



重慶理工大學

可编程器件原理及应用

题 目： 乐曲演奏电路

专业班级：

学 号：

姓 名：

联系电话：

指导老师：

时 间： 2023.6.21

目录

一、实验目的	1
二、实验原理	1
2.1 硬件发声原理	1
2.2 音符频率的获得	1
2.3 乐曲节奏的控制	2
2.4 音符及音阶的显示	2
2.5 蜂鸣器驱动电路	2
2.6 4×4 矩阵按键	2
2.7 PWM（脉冲宽度调制）	3
三、实验器材	3
四、实验要求及实验内容	4
4.1 系统设计思想	4
4.2 顶层模块设计	4
4.2.1 锁相环 PLL20	4
4.2.2 分频器 FDIV	4
4.2.3 计数器 CNT138T	4
4.2.4 数控分频模块 SPKER	4
4.2.5 存储器 ROM	5
4.2.6 四选一数据选择器	5
4.2.7 矩阵键盘驱动模块	5
4.2.8 PWM 模块	5
4.3 模块组合与管脚分配	6
五、实验数据处理与结果分析	6
5.1 数据处理	6
5.2 结果分析	7
六、实验结论	8
七、实验中的问题及解决办法、改进方向	8
八、参考文献	8
九、附录：	9
FDIV	9

CNT138T	9
RAM78.....	9
mux4_1	10
F_CODE.....	10
SPKER	11
Electric_Piano.....	11
Array_KeyBoard	12
Beeper.....	14
PWM.....	15
tone	15
DECL7S	16
.mif-maker.....	17
《Outer Wilds》	22
《蜜雪冰城》	22
《爱你》（部分）	23
《one last kiss》（部分）	23
《我爱北京天安门》（部分）	24

一、实验目的

1. 基于小脚丫 STEP MAX10 FPGA 开发板实现以下要求

基本要求：完成教材 195 页上的实验 7-4，能够演奏歌曲《梁祝》。

扩展要求：

(1) 替换歌曲为其它歌曲（初级扩展）

(2) 在 ROM 中装上多支歌曲，可手动或自动循环演奏歌曲。（高级扩展 1）

(3) 用 4*4 键盘，设计一个电子琴，可弹奏歌曲（高级扩展 2）

2. 进一步熟悉 Quartus II 及与 FPGA 硬件资源的使用方法。

3. 熟悉 PWM 信号发生驱动模块和矩阵键盘驱动模块的应用。

4. 了解无源蜂鸣器的驱动原理及方法。

二、实验原理

2.1 硬件发声原理

硬件发声的原理声音的频谱范围约在几十到几千赫兹，只要利用程序来控制小脚丫 STEP MAX10 FPGA 开发板某个引脚输出一定频率的矩形波，接上扬声器就能发出相应频率的声音。乐曲中的每一音符对应着一个确定的频率，因此，要想蜂鸣器发出不同音符的音调，实际上只要控制 FPGA 开发板输出相应音符的频率即可。乐曲都是由一连串的音符组成，要想让硬件电路准确地演奏出一首乐曲，不仅要控制电路能按照乐曲的乐谱依次输出这些音符所对应的频率，还必须准确地控制乐曲的节奏，即每个音符的持续时间。因此，乐曲中每个音符的发音频率及其持续的时间是乐曲能够连续演奏的两个关键因素。

2.2 音符频率的获得

在 FPGA 设计中，多个不同频率的信号，一般是通过对某个基准频率进行分频获得的。由于各个音符的频率多为非整数，而分频系数又不能为小数，故必须将计算得到的分频系数四舍五入取整。若基准频率过低，则分频系数过小，四舍五入取整后的误差较大。若基准频率过高，虽然可以减少频率的相对误差，但分频电路耗用的资源会增加。实际设计中应该综合考虑这两个方面的因素，在尽量减少频率误差的前提下，选取比较合适的基准频率。

在本实验中，选取基准频率为 1MHz。由于小脚丫 STEP MAX10 FPGA 开发板上有 12MHz 的晶振，故只需对其进行 12 分频，即可获得 1MHz 的基准频率信号。

音频	C (1)	D (2)	E (3)	F (4)	G (5)	A (6)	B (7)
中央 C	262	296	330	349	392	440	494
高音	523	587	659	698	784	880	988

$$\text{音符的发生频率} = \frac{\text{基准频率}}{2 \times (2048 - \text{预置数})}$$

由此可以算出不同音符对应的预置数，详见 5.1 数据处理。

对于乐曲中的休止符，只要将预置数设为最大值，为 $2^{11}-1=2047$ ，即 H7FF，此时扬声器将不会发声。

2.3 乐曲节奏的控制

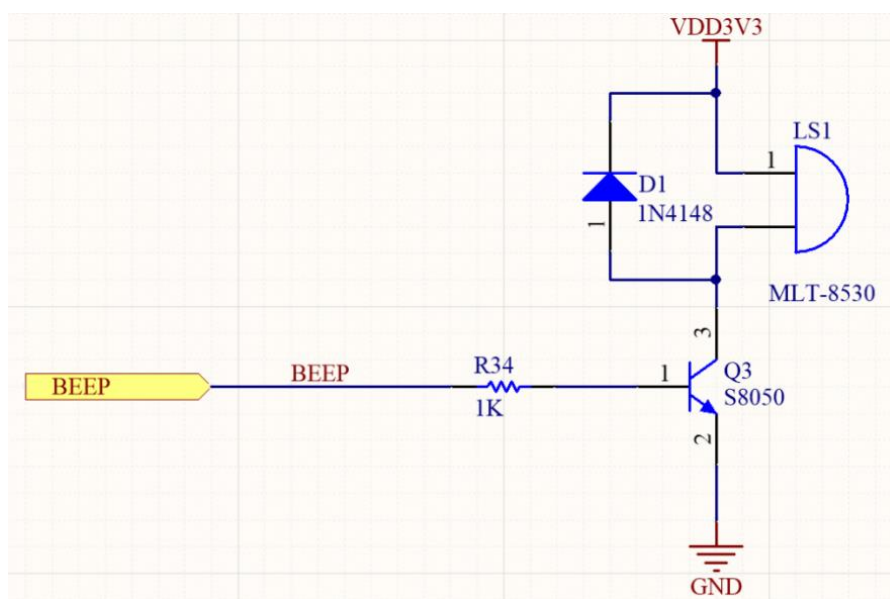
本实验中的梁祝乐曲，最小的节拍为 $1/4$ 拍，若将一拍的时长定为 $1s$ ，则只需要提供一个 $4Hz$ 的时钟频率即可产生 $1/4$ 拍的时长($0.25s$)，对于其它占用时间较长的节拍，如 $2/4$ 拍(必定是 $1/4$ 拍的整数倍)，则只需要将该音符连续输出两遍即可。

2.4 音符及音阶的显示

为提高电路的实用性，可以通过数码管来显示出乐曲演奏时的音符及其音调的高低。为此，本电路中采用两个数码管，一个数码管用来动态显示乐曲演奏时的音符，另一个数码管则显示乐曲演奏时音符所对应的音调，0 表示中音，1 表示高音。

2.5 蜂鸣器驱动电路

无源蜂鸣器没有集成振荡器，需要外部提供震荡激励，当震荡频率不同时发出不同的音调，对于数字系统来说，方波信号产生方便可靠，成为外部震荡激励的首选，方波信号输入谐振装置转换为声音信号输出，电磁式蜂鸣器需要的驱动电流较高，一般单片机和 FPGA 管脚驱动能力有限不能直接驱动，常用三极管增加驱动能力，另外电磁式蜂鸣器内部含有感应线圈，在电路通断瞬间会产生感应电势，为保证电路长期稳定的工作，最好增加续流二极管设计。



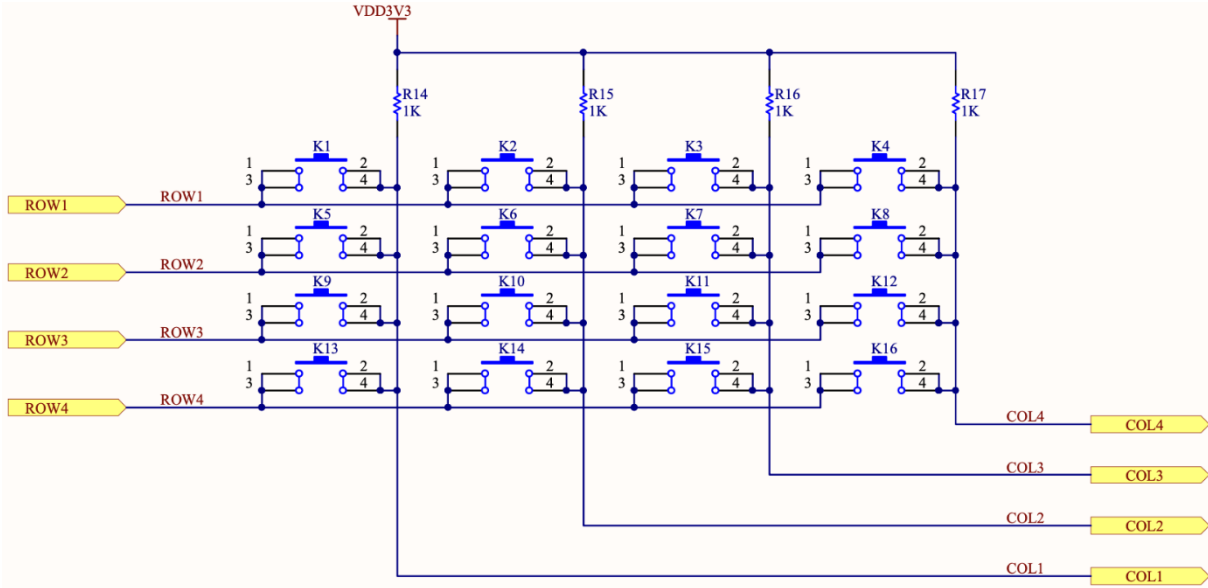
蜂鸣器驱动电路

2.6 4×4 矩阵按键

4 根行线（ROW1、ROW2、ROW3、ROW4）和 4 根列线（COL1、COL2、COL3、COL4），同时列线通过上拉电阻连接到 VCC 电压（3.3V），对于矩阵按键来讲：4 根行线是输入的，是由小脚丫 STEP MAX10 FPGA 开发板控制拉高或拉低，4 根列线数输出的，是由 4 根行线的输入及按键的状态决定，输出给小脚丫 STEP MAX10 FPGA 开发板，当某时刻，FPGA 控制 4 根行线分别为 ROW1=0、ROW2=1、ROW3=1、ROW4=1 时，对于 K1、K2、K3、K4 按键：按下时对应 4 根列线输出 COL1=0、COL2=0、COL3=0、COL4=0，不按时对应 4 根列线输出 COL1=1、COL2=1、COL3=1、

COL4=1, 对于 K5-K16 之间的按键: 无论按下与否, 对应 4 根列线输出 COL1=1、COL2=1、COL3=1、COL4=1。

通过上面的描述: 在这一时刻只有 K1、K2、K3、K4 按键被按下, 才会导致 4 根列线输出 COL1=0、COL2=0、COL3=0、COL4=0, 否则 COL1=1、COL2=1、COL3=1、COL4=1, 反之当 FPGA 检测到列线 (COL1、COL2、COL3、COL4) 中有低电平信号时, 对应的 K1、K2、K3、K4 按键应该是被按下了。按照扫描的方式, 一共分为 4 个时刻, 分别对应 4 根行线中的一根拉低, 4 个时刻依次循环, 在程序中以这 4 个时刻对应状态机的 4 个状态。



4×4 矩阵按键的硬件电路图

2.7 PWM（脉冲宽度调制）

PWM（Pulse Width Modulation）全称脉冲宽度调制，它通过对一系列脉冲的宽度进行调制，来等效地获得所需要波形（含形状和幅值）。面积等效原理是 PWM 控制技术的重要理论基础。

原理内容：冲量相等而形状不同的窄脉冲加在具有惯性的环节上时，其效果基本相同。冲量即指窄脉冲的面积。效果基本相同，是指环节的输出响应波形基本相同。如果把各输出波形用傅里叶变换分析，则其低频段非常接近，仅在高频段略有差异。

PWM 信号参数：周期：信号变化的过程中，某段波形重复出现，其某一次开始至结束的这段时间就称为“周期”。脉宽：在一个周期内正脉冲的持续时间。占空比：是在一串理想的脉冲系列中，脉宽与周期的比值。例如在上图中 t 为正脉冲的持续时间， T 为脉冲周期，占空比为 $\frac{t}{T}$ 。

三、实验器材

1. 小脚丫 STEP MAX10 FPGA 开发板
2. Step-Baseboard V3.0 底板
3. micro USB 数据线
4. Quartus II 软件

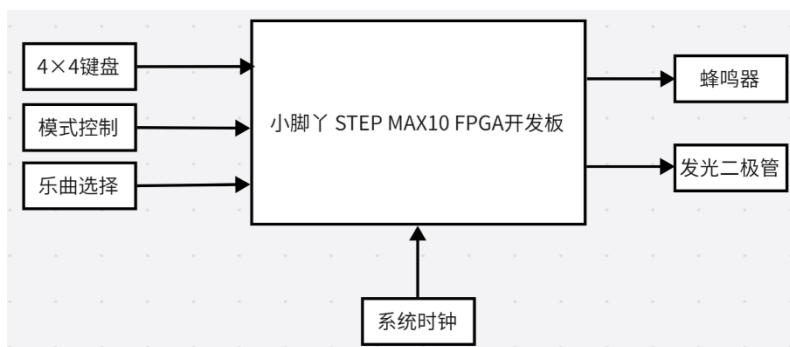
四、实验要求及实验内容

4.1 系统设计思想

本设计采用 Verilog 语言使用模块化的设计方法设计，可实现如下功能：

1. 音乐盒模式下，预存储 4 首歌，由乐曲选择键（拨码开关）选择播放。
2. 电子琴模式下，通过 4×4 键盘演奏，含高低 16 个音符。
3. 配有数码管高音指示及乐谱显示。

系统框图如下：



系统框图

4.2 顶层模块设计

4.2.1 锁相环 PLL20

通过调用 Quartus II 内 IP 核获得，并设置输出频率 $C_0=2\text{KHz}$ ， $C_1=1\text{MHz}$ 。

4.2.2 分频器 FDIIV

将来自锁相环的 2KHz 分频为 4Hz，提供给 CNT138T 与 RAM 用于控制每一个音符的停留时间，可以根据不同乐谱进行更改。

4.2.3 计数器 CNT138T

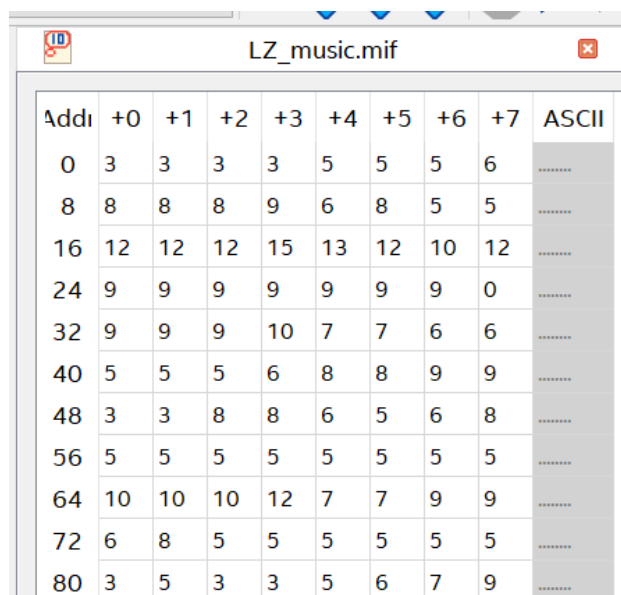
这个计数器的计数频率为 4Hz，即每一个计数值停留时间为 0.25s，在 4/4 拍的乐谱中，若假设全音符为 1s 则一个四分音符为 0.25s。可以根据乐曲长度修改计数器的最大计数值。

4.2.4 数控分频模块 SPKER

数控分频模块 SPKER 的功能就是当在输入端给定不同输入数据时，将对输入的时钟信号有不同的分频比，数控分频器就是用计数值可并行预置的加法计数器设计完成的，得出其对应的输出频率。

4.2.5 存储器 ROM

XX_music.mif 中的音符数据可以根据演奏乐曲更改，下图所示 ROM 中内容是乐曲《梁祝》中的一部分。



Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	3	3	3	3	5	5	5	6	*****
8	8	8	8	9	6	8	5	5	*****
16	12	12	12	15	13	12	10	12	*****
24	9	9	9	9	9	9	9	0	*****
32	9	9	9	10	7	7	6	6	*****
40	5	5	5	6	8	8	9	9	*****
48	3	3	8	8	6	5	6	8	*****
56	5	5	5	5	5	5	5	5	*****
64	10	10	10	12	7	7	9	9	*****
72	6	8	5	5	5	5	5	5	*****
80	3	5	3	3	5	6	7	9	*****

梁祝乐曲的.mif 文件

4.2.6 四选一数据选择器

输入端： 4 个数据输入，选择端： 2 位二进制选择，输出端： 1 个输出。二选一数据选择器与它类似，故不做多做赘述。

4.2.7 矩阵键盘驱动模块

由 2.64*4 矩阵按键的工作原理知可将矩阵键盘的扫描周期分为 4 个时刻，对应 4 个状态，使得状态机在 4 个状态上循环跳转，最终通过扫描的方式获取矩阵键盘的操作状态。

4.2.8 PWM 模块

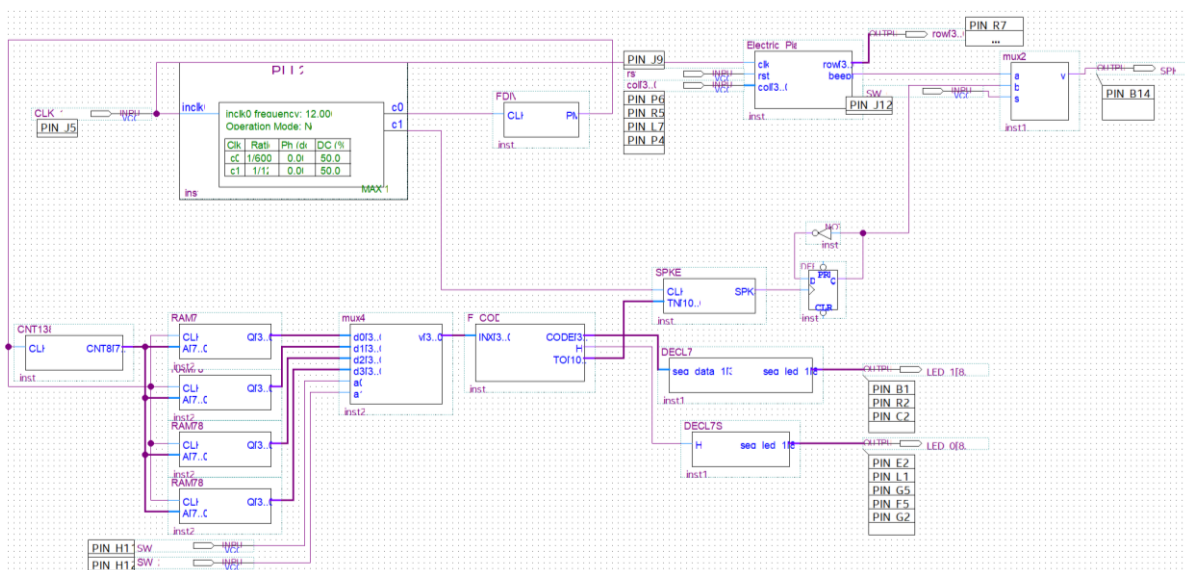
要驱动蜂鸣器就需要给蜂鸣器模块输出不同频率的脉冲信号就可以了，模块根据两个输入信号（cycle、duty）控制产生周期可控、占空比可控的脉冲信号（pwm_out），可以用来驱动无源蜂鸣器电路。默认产生 50% 占空比的脉冲信号，所以 duty 的输入取 cycle 的一半。

cycle 的值关系到蜂鸣器的音符，例如如果要蜂鸣器发出低音 1（do）的音符，脉冲信号频率控制为 261.6Hz，系统时钟采用 12MHz，计数器计数终值 cycle 就等于 $12M/261.6=45872$ ，即当 PWM 模块中 cycle 信号的值为 45872 时，得到低音 1 的音符输出。

这样将每个音符对应的 cycle 值计算出来，当按动不同按键时给 PWM 模块不同的 cycle 值就可以了，可以通过设计一个转码模块（tone）将按键信息转换成 PWM 需要的 cycle 信号，矩阵键盘共有 16 个按键，所以只能输出 16 个音节。

4.3 模块组合与管脚分配

对以上模块进行编译设计并生成 *.bsf 文件，将各个模块组合起来并进行管脚分配。



模块原理图

信号	引脚	信号	引脚
clk_12	PIN_J5	LED_0[7]	PIN_L1
col[3]	PIN_P6	LED_0[6]	PIN_G5
col[2]	PIN_R5	LED_0[5]	PIN_F5
col[1]	PIN_L7	LED_0[4]	PIN_G2
col[0]	PIN_P4	LED_0[3]	PIN_J2
row[3]	PIN_R7	LED_0[2]	PIN_K2
row[2]	PIN_P7	LED_0[1]	PIN_D2
row[1]	PIN_P8	LED_1[8]	PIN_B1
row[0]	PIN_P9	LED_1[7]	PIN_R2
rst	PIN_J9	LED_1[6]	PIN_C2
SPK	PIN_B14	LED_1[5]	PIN_C1
SW_0	PIN_J12	LED_1[4]	PIN_N1
SW_1	PIN_H11	LED_1[3]	PIN_P1
SW_2	PIN_H12	LED_1[2]	PIN_P2
LED_0[8]	PIN_E2	LED_1[1]	PIN_A2

管脚分配

如果仿真无误，构建并输出编程文件，烧写至 小脚丫 FPGA 的 Flash 之中。并观察输出结果。

五、实验数据处理与结果分析

5.1 数据处理

由 2.2 音符频率的获得 原理中的公式与表格可以计算得到预置数,用于修正 F CODE 中的音符

预置数。

音名	频率	系数	HEX	音名	频率	系数	HEX	音名	频率	系数	HEX
1	262	140	8C	高音 1	523	1092	444	高 2 度 1	1047	1571	623
2	296	359	167	高音 2	587	1197	4AD	高 2 度 2	1175	1623	657
3	330	533	215	高音 3	659	1290	50A	高 2 度 3	1319	1669	685
4	349	616	268	高音 4	698	1332	534	高 2 度 4	1397	1691	69B
5	392	773	305	高音 5	784	1411	583	高 2 度 5	1568	1730	6C2
6	440	912	390	高音 6	880	1480	5C8	高 2 度 6	1760	1764	6E4
7	494	1036	40C	高音 7	988	1542	606	高 2 度 7	1976	1795	703

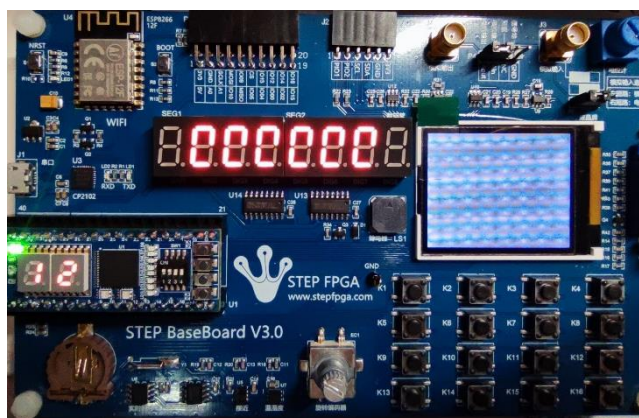
音符预置数

使用附录中.mif-maker 程序将简谱音符生成 ROM 所需要的.mif 文件。得到《Outer Wilds》与《蜜雪冰城》（哦、苏珊娜）《爱你》《one last kiss》《我爱北京天安门》等乐曲的.mif 文件：

Outer_Wilds.mif										MXBC										one_last_kiss.mif										AN_music.mif									
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII	Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII	Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII	Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1	1	1	1	5	5	5	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	7	7	7	7	0	0	5	5	8	0	0	0	0	0	0	8	8	8	0	0	0	0	0	0	8	8	8	0	0	0	0	0	0	8	8
16	8	8	7	7	6	6	5	5	16	8	8	8	8	8	0	8	6	16	8	8	8	8	8	0	8	6	16	8	8	8	8	8	0	8	6
24	6	6	7	7	5	5	5	5	24	6	6	8	8	0	0	0	8	24	6	6	8	8	0	0	0	8	24	6	6	8	8	0	0	0	8
32	2	2	2	2	5	5	5	5	32	5	5	5	5	9	9	7	1	32	5	5	5	5	9	9	7	1	32	5	5	5	5	9	9	7	1
40	7	7	7	7	0	0	5	5	40	8	8	5	5	0	0	0	0	40	8	8	5	5	0	0	0	0	40	8	8	5	5	0	0	0	0
48	8	8	7	7	6	6	5	5	48	5	8	8	8	0	8	10	10	48	5	8	8	8	0	8	10	10	48	5	8	8	8	0	8	10	10
56	6	6	7	7	9	9	7	7	56	8	8	8	8	6	6	0	0	56	8	8	8	8	6	6	0	0	56	8	8	8	8	6	6	0	0
64	3	3	3	3	5	5	5	5	64	0	0	6	6	5	5	5	5	64	0	0	6	6	5	5	5	5	64	0	0	6	6	5	5	5	5
72	7	7	7	7	0	0	5	5	72	9	9	7	8	8	8	8	8	72	9	9	7	8	8	8	8	8	72	9	9	7	8	8	8	8	8
80	8	8	7	7	6	6	5	5	80	3	3	4	4	5	5	5	5	80	3	3	4	4	5	5	5	5	80	3	3	4	4	5	5	5	5
88	6	6	7	7	5	5	5	5	88	8	8	8	8	7	0	7	7	88	8	8	8	8	7	0	7	7	88	8	8	8	8	7	0	7	7
96	2	2	2	2	5	5	5	5	96	6	0	6	6	0	0	0	0	96	6	0	6	6	0	0	0	0	96	6	0	6	6	0	0	0	0
104	7	7	7	7	0	0	5	5	104	0	0	0	0	0	0	0	0	104	0	0	0	0	0	0	0	0	104	0	0	0	0	0	0	0	0
112	8	8	7	7	6	6	5	5	112	0	0	0	0	0	0	0	0	112	0	0	0	0	0	0	0	0	112	0	0	0	0	0	0	0	0
120	6	6	7	7	9	9	7	7	120	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0

最后经过尝试选择了《梁祝》《Outer Wilds》、《蜜雪冰城》与《爱你》这四首曲目。

5.2 结果分析



小脚丫 STEP-Baseboard V3.0 实验平台

蜂鸣器成功发出声音，能够较为准确的演奏出指定曲目，且能通过第一个拨码开关切换功能（电子琴/八音盒），以及使用第二、三个拨码开关切换曲目（00/01/10/11）。

六、实验结论

本设计采用层次化和模块化的思想，使用 Verilog 语言设计了乐曲演奏电路，并在 Quartus II 开发平台上进行仿真，仿真结果达到预期的效果，将其下载到硬件电路中，可以看到它进行指定乐曲的演奏，并可以进行电子琴和音乐盒的模式更换与音乐更换。通过 Verilog 层次化和模块化设计方法，同时采用音符编码的设计思想，更好的优化了乐曲演奏数字电路的设计，在此基础上不必变化顶层文件架构可随意变更或增加任何乐曲，有效地提高了设计的灵活性和可靠性。

同时使用了 .mif-maker 程序快捷的生成乐曲 .mif 文件，提高了开发效率与准确性，同时也能优化曲目中的连音部分，能较好的区分出两个连续的不同音符。

七、实验中的问题及解决办法、改进方向

问题：

1. 在模式切换到电子琴模块时，八音盒仍在工作，只是音频信号未能得到输出。
2. 在工程管理中，有较多类似源代码，未能得到很好的归档，降低了工程的可读性。
3. 手动输入 .mif 较为繁琐并且易错。
4. 在对整个工程进行仿真时，由于经过多个分频器且分频系数较大，出现仿真时间过长而导致失败。

解决办法：

1. 可以在 FDIV 模块中加入使能信号，由拨码开关的 SW_0 的信号进行控制。
2. 建立多层文件夹，入 core 用于存放核心代码，piano 用于存放电子琴代码，加强可读性与可移植性。
3. 使用基于开源项目修改的 .mif-maker，以更方便、直观的方式生成乐曲 .mif 文件。
4. 独立仿真各个模块，当各个模块的仿真结果均满足预期目标时，只要各个模块功能正常，在正确连接的情况下，整个工程的功能也应该完全正常。

改进方向：

1. 可以尝试使用旋转编码器控制 PWM 的占空比，以达到音量大小的调节。
2. 可以尝试使用 esp8266 模块或者 uart 让小脚丫 FPGA 开发板直接与上位机通信，实现通过上位机更改曲目的 .mif 文件。
3. 可增加实现预置乐曲的暂停和继续播放实时控制功能。

八、参考文献

[1]潘松, 黄继业, 陈龙. EDA 技术与 Verilog HDL[M]. 清华大学出版社, 2017.

九、附录：

FDIV

```
module FDIV(CLK,PM);
input CLK;
output PM;
reg [8:0] Q1;
reg FULL;
wire RST;
always @(posedge CLK or posedge RST)
begin
    if(RST) begin Q1<=0;FULL<=1;end
    else begin Q1<=Q1+1; FULL<=0;end
end
assign RST=(Q1==499); //2k/500=4hz
assign PM=FULL;
assign DOUT=Q1;
endmodule
```

CNT138T

```
module CNT138T (CLK,CNT8);
input CLK;
output [7:0] CNT8;
reg[7:0]CNT;
wire LD;
always @(posedge CLK or posedge LD)
begin
    if(LD) CNT<=8'b00000000;
    else CNT<=CNT+1;
end
assign CNT8=CNT;
assign LD=(CNT==138);
endmodule
```

RAM78

```
module RAM78 (CLK,A,Q);
input CLK;
input [7:0] A;
output [3:0] Q;
reg [3:0] Q;
reg [3:0] mem[255:0]/* synthesis ram_init_file="LZ_music.mif" */;
```

```

        always @(posedge CLK)//可修改为其他.mif 文件
            Q<=mem[A];
endmodule

mux4_1

module mux4_1 (d0,d1,d2,d3,a0,a1,y);    //定义模块和端口变量
input [3:0]d0;
input [3:0]d1;
input [3:0]d2;
input [3:0]d3;
input a0,a1; //定义输入端
output [3:0]y; //定义输出端
reg [3:0]y; //定义寄存器存储结果
always@( a0,a1)//载入发生信号变化的端口
begin
    case({a1,a0})
        2'b00 : y<=d0;
        2'b01 : y<=d1;
        2'b10 : y<=d2;
        2'b11 : y<=d3;
        default : y<=d0; //可以不写，默认位
    endcase
end
endmodule

```

F_CODE

```

module F_CODE (INX,CODE,H,TO);
input [3:0]INX;//输入音符
output [3:0]CODE;//输出当前音调
output H;//高音标志
output [10:0]TO;//输出音频的频率
reg [10:0] TO;//音频预制数的寄存器
reg [3:0]CODE;
reg H;
always @(INX)
begin
    case (INX)
        0:begin TO<=11'H7ff;CODE<=0;H<=0;end
        1:begin TO<=11'H8C;CODE<=1;H<=0;end
        2:begin TO<=11'H167;CODE<=2;H<=0;end
        3:begin TO<=11'H215;CODE<=3;H<=0;end
        4:begin TO<=11'H268;CODE<=4;H<=0;end
        5:begin TO<=11'H305;CODE<=5;H<=0;end
    endcase
end

```

```

        6:begin TO<=11'H390;CODE<=6;H<=0;end
        7:begin TO<=11'H40C;CODE<=7;H<=0;end
        8:begin TO<=11'H444;CODE<=1;H<=1;end
        9:begin TO<=11'H4AD;CODE<=2;H<=1;end
        10:begin TO<=11'H50A;CODE<=3;H<=1;end
        11:begin TO<=11'H534;CODE<=4;H<=1;end
        12:begin TO<=11'H583;CODE<=5;H<=1;end
        13:begin TO<=11'H5C8;CODE<=6;H<=1;end
        14:begin TO<=11'H606;CODE<=7;H<=1;end
        15:begin TO<=11'H632;CODE<=1;H<=1;end
        default:begin TO<=11'H6c0;CODE<=1;H<=1; end
    endcase
end
endmodule

```

SPKER

```

module SPKER(CLK,TN,SPKS);
input CLK;
input [10:0] TN;
output SPKS;
reg SPKS;
reg [10:0] CNT11;
always @(posedge CLK)
begin: CNT11B_LOAD
    if(CNT11==11'h7FF) begin CNT11=TN; SPKS<=1'b1; end
    else begin CNT11=CNT11+1; SPKS<=1'b0; end
end
endmodule

```

Electric_Piano

```

module Electric_Piano
(
input                clk,                //system clock
input                rst_n,              //system reset
input                [3:0] col,
output               [3:0] row,
output               beeper
);
wire                [15:0] key_out;
wire                [15:0] key_pulse;
//Array_KeyBoard

```

```

Array_KeyBoard u1
(
.clk                (clk                ),
.rst_n              (rst_n              ),
.col                (col                ),
.row                (row                ),
.key_out            (key_out            ),
.key_pulse          (key_pulse          )
);
//beeper module
Beeper u2
(
.clk                (clk                ),
.rst_n              (rst_n              ),
.key_out            (~key_out           ),
.beeper             (beeper             )
);
endmodule

```

Array_KeyBoard

```

module Array_KeyBoard #
(parameter CNT_200HZ = 60000)
(
input                clk,
input                rst_n,
input                [3:0] col,
output reg [3:0] row,
output reg [15:0] key_out,
output [15:0] key_pulse
);
localparam          STATE0 = 2'b00;
localparam          STATE1 = 2'b01;
localparam          STATE2 = 2'b10;
localparam          STATE3 = 2'b11;
//计数器计数分频实现 5ms 周期信号 clk_200hz
reg [15:0] cnt;
reg clk_200hz;
always@(posedge clk or negedge rst_n) begin //复位时计数器 cnt 清
零, clk_200hz 信号起始电平为低电平
    if(!rst_n) begin
        cnt <= 16'd0;
    end
end

```



```

        clk_200hz <= 1'b0;
    end else begin
        if(cnt >= ((CNT_200HZ>>1) - 1)) begin
//数字逻辑中右移 1 位相当于除 2
            cnt <= 16'd0;
            clk_200hz <= ~clk_200hz; //clk_200hz 信号取反
        end else begin
            cnt <= cnt + 1'b1;
            clk_200hz <= clk_200hz;
        end
    end
end

reg [1:0] c_state;
//状态机根据 clk_200hz 信号在 4 个状态间循环，每个状态对矩阵按键的行接口
单行有效
always@(posedge clk_200hz or negedge rst_n) begin
    if(!rst_n) begin
        c_state <= STATE0;
        row <= 4'b1110;
    end else begin
        case(c_state)
            //状态 c_state 跳转及对应状态下矩阵按键的 row 输出
            STATE0: begin c_state <= STATE1; row <= 4'b1101; end
            STATE1: begin c_state <= STATE2; row <= 4'b1011; end
            STATE2: begin c_state <= STATE3; row <= 4'b0111; end
            STATE3: begin c_state <= STATE0; row <= 4'b1110; end
            default:begin c_state <= STATE0; row <= 4'b1110; end
        endcase
    end
end

reg [15:0] key,key_r;//因为每个状态中单行有效，通过对列接口的电平状态
采样得到对应 4 个按键的状态，依次循环
always@(negedge clk_200hz or negedge rst_n) begin
    if(!rst_n) begin
        key_out <= 16'hffff; key_r <= 16'hffff; key <= 16'hffff;
    end else begin
        case(c_state)//采集当前状态的列数据赋值给对应的寄存器位
            //对键盘采样数据进行判定，连续两次采样低电平判定为按键按下
            STATE0: begin key_out[ 3: 0] <= key_r[ 3: 0]|key[ 3:
0]; key_r[ 3: 0] <= key[ 3: 0]; key[ 3: 0] <= col; end
            STATE1: begin key_out[ 7: 4] <= key_r[ 7: 4]|key[ 7:
4]; key_r[ 7: 4] <= key[ 7: 4]; key[ 7: 4] <= col; end

```

```

        STATE2: begin key_out[11: 8] <= key_r[11: 8]|key[11:
8]; key_r[11: 8] <= key[11: 8]; key[11: 8] <= col; end
        STATE3: begin key_out[15:12] <=
key_r[15:12]|key[15:12]; key_r[15:12] <= key[15:12]; key[15:12] <=
col; end

        default:begin key_out <= 16'hffff; key_r <= 16'hffff;
key <= 16'hffff; end
    endcase
end
end
reg    [15:0]    key_out_r;
always @ ( posedge clk or negedge rst_n )
    if (!rst_n) key_out_r <= 16'hffff;
    else key_out_r <= key_out;    //将前一刻的值延迟锁存
    assign key_pulse= key_out_r & ( ~key_out);    //通过前后两个时刻的
值判断
endmodule

```

Beeper

```

module Beeper
(input  clk,
input  rst_n,
input  [15:0]  key_out,
output beeper
);
wire [15:0] cycle;
//将按键信息译成音节对应的周期 cycle 值
tone u1
(.key_in (key_out  ),.cycle (cycle  ));
//根据不同音节的周期 cycle 值产生对应的 PWM 信号
PWM #
(.WIDTH (16 )    //ensure that 2**WIDTH > cycle
)
u2
(.clk      (clk      ),
.rst_n     (rst_n     ),
.cycle     (cycle     ), //cycle > duty
.duty      (cycle>>1  ), //duty < cycle
.pwm_out   (beeper ));
endmodule

```

PWM

```
module PWM #
(
parameter    WIDTH = 32  //ensure that 2**WIDTH > cycle
)
(
input        clk,
input        rst_n,
input        [WIDTH-1:0] cycle,  //cycle > duty
input        [WIDTH-1:0] duty,   //duty < cycle
output reg   pwm_out
);

reg [WIDTH-1:0] cnt;
//counter for cycle
always @(posedge clk or negedge rst_n)
    if(!rst_n) cnt <= 1'b1;
    else if(cnt >= cycle) cnt <= 1'b1;
    else cnt <= cnt + 1'b1;

//pulse with duty
always @(posedge clk or negedge rst_n)
    if(!rst_n) pwm_out <= 1'b1;
    else if(cnt < duty) pwm_out <= 1'b1;
    else pwm_out <= 1'b0;

endmodule
```

tone

```
module tone
(
input        [15:0] key_in,
output reg   [15:0] cycle
);//不同按键按下对应得到不同的 PWM 周期
//已知蜂鸣器某音节对应的蜂鸣器震荡频率为 261.6Hz，系统时钟频率为 12MHz
//故每个蜂鸣器震荡周期时间等于 12M/261.6=45872 个系统时钟的周期时间之和
//我们把按键信息译成 PWM 模块周期 cycle 对应的值，输出给 PWM 模块
always@(key_in) begin
    case(key_in)
        16'h0001: cycle = 16'd45872;    //L1,
```

```

        16'h0002: cycle = 16'd40858;    //L2,
        16'h0004: cycle = 16'd36408;    //L3,
        16'h0008: cycle = 16'd34364;    //L4,
        16'h0010: cycle = 16'd30612;    //L5,
        16'h0020: cycle = 16'd27273;    //L6,
        16'h0040: cycle = 16'd24296;    //L7,
        16'h0080: cycle = 16'd22931;    //M1,
        16'h0100: cycle = 16'd20432;    //M2,
        16'h0200: cycle = 16'd18201;    //M3,
        16'h0400: cycle = 16'd17180;    //M4,
        16'h0800: cycle = 16'd15306;    //M5,
        16'h1000: cycle = 16'd13636;    //M6,
        16'h2000: cycle = 16'd12148;    //M7,
        16'h4000: cycle = 16'd11478;    //H1,
        16'h8000: cycle = 16'd10215;    //H2,
        default: cycle = 16'd0;          //cycle 为 0, PWM 占空比为 0, 低电
平
    endcase
end
endmodule

```

DECL7S

```

module DECL7S(seg_data_1,seg_led_1);
input  [3:0] seg_data_1;//数码管需显示 0~9, 故需要 4 位输入做译码
output [8:0] seg_led_1;
reg  [8:0] seg [15:0];
initial
begin
    seg[0]=9'h3f;
    seg[1]=9'h06;
    seg[2]=9'h5b;
    seg[3]=9'h4f;
    seg[4]=9'h66;
    seg[5]=9'h6d;
    seg[6]=9'h7d;
    seg[7]=9'h07;
    seg[8]=9'h7f;
    seg[9]=9'h6f;
    seg[10]=9'h77;
    seg[11]=9'h7c;
    seg[12]=9'h39;

```

```

        seg[13]=9'h5e;
        seg[14]=9'h79;
        seg[15]=9'h71;
    end
    assign seg_led_1 = seg[seg_data_1];
endmodule

```

.mif-maker

Main.java:

```

package com.frozen.sakura;
public class Main {
    public static void main(String[] args) {
        FileHelper fileHelper = new FileHelper();
        fileHelper.createFile();
    }
}

```

FileHelper.java:

```

package com.frozen.sakura;
import java.io.*;
import java.util.ArrayList;
public class FileHelper {
    private static class SheetMusic {
        public String note;
        public int count;
        public boolean tremolo = false;

        public SheetMusic (String note, int count) {
            super();
            this.note = note;
            this.count = count;
        }
        public SheetMusic (String note, int count, boolean tremolo) {
            super();
            this.note = note;
            this.count = count;
            this.tremolo = tremolo;
        }
    }
    // 计数器
    private int woc = 0;
    // 调性转换
    private final int offset = 0;
    // 以几分音符为一个音符单位

```

```

private final int unit = 16;
// 储存乐谱结果
private ArrayList<SheetMusic> sheetMusics = new ArrayList<>();
public void createFile() {
    String pathname = "src/com/frozen/sakura/input";
    StringBuilder data = new StringBuilder();
    try (FileReader reader = new FileReader(pathname);
        BufferedReader br = new BufferedReader(reader)
    ) {
        String line;
        while ((line = br.readLine()) != null) {
            if (line.length() == 0)
                continue;
            transformData(line.replace(" ", ""));
        }
        solveOverlapNote();
        int width = 0;
        for (SheetMusic item :
            sheetMusics) {
            for (int i = 0; i < item.count; i++, woc++) {
                if (!item.tremolo)
                    data.append("    ")
                        .append(woc)
                        .append(": ")
                        .append(item.note)
                        .append(";\\n");
                else {
                    data.append("    ")
                        .append(woc)
                        .append(": ");
                    // 模仿摇指
                    if ((i & 1) == 0)
                        data.append(item.note);
                    else
                        data.append("0");
                    data.append(";\\n");
                }
                if (Integer.parseInt(item.note) > width)
                    width = Integer.parseInt(item.note);
            }
        }
        String stringBuilder = "WIDTH = " +
            // 计算宽度

```

```

        (int) (Math.log(getNextPow2(width)) /
Math.log(2)) +
        ";\\nDEPTH = " +
        // 计算深度
        getNextPow2(woc) +
        ";\\n\\nADDRESS_RADIX = DEC;\\nDATA_RADIX =
DEC;\\n\\nCONTENT BEGIN\\n" +
        data.toString() +
        "END;";
        writeFile(stringBuilder);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
// 写入文件
private void writeFile(String content) {
    try {
        File writeName = new
File("src/com/frozen/sakura/output");
        try (FileWriter writer = new FileWriter(writeName);
            BufferedWriter out = new BufferedWriter(writer)
        ) {
            out.write(content);
            out.flush();
            System.out.println("Successful.");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
private void transformData(String data) {
    SheetMusic sheetMusic;
    String temp = data.replace("/", "");
    // 音符格式错误
    if (temp.contains(".") && temp.contains("-")) {
        sheetMusic = new SheetMusic("error", 1);
    } else {
        int count;
        // 计算单位个数
        if (temp.contains(".."))
            // 十六分音符
            count = unit / 16;
        else if (temp.contains("."))

```

```

2)        if (temp.length() - temp.replace(".", "").length() ==

            // 八分音符附点
            count = 3 * unit / 16;
        else if (temp.charAt(0) == '.')
            // 八分音符
            count = unit / 8;
        else
            // 四分音符附点
            count = 3 * unit / 8;
        else
            count = unit / 4 * (temp.length() - temp.replace("-",
"".length() + 1);
            // 将音符添加至乐谱中
            sheetMusic = new SheetMusic(number(temp.replace(".",
"".replace("-", "")), count, data.contains("/"));
        }
        sheetMusics.add(sheetMusic);
    }
    // 生成对应音高
    private String number(String index) {
        switch (index.length()) {
            case 1:
                if (index.equals("0"))
                    return index;
                // 中音
                return String.valueOf(Integer.parseInt(index) + 7 +
offset);
            case 2:
                int note =
Integer.parseInt(String.valueOf(index.charAt(1)));
                switch (index.charAt(0)) {
                    case '0':
                        // 低音
                        return String.valueOf(note + offset);
                    case '1':
                        // 高音
                        return String.valueOf(note + 14 + offset);
                    case '2':
                        return String.valueOf(note + 21 + offset);
                }
            default:
                return index;
        }
    }
}

```



```

    }
    // 处理相近音符
    private void solveOverlapNote() {
        for (int i = 1; i < sheetMusics.size(); i++) {
            SheetMusic temp = sheetMusics.get(i - 1);
            if (sheetMusics.get(i).note.equals(temp.note) &&
temp.count > 1) {
                temp.count -= 1;
                sheetMusics.add(i, new SheetMusic("0", 1));
                i++;
            }
        }
    }
    // 计算最接近的二次幂
    private long getNextPow2(int v) {
        long x = v - 1;
        x |= x >>> 1;
        x |= x >>> 2;
        x |= x >>> 4;
        x |= x >>> 8;
        x |= x >>> 16;
        return x + 1;
    }
}

```

《Outer Wilds》

星际拓荒主题曲

1=C $\frac{4}{4}$ ♩=90

(1)

||: 1 5 7 0 5 | 1̇ 7 6 5 6 7 5 | 2 5 7 0 5 | 1̇ 7 6 5 6 7 2̇ 7 |

(5)

| 3 5 7 0 5 | 1̇ 7 6 5 6 7 5 | 2 5 7 0 5 | 1̇ 7 6 5 6 7 2̇ 7 :||

《蜜雪冰城》

蜜雪冰城

苍强曲谱 制谱

1 =D $\frac{4}{4}$

哦，苏珊娜

♩=104

3 5 5. 6 5 3 1 1 2 | 3 3 2 1 2 0 |
你 爱 我， 我 爱 你， 蜜 雪 冰 城 甜 蜜 蜜

3 5 5. 6 5 3 1 1 2 | 3 3 2 2 1 0 | 4 4 4 6. |
你 爱 我， 我 爱 你， 蜜 雪 冰 城 甜 蜜 蜜 你 爱 我 呀，

5 5 3 2 0 | 3 5 5. 6 5 3 1 1 2 | 3 3 2 2 1 0 ||
我 爱 你， 你 爱 我， 我 爱 你， 蜜 雪 冰 城 甜 蜜 蜜

《爱你》（部分）

爱你

1 = $\flat E$ $\frac{4}{4}$

演唱：王心凌

$\text{♩} = 80$

演奏顺序：A B C1 A B C2 B D

A: 0 0 0 0 1 1 | 1 1 1 1 1 1 6 6 1 0. 1 |

如 果 你 突 然 打 了 个 喷 嚏 那
喜 欢 在 你 的 臂 弯 里 胡 闹 你

5 5 5 5 2 7 1 1 5 0 5 1 | 1 1 1 3 1 1 1 6 0 6 |

一 定 就 是 我 在 想 你 如 果 半 夜 被 手 机 吵 醒 啊
的 世 界 是 一 座 城 堡 在 大 头 贴 画 满 心 号 贴

5 5 5 5 2 7 1 1 3 4 | 5 1 7 7 6 6 6 |

那 是 因 为 我 关 心 常 常 想 你 说 的 话 是 不
在 手 机 上 对 你 微 笑 常 常 想 我 说 的 话 你 是

6 5 2 4 4 4 3 0 1 | 1 1 6 7 6 6 5 0 3 |

是 别 有 用 心 明 明 很 想 相 信 却
否 听 得 进 去 明 明 很 想 生 气 却

4 5 6 7 6 6 5 0 | 5 1 1 1 6 6 0 6 |

又 忍 不 住 怀 疑 在 你 的 心 里 我
又 止 不 住 笑 意 在 我 的 心 里 你

《one last kiss》（部分）

One Last Kiss

《新福音战士剧场版：终》op

歌手：宇多田光

作词：宇多田光

作曲：宇多田光

制谱：@日更简谱100年

1=B $\text{♩} = 112$

$\frac{4}{4}$ 6 6 6 6 6 3 2 3 2 0 | 6 6 6 6 3 3 2 3 2 1 | 1 6 6 6 6 3 3 2 3 2 0 1 | 6 0 6 6 6 3 2 3 2 0 |

5 6 6 6 6 3 3 2 3 2 0 1 1 | 5 6 6 6 6 3 3 2 3 2 1 | 1 6 6 6 6 3 3 2 3 2 0 | 3 2 2 1 1 0 |

6 - 0 5 6 | 3 6 5 6 6 6 | 6 - - 6 3 | 2 1 1 3 2 1 2 |

2 6 0 0 5 6 | 3 6 5 6 6 6 | 6 - 0 6 3 | 2 1 1 3 2 1 2 |

6 6 6 6 6 6 1 2 | 6 6 6 6 6 3 2 1 | 6 6 6 6 6 6 6 3 | 2 1 1 3 2 1 2 |

6 6 6 6 6 6 1 2 | 6 6 6 6 6 6 6 | 6 6 6 6 6 3 6 3 | 2 1 1 3 2 1 2 |

《我爱北京天安门》（部分）

我爱北京天安门

简谱

1= C $\frac{2}{4}$

热情 活泼

$\underline{5 \cdot \dot{1}} \underline{5 \ 4} \mid \underline{3 \ 2} \ 1 \mid \underline{1 \ 1} \underline{2 \ 3} \mid \widehat{3 \ 1} \widehat{3 \ 4} \mid \underline{5 -} \mid \underline{5 -} \mid \underline{5 \cdot \dot{1}} \underline{5 \ 4} \mid$
我 爱北京 天安 门， 天安门上 太 阳 升， 伟 大领袖

$\underline{3 \ 5} \underline{2} \mid \underline{4 \cdot 3} \underline{2 \ 5} \mid \underline{5 \ 2 \ 3} \mid \underline{1 -} \mid \underline{1 \ 0} \mid \underline{5 \cdot 3} \mid \underline{6 \ \dot{1}} \mid$
毛 主 席， 指 引我们 向 前 进。 我 爱 北 京

$\underline{7 \ 6 \ 7} \mid \underline{5 \ 3} \mid \underline{\dot{2} \cdot \dot{2}} \underline{\dot{2} \ \dot{1}} \mid \underline{7 \ 6 \ \dot{1}} \mid \underline{5 -} \mid \underline{5 -} \mid \underline{5 \cdot 3} \mid$
天 安 门， 天 安门上 太 阳 升， 伟 大

$\underline{6 \ \dot{1}} \mid \underline{7 \ 6 \ 7 \ \dot{1}} \mid \underline{\dot{2} -} \mid \underline{5 \cdot 6 \ 7 \ \dot{1}} \mid \underline{\dot{2} \ 5} \mid \underline{\dot{1} -} \mid \underline{\dot{1} \ 0} \mid$
领 袖 毛 主 席， 指 引我们 向 前 进。