

# EE345L – Lab 5: Music Player and Audio Amp

Ty Winkler and Jeremiah Bartlett

03/7/16

## 1.0 OBJECTIVE

The objective of this lab is to design and build a music player using a TLV5616 DAC and TPA731D audio amp. In this lab we will learn how to communicate with the DAC using SSI, how to store music in the TM4C123's memory, and how to assemble a circuit requiring a DAC and audio amp.

## Requirements document

### 1. Overview

#### 1.1. Objectives: Why are we doing this project? What is the purpose?

The objectives of this project are to design, build and test a music player. Educationally, students are learning how to interface a DAC, how to design a speaker amplifier, how to store digital music in ROM, and how to perform DAC output in the background. Your goal is to play your favorite song.

#### 1.2. Process: How will the project be developed?

The project will be developed using the TM4C123 board. There will be two or three switches that the operator will use to control the music player. The system will be built on a solderless breadboard and run on the usual USB power. The system may use the on board switches or off-board switches. A hardware/software interface will be designed that allows software to control the player. There will be at least three hardware/software modules: switch input, DAC output, and the music player. The process will be to design and test each module independently from the other modules. After each module is tested, the system will be built and tested.

#### 1.3. Roles and Responsibilities: Who will do what? Who are the clients?

EE445L students are the engineers and the TA is the client. Students are expected to make minor modifications to this document in order to clarify exactly what they plan to build. Students are allowed to divide responsibilities of the project however they wish, but, at the time of demonstration, both students are expected to understand all aspects of the design.

#### 1.4. Interactions with Existing Systems: How will it fit in?

The system will use the TM4C123 board, a solderless breadboard, and the speaker as shown in Figure 5.1. It will be powered using the USB cable. You may use a +5V power from the lab bench, but please do not power the TPA731 or the speaker with a voltage above +5V.

#### 1.5. Terminology: Define terms used in the document.

For the terms SSI, linearity, frequency response, loudness, pitch, instrument, tempo, envelope, melody and harmony. See textbook for definitions.

#### 1.6. Security: How will intellectual property be managed?

The system may include software from StellarisWare and from the book. No software written for this project may be transmitted, viewed, or communicated with any other EE445L student past, present, or future (other than the lab partner of course). It is the responsibility of the team to keep its EE445L lab solutions secure.

## 2. Function Description

### 2.1. Functionality: What will the system do precisely?

If the operator presses the play/pause button the music will play or pause. If the operator presses the play/pause button once the music should pause. Hitting the play/pause again causes music to continue. The play/pause button does not restart from the beginning, rather it continues from the position it was paused. If the rewind button is pressed, the music stops and the next play operation will start from the beginning. There is a mode switch that allows the operator to control some aspect of the player. Possibilities include instrument, envelope or tempo. *(Note to students: if you use the internal switches you could rename the switches Up Down Left Right or Select to match the switches you use)* *(Note to students: specify exactly what your mode button does.)*

There must be a C data structure to hold the music. There must be a music driver that plays songs. The length of the song should be at least 30 seconds and comprise of at least 8 different frequencies. Although you will be playing only one song, the song data itself will be stored in a separate place and be easy to change. The player runs in the background using interrupts. The foreground (main) initializes the player, then executes `for (;) {}` do nothing loop. If you wish to include LCD output, this output should occur in the foreground. The maximum time to execute one instance of the ISR is xxxx (*note to students: replace the xxxx with performance measure of your solution*). You will need public functions **Rewind**, **Play** and **Stop**, which perform operations like a cassette tape player. The **Play** function has an input parameter that defines the song to play. A background thread implemented with output compare will fetch data out of your music structure and send them to the DAC.

There must be a C data structure to store the sound waveform or instrument. You are free to design your own format, as long as it uses a formal data structure (i.e., **struct**). The generated music must sound beautiful utilizing the SNR of the DAC. Although you only have to implement one instrument, it should be easy to change instruments.

2.2. Scope: List the phases and what will be delivered in each phase.

Phase 1 is the preparation; phase 2 is the demonstration; and phase 3 is the lab report. Details can be found in the lab manual.

2.3. Prototypes: How will intermediate progress be demonstrated?

A prototype system running on the TM4C123 board and solderless breadboard will be demonstrated. Progress will be judged by the preparation, demonstration and lab report.

2.4. Performance: Define the measures and describe how they will be determined.

The system will be judged by three qualitative measures. First, the software modules must be easy to understand and well-organized. Second, the system must employ an abstract data structures to hold the sound and the music. There should be a clear and obvious translation from sheet music to the data structure. Backward jumps in the ISR are not allowed. Waiting for SSI output to complete is an acceptable backwards jump. Third, all software will be judged according to style guidelines. Software must follow the style described in Section 3.3 of the book (*note to students: you may edit this sentence to define a different style format*). There are three quantitative measures. First, the SNR of the DAC output of a sine wave should be measured. Second, the maximum time to run one instance of the ISR will be recorded. Third, you will measure power supply current to run the system. There is no particular need to optimize any of these quantitative measures in this system.

2.5. Usability: Describe the interfaces. Be quantitative if possible.

There will be three switch inputs. The DAC will be interfaced to a 32-ohm speaker. (*note to students: you could use 8 ohm speaker*)

2.6. Safety: Explain any safety requirements and how they will be measured.

If you are using headphones, please verify the sound it not too loud before placing the phones next to your ears.

### 3. Deliverables

3.1. Reports: How will the system be described?

A lab report described below is due by the due date listed in the syllabus. This report includes the final requirements document.

3.2. Audits: How will the clients evaluate progress?

The preparation is due at the beginning of the lab period on the date listed in the syllabus.

3.3. Outcomes: What are the deliverables? How do we know when it is done?

There are three deliverables: preparation, demonstration, and report.

## 2.0 ANALYSIS AND DISCUSSION

### 2.1 DAC Errors

DACs have three different errors an offset error which is the offset of actual output compared to ideal, a gain error which is the difference between ideal and actual output when a full spectrum of output is

sent to the DAC, and a differential nonlinearity which is the offset of the output when a linear set of values is output from the DAC.

## **2.2 SSI Interfacing**

The timing requirements for the DAC are as follows: setup time 8ns, hold time 5ns, and pulse duration 25ns. The SSI frequency is 8MHz = 125ns. Thus the data available is set to 125ns and the data required is 38ns making the data available longer than the time needed to input data to the DAC.

## **2.3 Spectrum Analyzer**

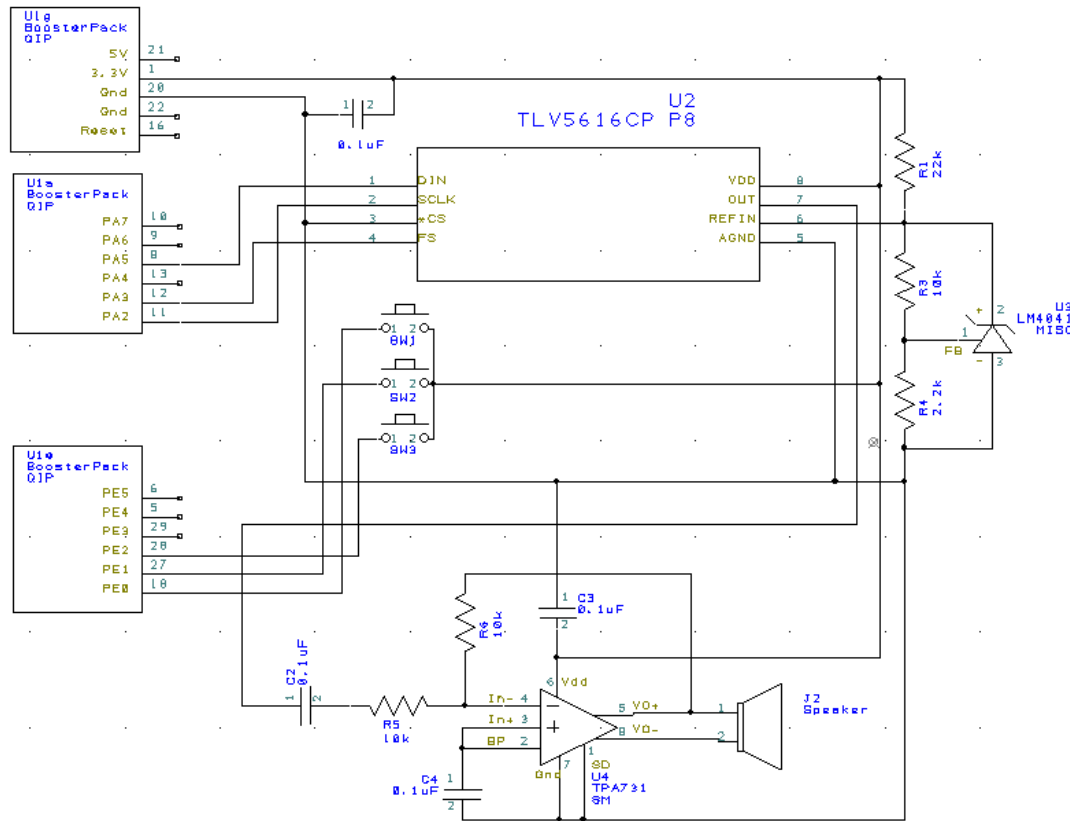
The spectrum analyzer takes the largest and lowest values of the input frequency and by adding an offset a frequency range is determined.

## **2.4 TPA731**

The TPA731 audio amp amplifies low power audio signals bring the frequencies to a suitable level for driving a speaker. If we were to use the speaker directly with the DAC, the sound coming from the speaker would not be as clear and at a much quitter volume.

### 3.0 Software & Hardware Design Solutions

#### 3.1 Hardware Design



#### 3.2 Measurement Data

Resolution, Range, Precision, and Accuracy

DAC_Out(X)	Expected V Out	Actual V Out
0	0	0
512	0.4125	.01
1024	0.825	.03
1536	1.2375	.05
2048	1.65	.10
2560	2.0625	.13
3072	2.475	.15
3584	2.8875	.17
4096	3.3	.18

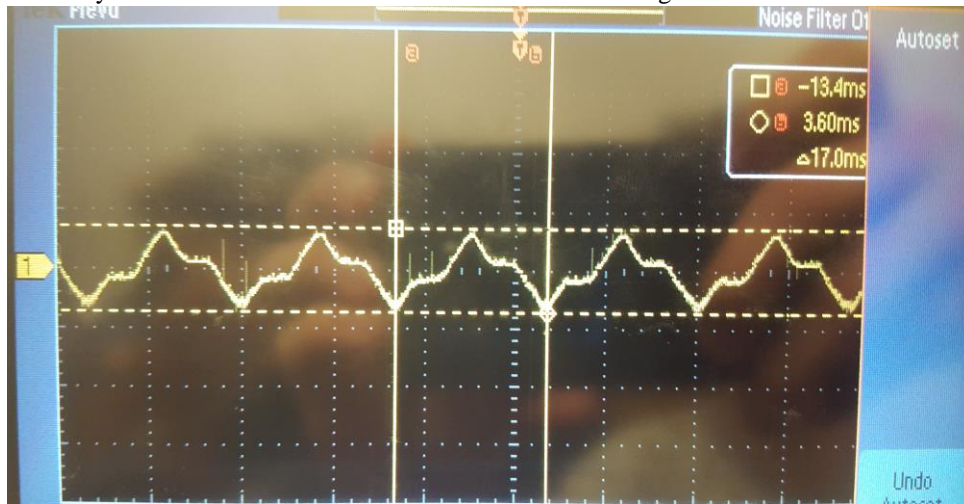
DAC_Out(X)	Expected V Out	Actual V Out
1024	0.825	.03
1024	0.825	.04
1024	0.825	.03
1024	0.825	.03
1024	0.825	.02
1024	0.825	.04
1024	0.825	.03
1024	0.825	.04
1024	0.825	.03

Resolution = 0.01V +/- 3V

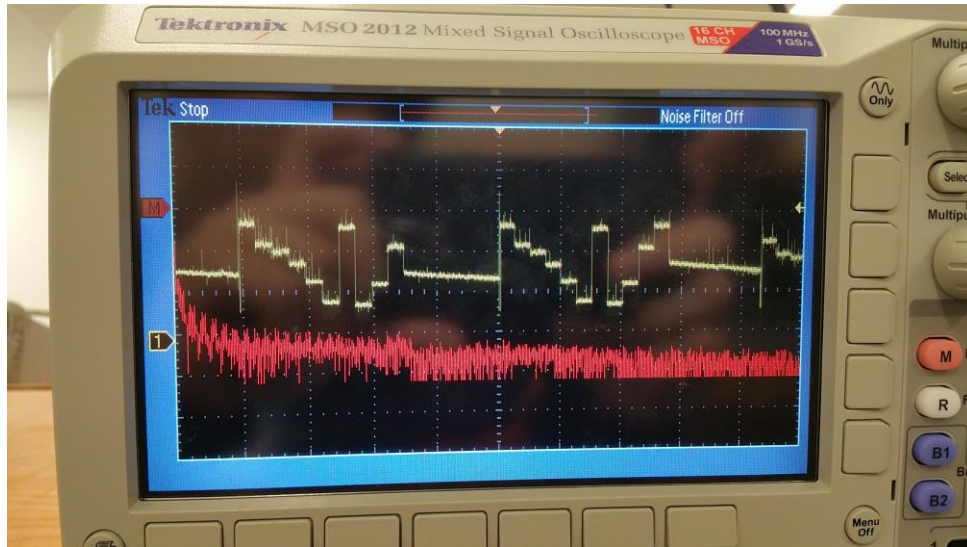
Range = 0-0.18V

Precision = 100%. Due to a weird coincidence, the standard deviation is 0. Thus our precision is 100%

Accuracy = 96.12% Error. 3.88% accurate because of a voltage difference.



## Experimental Response of the DAC & SNR



SNR = 20dB

### Debugging Data Dump of 100 interrupt times

```
0x20000034 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000044 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000054 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000064 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000074 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000084 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000094 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200000A4 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200000B4 9E 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200000C4 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200000D4 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200000E4 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200000F4 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000104 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000114 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000124 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000134 A2 00 00 00 9E 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000144 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000154 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000164 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000174 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000184 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x20000194 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200001A4 A2 00 00 00 A2 00 00 00 - A2 00 00 00 A2 00 00 00 .....
0x200001B4 A2 00 00 00 A2 00 00 00 - 9E 00 00 00 AE .....
```

The interrupt requires 162ns to complete

Current Required w/ and w/o music playing

