

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 数字系统设计实验 |
| 姓 名： | 黄嘉欣 |
| 学 院： | 信息与电子工程学院 |
| 系： | 信息与电子工程学系 |
| 专 业： | 信息工程 |
| 学 号： | 3190102060 |
| 指导教师： | 屈民军、唐奕 |

2021年6月8日

**实验报告**

专业： 信息工程

姓名： 黄嘉欣

学号： 3190102060

日期： 2021年6月8日

地点： 东四-223

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： 音乐播放实验 实验类型： 设计性实验

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、实验内容 六、实验结果与仿真分析

七、思考题 八、心得与体会

**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 ；

② 正弦信号频率范围为 ；

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的基本原理：

为了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。若要实时改变 输出信号的频率，可以通过改变查表寻址的频率或寻址的步长来实现。DDS技术即采 用了后面这种方法，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成线 性关系，其基本原理框图如图3.1.1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：

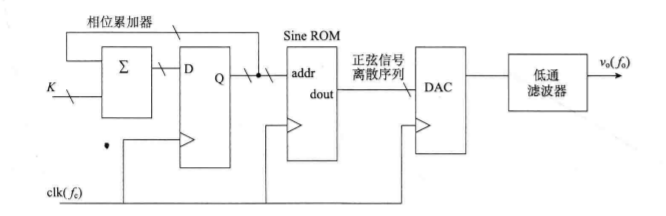


图3.1.1 DDS原理框图

正弦查询表（Sine ROM）中存放有一个完整的正弦信号样品，其由如下的映射关 系构成：



式中，为Sine ROM地址线位数，为ROM数据线宽度，的数据形式为补码。

为取样时钟clk的频率，为相位增量（也称频率控制字），输出正弦信号的频 率由和共同决定，即：



可以看出，正弦信号的频率与相位增量呈正比关系。相位累加器的位数由位整 数和位小数组成，其高位整数部分作为Sine ROM的地址。

在实际应用当中，DDS的最高输出频率由允许输出的杂散电平决定，一般取值为 因此的最大值一般为。除此之外，DDS可以很容易实 现正弦信号与余弦信号正交两路输出，只需用相位累加器的输出同时驱动固化有正弦信 号波形的Sine ROM和余弦信号波形的Cos ROM，并各自经数模转换器和低通滤波器 输出即可。另外，DDS也很容易实现调幅和调频，其原理框图分别如图3.1.2和图3.1.3 所示：

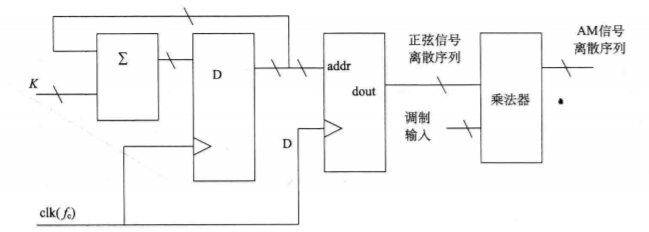


图3.1.2 DDS实现调幅原理框图

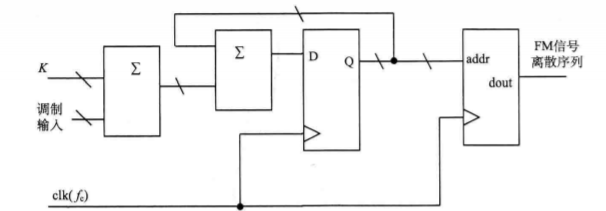


图3.1.3 DDS实现调频原理框图

2、DDS实验设计原理：

根据DDS输出信号的最低频率要求，有：



代入，可得，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，其高12位为Sine ROM的地址。

因为，所以存储一个完整周期的正弦信号样品就需要的ROM。 但由于正弦波形的对称性，如图3.2.1所示，可以将正弦波形分为四个区域，如此只需 要在 Sine ROM中存储四分之一的正弦信号样品（0区）即可。此时Sine ROM的容量 可减少为，即10位地址，存储四分之一的正弦信号样品（共1024个）。需 要注意的是，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）。

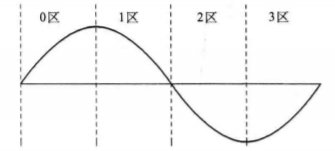


图3.2.1 正弦信号波形

经过上述优化后，DDS的结构也必须进行必要处理，其优化后的结构如图所示：

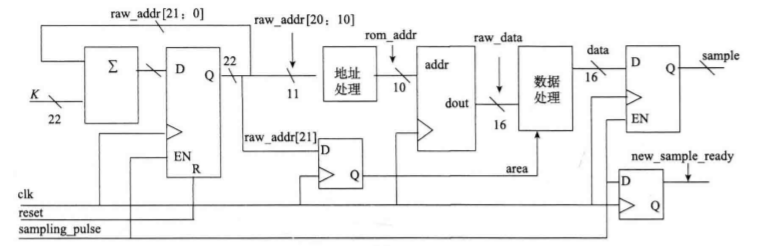
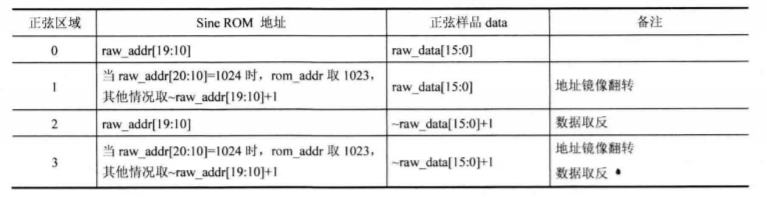


图3.2.2 优化后的DDS结构

图中，sampling\_pulse为采样脉冲，一般情况下，其频率应低于时钟clk频率，而宽度 必须为一个时钟clk周期。若采样脉冲与时钟信号为同一个信号，可将sampling\_pulse 接高电平即可。相位累加得到的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，其处理方法如表3.2所示：

表3.2 sine\_rom的地址和数据处理方法



3、音乐播放器的顶层设计：

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3.3.1所示。

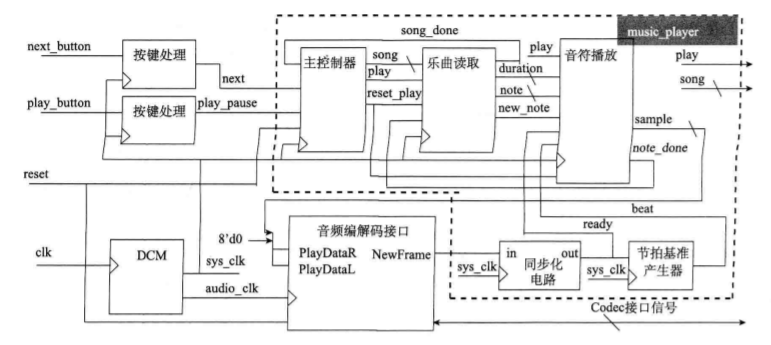


图3.3.1 音乐播放器的顶层设计

各主要子模块作用如下：

① 时钟管理模块（DCM）产生100MHz的系统时钟sys\_clk和的音频时钟 audio\_clk；

② 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）；

③ 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）；

④ 音符播放接收到需播放的音符，在音符的持续时间内，以速率送出该音符的 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；

⑤ 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；

⑥ 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；

⑦ 节拍基准产生器产生的节拍定时基准脉冲信号（beat），而ready信号频率为 ，因此，节拍基准产生器即为分频比为1000的分频器；

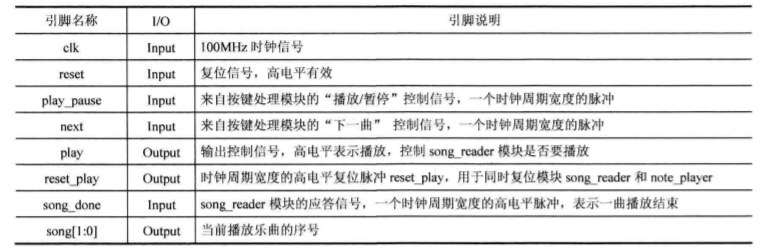
⑧ 按键处理模块完成输入同步化、防颤动和脉宽变换等功能。

4、音乐播放器设计原理：

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，下表为其端口含义：

表3.4.1 主控制模块mcu的端口含义



根据设计要求，模块mcu的原理框图如图3.4.1.1所示。图中的 2 位二进制计数器用来 计算乐曲序号（song）：

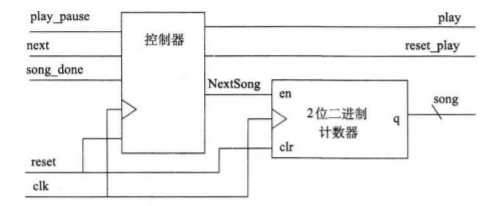


图3.4.1.1 mcu的结构框图

其控制器的工作流程图如图3.4.1.2所示：

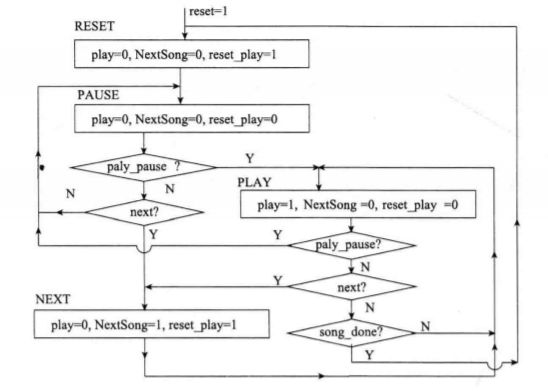


图3.4.1.2 mcu控制器的算法流程图

② 乐曲读取模块song\_reader的设计

乐曲读取模块song\_reader的任务有：

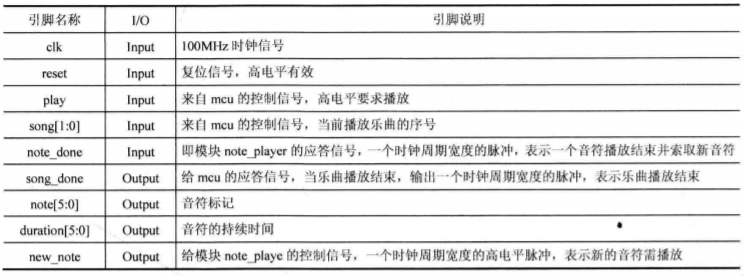
（1）根据mcu模块的要求，选择播放乐曲；

（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，其需包含表3.4.2所示的输入、输出端口：

表3.4.2 乐曲读取模块song\_reader的端口含义



song\_rom是一个只读存储器，用来存放乐曲，容量为。其中共存放有 四首乐曲，每首乐曲占用空间，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3.4.2.1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。

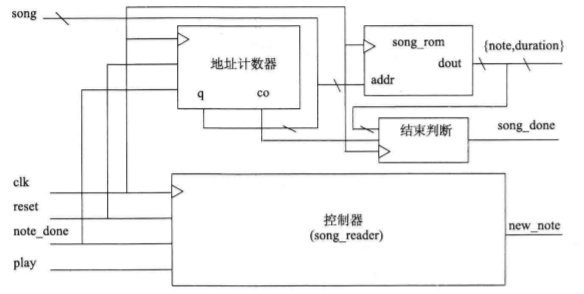


图3.4.2.1 song\_reader的结构框图

控制器算法流程图如图3.4.2.2所示：

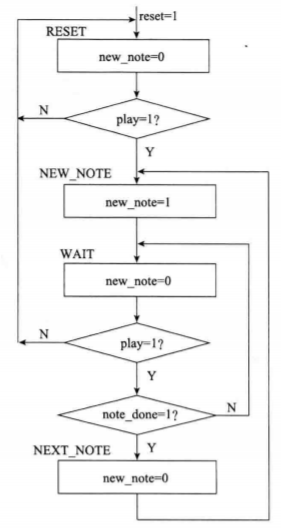


图3.4.2.2 song\_reader控制器的算法流程图

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，其主要任务包括：

（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；

（3）以速率从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，可进一步划分功能单元，如图3.4.3.1所示，图中FreqROM 为只读存储器，完成音符标记note与DDS模块的相位增量查找表关系。

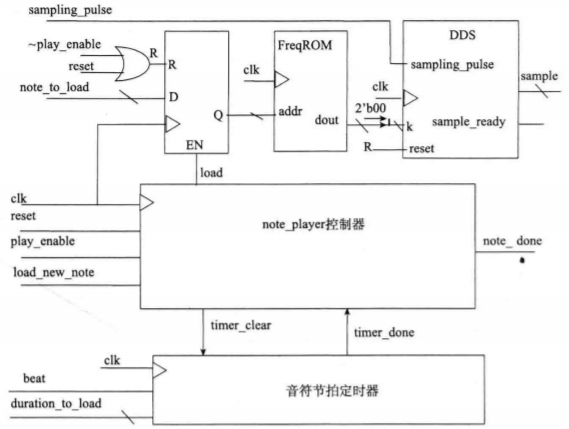
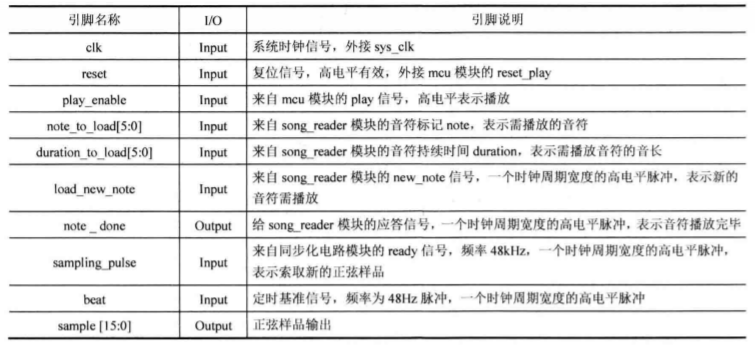


图3.4.3.1 note\_player的结构框图

表3.4.3为note\_player模块的端口含义：

表3.4.3 note\_player模块的端口含义



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，其算法流程如图3.4.3.2所示：

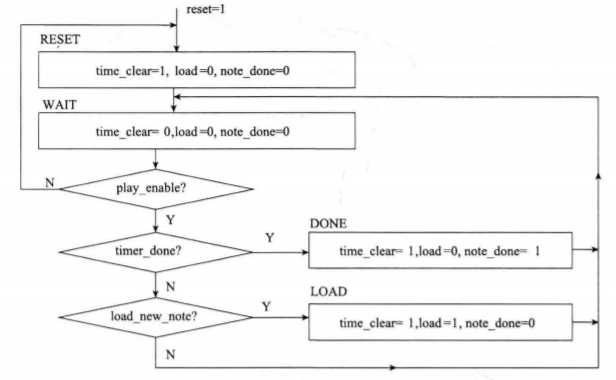


图3.4.3.2 note\_player控制器的算法流程图

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。需要注意的是，DDS模 块的输入为22位二进制，因此，Frequency ROM输出的20位相位增量需要高位加两 个0后再接入DDS。

④ 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。其由同步器和脉冲宽度变换电路组成，电路结果如图 所示：

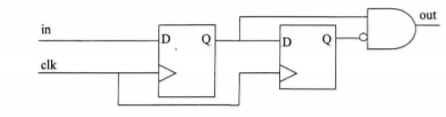


图3.4.4.1 同步化电路的结构框图

⑤ 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，产生的系统时钟和 的音频时钟。

**四、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多媒体音视频只能互联开发系统；

③ 有源音箱或耳机。

**五、实验内容**

① 从网站下载提供的文件；

② 编写DDS模块的Verilog HDL代码，并用ModelSim仿真验证；

③ 编写mcu模块的Verilog HDL代码，并用ModelSim仿真验证；

④ 编写song\_reader模块的Verilog HDL代码，并用ModelSim仿真验证；

⑤ 编写note\_player模块的Verilog HDL代码，并用ModelSim仿真验证；

⑥ 编写1000分频器模块的Verilog HDL代码及其测试代码，并用ModelSim仿真验证；

⑦ 编写同步化电路的Verilog HDL代码及其测试代码，并用ModelSim仿真验证；

⑧ 编写次顶层music\_player模块的Verilog HDL代码及其测试代码，并用ModelSim 仿真验证；

⑨ 新建music\_player的Vivado工程，编写顶层music\_player\_top模块的Verilog HDL 代码，生成符合要求的DCM内核，添加需要的文件，对工程进行综合、约束、实 现，并下载工程文件到Nexys Video开发板中；

⑩ 将耳机接入实验开发板音频输出插座，操作reset（中间按钮）、play/pause（右边 按钮）、next（下面按钮）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，验证设计结果是否正确。

**六、实验结果与仿真分析**（此部分代码语句内换行皆是由于页面限制，原文件可查阅Solutions）

① DDS模块

（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， 其顶层代码如下：

module dds (clk, reset, sampling\_pulse, k, sample, new\_sample\_ready);

input clk, reset, sampling\_pulse;

input[21:0] k; // 相位增量

output[15:0] sample; // 正弦信号

output new\_sample\_ready;

// 中间变量

wire[21:0] raw\_addr; // 地址处理输入

wire[9:0] rom\_addr; // 地址处理输出

wire[15:0] raw\_data, data; // 数据处理

wire[21:0] sum; // 加法器结果

wire area; // 区域

// 加法器实例

full\_adder adder0(.a(k), .b(raw\_addr), .s(sum), .co()); // 进位输出空脚

// D触发器实例

dffre #(.n(22)) dff0(.d(sum), .en(sampling\_pulse), .r(reset),

.clk(clk), .q(raw\_addr)); // 产生raw\_addr

dffre #(.n(1)) dff1(.d(raw\_addr[21]), .en(1), .r(0), .clk(clk),

.q(area)); // 产生area

dffre #(.n(16)) dff2(.d(data), .en(sampling\_pulse), .r(0), .clk(clk),

.q(sample)); // 产生正弦信号

dffre #(.n(1)) dff3(.d(sampling\_pulse), .en(1), .r(0), .clk(clk),

.q(new\_sample\_ready));

// 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];

// 正弦查找表

sine\_rom rom0(.clk(clk), .addr(rom\_addr), .dout(raw\_data));

// 数据处理

assign data[15:0] = area ? (~raw\_data[15:0]+1) : raw\_data[15:0];

endmodule

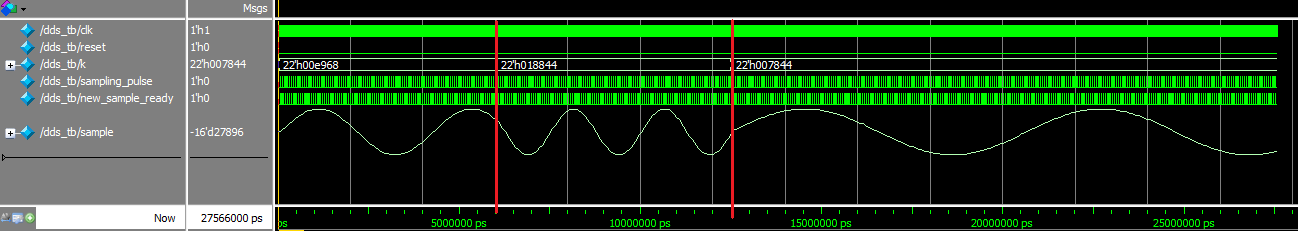
 （2）仿真结果

图6.1 DDS仿真结果

分析：如图，当增大时，正弦输出信号的sample的频率变大；当减小时，正弦输 出信号的sample的频率变小。值变化的点即为频率转折点（图中体现为波形发生变 化），满足频率关系。故设计符合要求。

② mcu模块

（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、D触发器dffre.v组成。 i）顶层代码：

module mcu (clk, reset, play\_pause, next,

song\_done, play, reset\_play, song);

input clk, reset;

input play\_pause; // play\_pause按钮

input next; // next按钮

input song\_done; // 乐曲播放结束

output play;

output reset\_play; // 脉冲复位

output[1:0] song; // 乐曲序号

wire NextSong; // 控制器输出

// 控制器实例

mcu\_ctrl ctrl0(.clk(clk), .reset(reset), .play\_pause(play\_pause),

.next(next), .song\_done(song\_done), .play(play),

.reset\_play(reset\_play), .NextSong(NextSong));

// 2位二进制计数器实例

counter\_n #(.n(4), .counter\_bits(2)) song\_cnt(.clk(clk),

.en(NextSong), .r(0), .q(song), .co());

endmodule

ii）控制器代码：

module mcu\_ctrl(clk, reset, play\_pause, next, song\_done,

play, reset\_play, NextSong);

input clk, reset;

input play\_pause; // play\_pause按钮

input next; // next按钮

input song\_done; // 乐曲播放结束

output reg play;

output reg reset\_play; // 脉冲复位

output reg NextSong; // 控制器输出

parameter RESET = 0, PAUSE = 1, PLAY = 2, NEXT = 3; // 状态编码

reg[1:0] state, nextstate; // 当前状态与下一个状态

// 状态寄存器

always @ (posedge clk)

begin

if (reset)

state = RESET; // 只要按下RESET按钮即复位

else

state = nextstate; // 否则继续运行

end

// 下一个状态和输出

always @ (\*)

begin

// 初始

play = 0; NextSong = 0; reset\_play = 0;

case (state)

// RESET状态

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 // PAUSE状态

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 // PLAY状态

NEXT: begin nextstate = PLAY; NextSong = 1; reset\_play = 1; end // NEXT状态

default: nextstate = RESET;

endcase

end

endmodule

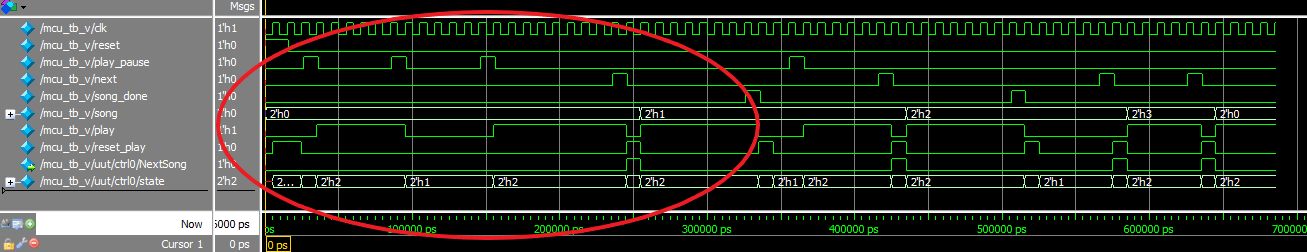
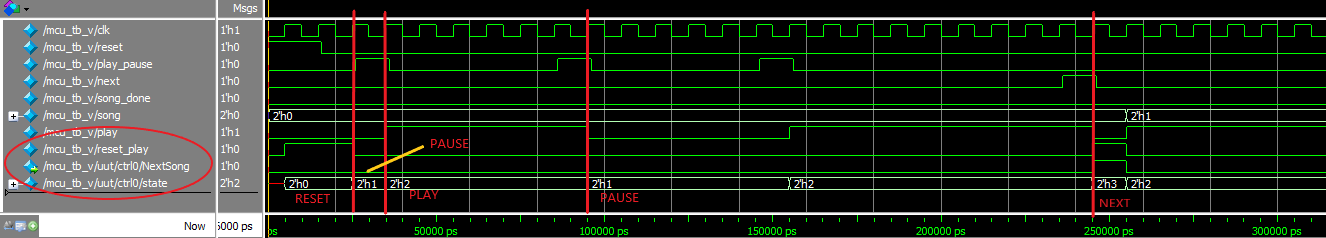
 （2）仿真结果

图6.2.1 mcu仿真结果（全图）

如图，将图中红线圈出部分放大，有：

图6.2.2 mcu仿真结果（局部放大图）

分析：当第一个时钟上升沿到来时，reset = 1，于是控制器状态跳为RESET（状态编 码0），此时输出play = 0，NextSong = 0，reset\_play = 1；

在第一条直线处，reset = 0，时钟上升沿到来后，控制器状态自然跳到PAUSE（状态 编码1），此时输出为play = 0，NextSong = 0，reset\_play = 0；

在第二条直线处，reset = 0，play\_pause = 1，时钟上升沿到来后，控制器状态跳为PLAY （状态编码2），此时输出play = 1，NextSong = 0，reset\_play = 0；

在第三条直线处，reset = 0，play\_pause = 1，时钟上升沿到来后，控制器状态跳为PAUSE， 此时输出play = 0，NextSong = 0，reset\_play = 0；

在第四条直线处，前一刻状态为PLAY，时钟上升沿到来时，reset = 0，play\_pause = 0， next = 1，控制器状态跳为NEXT，此时输出play = 0，NextSong = 1，reset\_play = 1；

综上可知，状态转换与输出均满足ASM图，模块设计正确。

③ song\_reader模块

（1）Verilog HDL代码设计

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、歌曲存储模块song\_rom.v和结束判断模块song\_over.v组成，结束判断 模块采用状态机方法实现。

i）顶层代码：

module song\_reader(clk, reset, play, song, note\_done,

song\_done, note, duration, new\_note);

input clk, reset;

input play; // 来自mcu的控制信号，高电平要求播放

input note\_done; // 模块note\_player的应答信号，表示音符播放结束

input[1:0] song; // 当前播放乐曲的序号

output song\_done; // 乐曲播放结束

output new\_note; // 表示新的音符需播放

output[5:0] note; // 音符标记

output[5:0] duration; // 音符持续时间

wire[4:0] q; // 乐曲音符地址

wire co; // 地址计数器进位

// 地址计数器(5bits)

counter\_n #(.n(32), .counter\_bits(5)) addr\_cnt(.clk(clk),

.en(note\_done), .r(reset), .q(q), .co(co));

// 只读存储器

song\_rom rom0(.clk(clk), .dout({note, duration}), .addr({song, q}));

// 结束判断

song\_over over0(.clk(clk), .ci(co), .din(duration),

.dout(song\_done));

// 控制器

song\_reader\_ctrl ctrl0(.clk(clk), .reset(reset),

.note\_done(note\_done), .play(play),

.new\_note(new\_note));

endmodule

ii）控制器代码：

module song\_reader\_ctrl(clk, reset, note\_done, play, new\_note);

input clk, reset;

input note\_done; // 音符播放结束

input play; // 高电平要求播放

output reg new\_note; // 新的音符需要播放

parameter RESET = 0, NEW\_NOTE = 1, WAIT = 2, NEXT\_NOTE = 3; // 状态编码

reg[1:0] state, nextstate;

always @ (posedge clk)

begin

if (reset) state = RESET; // 若复位，则系统处于RESET状态

else state = nextstate;

end

always @ (\*)

begin

new\_note = 0;

case (state)

RESET: begin // 处于RESET状态

if(play) nextstate = NEW\_NOTE;

else nextstate = RESET;

end

NEW\_NOTE: begin // 处于NEW\_NOTE状态

new\_note = 1;

nextstate = WAIT;

end

WAIT: begin // 若继续播放且乐符已播完，则NEXT\_NOTE

if (play)

begin

if (note\_done) nextstate = NEXT\_NOTE;

else nextstate = WAIT;

end

else nextstate = RESET;

end

NEXT\_NOTE: nextstate = NEW\_NOTE; // 准备播放下一个音符

default: nextstate = RESET;

endcase

end

endmodule

iii）结束判断模块代码：

module song\_over(clk, ci, din, dout);

input clk, ci; // ci为地址计数器进位，高电平表示乐曲结束

input[5:0] din; // song\_rom输出duration

output reg dout; // 即song\_done，乐曲播放结束

parameter PLAY = 0, OVER = 1; // 状态编码

reg state, nextstate;

// 状态寄存器

always @ (posedge clk)

begin

if (ci)

begin

state = OVER; // 若计数器进位，则播放结束

end

else state = nextstate;

end

// 下一个状态和输出

always @ (\*)

begin

// 初始

dout = 0;

case (state)

PLAY: begin

if (!din) // 若duration为0，则播放结束

begin

nextstate = OVER;

dout = 1;

end

else nextstate = PLAY; // 否则继续播放

end

OVER: begin

if (!din) nextstate = OVER;

// 若duration为0，则继续为OVER状态

else nextstate = PLAY; // 否则播放

end

default: begin nextstate = OVER; dout = 1; end

endcase

end

endmodule

iv）歌曲存储模块

新增有一首歌曲《送别》，新增代码为（详情参见song\_rom.v）：

// 《送别》

assign memory[ 96 ] = {6'd35, 6'd24} ; // Note: 3G

assign memory[ 97 ] = {6'd32, 6'd12} ; // Note: 3E

assign memory[ 98 ] = {6'd35, 6'd12} ; // Note: 3G

assign memory[ 99 ] = {6'd40, 6'd48} ; // Note: 4C

assign memory[ 100 ] = {6'd37, 6'd24} ; // Note: 4A

assign memory[ 101 ] = {6'd40, 6'd24} ; // Note: 4C

assign memory[ 102 ] = {6'd35, 6'd48} ; // Note: 3G

assign memory[ 103 ] = {6'd35, 6'd24} ; // Note: 3G

assign memory[ 104 ] = {6'd28, 6'd12} ; // Note: 3C

assign memory[ 105 ] = {6'd30, 6'd12} ; // Note: 3D

assign memory[ 106 ] = {6'd32, 6'd24} ; // Note: 3E

assign memory[ 107 ] = {6'd30, 6'd12} ; // Note: 3D

assign memory[ 108 ] = {6'd28, 6'd12} ; // Note: 3C

assign memory[ 109 ] = {6'd30, 6'd48} ; // Note: 3D

assign memory[ 110 ] = {6'd0, 6'd24} ; // Note: rest

assign memory[ 111 ] = {6'd0, 6'd24} ; // Note: rest

assign memory[ 112 ] = {6'd35, 6'd24} ; // Note: 3G

assign memory[ 113 ] = {6'd32, 6'd12} ; // Note: 3E

assign memory[ 114 ] = {6'd35, 6'd12} ; // Note: 3G

assign memory[ 115 ] = {6'd40, 6'd36} ; // Note: 4C

assign memory[ 116 ] = {6'd39, 6'd24} ; // Note: 4B

assign memory[ 117 ] = {6'd37, 6'd24} ; // Note: 4A

assign memory[ 118 ] = {6'd40, 6'd24} ; // Note: 4C

assign memory[ 119 ] = {6'd35, 6'd48} ; // Note: 3G

assign memory[ 120 ] = {6'd35, 6'd24} ; // Note: 3G

assign memory[ 121 ] = {6'd30, 6'd12} ; // Note: 3D

assign memory[ 122 ] = {6'd32, 6'd12} ; // Note: 3E

assign memory[ 123 ] = {6'd33, 6'd36} ; // Note: 3F

assign memory[ 124 ] = {6'd27, 6'd12} ; // Note: 2B

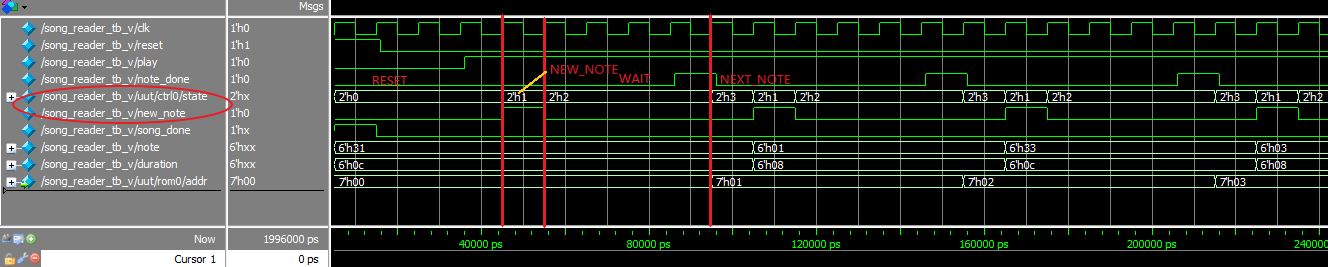
assign memory[ 125 ] = {6'd28, 6'd48} ; // Note: 3C

assign memory[ 126 ] = {6'd0, 6'd24} ; // Note: rest

assign memory[ 127 ] = {6'd0, 6'd24} ; // Note: rest

（2）仿真结果

i）顶层及控制器

图6.3.1 song\_reader控制器仿真结果

分析：当第一个时钟上升沿到来时，reset = 1，于是控制器状态state = RESET，输出 new\_note = 0。在后续时钟周期中，由于play = 0，故控制器状态始终保持为RESET；

在第一条直线处，reset = 0，play = 1，当时钟上升沿到来时，控制器状态跳为NEW\_ NOTE，输出new\_note = 1；

在第二条直线处，控制器自然跳到WAIT状态，输出new\_note = 0。此时由于play始 终为1，note\_done始终为0，故控制器保持为WAIT状态；

在第三条直线处，由于play = 1，note\_done = 1，当时钟上升沿到来时，控制器跳转 为NEXT\_NOTE状态，输出new\_note = 0；

综上可知，控制器状态转换与输出均满足ASM图，设计符合要求。

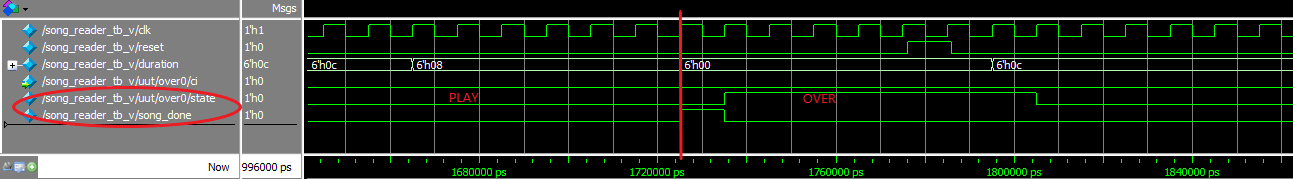
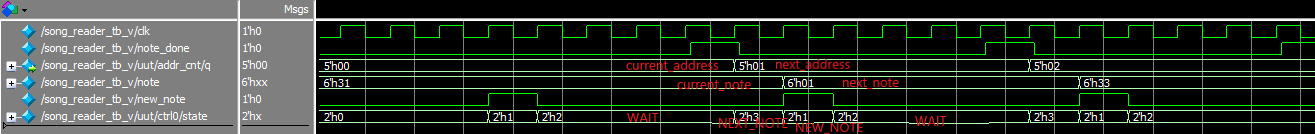
 ii）结束判断

图6.3.2 song\_over仿真结果

分析：如图，地址计数器进位输出始终为0，当duration = 0时（图中红线处），结束 判断模块输出song\_done = 1，其状态在下一个时钟信号到来时由PLAY（状态编码为 0）跳转为OVER（状态编码为1）。当duration不再等于0时，其状态又将回到PLAY 状态，满足设计要求。

iii）控制器、地址计数器与ROM的时序关系

图6.3.3控制器、地址计数器与ROM的时序关系图

分析：如图，song\_reader控制器、地址计数器与ROM的时序关系与书中要求一致，设 计正确。

④ note\_player模块

（1）Verilog HDL代码设计

该模块由顶层模块note\_player.v和子模块控制器note\_player\_ctrl.v、音符节拍定时 器note\_timer.v、D触发器dffre.v、只读存储器FreqROM.v、DDS模块dds.v组成。

i）顶层代码：

module note\_player(clk, reset, play\_enable, note\_to\_load,

duration\_to\_load, load\_new\_note, note\_done,

sampling\_pulse, beat, sample, sample\_ready);

input clk, reset;

input play\_enable; // 高电平表示播放

input[5:0] note\_to\_load; // 需播放的音符

input[5:0] duration\_to\_load; // 需播放音符的音长

input load\_new\_note; // 新的音符需播放

input sampling\_pulse; // 索取新的正弦样品

input beat; // 定时基准信号

output note\_done; // 音符播放完毕

output[15:0] sample; // 正弦样品输出

output sample\_ready; // 下一个正弦信号

wire[5:0] q; // FreqROM地址输入

wire[19:0] dout; // FreqROM输出

wire timer\_clear, timer\_done; //音符定时器输入输出

wire load; // D触发器使能输入

// D触发器实例

dffre #(.n(6)) d0(.d(note\_to\_load), .en(load),

.r(reset || ~play\_enable),.clk(clk), .q(q));

// Frequency ROM，dout为DDS模块k的后20位

frequency\_rom rom0(.clk(clk), .dout(dout), .addr(q));

// DDS实例

dds dds0(.clk(clk), .reset(reset || ~play\_enable),

.sampling\_pulse(sampling\_pulse), .k({2'b00, dout}),

.sample(sample), .new\_sample\_ready(sample\_ready));

// note\_player控制器

note\_player\_ctrl ctrl0(.clk(clk), .reset(reset),

.play\_enable(play\_enable),

.load\_new\_note(load\_new\_note), .load(load),

.timer\_clear(timer\_clear),

.timer\_done(timer\_done),

.note\_done(note\_done));

// 音符节拍定时器

note\_timer timer0(.clk(clk), .beat(beat),

.duration\_to\_load(duration\_to\_load),

.timer\_clear(timer\_clear),

.timer\_done(timer\_done));

endmodule

ii）控制器代码：

module note\_player\_ctrl(clk, reset, play\_enable, load\_new\_note,

load, timer\_clear, timer\_done, note\_done);

input clk, reset;

input play\_enable; // 来自mcu的play信号，高电平表示播放

input load\_new\_note; // 来自song\_reader的new\_note信号，表示新音符需要播放

input timer\_done; // 定时结束标志

output reg load; // D触发器的使能输入

output reg timer\_clear; // 清0信号

output reg note\_done; // 给song\_reader的应答信号，表示音符播放完毕

parameter RESET = 0, WAIT = 1, DONE = 2, LOAD = 3; // 状态编码

reg[1:0] state, nextstate;

always @ (posedge clk)

begin

if (reset) state = RESET; // 若复位，则处于RESET状态

else state = nextstate;

end

always @ (\*)

begin

timer\_clear = 0; load = 0; note\_done = 0; // 初始化

case (state)

RESET: begin timer\_clear = 1; nextstate = WAIT; end //RESET状态

WAIT: begin

if (play\_enable)

begin

if (timer\_done) nextstate = DONE; // 定时结束

else

begin

if (load\_new\_note) nextstate = LOAD;//读取新音符

else nextstate = WAIT;

end

end

else nextstate = RESET;

end

DONE: begin // 音符播放完毕

timer\_clear = 1; note\_done = 1;

nextstate = WAIT;

end

LOAD: begin // 读取新的音符

timer\_clear = 1; load = 1;

nextstate = WAIT;

end

default: nextstate = RESET; // 否则处于RESET状态

endcase

end

endmodule

iii）音符节拍定时器代码：

module note\_timer(clk, beat, duration\_to\_load, timer\_clear, timer\_done);

input clk;

input beat; // 定时基准信号，频率为48Hz的脉冲(使能输入)

input[5:0] duration\_to\_load; // 播放音符的音长

input timer\_clear; // 清0信号

output timer\_done; // 定时结束

reg[5:0] cnt = 0; // 计数，因为音长为5bits(32)，故cnt长为5位

assign timer\_done = (cnt == (duration\_to\_load - 1)); // 定时结束

always @ (posedge clk)

begin

if (timer\_clear) cnt = 0; // 清0

else

begin

if (beat) cnt = cnt + 1;

else cnt = cnt; // 若beat高电平则计数，否则保持

end

end

endmodule

（2）仿真结果

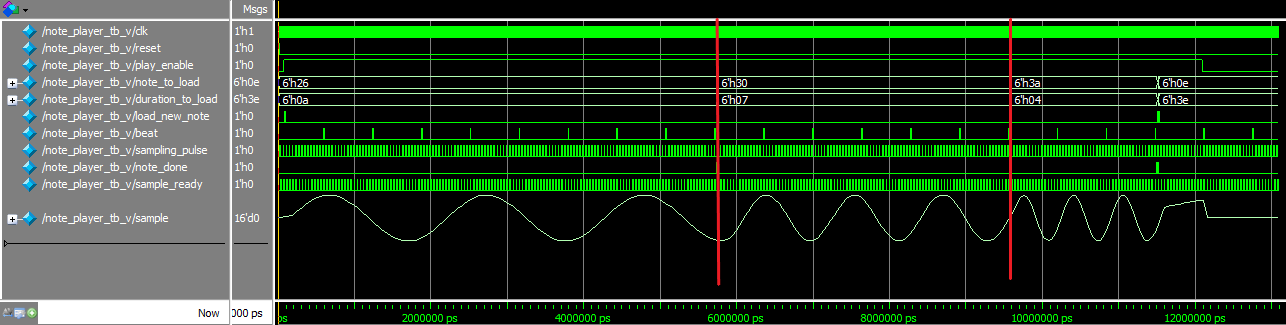
i）顶层设计

图6.4.1 note\_player仿真结果

分析：如图，当note\_to\_load增大时，输出sample正弦信号频率增大，两者呈正比关系；note\_to\_load变化的点即为频率转折点。当play\_enable = 0时，sample = 0，符合设计要求。

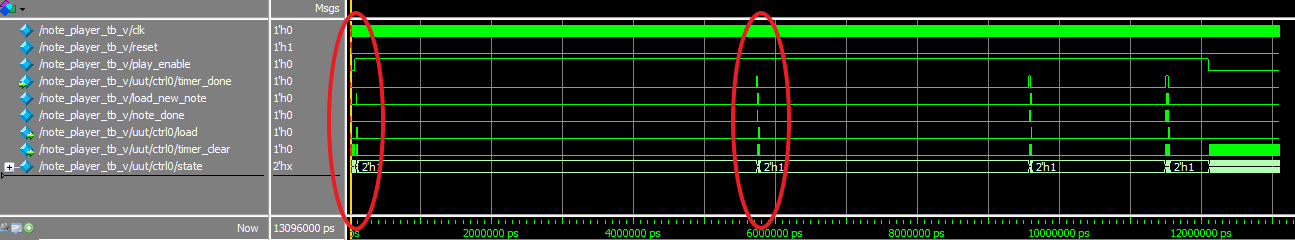
ii）控制器

图6.4.2.1 控制器仿真结果（全图）

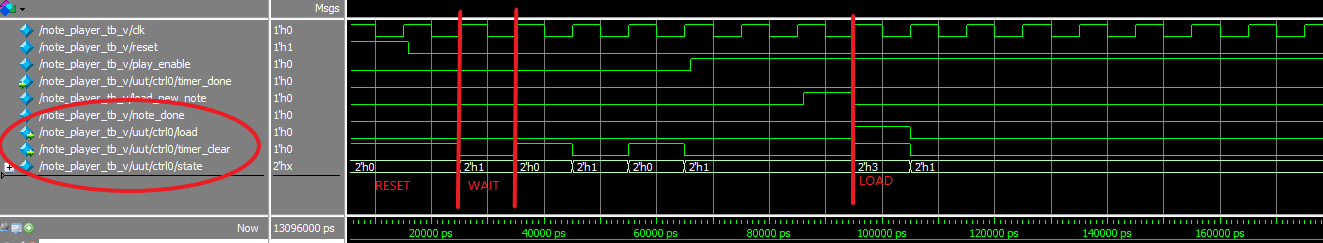
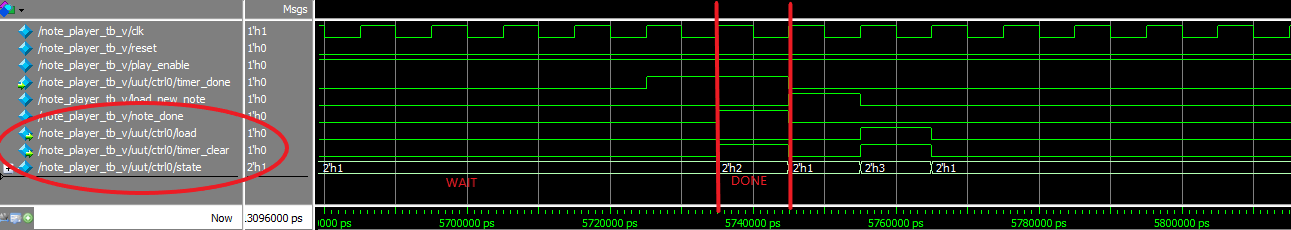
前、后两处放大图分别为：

图6.4.2.2 控制器仿真局部放大图（前）

图6.4.2.3 控制器仿真局部放大图（后）

分析：如图6.4.2.2所示，当reset = 1时，控制器state = 0，即为RESET状态，此时输出time\_clear = 1，load = 0，note\_done = 0；

在第一条直线处，时钟上升沿到来时，控制器自然进入WAIT状态（状态编码为1），此时time\_clear = 0，load = 0，note\_done = 0；

在第二条直线处，由于play\_enable = 0，控制器回到RESET状态，与ASM图要求一致；

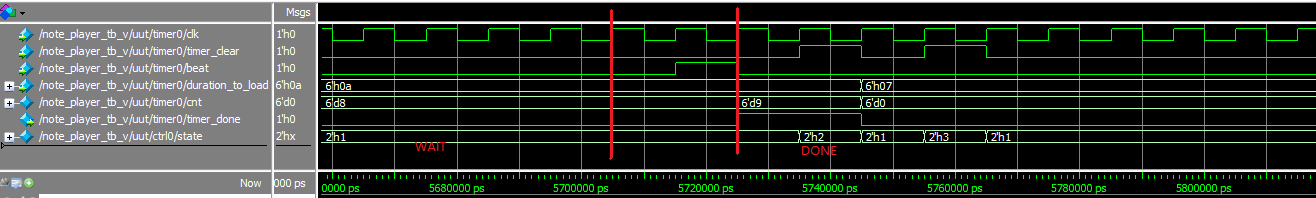
在第三条直线处，前一刻控制器状态为WAIT，由于时钟上升沿到来时，play\_enable =1，timer\_done = 0，load\_new\_note = 1，控制器状态跳转为LOAD，输出time\_clear = 1，load = 1，note\_done = 0；

如图6.4.2.3，在第一条直线处，前一刻控制器状态为WAIT，当时钟上升沿到来时，play\_enable =1，timer\_done = 1，使得控制器状态跳为DONE，输出time\_clear = 1，load = 0，note\_done = 1；

在第二条直线处，控制器自然跳转到WAIT状态，符合ASM图要求。

综上可知，控制器设计正确。

iii）音符节拍定时器

图6.4.3 音符节拍定时器仿真局部放大图

分析：如图，在第一条直线处，beat = 0，即使能信号为低电平，定时器不计数，cnt保 持为8；在第二条直线处，beat = 1，定时器计数，由于此时duration = 10，而cnt = 9 = 10-1， 定时结束，timer\_done = 1，符合要求。可以看见，此时控制器的状态由WAIT转为DONE， 与ASM图一致。综上，定时器设计正确。

⑤ 1000分频器模块

（1）Verilog HDL代码设计

1000分频器即为模1000的计数器，其状态位数应设为10位，因此，只需将 counter\_n.v中的两个参数n、counter\_bits设为对应值即可，对应测试代码可查看 counter\_n\_tb.v文件。

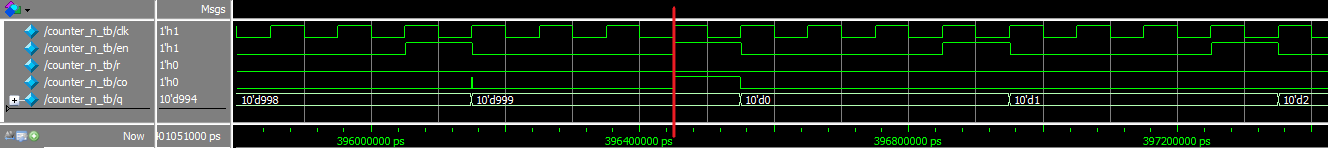
 （2）仿真结果

图6.5 1000分频器仿真结果

分析：如图，当使能信号en = 1时，分频器正常计数；当分频器计数到999，en = 1， 且时钟上升沿到来时，分频器进位输出co = 1，随后计数值清0，满足设计要求。

⑥ 同步化电路

（1）Verilog HDL代码设计

由原理图可知，同步化电路由两个D触发器构成，其设计代码如下：

module syn\_circ(in, clk, out);

input in, clk; // 输入

output out;

reg q1, q2; // 两个触发器的输出

always @ (posedge clk)

begin

q1 <= in; // 非阻塞赋值

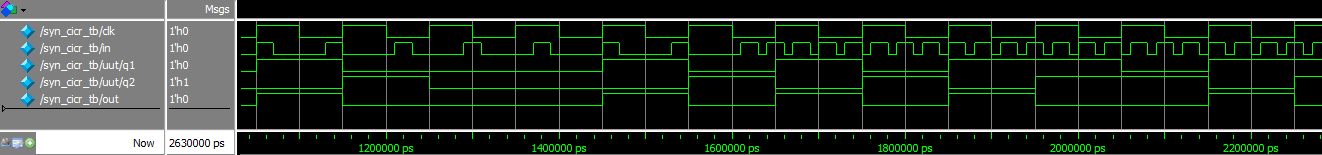
q2 <= q1;

end

assign out = q1 && (~q2);

endmodule

（2）仿真结果

图6.6同步化电路仿真结果

分析：如图，经过一段时间后，输出信号out与输入时钟in同步，且其脉冲宽度为一个in的时钟周期，设计符合要求。

⑦ 次顶层模块

（1）Verilog HDL代码设计

music\_player次顶层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，其代码如下：

module music\_player(clk, reset, play\_pause, next,

NewFrame, sample, play, song);

input clk, reset; // 高电平有效

input play\_pause; // 播放/暂停

input next; // 下一首输入

input NewFrame; // 高电平非同步脉冲，索取新的样品

output[15:0] sample; // 正弦样品输出

output play; // 播放状态指示

output[1:0] song; // 曲号指示

wire reset\_play; // mcu复位模块输出

wire song\_done, new\_note; // song\_reader输出信号

wire[5:0] note, duration; // 音符标记及持续时间

wire note\_done, sample\_ready; // note\_player输出信号

wire ready; // 同步化电路输出

wire beat; // 节拍基准产生器输出

parameter sim = 0;

// 主控制器mcu实例

mcu m0(.clk(clk), .reset(reset), .play\_pause(play\_pause), .next(next), .song\_done(song\_done), .play(play), .reset\_play(reset\_play),

.song(song));

// 乐曲读取song\_reader实例

song\_reader s0(.clk(clk), .reset(reset\_play), .play(play),

.song(song), .note\_done(note\_done),

.song\_done(song\_done), .note(note),

.duration(duration), .new\_note(new\_note));

// 音符播放note\_player实例

note\_player n0(.clk(clk), .reset(reset\_play), .play\_enable(play),

.note\_to\_load(note), .duration\_to\_load(duration),

.load\_new\_note(new\_note), .note\_done(note\_done),

.sampling\_pulse(ready), .beat(beat), .sample(sample),

.sample\_ready(sample\_ready));

// 同步化电路syn\_circ实例

syn\_circ syn0(.in(NewFrame), .clk(clk), .out(ready));

// 节拍基准产生器实例，根据sim的值选择对应模式

counter\_n #(.n(sim ? 64 : 1000), .counter\_bits(sim ? 6 : 10))

c0(.clk(clk), .en(ready), .r(0), .q(), .co(beat));

endmodule

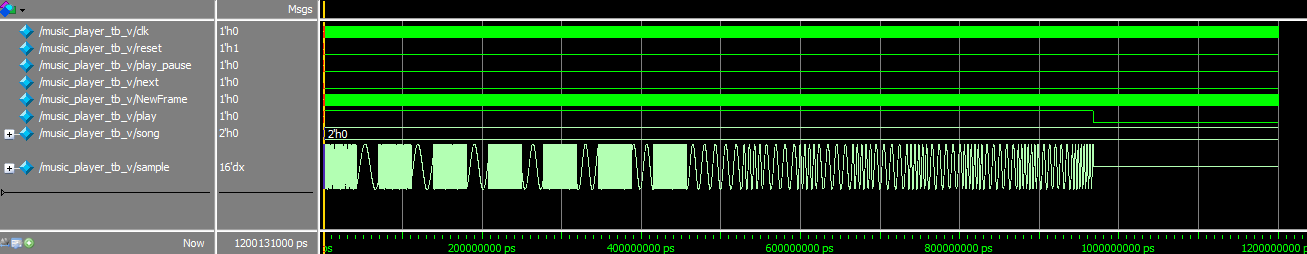
 （2）仿真结果

图6.7次顶层模块仿真结果

分析：如图，仿真结果与书中要求基本一致。

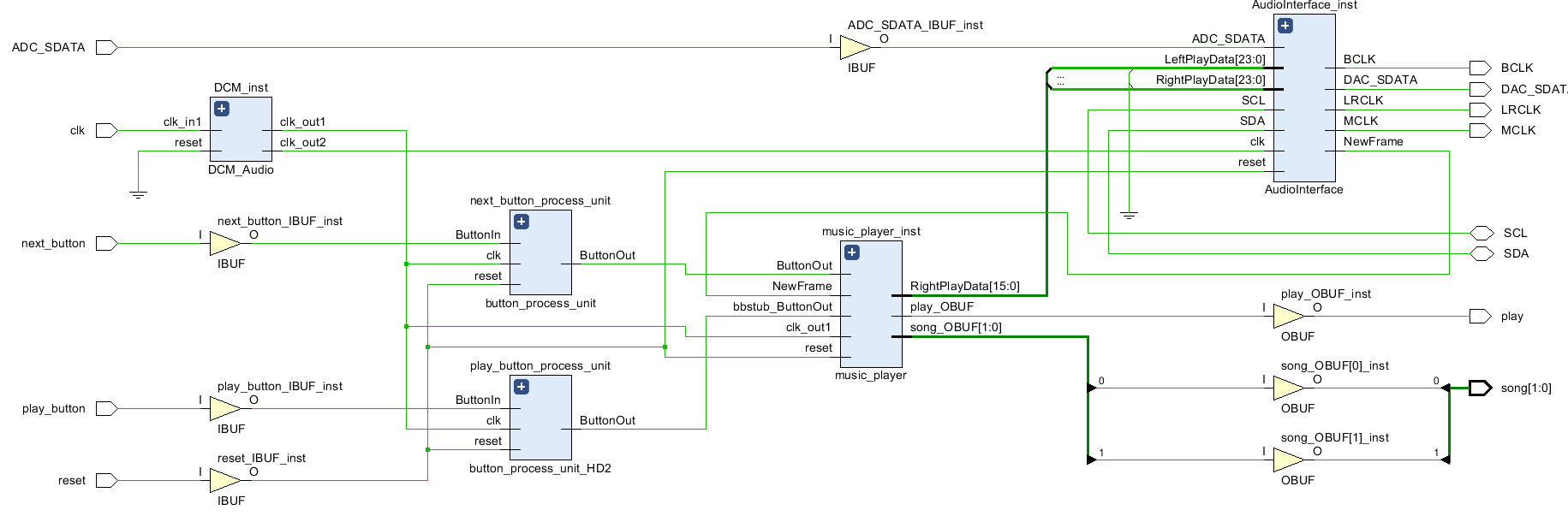
 ⑧ Vivado工程建立

图6.8 Vivado工程建立

**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？

答：因为reset为置零信号，当reset = 1时，系统置零，之后不论reset为0或1，系统内部 应置零的信号都为0，其颤动对系统无影响。而next\_button和play\_pause\_button都会 影响系统内部状态的转换，若next输入不稳定，可能会使播放的下一首歌连续跳转， 引起不确定；若play\_pause输入不稳定，则系统会在播放和暂停之间不断切换，导致输 出断断续续，并增加系统的损耗，影响系统的使用效果。

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。由控制器ASM图可知，从RESET到PAUSE转换，以及从NEXT到PLAY转换 时，并没有判断play\_pause和next，这期间按下按钮将接收不到按键信息。由于状态转 换的间隔只有一个时钟周期，接收不到按键信息的概率很小，因此没有必要修改设计。

**八、心得与体会**

此次实验，我们在完成前期组合逻辑电路和时序逻辑电路的基础上，设计完成了一个音乐播放器，不仅学会了状态机的描述，也能够自主编写测试代码，对模块进行测试、纠错，更重要的是对“自顶而下”的数字系统设计方法有了更加深刻的认识，受益颇丰。

总的来说，这次实验是比较困难的。由于最初看不懂算法流程图，我花了较长的时间进行摸索，对自己设计出来的控制器也将信将疑。直到学习了学在浙大上的视频，我对状态机描述的方法有了一个大体的了解，结合书中的ASM图，方才掌握了一些门道。根据模块化的设计方法，借助书中的结构框图和端口含义，我在逐一完成各个功能模块的同时，也对数字系统设计的流程和方法有了更新的认识。子功能模块的编写、顶层设计中的调用，将功能和结构两者有机地结合起来，不仅使代码结构清晰明了，也有利于我们的编写和调试。当然，在实验过程当中，我也遇到了其他一些问题，主要有以下两个：

1、端口、变量的名称写错。在对音符播放部分进行仿真时，我发现timer\_done信号始终为0，导致控制器出错。通过对音符定时器模块进行调试，对整个部分的代码进行检查，我发现原来是因为自己将顶层设计中的单词load错写成了laod，导致变量名出错。修正之后进行仿真，所得结果正确；

2、变量长度忘了设置。同样是在音符播放部分，在调试过程中，我发现DDS模块的输入始终为高阻态，使得sample无法输出正弦信号。通过检查相关模块的波形，我发现dout存在错误。查看代码，原来是其长度忘了设置，导致传递给DDS的变量出错。修正之后进行仿真测试，所得结果正确。

从上述两个问题当中，我已经深刻认识到了细致的重要性。相较而言，任何一点小小的错误，在工程当中都可能会产生巨大的影响。不论是在课堂中的实验，还是我们以后的工作生涯，都要时刻保持细心谨慎、掌握原理和方法，才能把事情做得更好。