音乐播放实验报告

卢泽熙 3220102478 周四 9、10 节

目录

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） 掌握系统自上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，并设置 play/pause_button、next_button、reset 三个按键，按下 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。

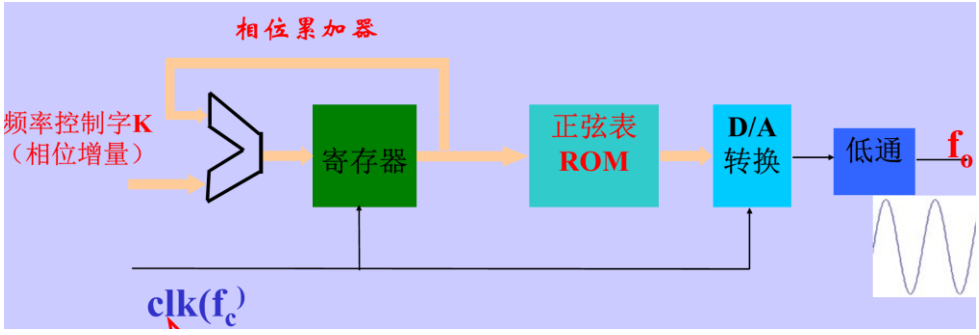
三. 实验原理及设计思路

1. DDS

1）基本原理

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。在此过程中，DDS 技术通过改变寻址的步长（对数字波形查表的相位增量）来改变输出信号的频率。

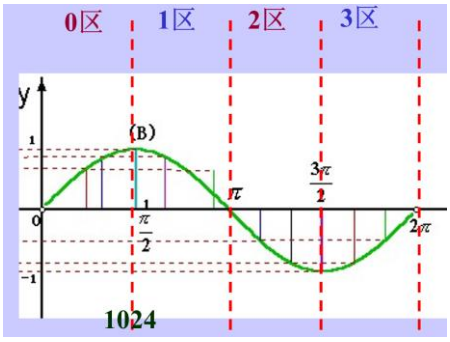
DDS 原理框图如下所示：



输出正弦信号频率满足： $f_0 = \frac{K}{2^m} f_c$ ，其中 m 为 Sine ROM 地址线位数。要得到某一频率的正弦信号，相位增量 K 值不一定为整数。因此，为了得到更准确的正弦信号频率， K 的位数会增加几位小数。

相位累加器的位数由 m 位整数和 p 位小数组成，高 m 位整数部分作为 Sine ROM 的地址。根据 DDS 输出信号的最低频率要求，可计算出 m = 12；同时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。

根据分析，存储一个完整周期的正弦信号样品可采用 2^{12} × 16bits 的 ROM。但由于正弦波形的对称性，可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（0 区）样品即可，如下图所示：

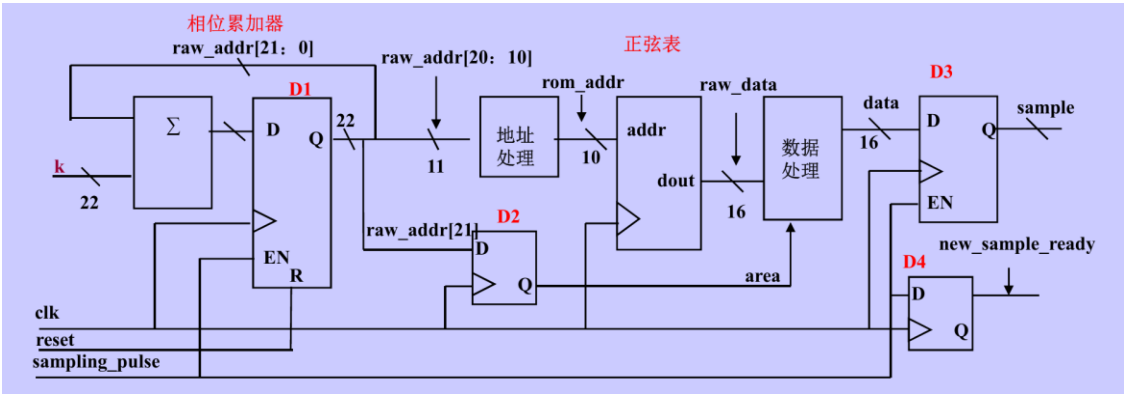


0 区	从 0 区中复制另外 3 个区的数据
1 区	1 区第 N 个样品与 0 区第（1024-N）个样品相同
2 区	2 区第 N 个样品与 0 区第 N 个样品幅度相同，极性相反
3 区	3 区第 N 个样品与 0 区第（1024-N）个样品幅度相同，极性相反

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 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）。

2） 总体设计

DDS 总体电路框图如下所示：



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，处理方法如下表所示：

正弦区域	**sine ROM 地址**	**正弦样品 data**	**说明**
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	地址镜像翻转并且数据取反

3） Verilog 描述

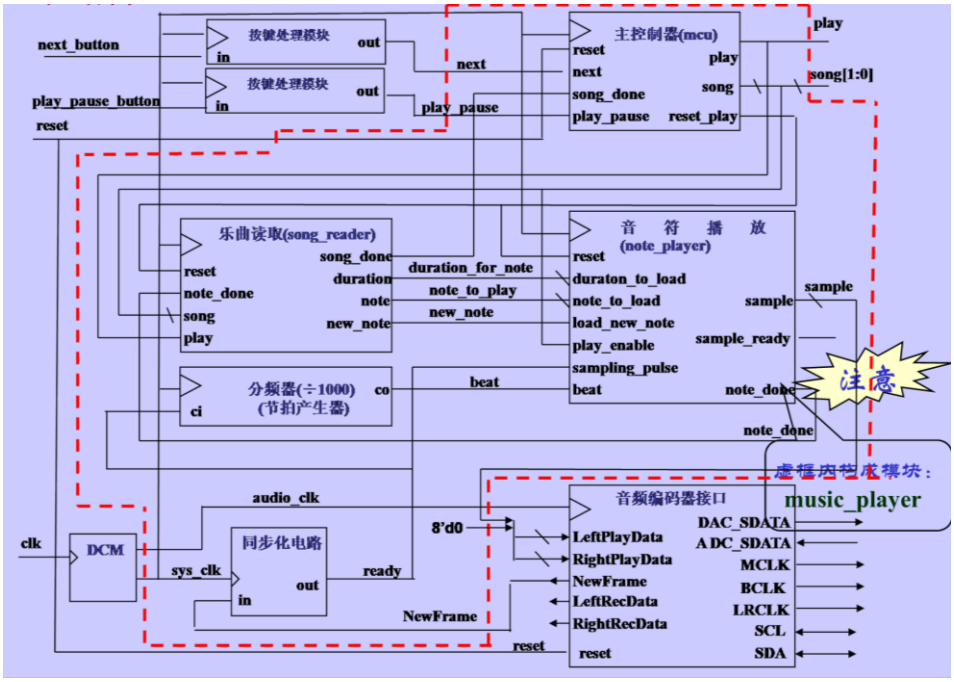
DDS 顶层模块的 Verilog 代码如下所示：

```
module dds(clk, reset, sampling_pulse, k, sample, new_sample_ready);
    input clk, reset, sampling_pulse;
    input [21:0] k;          // Phase Increment value
    output [15:0] sample;
    output new_sample_ready;

    wire [21:0] raw_addr;
    wire [21:0] temp;        // Temporary variable for holding adder's output
    wire [9:0] rom_addr;
    wire [15:0] raw_data;
    wire [15:0] data;
    wire area;

    // Full Adder: temp = k + raw_addr, adding phase increment to the current address
    full_adder#(.n(22)) adder_inst(.a(k), .b(raw_addr), .s(temp), .co(), .ci(0));

    dffr#(.n(22)) dffr1(.d(temp), .r(reset), .en(sampling_pulse), .clk(clk), .q(raw_addr));

    // Address Processor
    addr_process addr_process_inst(.addr(raw_addr[20:10]), .rom_addr(rom_addr));

    // ROM outputting raw sine wave samples
    sine_rom sine_rom1(.clk(clk), .addr(rom_addr), .dout(raw_data));

    dffr#(.n(1)) dffr2(.r(0), .en(1), .clk(clk), .d(raw_addr[21]), .q(area));

    assign data[15:0] = area? (~raw_data[15:0]+1) : raw_data[15:0]; // Data Processing

    dffr#(.n(16)) dff3(.r(0), .en(sampling_pulse), .clk(clk), .d(data), .q(sample));

    dffr#(.n(1)) dff4(.r(0), .en(1), .clk(clk), .d(sampling_pulse), .q(new_sample_ready));

endmodule
```

2. 音乐播放器

1） 总体设计

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(mcu)、乐曲读取(song_reader)、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，如下图所示：



各主要子模块作用如下：

- **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 模块是否要播放及播放哪首乐曲；
- **乐曲读取模块(song_reader)**根据 mcu 模块的要求，逐个取出音符信息{note, duration}送给 note_player 模块

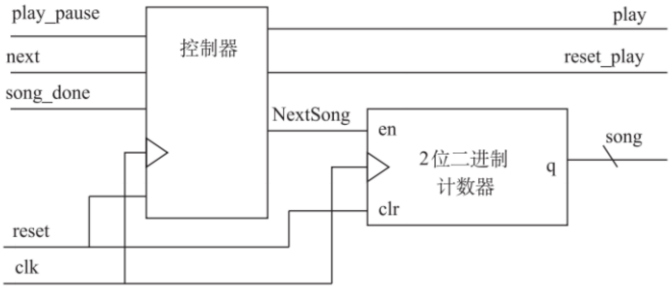
播放，当一首乐曲播放完毕，回复 mcu 模块乐曲播放结束信号 song_done；

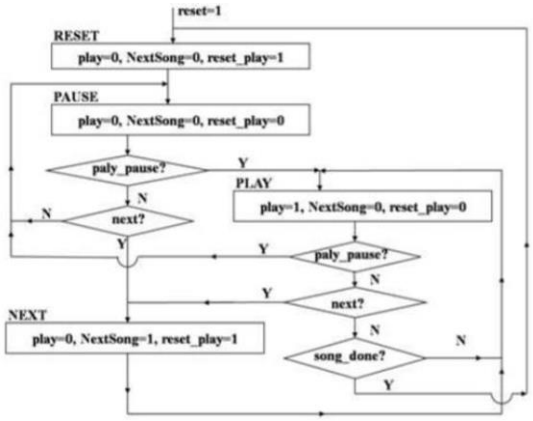
- **音符播放模块(note_player)**接收到需播放的音符，在音符的持续时间内，以 48kH 速率送出该音符的正弦波样品至音频编解码接口模块。当一个音符播放结束，向 song_reader 模块发送一个 note_done 脉冲索取新的音符。
- **音频编解码接口模块**负责将音符的正弦波样品转换为串行输出并发送给音频编解码芯片 ADAU1761。音频编解码芯片 ADAU1761 接收正弦波样品，再进行 AD 转换并放大，最后送至扬声器播放。注意，note_player 模块产生的正弦波样品为 16 位二进制，需在低位加 8 个 0 后送入音频编解码接口模块。
- **节拍基准产生器**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。
- **按键处理模块**完成输入同步化、防颤动和脉宽变换等功能。

2）子模块设计思路

(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。其原理框图如左图所示；控制器的算法流程图如右图所示。二进制计数器用来计算乐曲序号。



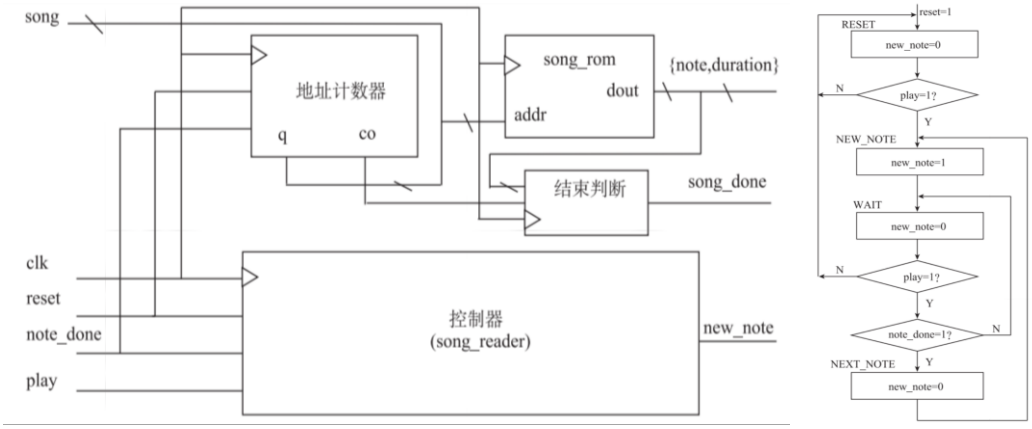


主控制模块 mcu 的顶层代码如下所示：

```
module mcu(
    input clk,
    input reset,
    input play_pause,
    input next,
    input song_done,
    output play,
    output reset_play,
    output [1:0] song
);
    wire NextSong;  // Internal signal to trigger the next song

    // Instantiate the counter module to track the current song
    counter_n #(.n(4), .counter_bits(2)) counter1(
        .clk(clk),
        .en(NextSong),
        .r(reset),
        .q(song),
        .co()
    );
    // Instantiate the control module to manage play/pause, next song, and reset signals
    mcu_control mcu_control_inst(
        .clk(clk),
        .reset(reset),
        .play_pause(play_pause),
        .next(next),
        .song_done(song_done),
        .NextSong(NextSong),
        .reset_play(reset_play),
        .play(play)
    );
endmodule
```

(2) 乐曲读取 song_reader 模块的设计

song_reader 模块结构框图与控制器算法流程图如下所示：



song_rom 是一个只读存储器，用来存放乐曲，容量为 27 × 12bits，共存放四首乐曲，每首乐曲占用 27 ×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。

地址计数器为 5 位二进制计数器，其中 note_down 为计数器使能输入，当 note_down 为高电平时允许计数。

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；结束判断模块采用状态机实现脉宽变换电路。

song_reader 顶层模块 Verilog 代码如下：

```
module song_reader(clk, reset, note_done, play, song, song_done, new_note, duration, note);
    input clk, reset, note_done, play;
    input [1:0] song;
    output song_done, new_note;
    output [5:0] duration, note;

    wire co;
    wire [4:0] q;

    // 5-bit counter with 32 as modulus
    counter_n #(.n(32), .counter_bits(5)) conter_inst2(.clk(clk), .en(note_done), .r(reset), .q(q), .co(co));
    // Control logic for song reading
    song_reader_control song_reader_control_inst(.clk(clk), .reset(reset), .note_done(note_done), .play(play), .new_note(new_note));
    // ROM for storing song notes and durations
    song_rom song_rom_inst(.clk(clk), .dout({note[5:0], duration[5:0]}), .addr({song[1:0], q[4:0]}));
    // Checks song duration and completion
    judgment judgment_inst(.clk(clk), .duration(duration), .co(co), .song_done(song_done));

endmodule
```

(3) 音乐播放 note_player 模块的设计

音乐播放模块 music_player 是本实验的核心模块，它主要任务有：

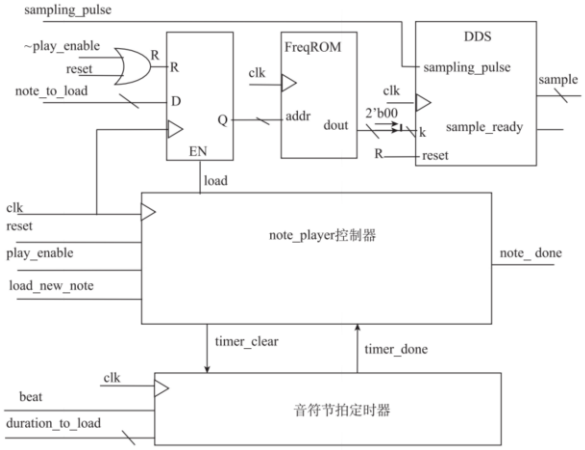
（1）从 song_reader 模块接收需播放的音符{note, duration}；

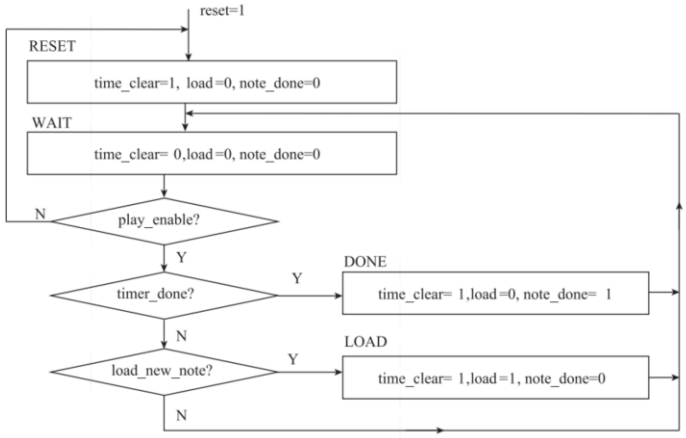
（2）根据 note 值找出 DDS 的相位增量 k；

（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。

note_player 结构框图如下左图所示：





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。

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上右图所示：

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。

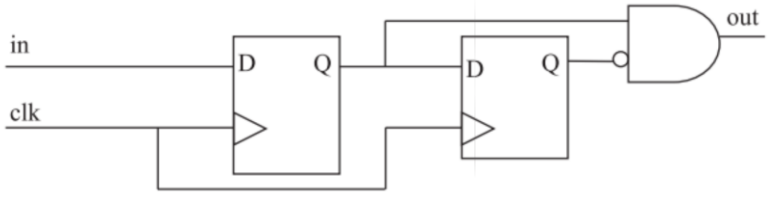
子模块 DDS 的功能就是利用 DDS 技术产生正弦样品，需要注意的是，DDS 模块的输入 k 为 22 进制，因此需要 FreqROM 输出的 20 位相位增量高位加 2 个 0 后接入 DDS。

note_player 顶层模块 Verilog 代码如下所示：

```
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, 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;

    // Flip-flop to hold current note
    wire [5:0] qout;
    wire load;
    dffr #(.n(6)) dffr_inst(.r(~play_enable || reset), .en(load), .clk(clk), .d(note_to_load), .q(qout));
    // ROM to get frequency value for note
    wire [19:0] dout;
    frequency_rom frequency_rom_inst(.clk(clk), .addr(qout), .dout(dout));
    // DDS to generate sample
    dds dds_inst(.clk(clk), .reset(~play_enable || reset), .sampling_pulse(sampling_pulse), .k({2'b00, dout}), .sample(sample),
.new_sample_ready(sample_ready));
    // Control unit for note player
    wire timer_clear, timer_done;
    note_player_control note_player_control_inst(.clk(clk), .reset(reset), .play_enable(play_enable), .load_new_note(load_new_note),
.note_done(note_done), .load(load), .timer_clear(timer_clear), .timer_done(timer_done));
    // Timer to keep track of note duration
    timer timer_inst(.clk(clk), .en(beat), .timer_clear(timer_clear), .duration_to_load(duration_to_load), .q(), .timer_done(timer_done));
endmodule
```

（4）同步化电路的设计

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。电路由同步器和脉冲宽度变换电路组成：



同步化电路 Verilog 描述如下：

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

    wire q1, q2;
    assign out = q1 && ~q2;

    dffr #(.n(1))dffr1(.r(0), .en(1), .clk(clk), .d(in), .q(q1));
    dffr #(.n(1))dffr2(.r(0), .en(1), .clk(clk), .d(q1), .q(q2));
endmodule
```

(5) 时钟管理模块

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 12.5MHz 的音频时钟。

(6) 节拍基准产生器模块

节拍基准产生器产生 48kHz 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一个分频比为 1000 的分频器。Verilog 代码如下：

```
module counter_n(clk, en, r, q, co);
  parameter  n = 1000;
  parameter  counter_bits = 10;
  input   clk, en, r;
  output co;
  output reg [counter_bits-1:0]  q = 0;

  assign  co = (q == (n-1)) && en;               // carry generates when q equals to n-1 and en signal is true
  always @(posedge clk)
  begin
      if(r) q = 0;                               // if r is true, then reset q to 0
      else if(en)
        begin
          if(q == (n-1))  q = 0 ;          // if q equals to n-1, then reset q to 0
          else q = q + 1;
        end
      else q = q;
  end

endmodule
```

3）顶层模块 Verilog 描述

music_player 顶层模块 Verilog 描述如下：

```
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 [1:0] song;
    output play;

    wire [5:0] note, duration;
    wire [1:0] song;
    wire play, reset_play, song_done, ready, note_done, new_note, beat;

    mcu mcu_inst(.clk(clk), .reset(reset), .play_pause(play_pause), .next(next), .song_done(song_done), .song(song), .reset_play(reset_play),
.play(play));

    song_reader song_reader_inst(.clk(clk), .reset(reset_play), .note_done(note_done), .play(play), .song(song), .song_done(song_done),
.new_note(new_note), .duration(duration), .note(note));

    note_player note_player_inst(.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());

    synchronizer synchronizer_inst(.in(NewFrame), .clk(clk), .out(ready));

    counter_n #(.n(sim?64:1000), .counter_bits(sim?6:10))counter_inst1(.clk(clk), .en(ready), .r(0), .q(), .co(beat));
endmodule
```

四. 主要仪器设备

（1） 装有 Vivado 和 ModelSim SE 软件的计算机；

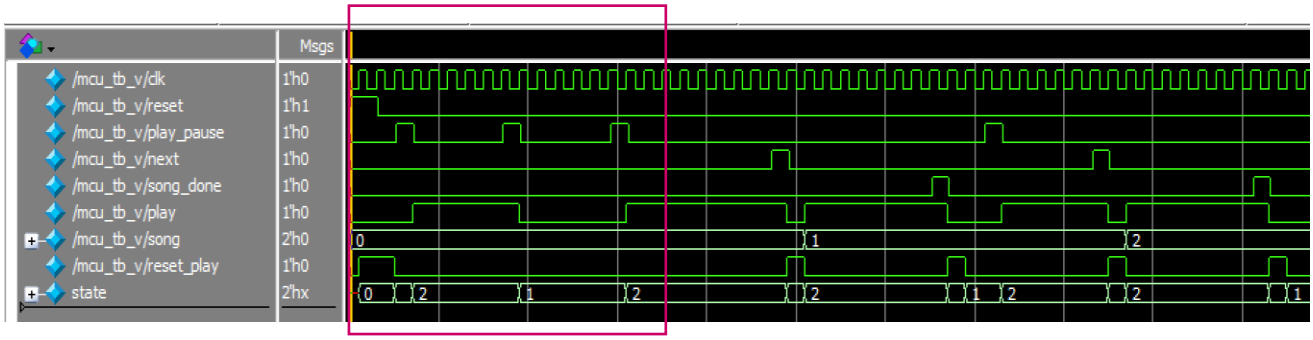
（2） Nexys Video Artix-7 FPGA 多媒体音视频智能互联开发系；

（3） 有源音箱或耳机。

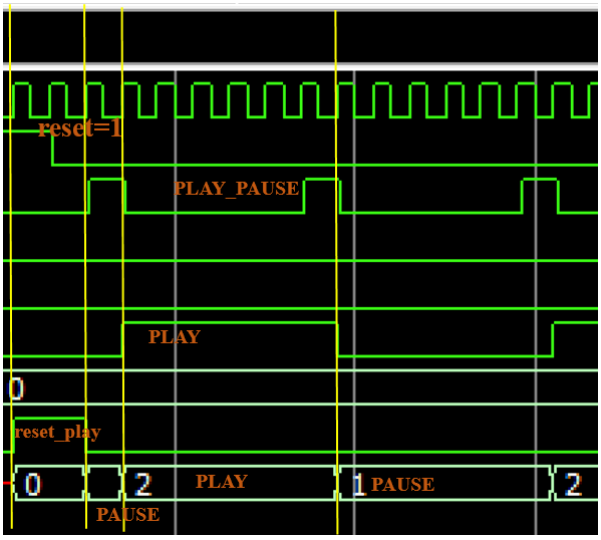
五. 仿真结果

1. 主控制 mcu 模块

仿真波形部分截图如下所示：



对矩形框内波形放大显示：

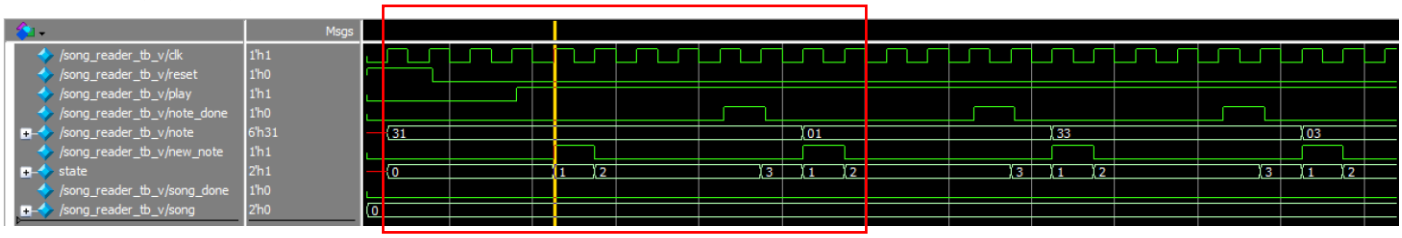


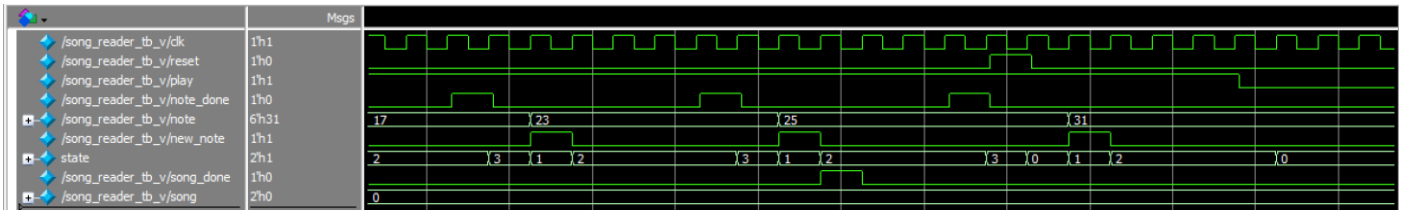
结合 ASM 图，以上面四条黄线为例分析仿真波形的正确性：

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。 后续部分波形分析方法与该部分类似，经过仔细分析得出结论该模块设计符合要求。

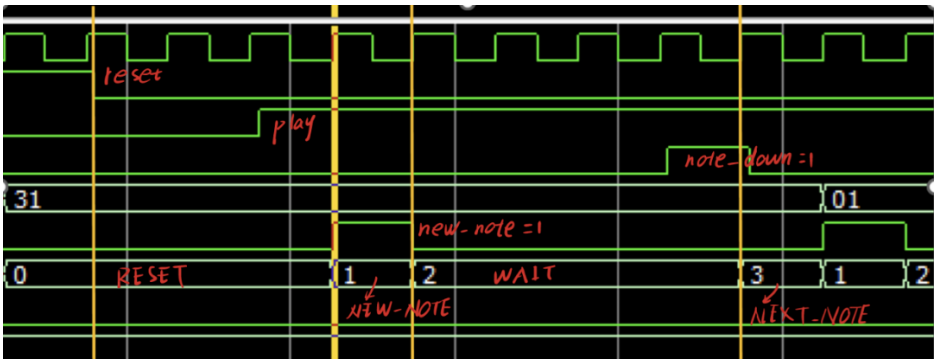
2. 乐曲读取 song_reader 模块

该部分完整仿真波形如下所示：





截取红色框内部分波形进行分析：

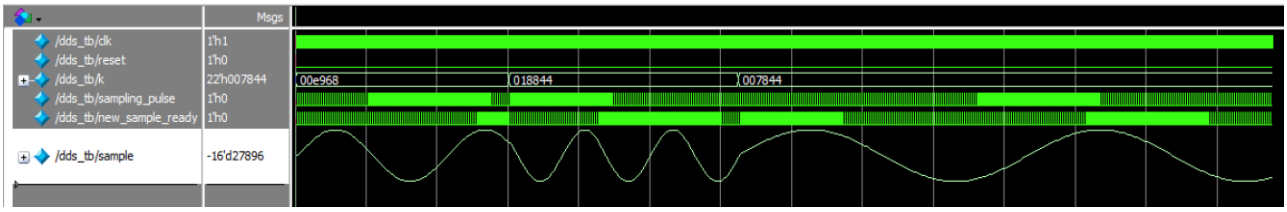


由波形图可见，当 reset = 1 时，进入 RESET 状态；当 play = 1 时，进入 NEW_NOTE 状态并输出 new_note = 1；然后进入 WAIT 状态，输出 new_note = 0；当 note_done = 1 时，下一个时钟上升沿到来时，进入 NEXT_NOTE 状态，输出 new_note = 0，并读取下一个音符的地址；然后下一个时钟上升沿到来时，进入 NEW_NOTE 状态，输出 new_note = 1，下一个音符播放。该波形图与设计相符合。

3. 音符播放 note_player 模块

1） 子模块 DDS

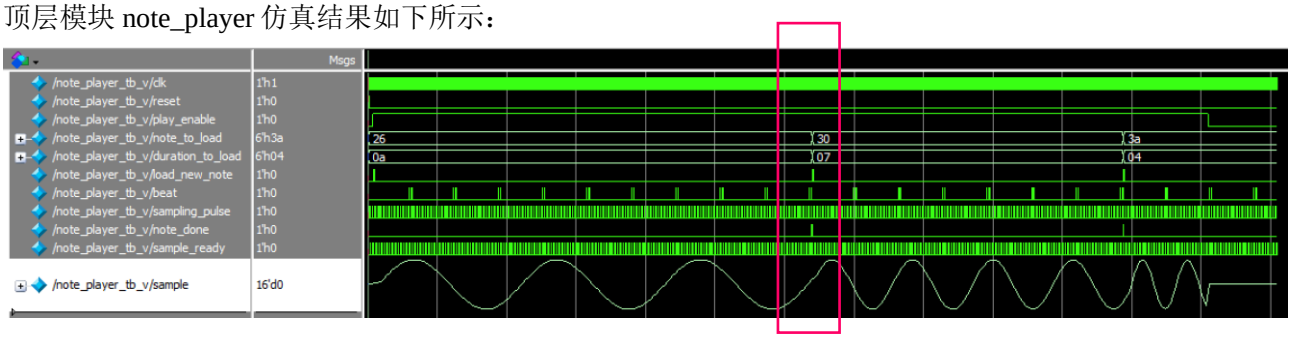
子模块 DDS 仿真结果如下所示：

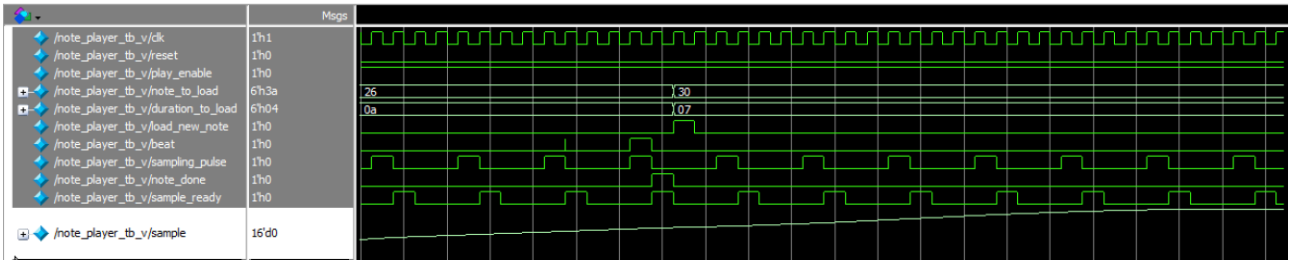


由仿真结果可见，当 k 较大时，正弦波频率较大；k 值变化的转折点即频率变化转折点，设计符合要求。

2） 顶层模块 note_player

顶层模块 note_player 仿真结果如下所示：



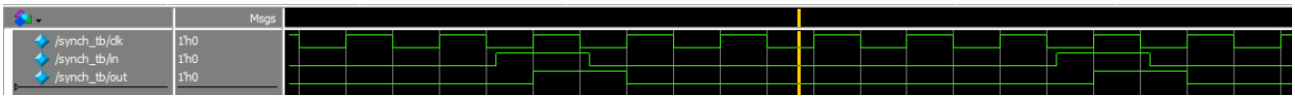


根据第一张波形图可见，正弦信号频率与 note_to_load 的值呈正相关，且当 play_enable=0 时，sample=0。

第二张图为第一张图红色矩形框内放大部分，当 note_down 信号变为高电平，load_new_note 信号产生，自动更新播放的音符。综上，波形符合预期，该子模块设计成功。

4. 同步电路模块

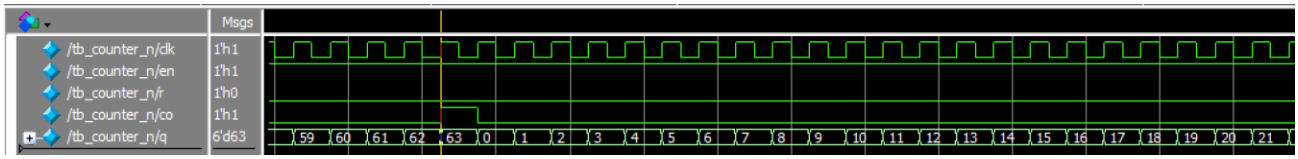
同步电路模块波形仿真结果如下：



根据仿真结果，out 信号总是跟随 in 信号出现，其脉冲宽度为一个 clk 周期且与 clk 同步，设计符合要求。

5. 计数器模块

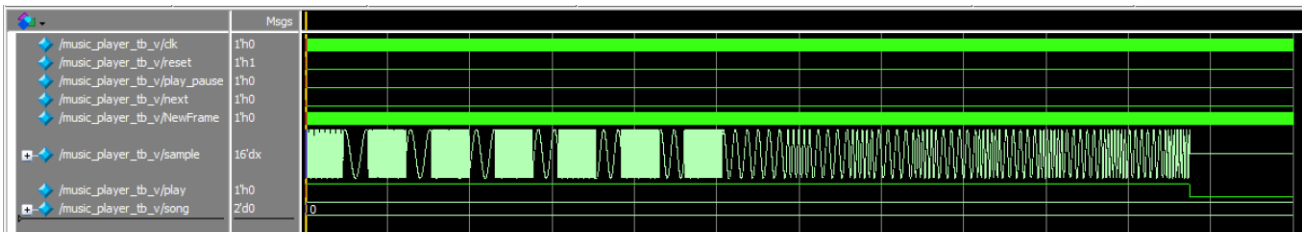
仿真时将分频器改为小分频比的分频器模块以减少仿真时间，设置 n=64，counter_bits=6，仿真波形如下：



可以看到，在 q 等于 63 时进位信号 co 产生，随后 q 置零重新开始计数。设计结果符合预期。

6. 项目顶层模块 music_player

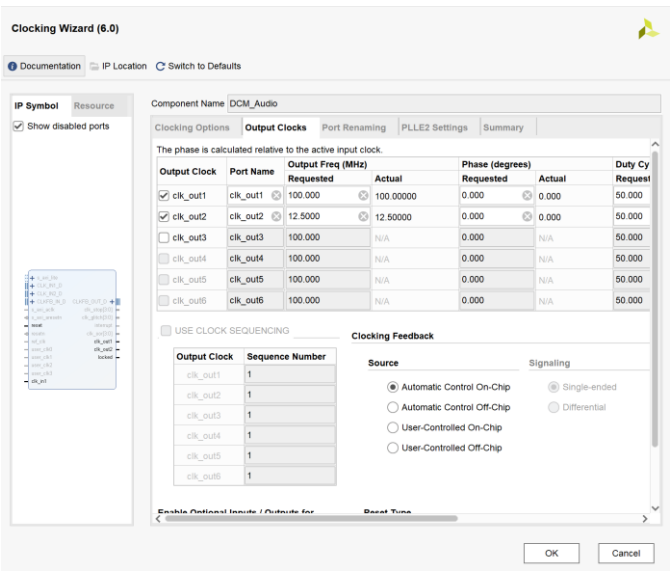
仿真波形如下图所示：



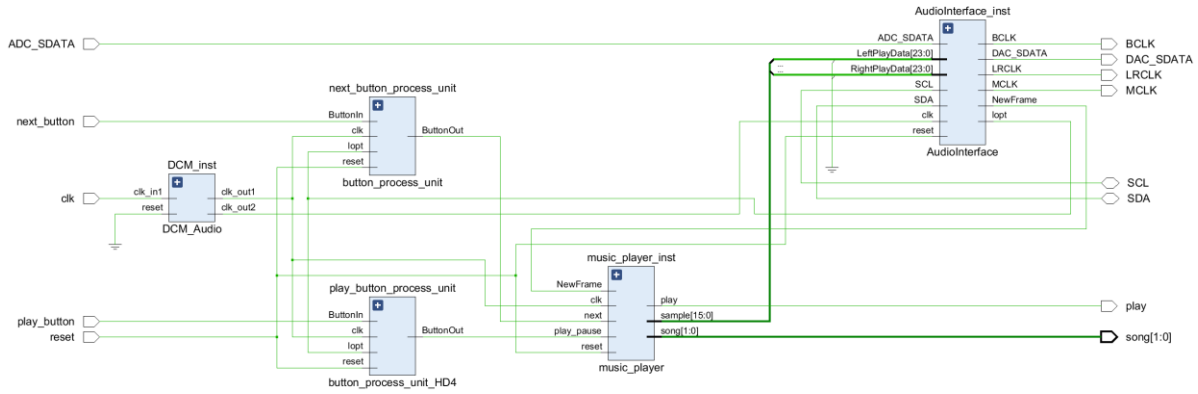
play 信号为 1 时 sample 产生变化的正弦信号，波形基本符合要求。

六. Vivado 工程

编写项目最顶层 MusicPlayerTop 的 Verilog HDL 代码（实验中已给出）建立 Music_player 的 Vivado 工程，添加 music_player 模块及其子模块、音频编解码接口模块的网表文件和端口文件、按键处理模块的网表文件和端口文件以及约束文件，同时生成符合要求的 DCM 内核，如下所示：



对工程进行综合，实验综合后得到电路原理图如下所示：



下载工程文件到开发板后，按下相关按键：

1. 按下中间键实现复位；
2. 按下右边键实现暂停/播放；
3. 按下下方键实现切换歌曲的功能。

验收过程中播放音乐正常，功能正常实现，至此整个项目成功完成设计！

七. 思考题

思考题 1

因为 reset 是置零信号，当 reset=1 时，系统置零，之后无论 reset 为 0 或 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 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。

思考题 2

存在接收不到按键信息的可能。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

本次实验我独立完成了音乐播放器的设计、Verilog 代码编写、仿真以及综合测试工作，从入手子实验到最终完成大作业的验收，前前后后花费了我大约一个月的时间。在这个过程中我巩固了专业课程中的理论知识，包括控制器、计数器、同步电路等内容，同时积累了许多 Verilog 设计经验。

一开始的子实验 DDS 设计充满了挑战，实验过程中，我遇到了许多困难。首先，正弦表对我来说是一个全新的概念，并且数模转换这部分的理论知识我们也还没有学，这对我理解设计思路造成了一定的阻碍；其次，地址

处理、数据处理两个模块需要自己基于实验原理独立设计，为了确保这两个关键模块的正确性，我花费了较长时间去彻底弄懂设计的基本原理。

完成子实验后我就开始正式着手音乐播放器的设计。音乐播放器的设计思路与原理框图教材上都给了，在经过前面一系列 Lab 的训练之后，音乐播放器各子模块以及 music_player 总体模块的 Verilog 代码编写反而意想不到的轻松，唯一有一点挑战性的是根据 ASM 图设计各模块的控制器。我仿照教材的示例代码并结合《数字系统设计》课程所掌握的理论知识，慢慢完成了第一个子模块 mcu 中的控制器设计部分，并在此过程中熟悉了控制器 verilog 代码编写的标准流程与规范形式。后面几个子模块的控制器设计我按部就班的完成，基本没有什么困难。

不过毕竟是第一次设计这么大的一个工程，实验过程中我还是遇到了不少大大小小的问题的。一些小问题包括：full_adder 忘记给输入数据设置位数参数，在 DDS 相位累加器模块无法正常工作；总模块仿真忘记设定分频比；端口命名和提供的_tb 文件不符合等等。这些问题都能很快发现并及时解决。实验中困扰我最久的一个故障发生在顶层模块 music_player 的仿真——之前子模块 mcu、song_reader、music_player 仿真波形都没有什么问题，但是顶层模块 sample 信号经历了极短的波动后保持为零。这个问题我一开始根本不知道为什么会出现，后来经过反复排查才发现 song_reader 模块中的计数器 counter_n 使能控制信号 en 应该接 note_done，但我却直接将它置 1 了。这个问题也再提醒我设计过程中一个看似肯细微的差别可能会对最终结果造成巨大的影响。

最终完成验收的那一刻我的内心是十分激动的，在本学期的实验学习中我了解熟悉了之前未接触的 Verilog 语言，同时掌握了数字系统设计的基本流程，领会到了自顶向下的设计思想。做完实验后感觉实际的电路设计真的充满挑战，毕竟我们在有教材、老师辅导，有设计流程提示，有正确结果示例的情况下设计都感到困难重重，完全独立的数字系统设计过程显然具有远高于此的能力要求。我意识到在这个领域我还需要学习大量知识，进行大量练习来提升自己的数字系统设计能力。