Lab 8: Music Player

Submission Due Dates:

Source Code: 2020/12/22 18:30

Report: 2020/12/27 23:59

Objective

Getting familiar with the audio peripheral.

Action Items

Design a music player with a song. The player should have these functions:

1. Be able to **Play / Pause**
2. Be able to **Mute**
3. Be able to **Repeat** playing after the song ends
4. Be able to **Rewind** the song
5. Supporting the 5-level volume control
6. Supporting the 3-level octave control
7. Be able to display the current note with the 7-segment display
8. Supporting 2 Tracks (one for the melody, the other for the accompaniment part) per song
9. Be able to switch between two songs (optional bonus)

I/O signal specification

BtnC	**Reset**	
BtnU	**Volume Up**	
BtnD	**Volume Down**	
BtnR	**Higher Octave**	
BtnL	**Lower Octave**	
SW 0	**Play / Pause**	1: Play; 0: Pause
SW 1	**Mute / Normal**	1: Mute; 0: Normal
SW 2	**Repeat / No-repeat**	1: Repeat; 0: Non-repeat
SW 3	**Rewind / Normal**	1: Rewind; 0: Normal
SW 4	**Music** (optional bonus)	0: the first song; 1: the second song
LED 0~4	**Volume Indicator**	
LED 15~13	**Octave Indicator**	
Pmod JB 1~6	**Pmod I2S**	
7-Segment	**Displaying Current Note**	

Description

1. Upon the **Reset**, the player should not play any tone until the **Play/Pause** is switched to the **Play** mode. Also, it should play the music from the beginning. Upon the **Reset**, the default volume level is **3**; the default octave level is **2** (normal).

2. By switching the **Play/Pause** to "Pause" while playing, the player should pause immediately right at the moment and play no tone. The player should resume the playing from where it paused by switching to "Play" mode again.

3. When reaching the end of the music, the player should
   a. Play it again if it is in the **Repeat** mode;
   b. Stop otherwise.

4. By switching to the **Rewind** mode, the player should rewind the song from its current note. When reaching the beginning of the music, the player should stop the music.

   Note: Repeat function has NO effect in the **Rewind** mode.

5. If you press **Reset** in the **Rewind** mode, the player should play no sound and stop at the beginning. It plays no sound until you turn off the **Rewind** mode and turn the **Play/Pause** switch to the **Play** mode. (Please refer to point 1.)

6. Properly process the signals from pushbuttons with the debouncing and one-pulse converters.

7. Display the melody pitch (C, D, E, F, G, A, and B) on the 7-segment display. Sharp/flat notation and octave notation can be ignored. For example, you can display "- - - G" when the first note of *Lightly Row* is being played. You should display "- - - -" when the player is paused.

8. Design 5 distinguishable levels of volume, which is controlled by **Volume Up** and **Volume Down**. At the lowest level, the sound should be louder than the **Mute** mode. Pressing **Volume Up** at Level 5 or pressing **Volume Down** at Level 1 has no effect. (You should define your own sound levels.)

9. When **Muted**, the music keeps playing with no sound. This is different from the **Pause** mode.

10. LED 0~4 indicates the current volume:

Volume Level	LED 4	LED 3	LED 2	LED 1	LED 0
Muted	○	○	○	○	○
Level 1	○	○	○	○	●
Level 2	○	○	○	●	●
Level 3	○	○	●	●	●
Level 4	○	●	●	●	●
Level 5 (Loudest)	●	●	●	●	●

11. Design 3 levels of octaves, which is controlled by **Higher Octave** and **Lower Octave**. Pressing **Higher Octave** at level 3 or pressing **Lower Octave** at level 1 has no effect.

12. Both melody and accompaniment parts should be controlled by the **Octave** function.

13. Raising a note an octave higher means to double its frequency. (ex. A4 with frequency 440 Hz -> A5 with frequency 880 Hz)

14. A little difference can be ignored when doubling frequencies. (ex. C4 with frequency 262 Hz -> C5 with frequency 523 Hz)

15. LED 15~13 indicates the current octave:

Octave Level		LED 15	LED 14	LED 13
Level 1	lower	●	○	○
Level 2	normal	○	●	○
Level 3	higher	○	○	●

Bonus

1. You may choose 2 pieces of music from the samples below. The player plays one of them when the Music switch is 0, and the other when the switch is 1. You may define the song list by yourself.

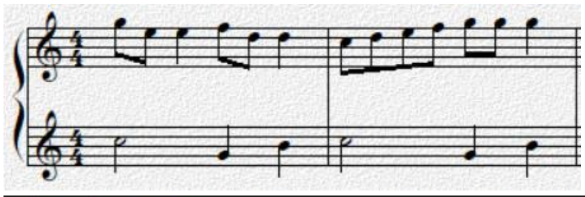
2. Every time the Music switch is changed, the player starts playing the corresponding song from the beginning.

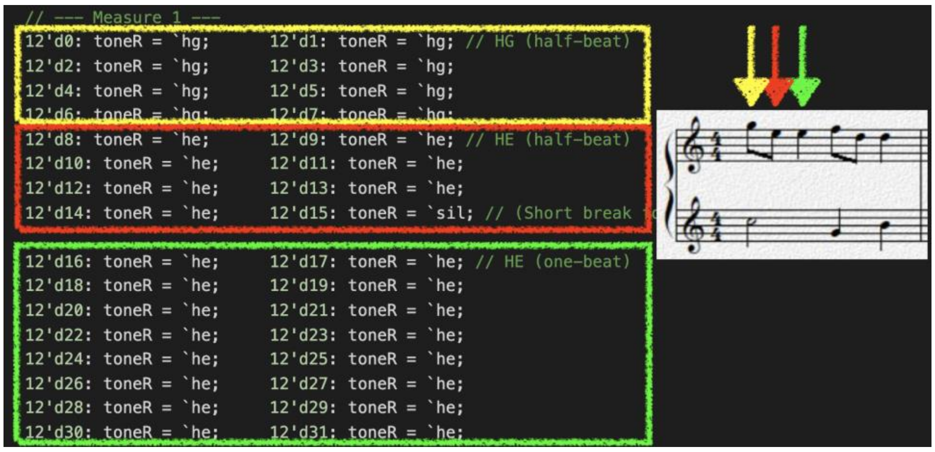
Attention

1. You should hand in only one Verilog file, lab8.v.
2. Finish the modules from the template, and integrate them in lab8.v. (You don't need to put debounce, clock_divider and one-pulse in the file you hand in. Please do not integrate them in lab8.v)
3. You should also hand in your report as **lab8_report_StudentID.pdf** (e.g. **lab8_report_108062666.pdf**).
4. Please do not hand in any compressed files, which will be considered as an incorrect format.
5. You should be able to answer the questions of this lab from TA during the demo.
6. You need to generate the bitstream before the demo.
7. If you have any questions about the spec, feel free to ask on the iLMS forum.

Pick a Song

The design template plays the following music.

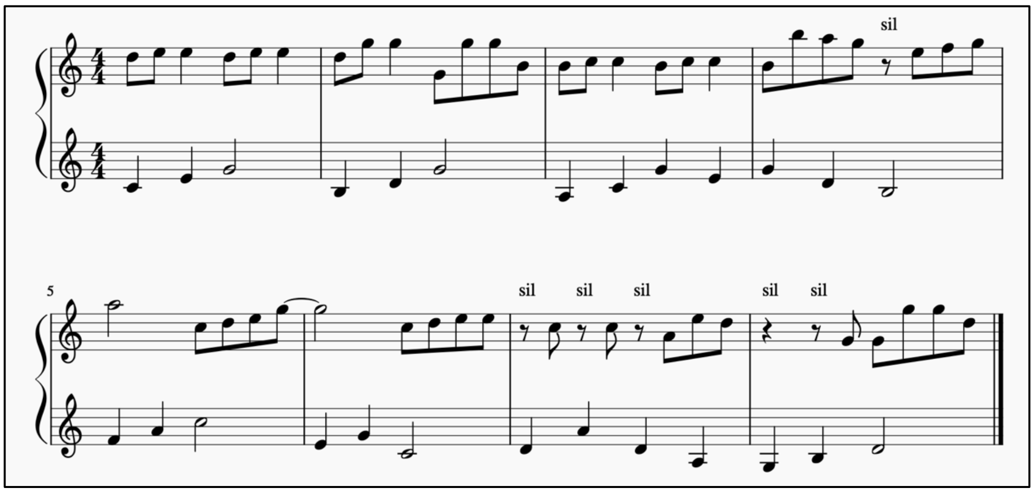




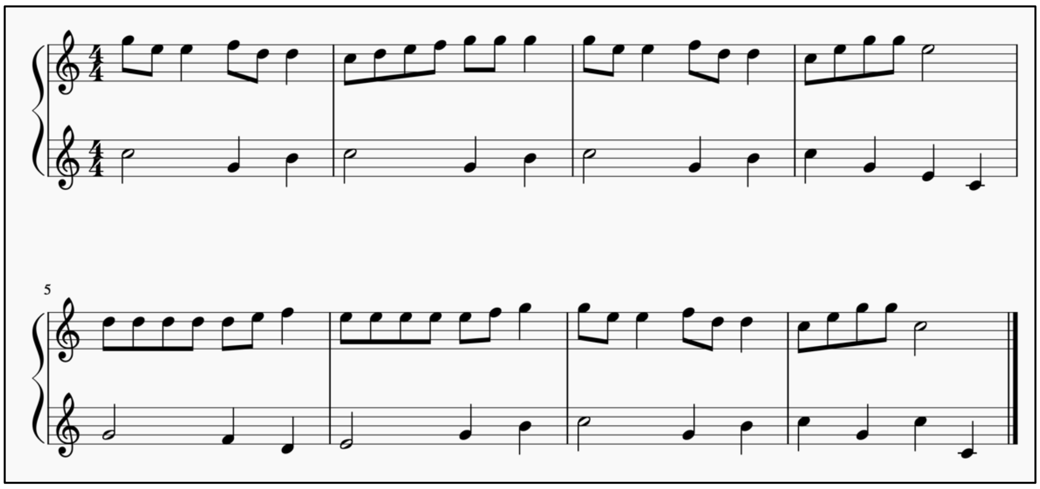
```
// --- Measure 1 ---
12'd0: toneR = `hg;       12'd1: toneR = `hg; // HG (half-beat)
12'd2: toneR = `hg;       12'd3: toneR = `hg;
12'd4: toneR = `hg;       12'd5: toneR = `hg;
12'd6: toneR = `hg;       12'd7: toneR = `hg;
12'd8: toneR = `he;       12'd9: toneR = `he; // HE (half-beat)
12'd10: toneR = `he;      12'd11: toneR = `he;
12'd12: toneR = `he;      12'd13: toneR = `he;
12'd14: toneR = `he;      12'd15: toneR = `sil; // (Short break

12'd16: toneR = `he;      12'd17: toneR = `he; // HE (one-beat)
12'd18: toneR = `he;      12'd19: toneR = `he;
12'd20: toneR = `he;      12'd21: toneR = `he;
12'd22: toneR = `he;      12'd23: toneR = `he;
12'd24: toneR = `he;      12'd25: toneR = `he;
12'd26: toneR = `he;      12'd27: toneR = `he;
12'd28: toneR = `he;      12'd29: toneR = `he;
12'd30: toneR = `he;      12'd31: toneR = `he;
```

You can pick any song of the sample songs listed below. Pick two songs to design the bonus. The Music switch controls which one to play by your definition.
They are all 8 measures (小節) in length.

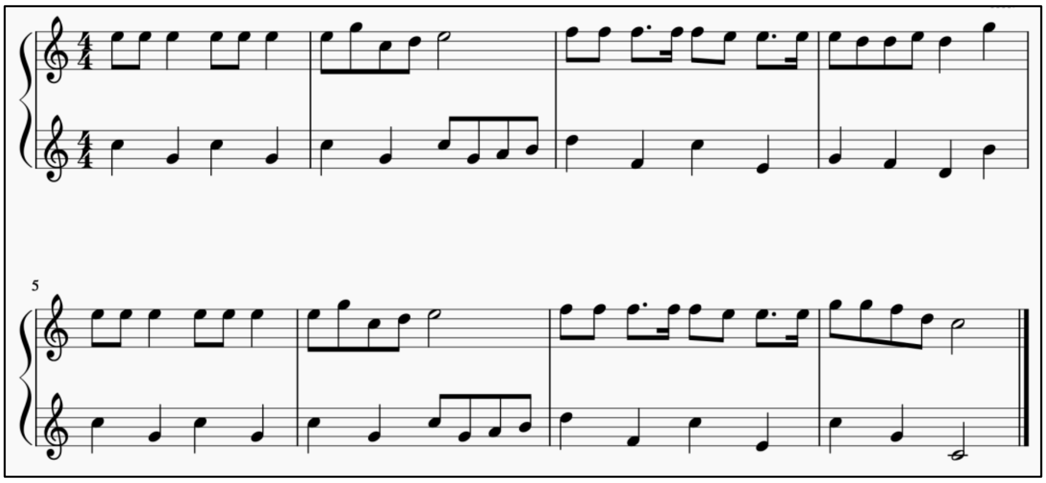
1. 你，好不好



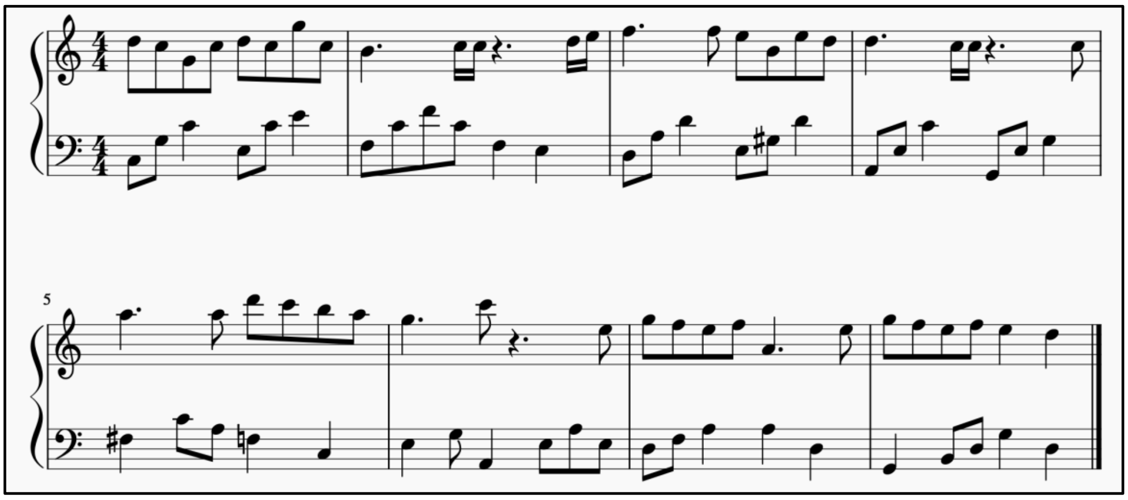
2. Lightly Row



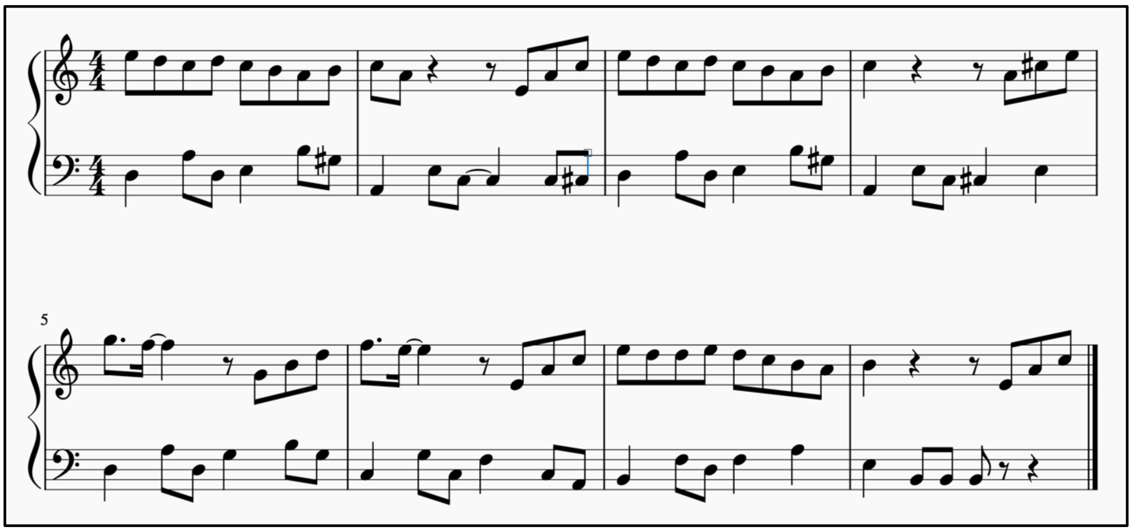
3. Jingle Bells



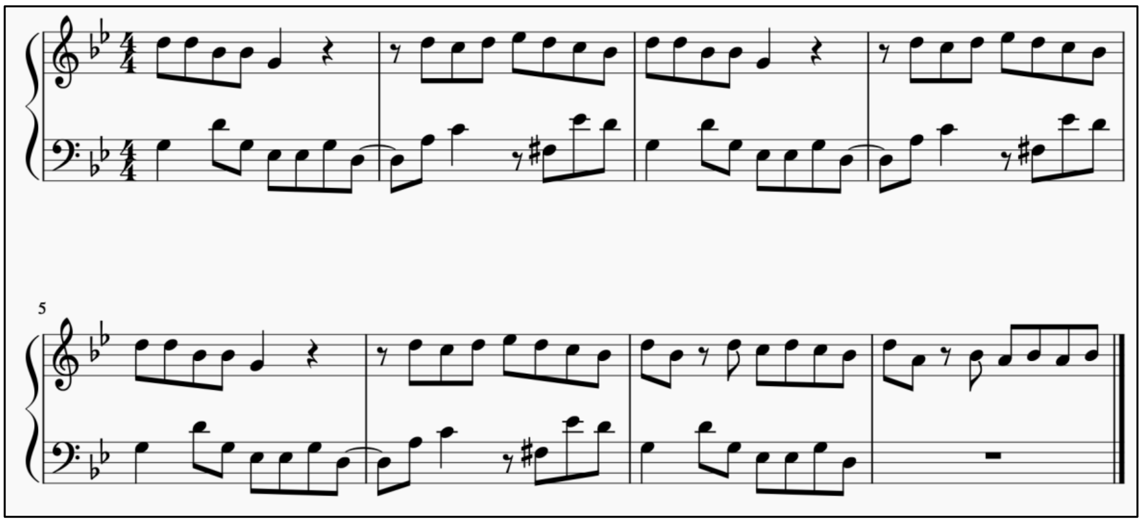
4. 等你下課 (Note some temporary F♯ and the bass clef 𝄢)



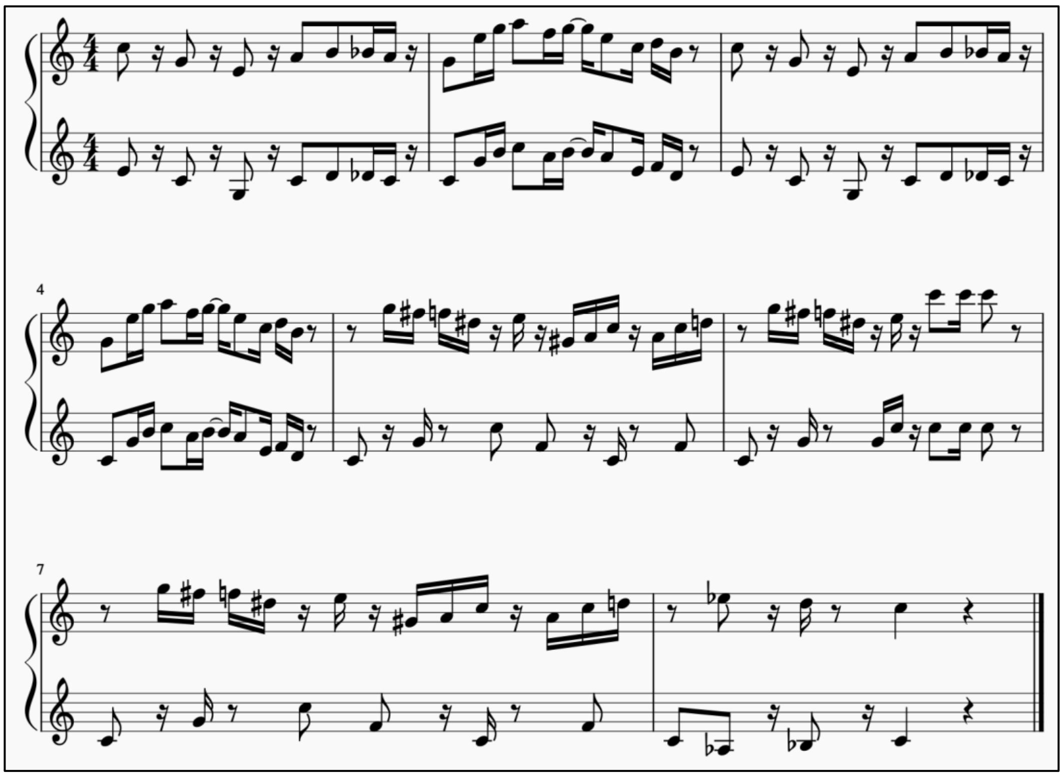
5. Mojito (Note some temporary G♯, C♯ and the bass clef 𝄢)



6. Havana (Note the B♭, E♭ and temporary F♯ and the bass clef 𝄢)



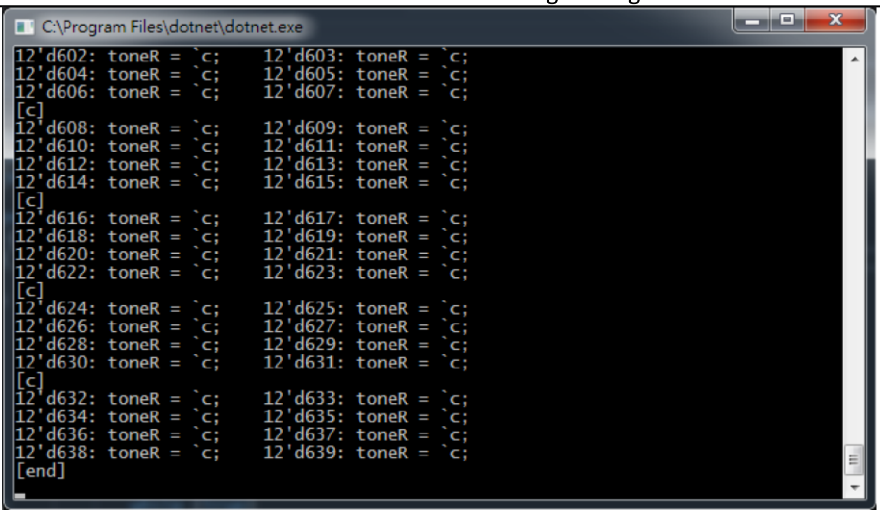
7. Mario Theme Music (Note some temporary sharp ♯, flat ♭ and natural ♮ symbols)



8. Any (beautiful) songs you like!
   Music of your choice/creation/arrangement should contain an accompaniment part and not too short (with at least 8 measures); you should provide your score and be confident in convincing TAs that it is good enough.

Hints

- You can add or modify some modules in the template. For example, an FSM may be a good option for controlling the player.
- Use one track for melody, another for the accompaniment (toneR, toneL).
- Remember that the buzzer/speaker uses 2's complement numbers for audio signals. When designing the volume level, choose the peak value to be some certain val and $-val$.
- If two consecutive notes have the same frequency, it may not be possible to tell when the second note starts. Therefore, in the template, one Quarter Note ♩ is divided into 16 beats further, for the sophisticated note arrangement. You may use a short rest (silence) so that the 2 same-frequency notes can be separated by a short break. (Refer to music_example.v)
- Since a quarter note is 16 beats, it may be tedious to enter all the notes one by one (8 measures consist of 8*4*16=512 beats in total). In that case, you may want to write an aid program like:



```
12'd602: toneR = `c;     12'd603: toneR = `c;
12'd604: toneR = `c;     12'd605: toneR = `c;
12'd606: toneR = `c;     12'd607: toneR = `c;
[c]
12'd608: toneR = `c;     12'd609: toneR = `c;
12'd610: toneR = `c;     12'd611: toneR = `c;
12'd612: toneR = `c;     12'd613: toneR = `c;
12'd614: toneR = `c;     12'd615: toneR = `c;
[c]
12'd616: toneR = `c;     12'd617: toneR = `c;
12'd618: toneR = `c;     12'd619: toneR = `c;
12'd620: toneR = `c;     12'd621: toneR = `c;
12'd622: toneR = `c;     12'd623: toneR = `c;
[c]
12'd624: toneR = `c;     12'd625: toneR = `c;
12'd626: toneR = `c;     12'd627: toneR = `c;
12'd628: toneR = `c;     12'd629: toneR = `c;
12'd630: toneR = `c;     12'd631: toneR = `c;
[c]
12'd632: toneR = `c;     12'd633: toneR = `c;
12'd634: toneR = `c;     12'd635: toneR = `c;
12'd636: toneR = `c;     12'd637: toneR = `c;
12'd638: toneR = `c;     12'd639: toneR = `c;
[end]
```

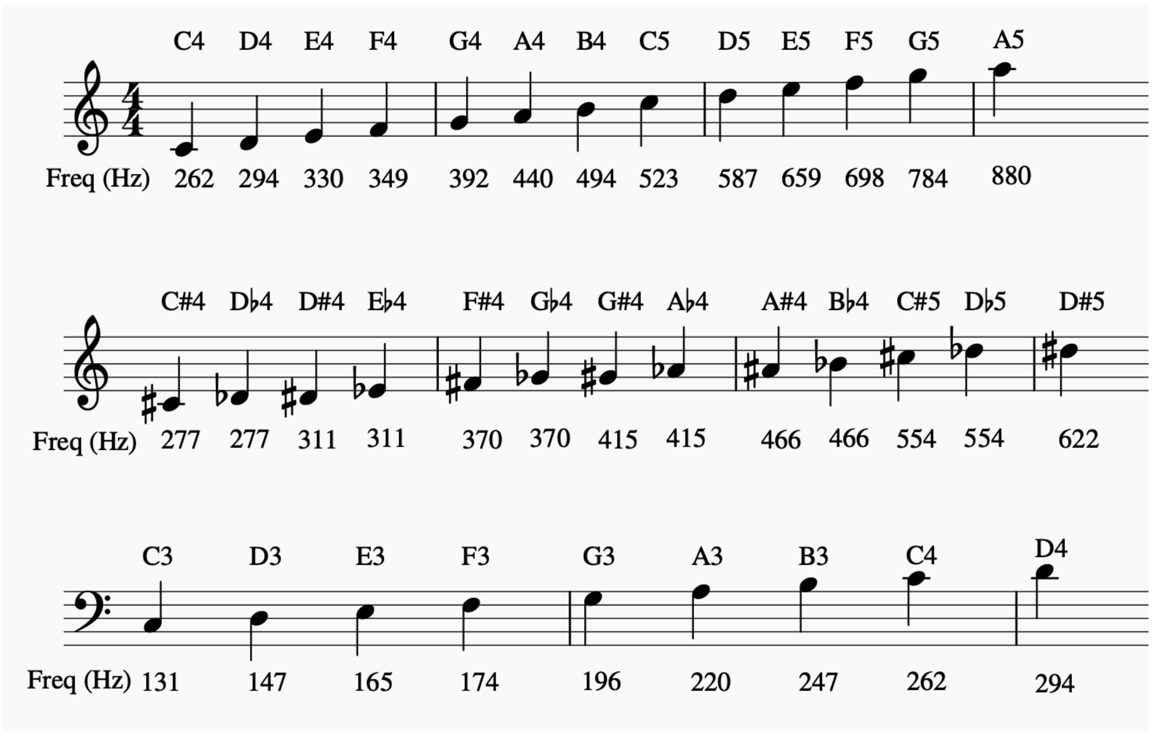
So that you can copy-n-paste them, saving a tremendous amount of time.

- A note-to-frequency table is in the appendix for your reference.
- You can use multiple music modules or player modules if that helps.
- We use square waves. So, you know that the music may not sound smooth.

Appendix

Pitch-to-Frequency Table

The number after the notation indicates how high the pitch is.


C4 D4 E4 F4 G4 A4 B4 C5 D5 E5 F5 G5 A5
Freq (Hz) 262 294 330 349 392 440 494 523 587 659 698 784 880
C#4 D♭4 D#4 E♭4 F#4 G♭4 G#4 A♭4 A#4 B♭4 C#5 D♭5 D#5
Freq (Hz) 277 277 311 311 370 370 415 415 466 466 554 554 622
C3 D3 E3 F3 G3 A3 B3 C4 D4
Freq (Hz) 131 147 165 174 196 220 247 262 294

You can use these frequencies in your code like that in music_example.v.

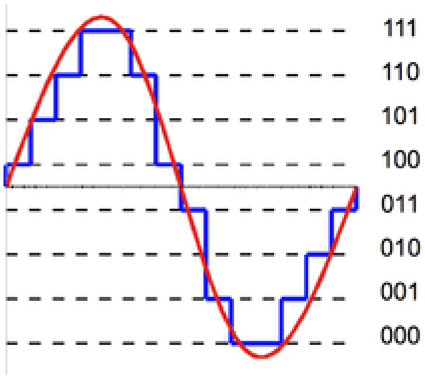
Or you can refer to this page.

Square Waves and Sine Waves

We use square waves to control the speaker, which inevitably results in a buzzing sound.

Triangle waves will do, but sine waves are usually the most natural.

Sine waves can be emulated in Verilog with a look-up table. But it results in some **quantization noises** (if no interpolation is involved) that make the sound terrible. (like the picture below, from Wikipedia)



However, there is an algorithm known as **CORDIC**, or **Volder's algorithm**, a simple and efficient way to calculate trigonometric functions, even when using the FPGA. Refer to Wikipedia for more information. Vivado also provides the CORDIC IP in the IP Catalog. So you may find it handy in this lab if you really cannot stand the square wave sound or if you are going to improve the audio effect in your final project. (You can also generate audio data in your PC and store it in the block memory of Basys3. But it will consume a lot of memory.)

Different Timbres (音色)

Refer to this video to gain an insight of how timbres are made of. Actually, the frequency change (vibrations on violins), amplitude change (like the fading sound of pianos), and how the sound waves start can determine how we perceive the timbres. Even the ratio of overtones can differ on the same musical instrument when it plays different frequencies.