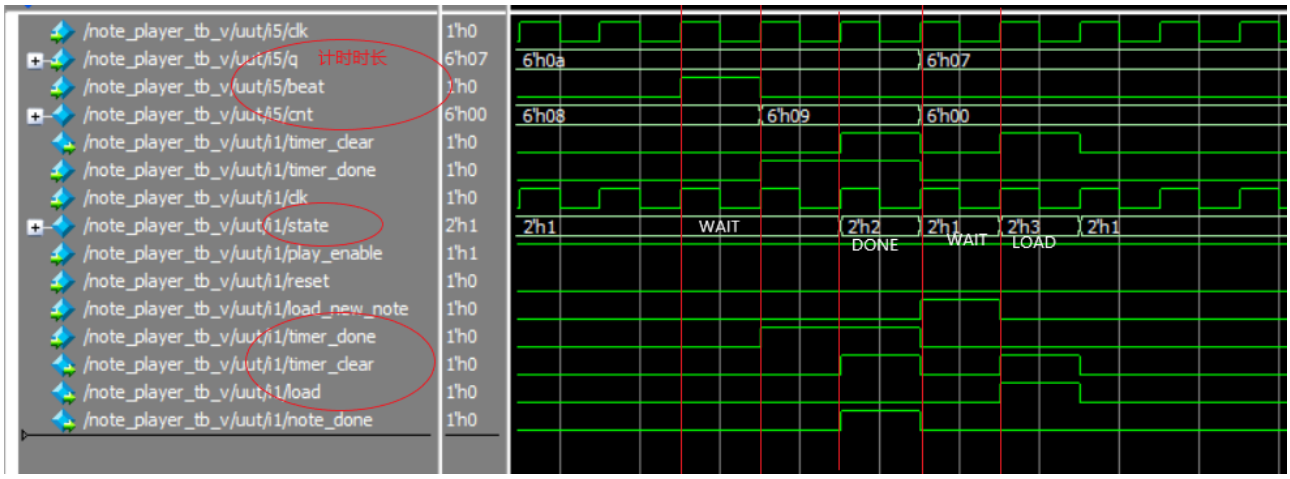


分析：波形图可见，正弦信号频率与 note_to_load 的值呈正相关。当 play_enable=0 时，sample=0。

控制器和节拍定时器测试：

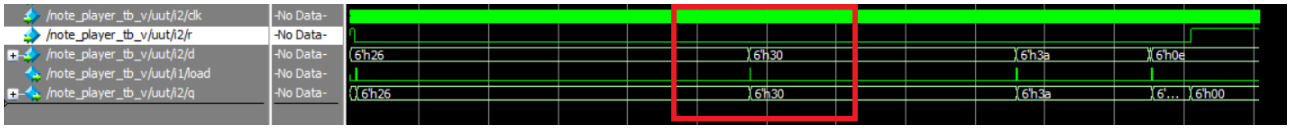


局部放大：

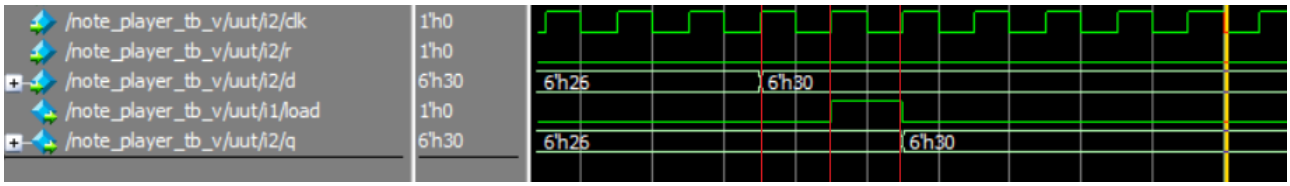


分析：波形图可见，当 note_player_ctrl 处于 WAIT 状态时，计时 cnt 到 duration-1 时，输出 timer_done=1，时间为两个时钟周期，然后控制器进入 DONE 状态，输出 timer_clear=1，note_done=1，cnt 复位为 0,随后进入 WAIT 状态,timer_clear=0,note_done=0,因为此时 play_enable=1,timer_done=0，load_new_note=1，控制器进入 LOAD 状态，输出 timer_clear=1，load=1。

D 触发器测试：



局部放大：



分析：波形图可见，在使能信号 load=1 的下一个时钟周期，q=d，符合设计要求。

5、同步化电路设计

5.1 设计原理

由于音频编解码接口模块和其它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。

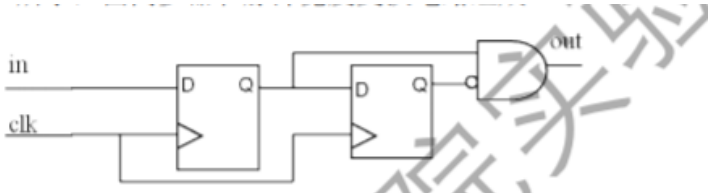


图 6.52　同步化电路

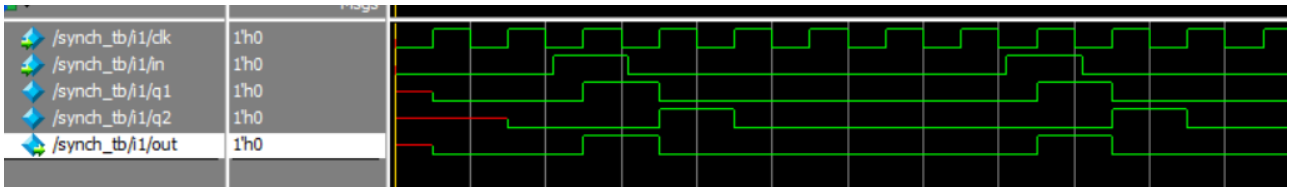
5.2 Verilog HDL 代码设计

通过非阻塞赋值的方式设计两个 D 触发器。再通过持续赋值语句得到 out 输出。

代码如下：

```
module synch(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 // synch
```

5.3 仿真测试



6、music_player 模块设计

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。

Verilog HDL 设计如下：

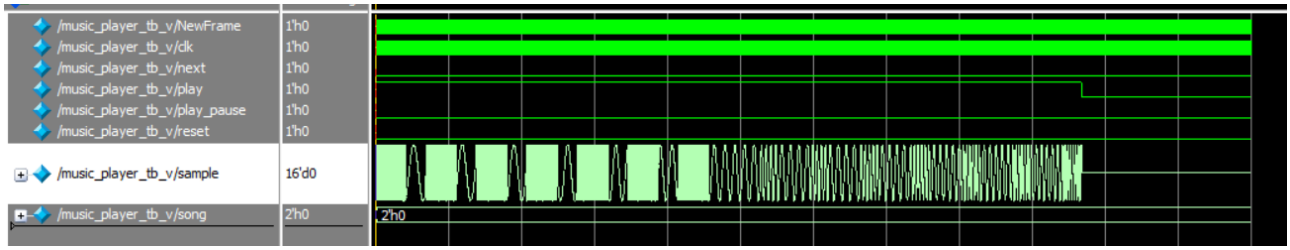
顶层设计：

```
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 ready, beat, reset_play, song_done, new_note, note_done;
wire [5:0] note, duration;
wire sample_ready;
//同步化电路
synch i1(
    .clk(clk),
    .in(NewFrame),
    .out(ready)
);
//节拍基准产生器
counter_n #(.n(sim?64:1000), .counter_bits(sim?6:10)) i2(
    .clk(clk),
    .r(0),
    .en(ready),
    .q(),
    .co(beat)
);
//主控制器
mcu c1(
    .clk(clk),
    .reset(reset),
    .play_pause(play_pause),
    .next(next),
    .play(play),
    .song(song),
    .reset_play(reset_play),
    .song_done(song_done)
);
//乐曲读取
song_reader c2(
    .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)
);
//音符播放
note_player c3(
    .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)
);
endmodule // music_player
```

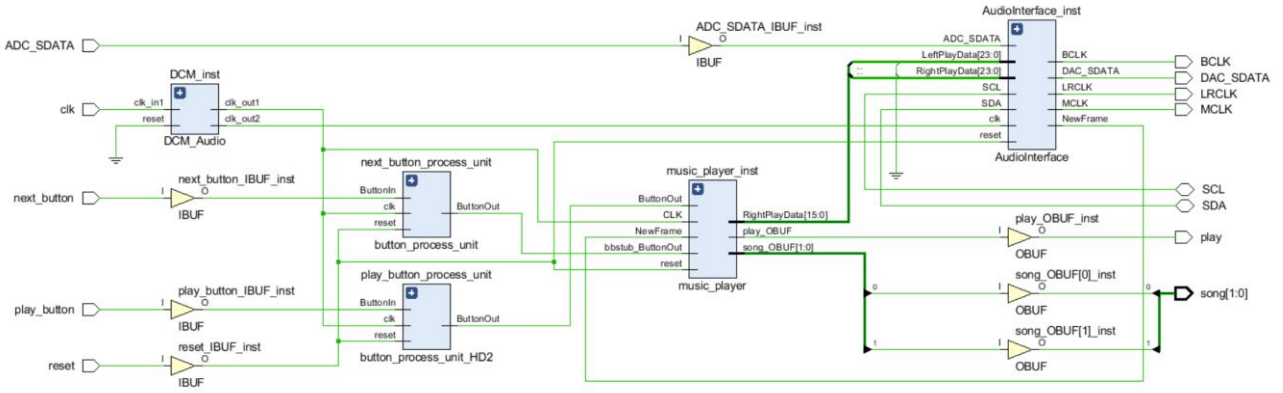
仿真结果：



分析：波形基本符合要求。

五、Vivado 工程建立

电路原理图：



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？

答：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？

答：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

这次实验我觉得开头比较难，在没有老师讲解的情况下，要设计一个这么大的工程，开始时觉得无从下手。第一天花了很久时间理解实验设计的思路和 DDS 设计的原理，到了下午终于完成了 DDS 的设计和仿真。在设计 mcu 控制器时，又因为不会设计去看了 HDL 语言的讲义，跟着讲义上控制器设计的例子设计了控制器，虽然波形出来了，但是也不知道设计的对不对。后来就跟着 mcu 的 ASM 图一个个分析下去。开始时还不清楚模块化设计的思想，以为 mcu 就是一个子模块，在 mcu 里面调用了 counter_n 模块，还写了状态控制的代码，后来做到 song_reader 的时候，代码写着写着就意识到自己写的代码有些乱，又调用模块又写状态控制的代码，然后意识到设计应该是根据每一个模块的原理框图写的，通过调用原理框图里面的子模块来写这个模块。这样既方便调试，代码结构也清晰了。有了这样的意识之后，后面的模块设计就比较快了，根据每一个模块的原理框图和端口写代码，然后仿真。

这次实验遇到的主要问题有三个：

1、一开始不会看控制器设计的波形，那时候没有注意到老师在学在浙大传的视频，就自己在那里想，想了很久也没想出来，然后去设计下一个了，但心里总是不舒服，不知道自己的设计对不对，后来看了老师的视频，就知道要和 ASM 图结合起来看了。

2、music_player 模块输出始终不对，观察波形发现 ready 总是出现高阻的状态，而在同步化电路单独测试的时候并不会出现输出高阻的情况，反复检查自己的代码，也没找出原因。后来是问了唐老师，才发现自己端口设置错了，ready 应该是节拍基准产生器的使能输入，而不是计数器的 q，当把 ready 作为计数器的 q 时，会修改 ready 的值，因此出现高阻状态。修改端口设置后，波形看上去和教材中的一样了。

3、song_reader 的结束判断模块有误。虽然在 music_player 仿真时，我的波形和教材中的差不多，但在写报告时，我重新仿真了全部模块并且观察了各子模块的波形后发现了我的结束判断子模块 is_over 输出的 song_done 高电平脉冲大于一个时钟周期，发现是状态设置有误，后来反复修改代码终于使其为一个时钟周期之后又出现比 duration 为 0 的时刻延迟一个时钟周期的情况。通过修改输出的时刻，使其在判断之后马上输出而不是进入下一个状态后输出，终于得到了正确结果。

其实这次实验我做完也不能确定自己的仿真结果是否正确，也许波形是和教材差不多，但我觉得在一些子模块里面的输出不一定是预期的那样，在板子上测试的时候可能会有很多 bug。

做完实验后感觉实际的电路设计真的很难，毕竟我们在有教材，有设计流程，有正确结果的情况下设计都感到困难，在实际中无教材，无正确结果，设计流程全得自己做，这样的电路设计想想都觉得很难。