

Lab 08: Music Player

(Nov. 21, 2019)

Submission deadlines:

Source Code:	18:30, Dec. 03, 2019
Report	23:59, Dec. 08, 2019

Objective

- To be familiar with the audio peripheral.

Action Items

Design a music player with 2 songs. The player should have these functions:

- Supporting 2 Tracks (one for melody, one for accompaniment part) per song
- Be able to **Play / Pause**
- Be able to **Mute**
- Be able to switch between two songs
- Be able to repeat playing after the song ends
- Supporting the 5-level volume control
- Be able to display the current note with 7 segment display

I/O ports are specified as follows:

BtnC	Reset
BtnU	Volume Up
BtnD	Volume Down
SW 0	Play / Pause (1: Play; 0: Pause)
SW 1	Mute / Normal (1: Mute; 0: Normal)
SW 2	Repeat / Non-repeat (1: Repeat; 0: Non-repeat)
SW 3	Music (0: the first song; 1: the second song)
LED 0~4	Volume Indicator
Pmod JB 1~6	Pmod I2S
7 Segment	For Displaying Current Note

- ✓ Upon the reset, the player should not play any tone until the **Play/Paused** is switched to the "Play" mode. Also, reset the volume level to 3.
- ✓ By switching the **Play/Paused** to "Pause" while playing, the player should pause immediately right at the moment you turn it off. The player should be able to resume the playing from where it paused by switching to "Play" mode again.
- ✓ You may choose 2 pieces of music from the samples below. The player plays one of them when the **Music** switch is 1, and the other when the switch is 0. You may define the song list by yourself.
- ✓ Every time after the Music switch is changed, the player starts playing the corresponding song from the beginning.
- ✓ When reaching the end of the music, the player should
 - Play it again if it is in the **Repeat** mode;
 - Stop otherwise.
- ✓ Design 5 distinguishable levels of volumes, 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.) After reset, volume level should be 3.
- ✓ When **Muted**, the music keeps playing with no sound. This is different from the **Pause** mode.
- ✓ 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)	●	●	●	●	●

- ✓ Properly process the signals from buttons, with debounce and onepulse.
- ✓ Display the music pitch with 7-segment display, C, D, E, F, G, A, and B. sharp notation/flat notation/high- or low-pitch notation can be ignored. For example, you can display "---G" when the first note of *Lightly Row* is being played. (Only for the melody part.)

Pick 2 Songs

The template is a loop of the following music.



You can pick 2 out of the sample songs listed below. The Music switch controls which one to play by your definition. They are all 8 measures (小節) in length:

1. Lightly Row



2. Jingle Bells (Note the F#, C#, and the Natural F \natural notation)



3. Pachelbel's Canon in D (Note the F# and C#)



4. Havana (Note the Bb, Eb, and temporary F#)



5. Frere Jacques




6. 告白氣球 (Note some temporary G#)

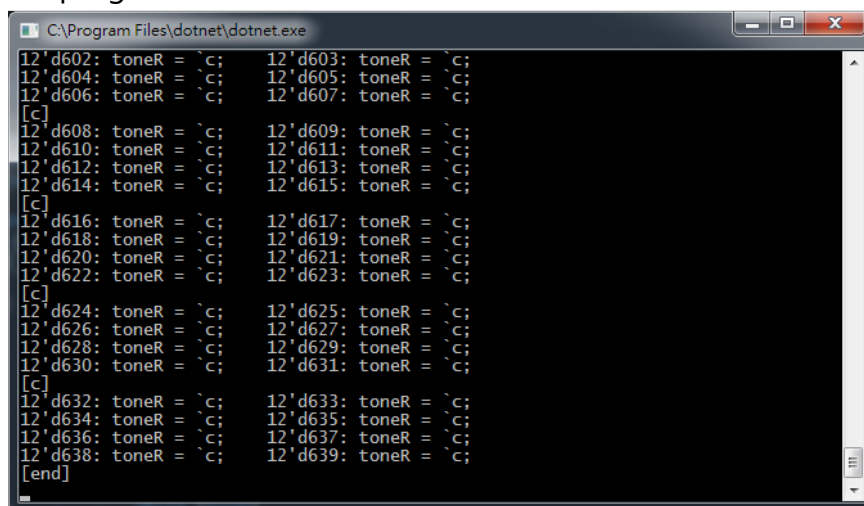


7. Any (beautiful) songs you like! (Music of your choice/creation/arrangement should contain accompaniment part and not too short; you should be confident in convincing TAs that it is good enough.)

Hints

- You can add or modify some modules in the template. For example, an FSM may be a good option for modeling player behavior.

- If two same-frequency notes appear one after another, it may be impossible to tell what time the second note starts at. 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 $8 \times 4 \times 16 = 512$ beats in total). In that case, you may want to write an aid program like:



```

C:\Program Files\dotnet\dotnet.exe
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 great amount of time.

- 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 volume level, choose the peak value to be some certian `val` and `-val`.
- 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](#). Just so you know that the music may not sound smooth.

Appendix

Pitch-to-Frequency Table

The subscript number indicates how high the pitch is.



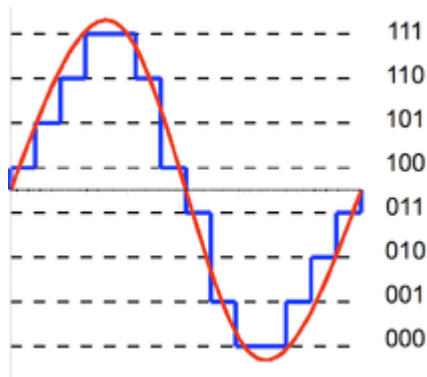
You can use these frequencies in your code like that in `music_example.v`. Or you can refer to [this page](#).

Square Waves and Sine Waves

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

[Triangle waves](#) will do, but [sine waves](#) are usually the most natural.

Sine waves can be simulated 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. In Vivado, the CORDIC is 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 make something audio-related in your final project. (You can also generate audio in your PC, and store it in the block memory of Basys3 though.)

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.