Lab 5 Music Player and Audio Amp

This laboratory assignment accompanies the book, Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154, by Jonathan W. Valvano, copyright © 2012.

Goals
- DAC conversion,
- SPI/SSI interface,
- Design a data structure to represent music,
- Develop a system to play sounds.

Review
- Search http://www.ti.com/ for a data sheet on the TLV5618AIP 12-bit DAC,
- Valvano Section 6.2 on periodic interrupts using the timer,
- Valvano Section 7.5 on SSI interfacing,
- Valvano Section 8.4 on DAC parameters and waveform generation.

Starter files • `PeriodicTimer0AInts_1968.zip` project, Excel files starting with `dac_`. You will find the TLV5618, the MC34119, and the speaker in the EE345L library **PCBArtistLibrary.zip**.

Background

Most digital music devices rely on high-speed digital to analog converters (DAC) to create the analog waveforms required to produce high-quality sound. First, you will interface a 12-bit DAC, and use it to create a sine-wave output. In particular, you will interface a TI TLV5618 12-bit DAC to an SSI port, as shown in Figure 5.1. Please refer to the DAC data sheets for the synchronous serial protocol. During testing, the output of the DAC will be connected to a voltmeter, an oscilloscope or a spectrum analyzer. You are allowed to use any DAC chip you want, as long as it runs on a single +3.3V supply and has an SSI interface. Many DACs, such as the TLV5618, require a reference voltage. A stable 1.50V reference can be created using a reference chip such as the LM4041C. The LM4041CILPR is an adjustable shunt reference that can be powered from the +3.3V supply, and requires three external resistors to create the 1.50 V reference. Look up in the TLV5618 data sheet to find how much current the DAC needs on its Refin input. In the data sheet you will find the input impedance R_{in} of the Refin pin, you can calculate this load current $I_L = 1.5V/R_{in}$. Next, look up page 8 of the LM4041CILPR data sheet to find I_Z (80 µA) and V_{REF} (1.233V). Select R1 and R2 to set the reference output $V_Z = V_{REF}$ (1+R2/R1) to 1.5V (%%%). The Rs resistor in Figure 5.1 (data sheet page 15) sets the available current for the shunt reference. Make Rs smaller than $(3.3-V_Z)/(I_L+I_Z)$. See Figure 15 and Equation 1 of the LM4041CILPR data sheet. The TLV5618 has no data output, and the data sheet shows which pins to use for an SPI interface.

The second step is to design a low-level device driver for the DAC. For the TLV5618, a 16-bit SSI frame is required to set the DAC output. Next, you will design a data structure to store the sine-wave. The main program will input from switches and allow the operator to select from three pre-defined frequencies.

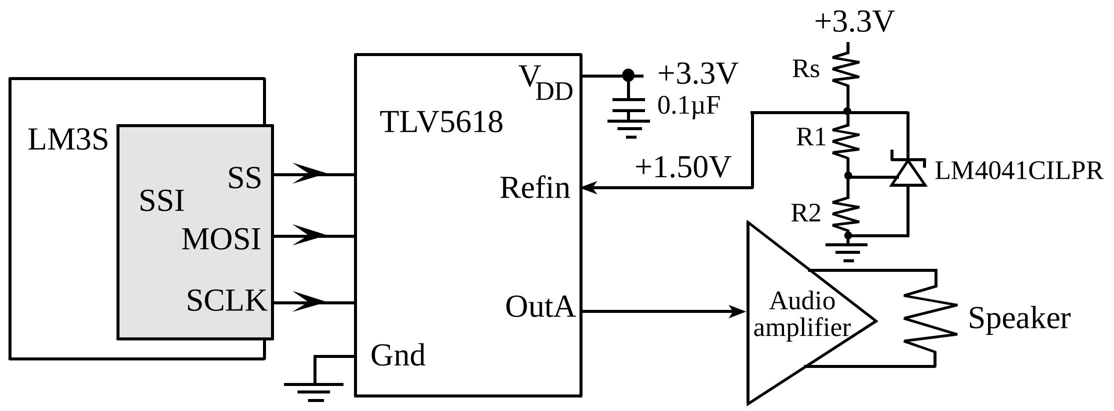
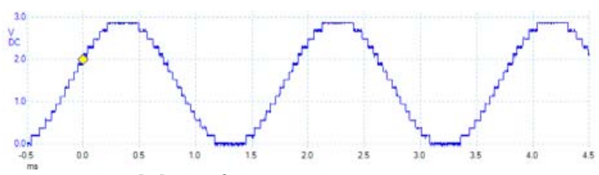
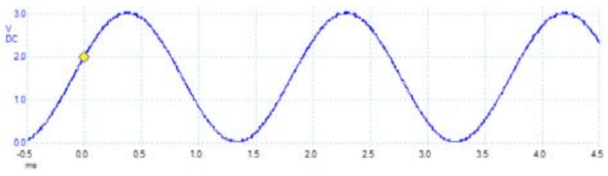


Figure 5.1. Block diagram of the DAC interface using the TLV5618.

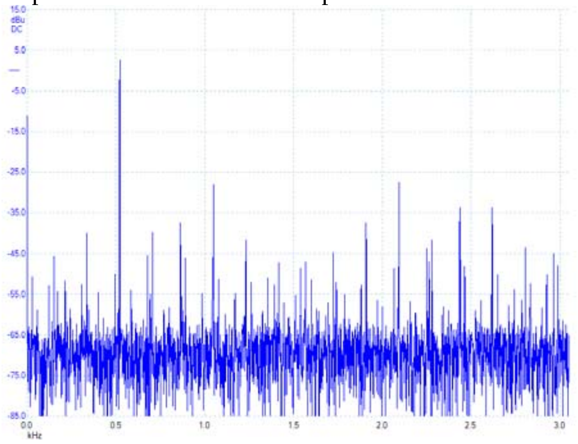
If you output a sequence of numbers to the DAC that form a sine-wave, then you should be able to see the wave on an oscilloscope, as shown in Figure 5.2. This measured data was collected using a 12-bit DAC (TLV5618) having a range of 0 to +3 V. The plot on the left was measured with a digital scope (without a speaker attached). The plot on the right shows the frequency response of this data, plotting amplitude (in dB) versus frequency (in kHz). This measured waveform is approximately 1.5+1.5*sin(2π523πt) volts. The two peaks in the spectrum are at DC and 523 Hz (e.g., 20*log(1.5)= 3.5 dB).



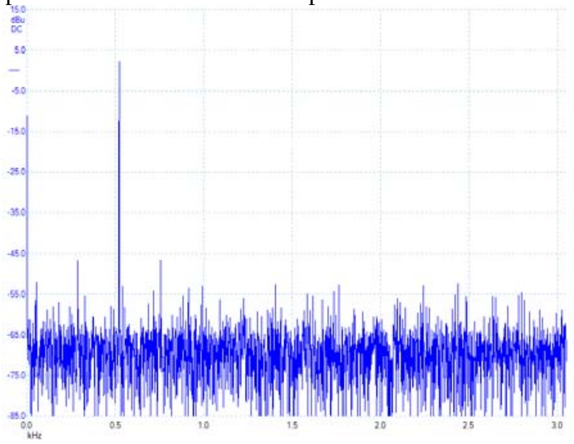
Experimental data of a 32-output 523 Hz sine-wave.



Experimental data of a 256-output 523 Hz sine-wave.



Signal/noise ratio is 31 dB (3dB- -28dB)



Signal/noise ratio is 49 dB (3dB- -46dB).

Figure 5.2. Voltage versus time data on top and the Fourier Transform (frequency spectrum dB versus kHz) of the data on the bottom.

The next step is to convert the DAC analog output to speaker current using a current-amplifying audio amplifier, as shown in Figure 5.3. It doesn’t matter what range the DAC is, as long as there is an approximately linear relationship between the digital data and the speaker current. To do this you will have to run the amplifier in its linear range. The performance score of this lab is not based on loudness, but sound quality. The quality of the music will depend on both hardware and software factors. The precision of the DAC, the linearity of the audio amp, the frequency response of the audio amp and the dynamic range of the speaker are some of the hardware factors. Software factors include the DAC output rate and the complexity of the stored music data. Consider using the MC34119 when designing the audio amp, choosing Rf and Ri so the gain is one or less than one.

The SSI module must be written at a low level, like the book, without calling StellarisWare driver code. Other code (SysTick, Timer, OLED, GPIO, and PLL) can use StellarisWare driver code.

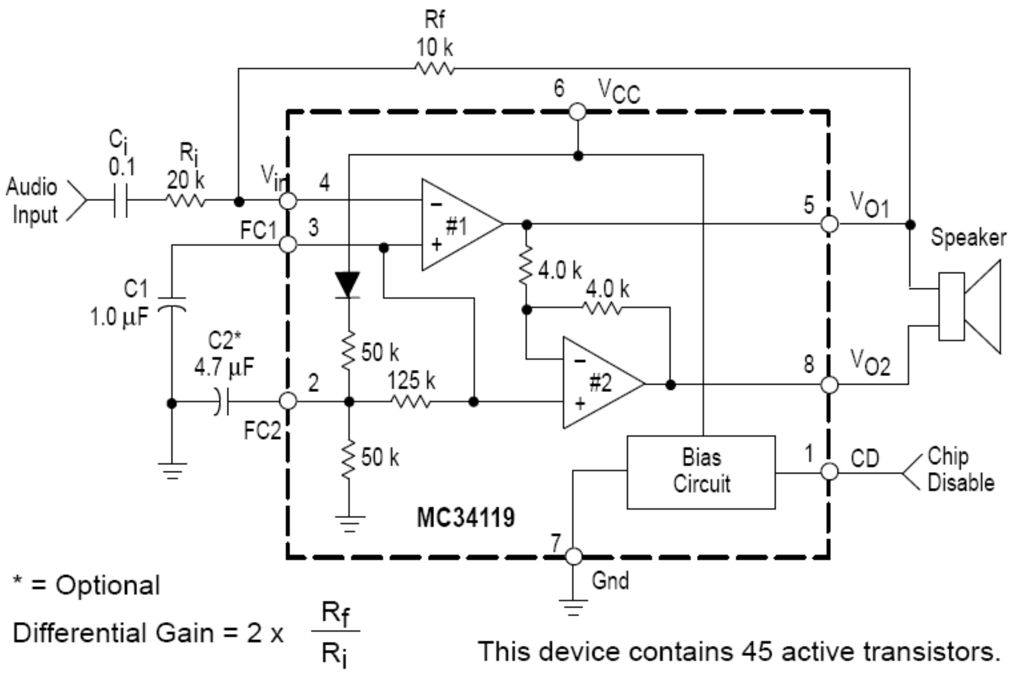


Figure 5.3. The MC34119 is one way to convert DAC voltage into speaker current (ground CD).

If you output a sequence of numbers to the DAC that form a sine-wave, then you will hear a continuous tone on the speaker, as shown in Figures 5.2 and 5.4. The **loudness** of the tone is determined by the amplitude of the wave. The **pitch** is defined as the frequency of the wave. Table 5.1 contains frequency values for the notes in one octave.

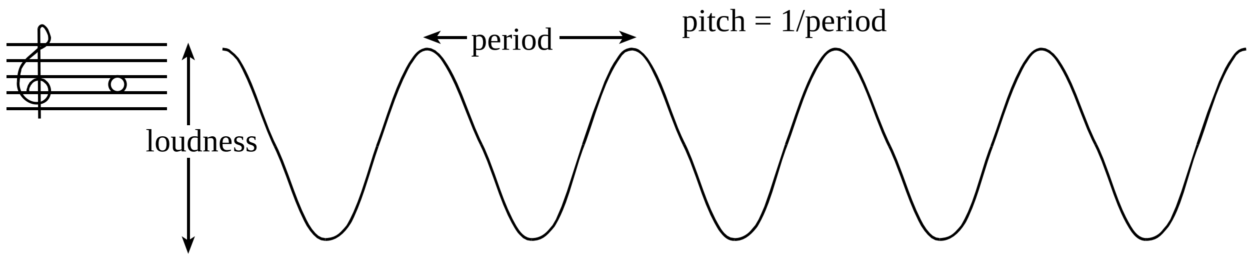


Figure 5.4. A sine-wave generates a pure tone.

Note	frequency
C	523 Hz
B	494 Hz
B^b	466 Hz
A	*440 Hz*
A^b	415 Hz
G	392 Hz
G^b	370 Hz
F	349 Hz
E	330 Hz
E^b	311 Hz
D	294 Hz
D^b	277 Hz
C	262 Hz

Table 5.1. Fundamental frequencies of standard musical notes. The frequency for 'A' is exact.

The frequency of each note can be calculated by multiplying the previous frequency by $\sqrt[12]{2}$. You can use this method to determine the frequencies of additional notes above and below the ones in Table 5.1. There are twelve notes in an octave, therefore moving up one octave doubles the frequency. Figure 5.5 illustrates the concept of **instrument**. You can define the type of sound by the shape of the voltage versus time waveform. Brass instruments have a very large first harmonic frequency.

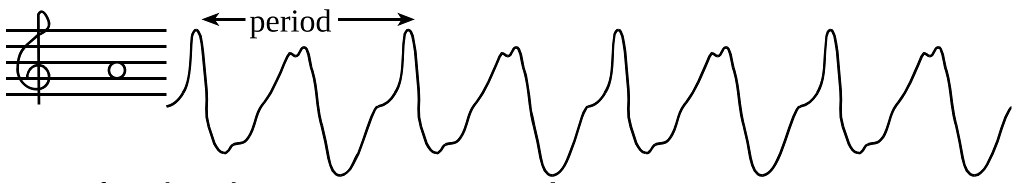


Figure 5.5. A waveform shape that generates a trumpet sound.

The **tempo** of the music defines the speed of the song. In 2/4 3/4 or 4/4 music, a **beat** is defined as a quarter note. A moderate tempo is 120 beats/min, which means a quarter-note has a duration of ½ second. A sequence of notes should be separated by pauses (silences) so that each note is heard separately. The **envelope** of the note defines the amplitude versus time. A very simple envelope is illustrated in Figure 5.6.

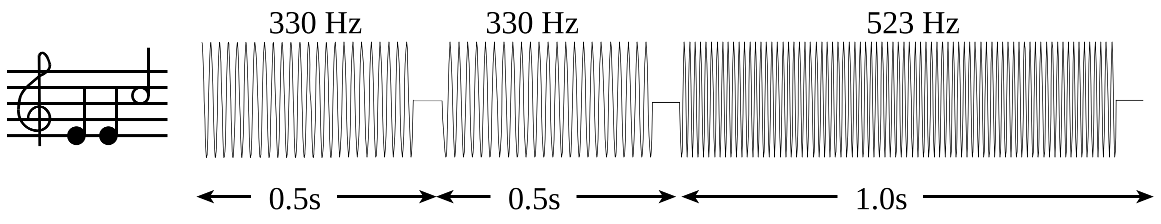


Figure 5.6. You can control the amplitude, frequency and duration of each note (not drawn to scale).

The smooth-shaped envelope, as illustrated in Figure 5.7, causes a less staccato and more melodic sound. The Arm Cortex M3 has plenty of processing power to create these types of waves.

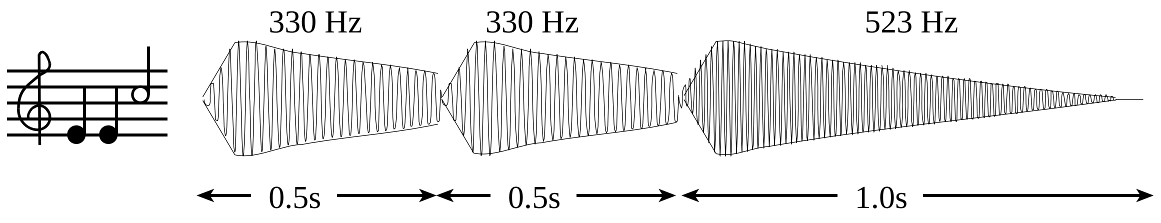


Figure 5.7. The amplitude of a plucked string drops exponentially in time.

A chord is created by playing multiple notes simultaneously. When two piano keys are struck simultaneously both notes are created, and the sounds are mixed arithmetically. You can create the same effect by adding two waves together in software, before sending the wave to the DAC. You can produce this effect by using two interrupts and adding two waves together in software. Figure 5.8 plots the mathematical addition of a 262 Hz (low C) and a 392 Hz sine wave (G), creating a simple chord.

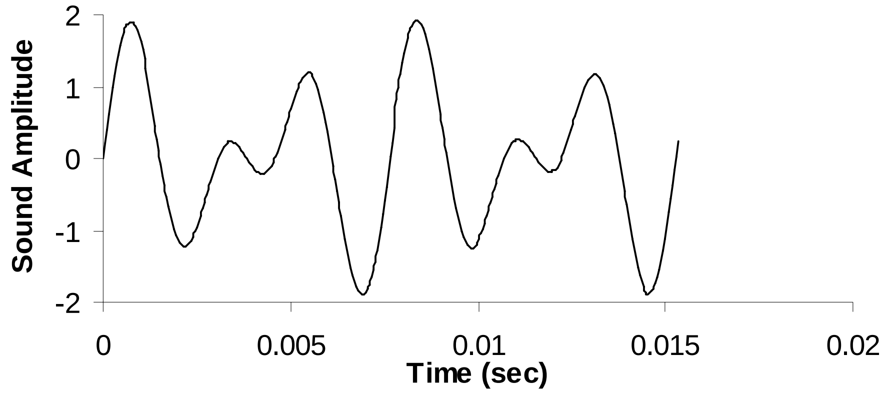


Figure 5.8. A simple chord mixing the notes C and G.

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 LM3S1968 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 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 LM3S1968 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 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 sounds. 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 OLED 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 LM3S1968 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.

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. *(Note to students: you should remove all notes to students in your final requirements document).*

Preparation (do this before your lab period)

1) Copy and paste the requirements document from the lab manual. Edit it to reflect your design.

2) Draw the circuit required to interface the DAC to the LM3S SSI port. Include signal names and pin numbers. The bypass capacitor on the +3.3 V supply of the DAC should be 0.1 µF. Draw the circuit required to interface three push button switches. Design the audio amplifier that runs on the +5V power from the board. Collect all the external parts needed. You do not need to construct the circuit as part of the preparation, but you should draw the circuit and collect all parts.
3) Write a low-level device driver for the SSI interface. Include two functions that implement the SSI/DAC interface. Look very carefully at the four Freescale SPI modes possible. Only one of these four modes matches exactly the shape and polarity of the clock needed by the TLV5618. The function **`DAC_Init()`** initializes the SSI protocol, and the function **`DAC_Out()`** sends a new data value to the DAC. Create separate **`DAC.h`** and **`DAC.c`** files. Write a second low-level device driver for the three switches, creating separate **`Switch.h`** and **`Switch.c`** files.
4) Design and write the music device driver software. Create separate **`Music.h`** and **`Music.c`** files. Place the data structure format definition in the header file. For example, you could implement a **`Music_Play`** function that takes as an input parameter a pointer to a song data structure. Add minimally intrusive debugging instruments to allow you to visualize when interrupts are being processed.
5) Write a main program to run the entire system.
6) Find one or two boxes you can cut up. E.g., a pop tart box and/or small cereal box.

A “syntax-error-free” hardcopy listing for the software is required as preparation. The TA will check off your listing at the beginning of the lab period. You are required to do your editing before lab. The debugging will be done during lab. Document clearly the operation of the routines. Figure 5.9 shows one possible data flow graph of the music player.

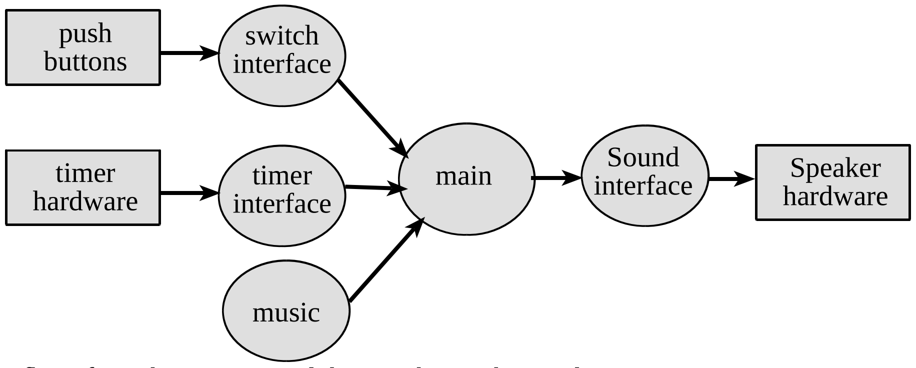


Figure 5.9. Data flows from the memory and the switches to the speaker.

Figure 5.10 shows a possible call graph of the system. Dividing the system into modules allows for concurrent development and eases the reuse of code.

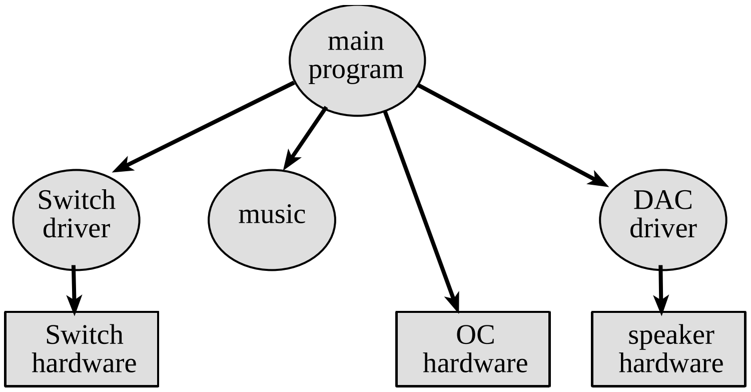


Figure 5.10. A call graph showing the three modules used by the music player.

Procedure (do this during your lab period)
1. Build the SSI/DAC hardware including voltage reference. Use simple main programs to debug the SSI/DAC interface. Experimentally measure the DAC output versus digital input for 8 different digital inputs. Compare the measured data with the expected values. Calculate resolution, range, precision and accuracy of the DAC.

2. Using an oscilloscope and spectrum analyzer, measure the time-domain and frequency-domain outputs from your system at one frequency, like Figure 5.2.
3. Using debugging instruments, measure the maximum time required to execute the periodic interrupt service routines. In particular, create a debugging profile to measure the percentage processor time required to play the song. Adjust the interrupt rate to guarantee no data are lost.
4. Debug the music system. Cut up a box, placing the speaker inside, and notice how much better it sounds.
5. Remove the USB cable and carefully power your system using a lab power supply connected to the +5V line. Set the voltage to +5V and measure the required current to run the system. Take a measurement with and without the music playing. Double check the positive and negative connections before turning it on. If you are at all unsure about this measurement ask your TA for help.

Deliverables (exact components of the lab report)

A) Objectives (final requirements document)
B) Hardware Design
    Detailed circuit diagram of all hardware attached to the LM3S (preparation 2)
C) Software Design (upload your files to Blackboard as instructed by your TA)
    Draw pictures of the data structures used to store the sound data
    If you organized the system different than Figure 5.9 and 5.10, then draw its data flow and call graphs
D) Measurement Data
    Show the data and calculated resolution, range, precision and accuracy (procedure 1)
    Show the experimental response of DAC (procedure 2)
    Show the results of the debugging profile (procedure 3)
    Measurements of current required to run the system, with and without the music playing
E) Analysis and Discussion (give short answers to these questions)
  1) Briefly describe three errors in a DAC.
  2) Calculate the data available and data required intervals in the SSI/DAC interface. Use these calculations to justify your choice of SSI frequency.
  3) How is the frequency range of a spectrum analyzer determined?
  4) Why did we not simply drive the speaker directly from the DAC? I.e., what purpose is the MC34119?

Checkout (show this to the TA)

You should be able to demonstrate the three functions as described in the requirements document. The TA will ask you to connect your DAC output to an oscilloscope and spectrum analyzer, and ask you questions about the frequency spectrum of your output. You should be prepared to discuss alternative approaches and be able to justify your solution.

A software and report files must be uploaded as instructed by your TA.

Do you want your machine to sound better than everyone else's? Extra credit bonus

You may (for a +5% bonus) create multiple sine-waves at the same time. This way, you can play music containing melody and harmony. For this bonus you will use two sine-wave generators and add them together in software; be careful not to overflow and cause clipping. You will need three interrupts: one for outputting the sine-wave for the melody, one for outputting the sine-wave for the harmony, and a third to interpret the music (updating the frequencies and envelopes for the other two.) You will have to add the two sine-waves together in software.

You may (for another +5% bonus) create sine-waves with envelopes similar to Figure 5.5. To get extra credit, these envelopes must have shapes that sound pretty and are independent of pitch. Notice in Figure 5.5 that the decay slope of the envelopes for 330 and 523 Hz are the same. A sinusoidal envelope sound like a violin.

Simple Music Theory	http://library.thinkquest.org/15413/theory/theory.htm
Free sheet music	http://www.8notes.com/piano/
Wav files of instruments	https://ccrma.stanford.edu/~jos/pasp/Sound_Examples.html

Do an internet search on "midi format specification", they may give you ideas on how to encode music digitally.