EE 445L - Lab 3: Alarm Clock

Joshua Bryant jmb6357

James Morris jsm3288

October 13, 2014

1 Objective

1.1 Overview

1.1.1 Objectives

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 outputs in the background. Your goal is to play your favorite song.

1.1.2 Process

The project will be developed using the TM4C1294 board. There will be 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 will use 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.

1.1.3 Roles and Responsibilities

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.

1.1.4 Interactions with Existing Systems

The system will use the TM4C1294 board, a solderless breadboard, and the speaker as shown in figure 5.1 in the Lab05.doc. It will be powered using the USB cable.

1.1.5 Terminology

SSI A widely used serial interface standard for industrial applications between a master and a slave. SSI is based on RS-422 standards and has a high protocol efficiency.

Linearity A linear function is a function that satisfies two properties: additivity and homogeneity.

Frequency Response The frequency at which gain drops to 0.707 of the normal value. For a low pass system, the frequency response ranges from 0 to a maximum value. For a high pass system, the frequency response ranges from a minimum value to infinity. For a bandpass system, the frequency response ranges from a minimum to a maximum value.

Loudness The characteristic of a sound that is a subjective measure and is often confused with objective measures of sound strength such as sound pressure, sound pressure level, sound intensity or sound power.

Pitch A perceptual property that allows the ordering of sounds on a frequency-related scale.

Instrument An embedded system that collects information, same as data acquisition system.

Tempo Speed or pace of a given piece of music.

Envelope A function used for shaping the amplitude, frequency, or phase of a signal.

Melody A linear succession of musical tones that the listener perceives as a single entity.

Harmony The use of simultaneous pitches or chords.

1.1.6 Security

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.

1.2 Function Description

1.2.1 Functionality

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.
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 sounds. Although you will be playing 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 OLED output, this output should occur in the foreground. The maximum time to execute one instance of the ISR will be measured in lab and will be included in this document once it has been tested. 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 won 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.

1.2.2 Scope

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

1.2.3 Prototypes

A prototype system running on the TM4C1294 board and solderless breadboard will be demonstrated. Progress will be judged by the preparation, demonstration, and lab report.

1.2.4 Performance

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 structure 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. 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.

1.2.5 Usability

There will be three switch inputs. The DAC will be interfaced to a 32-ohm speaker.

1.2.6 Safety

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

1.3 Deliverables

1.3.1 Reports

A lab report described in Lab05.doc is due by the date listed in the syllabus. This report includes the final requirements document.

1.3.2 Audits

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

1.3.3 Outcomes

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

2 Hardware Design

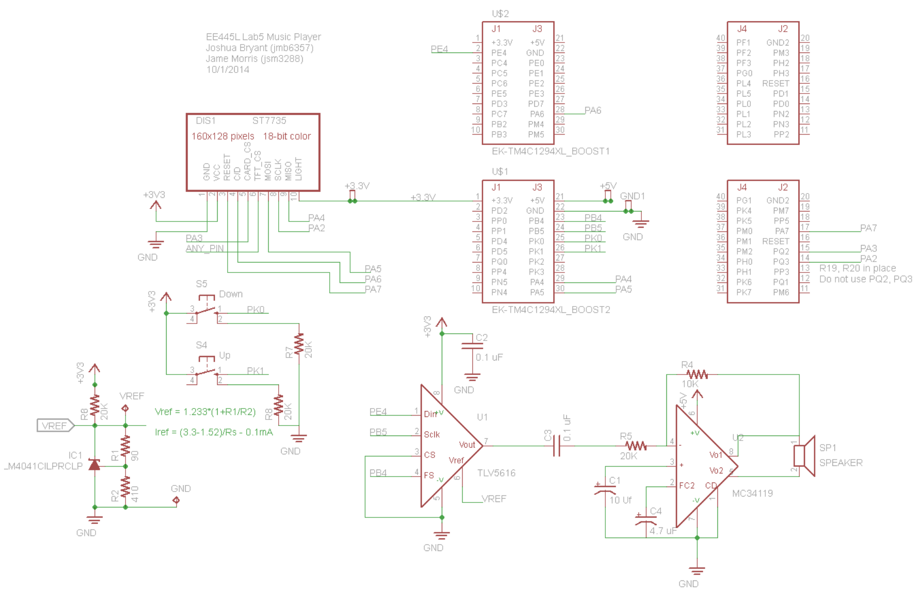

Figure 1: Schematic for Lab 5.

3 Software Design

```
uint32_t notePeriod[] = {0, 896, 948, 1005, 1065, 1129, 1195,
    1266, 1343, 1420, 1507, 1594, 1692};

uint16_t tone[] =
    {128,131,134,137,140,143,146,149,152,156,159,162,165,168,171,174,
176,179,182,185,188,191,193,196,199,201,204,206,209,211,213,216,
218,220,222,224,226,228,230,232,234,236,237,239,240,242,243,245,
246,247,248,249,250,251,252,252,253,254,254,255,255,255,255,255,
255,255,255,255,255,255,254,254,253,252,252,251,250,249,248,247,
246,245,243,242,240,239,237,236,234,232,230,228,226,224,222,220,
218,216,213,211,209,206,204,201,199,196,193,191,188,185,182,179,
176,174,171,168,165,162,159,156,152,149,146,143,140,137,134,131,
128,124,121,118,115,112,109,106,103,99, 96, 93, 90, 87, 84, 81,
79, 76, 73, 70, 67, 64, 62, 59, 56, 54, 51, 49, 46, 44, 42, 39,
37, 35, 33, 31, 29, 27, 25, 23, 21, 19, 18, 16, 15, 13, 12, 10,
9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 2, 3, 3, 4, 5, 6, 7, 8,
9, 10, 12, 13, 15, 16, 18, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 42, 44, 46, 49, 51, 54, 56, 59, 62, 64, 67, 70, 73, 76,
79, 81, 84, 87, 90, 93, 96, 99, 103,106,109,112,115,118,121,124};

struct Song {
        uint8_t notes1[60];
        uint8_t duration1[60];
        uint16_t length1;

        uint8_t numInstruments;

        uint8_t notes2[60];
        uint8_t duration2[60];
        uint16_t length2;
    };
```

The above code shows both the structures used to store the sound data as well as the structure used to store the song file. The **notePeriod**[] array contains the periods of middle notes from C down to $D^\flat$. The first entry of "0" is used for rests. The values in the **notePeriod**[] array are used to specify the time between interrupts for outputting values to the DAC. Each note is defined in the code to correspond to an index in this array. In this way, "C" is defined to be 1 so it's corresponding period is 896 clock cycles.
The **tone**[] array is an array that contains all values needed to generate an 8-bit sine-wave. These values are called consecutively by the note interrupt to create the frequencies needed for each note.
The **Song** struct is used to store the notes for a piece of music. The **notes** arrays store the name of the note that is defined in the header or a "0" for a rest. The **duration** array is used to store the length of each note in the same index in **notes**. The duration is stored as the length of the note in terms of 32^{nd} notes. An example is if you wanted to play a middle C as a half-note, you would store C in the 0 index of **notes** and 8 in the 0 index of **duration**. The **length** variable is used to determine how many notes are in the song.
In the **Song** struct, there are two sets of notes and durations stored. This was designed to be able to support playing two notes at a time. Although we ended up not being able to finish it, this code would have been expandable up to n number of instruments given that you had an n number of interrupts. In our code, each independent line of notes is called an instrument. The number of these instruments that each song would contain would be stored in the **numInstruments** variable. The only modification needed to expand to more instruments would be initializing more time interrupts for each instrument and adding the corresponding **notes**, **duration**, and **length** variables for each new instrument. This structure would also allow for an easy addition of different profiles for each instrument.

4 Measurement Data

a. The range of the DAC was 5V, the resolution was $2^8 = 256$ discrete levels, the precision was 19.5 mV.

b. The current required to run the system with the music playing was ∼50

μA and was ~15 μA when the music was not playing.

5 Analysis and Discussion

a. Three errors in a DAC include: (1) Settling Time, (2) Thermal Noise, and (3) Clock Jitter. Settling time is the interval of time between when a new value is sent to the DAC and when the DAC actually outputs the desired value. Thermal noise is a relatively flat spectral noise added into the output of the DAC that increases the SNR of the output signal. Clock jitter is the timing variations of the clock signal from its ideal values.

b. The shortest period for any note our music player was designed to support was 896 clock cycles at 20 MHz and 16 bits were to be transferred at each execution of the interrupt service routine. This leads to transmitting 16 bits in, at most, 44.8 μs. This leads to $\sim 3.6 * 10^5$ bits per second transfer rate required for the SSI. Given this required data transfer rate, the chosen 8 MHz clock for the SSI is more than adequate for our current system and allows for higher notes to be added in at future times.

c. The frequency range of a spectrum analyzer is determined to be from 0 to less than half its maximum sampling frequency as per the Nyquist theorem.

d. The purpose of the MC34119 in the circuit is to turn the digital values of the output signal from the SSI line into analog voltages to drive the speaker. The MC34119 acts as the DAC for out lab.