Lab 6. Digital Piano using a Digital to Analog Converter (Spring 2023)

NEW Modularity in C Programming - The header (.h) file: https://youtu.be/2MLsbyNt8mk

Preparation

Read all of ebook Chapter 6 (skip 6.6) or textbook Chapter 6, http://users.ece.utexas.edu/%7Evalvano/Volume1/IntroToEmbSys/Ch6_DACSound.html

Starter file **PeriodicSysTickInts_123** in EE319Kware,
Lab6_EE319K project is in the original EE319K installer.

Purpose

There are three objectives for this lab:

1. to learn about Digital to Analog Converters (DACs)
2. to understand how digital data stored in a computer could be used to represent sounds and music;
3. to study how the DAC can be used to create sounds.

Programming Requirements

All software for this lab must be written in C. You can debug your code in the simulator but your final code must run on the board with a DAC circuit. Notice the startup.s files are different between C and assembly. The assembly startup.s file simply jumps to your **Start** program. The C startup.s file will initialize all global variable (called premain), and then call to your **main** function. The Lab 6 starter file has the appropriate connections to the Lab 6 simulator/grader. However, you should also look at **SysTickInts.c**

System Requirements

In this lab you will create a very simple sound generation system that illustrates an application of the DAC. Your goal is to create an embedded system that plays four notes, which will be a digital piano with four keys.

- Design a minimum of a 6-bit, binary weighted DAC
- Design a device driver for your DAC
- Interface a four switches to act as synthesizer keys
- Implement a synthesizer with four notes with the switches and device driver
- Connect the DAC output to a speaker or headphones

Procedure

All code written for this lab must be written in C. We have exposed the autograder code for Lab 6. Unless the **Lab6grader.c** has bugs reported by the TAs or professors, please do not edit **Lab6grader.c**.

Part a - Port and Frequency Assignment

Unless your LaunchPad has broken pins, the Lab 6 autograder will select which port pins you will use for the inputs and outputs. Add your two EIDs to the **Lab6.c** file, and run the project with **TExaS_Init(NONE);** to see the exact requirements for you. Figure 6.1 shows one of many possibilities.

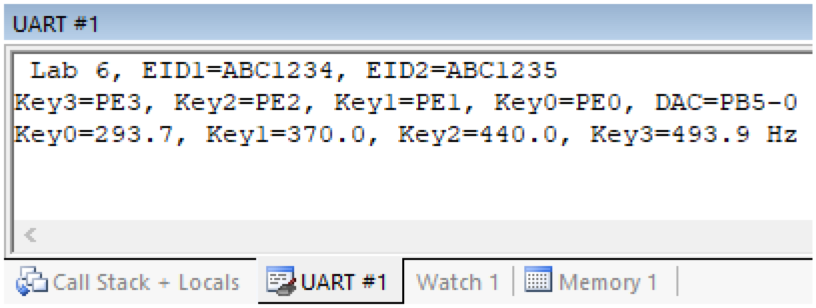


Figure 6.1. The autograder chose Port E for inputs, Port B for outputs, and frequencies D, F#, A and B. Your window will show Spring 2023.

Table 6.1 shows you three possibilities for how you can connect the DAC output. Table 6.2 shows you three possibilities for how you can connect the four positive logic switches that constitute the piano keys. Obviously, you will not connect both inputs and outputs to the same pin. Since we are going to use the DAC in Lab10, everyone will use Port B for DAC. There are no piano keys in Lab10 but sound (DAC) is a required feature.

DAC bit 5	PA7	PB5	PE5
DAC bit 4	PA6	PB4	PE4
DAC bit 3	PA5	PB3	PE3
DAC bit 2	PA4	PB2	PE2
DAC bit 1	PA3	PB1	PE1
DAC bit 0	PA2	PB0	PE0

Table 6.1. Possible ports to interface the DAC outputs.

Piano key 3	PA5	PD3	PE3
Piano key 2	PA4	PD2	PE2
Piano key 1	PA3	PD1	PE1
Piano key 0	PA2	PD0	PE0

Table 6.2. Possible ports to interface the four piano key inputs.

Part b - Design the DAC

Draw the circuit required to interface the binary-weighted DAC to the TM4C123. This DAC should operate using a simple resistor network. A 6-bit binary-weighted DAC uses resistors in a 1/2/4/8/16/32 resistance ratio. Select values in the 0.75 kΩ to 240 kΩ range. For example, you could use 0.75kΩ, 1.5 kΩ, 3 kΩ, 6 kΩ, 12 kΩ, and 24kΩ. Notice that you could create double or half resistance values by placing identical resistors in series or parallel, respectively. It is a good idea to email a pdf file of your design to your TA and have him/her verify your design before you build it.

In simulation, open the DAC window and match the input and output ports with the selection given to you by the autograder. In Figure 6.2, we have selected Port E for input and Port B for output, matching the choices in Figure 6.1 (your ports may be different). Enter the six resistance values for connections to the 6-bit DAC in the DAC window. Figure 6.2 shows a 4-bit DAC, but you will create a 6-bit DAC.

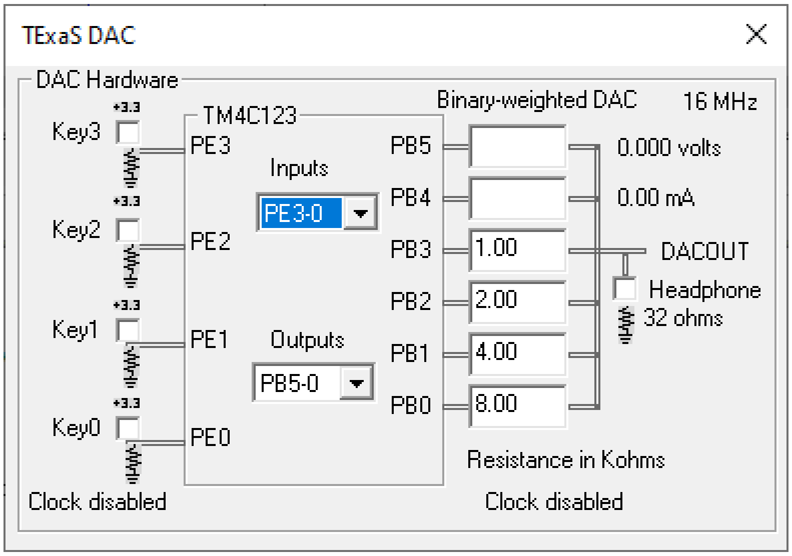


Figure 6.2. Simulation using Port E for inputs, and Port B for outputs.

The default configuration will connect an actual speaker to the output of the DAC, see Figure 6.3. If you would rather use headphones you can get a jack from the TA, Figure 6.4, The 150Ω speaker was included in the lab kit. You will need to solder two wires to the two pads. Please use solid tinned wire. DO NOT TRY TO SOLDER BARE COPPER. Connect one wire to the DACOUT and the other wire to ground.

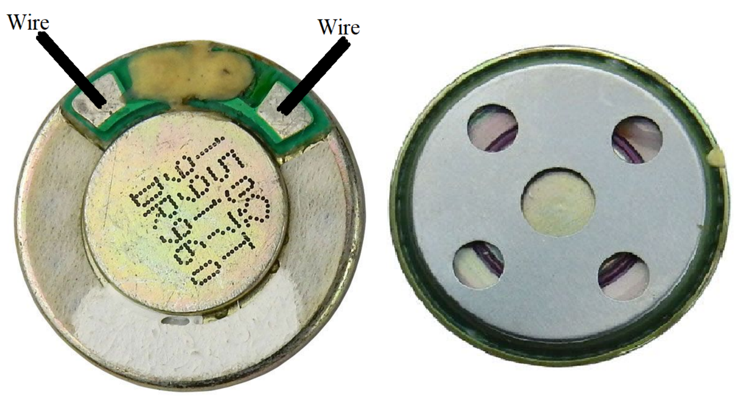
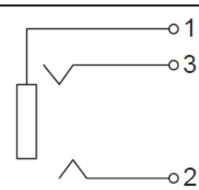


Figure 6.3. The 150-ohm speaker has two electrical connections.

A short soldering guide can be found at http://users.ece.utexas.edu/~valvano/Volume1/SolderingGuide.pdf

A second option is plug the audio jack into the solderless breadboard, see Figure 6.4. Plug your headphones into your audio jack and use your ohmmeter to determine which two wires to connect. If you wish to connect headphones, ask your TA for an audio jack. You have the option of connecting just the left, just the right, or both channels of the headphones.

Model No.	SJ1-3543N
Schematic	
PIN	
1	sleeve
2	tip
3	ring

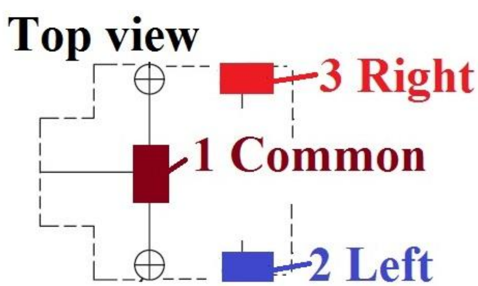


Figure 6.4. A stereo jack has three connections. Channel connect pin 1 (sleeve) to ground and connect the DAC output to pin 2 or 3. You could also connect the DAC to both pins 2 and 3.

Note: all of the testing and grading will be performed without connecting the speaker to the DAC. You will connect the speaker only when you wish to create sound.

Part c - Write the DAC Driver and perform a static test

PLEASE perform this test first in simulation, then on the board before trying to run interrupts or the grader.

Implement a device driver for your binary weighted DAC. Include at least two functions that implement the DAC interface. For example, you could implement the function **DAC_Init** to initialize the DAC, and the function **DAC_Out** to send new data value to the DAC. Place all code that accesses the DAC in a separate **DAC.c** file. Add a **DAC.h** header file with the prototypes for public functions. Describe how to use the functions defined by the module in the comments of the header file. We suggest you set **GPIO_PORTB_DR8R_R |= 0x3F;** so the TM4C123 can drive more current into the resistor DAC.

Measure the range, precision, resolution, and accuracy of your DAC's analog output by connecting it to your voltmeter or oscilloscope. This requires you to run a simple main program to test the DAC. There are two options depending on whether or not you have access to a voltmeter.. You are free to debug this system however you wish, but you must debug the DAC module separately before continuing to the subsequent steps. If you have a voltmeter, attach it to DACOUT, and run the first main, and placing a breakpoint after **DAC_Out**. You do not have a voltmeter, attach PD3 to DACOUT, and implement the LaunchPad functions in the second main. Observe the 16 possibilities, comparing digital **`Data`** to the analog voltage at the V_{out} without the speaker attached.

Using Ohm's Law and fact that the digital output voltages will be approximately 0 and 3.3 V, make a table of the theoretical DAC voltage and as a function of digital value (without the speaker attached). Calculate resolution, range, precision and accuracy. Complete Table 6.3 and attach it as a deliverable. See **DACdatastatic.xls** sheet in Lab6 on your computer

Digital Bits 5-0	Theoretical DAC voltage	Measured DAC voltage
0		
1		
7		
8		
15		
16		
17		
18		
31		
32		
33		
47		
48		
49		
62		
63		

Table 6.3. Static performance evaluation of the DAC.

Part d - Dynamic DAC Test

Using SysTick interrupts, write code that outputs a sine wave at one of the frequencies given to you by the autograder. At this pointer there are no inputs, and this sine wave should occur continuously. In simulation, you can observe the DACOUT using the logic analyzer, with **DisplayType=analog** and **Max=3.3**, as shown in Figure 6.5. Figure 6.6 shows the output in simulation and on the real board. To see the output on the real board, connect PD3 to DACOUT and initialize TExaS in scope mode: **TExaS_Init(SCOPE);**

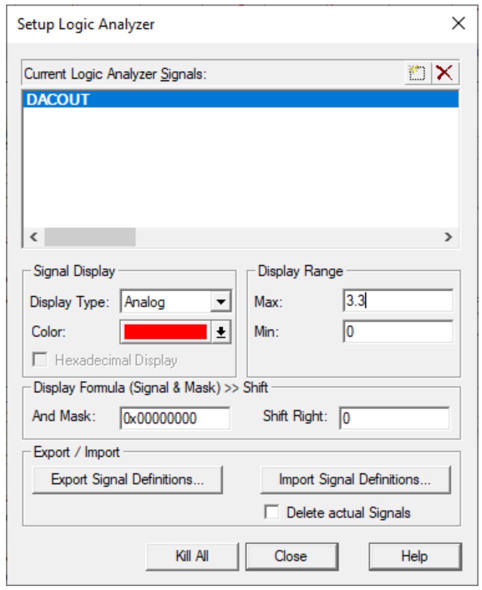


Figure 6.5. The logic analyzer in simulation mode can actually be configured as an oscilloscope.

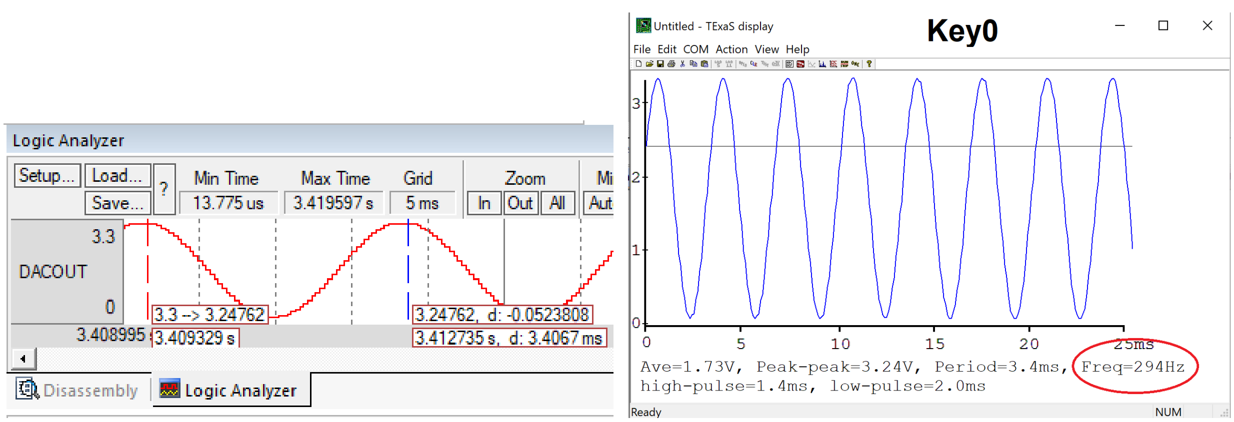


Figure 6.6. Sine wave with frequency 293.7 Hz (period = 3.405 ms) in simulation and on real board (without speaker). The black horizontal line at 2.5V is the scope trigger.

If you connect the headphones you will hear the sound, but the amplitude will be reduced. Figure 6.7 was created with a six-bit DAC and a 150 ohm speaker.

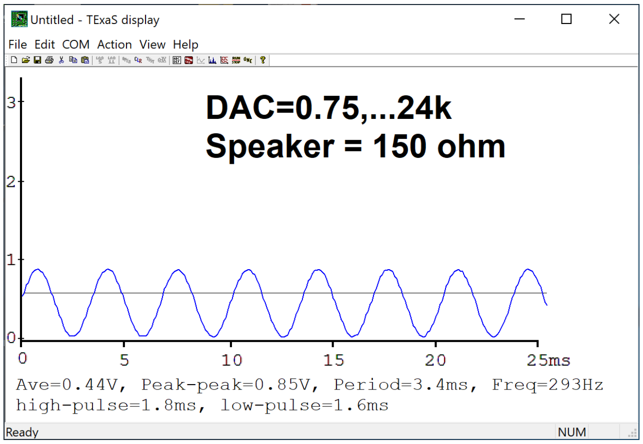


Figure 6.7. TExaSdisplay scope trace with speaker attached. Notice, I moved the trigger threshold (black horizontal line) down to 0.6V so the scope could measure frequency.

Part e - Write Keyboard Driver

Draw interface circuits for four switches, which will be used for the piano keyboard. The circuit with both switches and DAC will be included in the deliverables.

Design and write the piano keyboard device driver software. These routines facilitate the use of the four piano keys. Include at least two functions that implement the piano keyboard interface. For example, you could implement the function **Key_Init** to initialize the switch inputs, and the function **Key_In** that returns a logical key code for the pattern of switches that are pressed. Place all code that directly accesses the switches in a separate **Key.c** code file. Add a **Key.h** header file with the prototypes for public functions. Add comments that describe what it does in the **Key.h** file and how it works in the **Key.c** file.

Part f - Write Sound Device Driver

Design and write the sound device driver software. The input to the sound driver is the pitch of the note to play. SysTick interrupts will be used to set the time in between outputs to the DAC. Add minimally intrusive debugging instruments to allow you to visualize when interrupts are being processed. Include at least two functions that implement the sound output. For example, you could implement the function **Sound_Init** to initialize the data structures, calls **DAC_Init**, and initializes the SysTick interrupt. You could implement a function **Sound_Start(note)** that starts sound output at the specified pitch. Place all code that implements the waveform generation in a separate **Sound.c** code file. Add a **Sound.h** header file with the prototypes for public functions. Add comments that describe what it does in the **Sound.h** file and how it works in the **Sound.c** file.

When you wish to play a new note you could write to `NVIC_ST_RELOAD_R`, changing the interrupt period, without writing to CURRENT or CTRL. In other words, you do not wish to restart SysTick when changing periods.

To see the output on the real board, connect PD3 to DACOUT and initialize TExaS in scope mode: **TExaS_Init(SCOPE);** Figure 6.8 shows the testing of four frequencies D, F#, A and B (293.7, 370.0, 440.0, and 493.9 Hz respectively). Your frequencies will be chosen for you by the autograder.

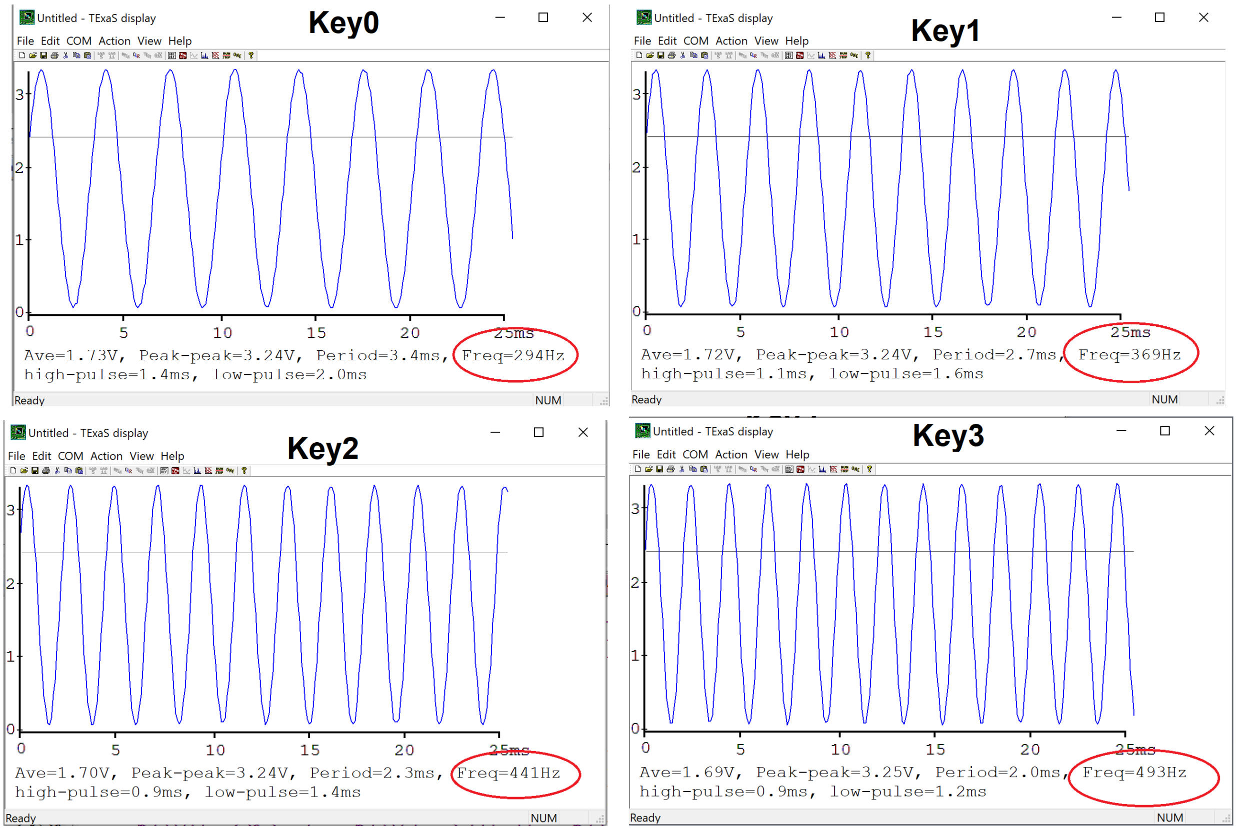
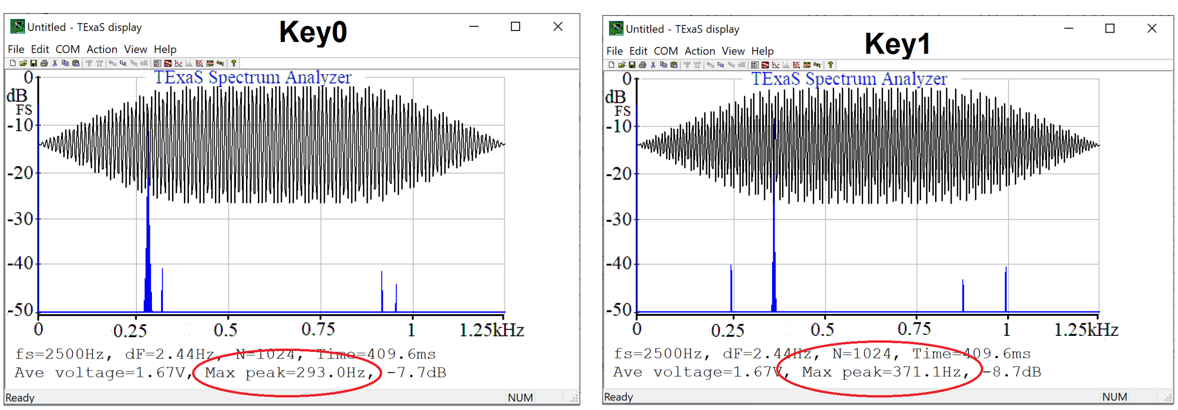


Figure 6.8. The 6-bit DAC with a 64-element table creates four different frequencies without speaker.

A scope measures voltage versus time, but a spectrum analyzer measures voltage (in dB) versus frequency. To see your DAC output in the frequency domain, set the view in TExaS to spectrum analyzer. Figure 6.9 shows the same four frequencies D, F#, A and B. Observing the signals in the frequency domain is an optional, but fun activity.



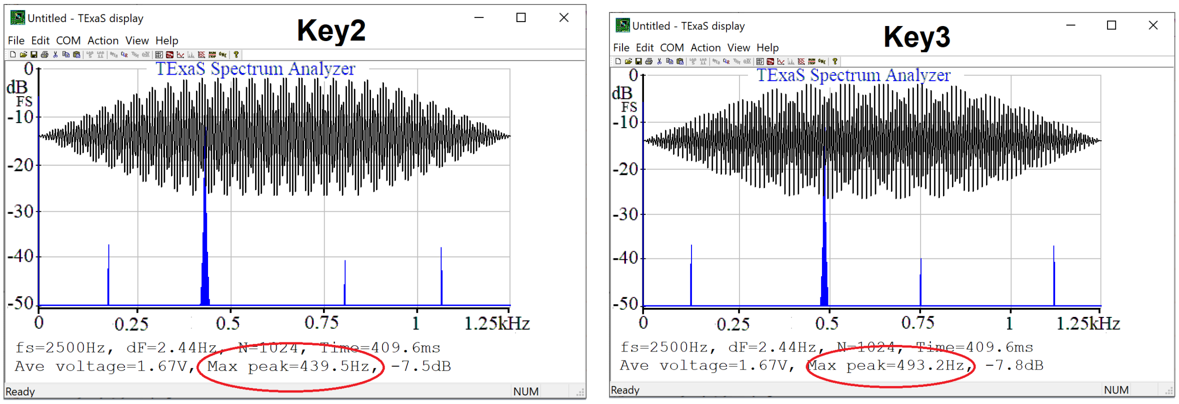


Figure 6.9 Four sine wave plotted in the frequency domain. The trace is the DAC output (without speaker).

Part g - Write Main Program

Write a main program to implement the four-key piano. Make a heartbeat connected to an LED (use PortF) so you can see that your program is running. Document clearly the operation of the routines. Figure 6.10 shows a possible data flow graph of the music player. Debug the system first in the simulator then on the real TM4C123 with the TExaS oscilloscope. Take a screenshot of a scope trace (like Figure 6.8) to capture the waveform generated by your digital piano. When no buttons are pressed, the output will be quiet. When switch 1 is pressed, output a sine wave at one frequency. When switch 2 is pressed, output a sine wave at a second frequency. When switch 3 is pressed, output a sine wave at a third frequency. When switch 4 is pressed, output a sine wave at a fourth frequency. Only one button will be pressed at a time. The sound lasts until the button is released.

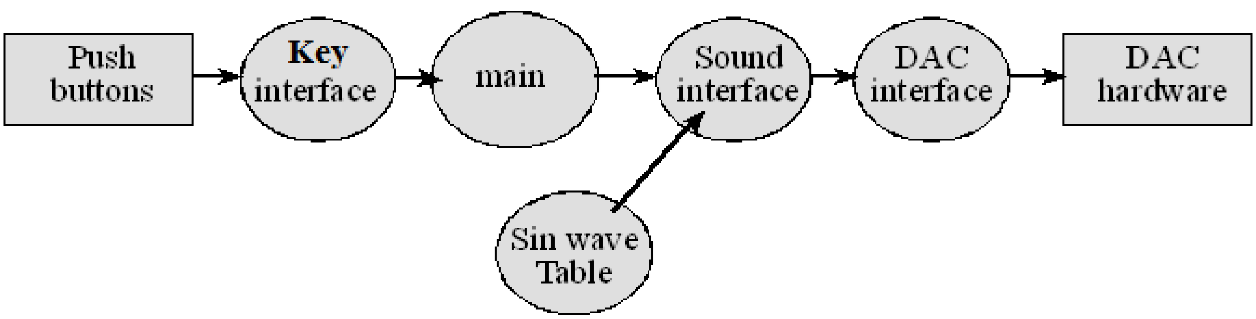


Figure 6.10. Data flows from the memory and the switches to the speaker.

Figure 6.11 is one possible call graph for Lab 6. Dividing the system into modules allows for concurrent development and eases the reuse of code. *The two modules used by the extra credit part of Lab 6 are shown in orange. Notice there are two interrupts if you do the extra credit.*

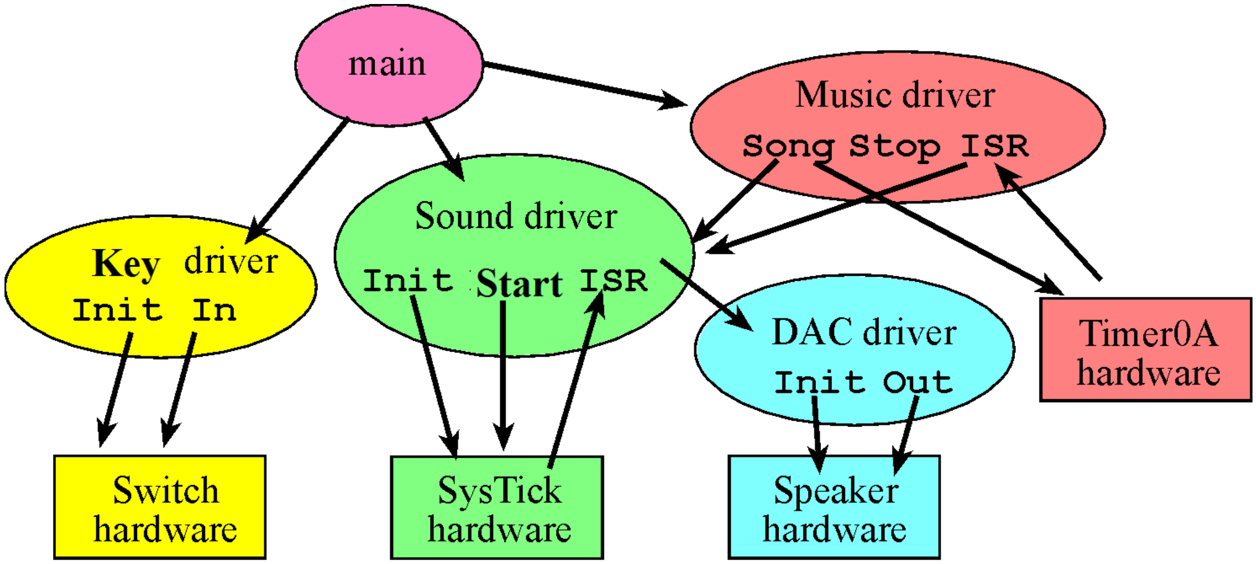


Figure 6.11. Possible call graph of the Lab 6 system. "Music driver" will be Song.c for extra credit 1 or "wave.c" for extra credit 2). timer0, timer1, or timer2 can be used for extra credit (the grader uses Timer3, TExaS uses Timer5)

You must disconnect the speaker to run the grader. Notice the voltage is Figure 6.7 (with speaker) is very small compared to the voltage in Figure 6.8 (without speaker).

Demonstration

There are grading sheets for every lab so you know exactly how you will be evaluated. **(both partners must be present, and demonstration grades for partners may be different)**

Do all these well in advance of your checkout

1. **Signup for a time with a TA. If you have a partner, then both must be present**
2. **Upload your software to canvas, make sure your names are on all your software**
3. **Upload your one pdf with deliverables to Canvas**

Do all these during the TA checkout meeting

1. **Have your one pdf with deliverables open on your computer so it can be shared**
2. **Have Keil Lab 6 open so TA can ask about your code**
3. **Start promptly, because we are on a schedule. If you have a partner, then both must be present**
4. **Demonstrate lab to TA**
5. **Answer questions from TA to determine your understanding**
6. **TA tells you your score (later the TA will upload scores to Canvas)**

You should be able to demonstrate the four notes. Be prepared to explain how your software works. You should be prepared to discuss alternative approaches and be able to justify your solution. The TA may look at your data and expect you to understand how the data was collected and how DAC works. In particular, you should be able to design a DAC with 3, 4, 5 or 7 bits using the binary weighted approach. What is the range, resolution and precision? You will tell the TA what frequency you are trying to generate, and they may check the accuracy with a frequency meter or scope. TAs may ask you what frequency it is supposed to be, and then ask you to prove it using calculations. Just having four different sounding waves is not enough, you must demonstrate the frequency is proper and it is a sinewave (at least as good as you can get with a 6-bit DAC). You will be asked to attach your DAC output to the scope. Many students come for their checkout with systems that did not operate properly. You may be asked SysTick interrupt and DAC questions. If the desired frequency is **f**, and there are **n** samples in the sine wave table, what SysTick interrupt period would you use?

This lab mentions 64 samples per cycle. Increasing the DAC output rate and the number of points in the table is one way of smoothing out the "steps" that in the DAC output waveform. If we double the number of samples from 64 to 128 to 256 and so on, keeping the DAC precision at 6-bit, will we keep getting a corresponding increase in quality of the DAC output waveform?

As you increase the number of bits in the DAC you expect an increase in the quality of the output waveform. If we increase the number of bits in the DAC from 6 to 8, and keep the number of points in the table fixed at 64, there will be no increase in quality of the DAC output waveform. If you increase the number of bits in the DAC you must also increase the size of the table.

Deliverables

Combine all your C files for this lab into one text file called Lab6.c, and upload this Lab6.c file to Canvas. Combine the following components into one pdf file and upload this file also to Canvas. UPLOAD ONLY ONE COPY PER TEAM (names on both). Have the pdf file and Keil open on the computer during demonstration

1. Your names, professors, and EIDs.
2. Circuit diagram showing the DAC and four switches, using any drawing method
3. Software Design
    a. Draw pictures of the data structures used to store the sound data
    b. If you organized the system different than Figure 6.10 and 6.11, then draw its data flow and call graphs
4. A picture of the output on TExaS scope (like once trace Figure 6.8)
5. TExaSdisplay window showing real board grading
6. Measurement Data
    a. Show the theoretical response of DAC voltage versus digital value (Table 6.3)
    b. Show the experimental response of DAC voltage versus digital value (Table 6.3)
    c. Calculate resolution, range, precision and accuracy
7. Brief, one sentence answers to the following questions
    a. When does the interrupt trigger occur here in Lab 6?
    b. In which file is the interrupt vector?
    c. List the steps that occur after the trigger occurs and before the processor executes the handler.
    d. Previously we learned that the `BX LR` instruction simply moves LR into PC; it does not do this for interrupts, rather it returns from interrupt; how does `BX LR` return from interrupt?

Optional Feedback : http://goo.gl/forms/rBsP9NTxSy

Extra Credit Option 1 10 points (similar to a MIDI file)

Extend the system so that it plays your favorite song (a sequence of notes, set at a specific tempo and includes an envelope. An envelope is the changing of amplitude of the sound wave over time, you can use these as references: http://www.eetimes.com/document.asp?doc_id=1279248, http://www.teachmeaudio.com/recording/sound-reproduction/sound-envelopes/,

The song should contain at least 5 different pitches and at least 20 notes to earn the full 10 points. But, what really matters is the organization is well-done using appropriate data structures that are easy to understand and easy to adapt for other songs.

This extra credit provides for up to 10 additional points, allowing for a score of 110 out of 100 for this lab. Extra credit is for playing a song. To earn the credit you must use more than one interrupt. A fast SysTick ISR outputs the sinewave to the DAC. The rate of this interrupt is set to specify the frequency (pitch) of the sound. A second slow Timer ISR occurs at the tempo of the music. For example, if the song has just quarter notes at 120, then this interrupt occurs every 500 ms. If the song has eight notes, quarter notes and half notes, then this interrupt occurs at 250, 500, 1000 ms respectively. During this second ISR, the frequency of the first ISR is modified according to the note that is to be played next.   To earn your credit, there must be at least 3 distinct notes in the song.

Compressed data occupies less storage, but requires runtime calculations to decompress. On the other hand, a complete list of points will be simpler to process, but requires more storage than is available on the TM4C123. Compression is not required for this lab or the extra credit.

Although you will be playing only one song, the song data itself will be stored as a data structure in flash ROM, and the device driver will perform all the I/O and interrupts to make it happen. You will need public functions `Song_Play` and `Song_Stop`. The `Song_Play` function could have an input parameter that defines the song to play. You only need to implement one song.

If you complete the extra credit (with input switches that can be used to start and stop), then the four-key piano functionality still must be completed. In other words, the extra credit part is in addition to the regular part. You could interface more switches (which you can get from checkout) or you could use the on-board switches to activate the `Music_PlaySong` and `Music_StopSong` operations.

Option 1 extra credit will not be useful for Lab 10.

Extra Credit Option 2 10 points (similar to wav file)

Option 2 extra credit will be to implement one of the required components of Lab 10. Sounds are represented by a finite sequence of DAC output values. The sound is created by outputting the digital values at a fixed rate, see **wave.h** and **wave.c**. Convert the digital numbers to 6-bit integers and output the numbers at that same fixed rate. Full credit requires four sounds, but you can use the SpaceInvader sounds in **wave.c**. If you wish to create addition sounds [see ebook section 10.4]

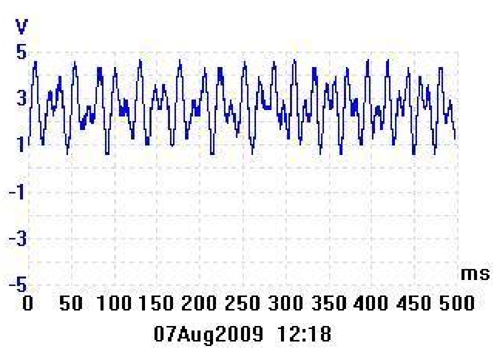
See [http://users.ece.utexas.edu/~valvano/Volume1/WavConv.m]

You can only do Option 1 or Option 2, not both

Some web links about music

Music Theory   [https://en.wikipedia.org/wiki/Music_theory]

Free sheet music   [http://www.8notes.com/piano/]



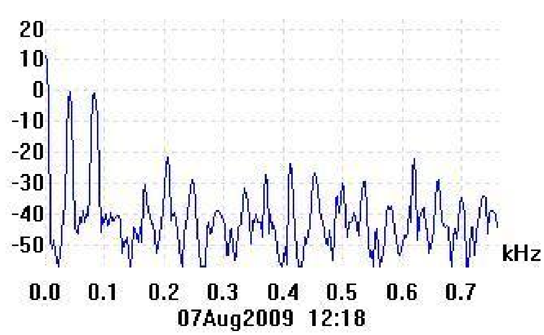


Figure 6.12. A song being played with a harmony and melody (see two peaks in the spectrum).

Voices

One of the optional activities is to create other voices. The sine wave is a pure tone, and it is required to pass the autograder. However, other sound shapes are more pleasant to the ear. Figure 6.13 shows four different voices all at the same fundamental pitch. Figure 6.14 shows the horn voice measured with a real oscilloscope.

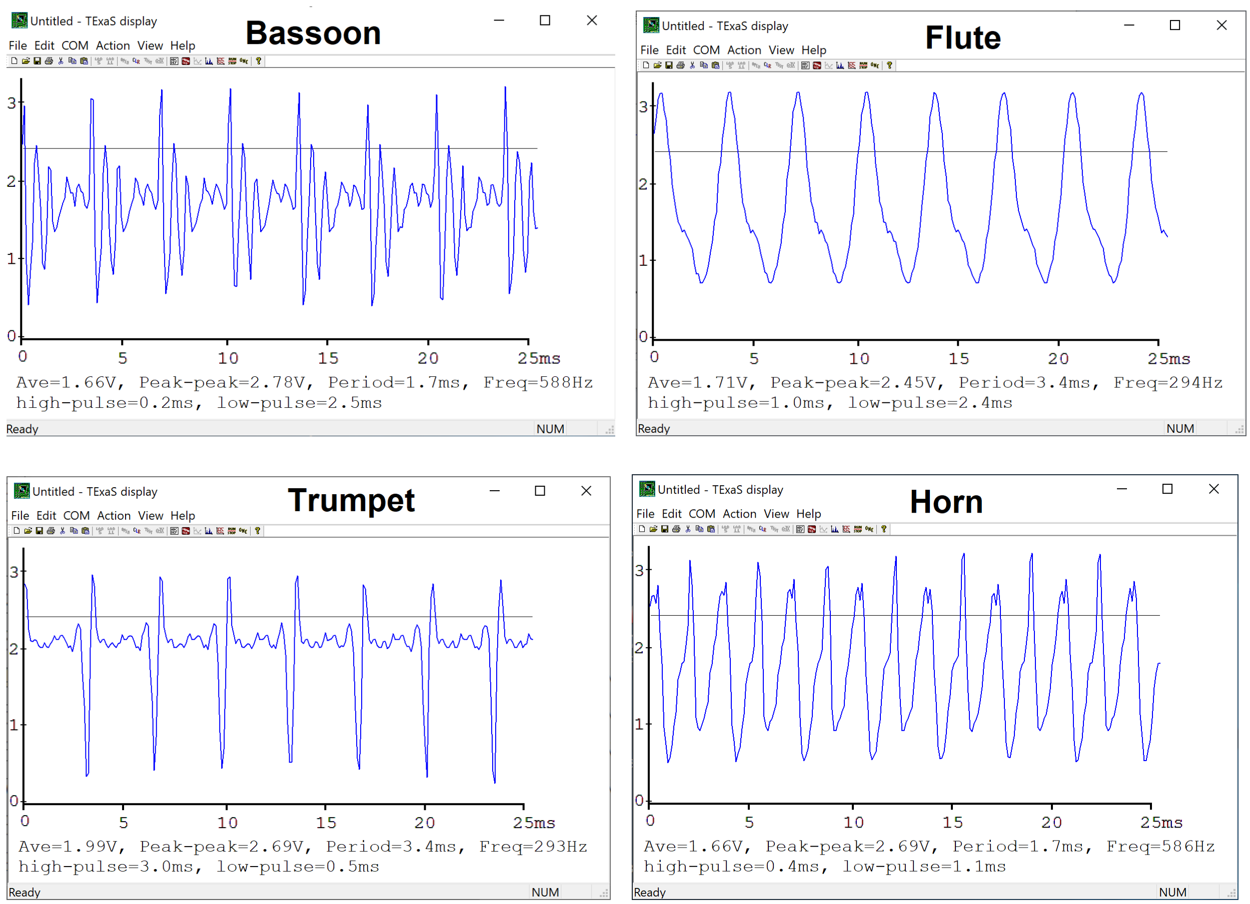


Figure 6.13. These voices were created with a 6-bit DAC with a 64-element tables

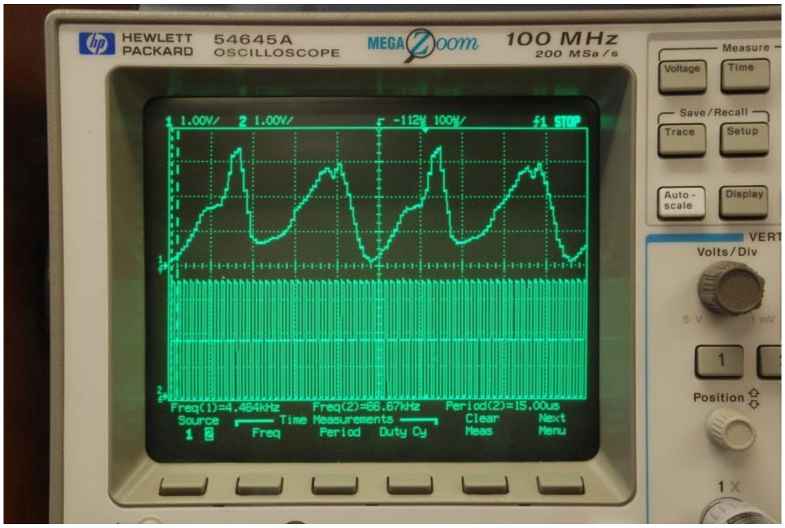


Figure 6.14. This waveform sounds like a horn (6-bit DAC, 64-element table). The top trace is the 6-bit DAC output (without headphones) and the bottom trace is a debugging monitor that toggles at each SysTick interrupt.

Multithreading

There should be no calls to **`DAC_Out`** in the main loop. There should be zero or one call to **`DAC_Out`** each execution of the ISR. Some students will discover they can make sounds without interrupts, by simply using a software delay like Labs 4, and 5. This approach is not allowed. Other students will discover they can create sounds by calling **`DAC_Out`** in the main loop. This approach is also not allowed. 20 points will be deducted if zero or one calls to **`DAC_Out`** do not occur in the ISR. These other approaches are not allowed because we will be employing sounds in Lab 10, and we will need the sound to completely run within the ISR. The Lab 10 main program will start and stop sounds, but the ISR will do all the **`DAC_Out`** calls. This lab implements **multithreading** with two threads. The main program (foreground thread) reads from the piano keyboard, and the SysTick ISR (background thread) produces the sound.

FAQ: Frequently Asked Questions

1. **The deliverables say to have pictures of the data structures. Does it mean data flow graphs/flow charts?**

   No, it's talking about diagrams of the structs or arrays you define in C. You should include all the members of your struct in the diagram.

2. **What is the purpose of the SysTick_Handler? Is it called automatically every time the CURRENT_R reaches 0?**

   Yes, as long as you initialized SysTick and enabled the SysTick interrupt.

3. **What is the purpose of the Key_Init function and the Sound_Init functions ? What does the implementation for these functions do ?**

   Those functions will essentially setup your inputs and outputs. For example, Key_Init will configure the four pins your external switches are connected to as inputs and enable them. If you have any data structures or arrays, you can also set their initial values in your init functions.

4. **How do I export variables in C? (C file calling another C file) More specifically, I am using bit specific addressing and want my dac.c file to be able to use the variable I defined in lab6.c**

   I'm guessing the variable you want to share is a global variable? In that case, you'll want to use the "extern" declaration. You said you already have a definition for your variable in lab6.c. Keep that there as it is, but in dac.c, declare the same variable again, except this time with the "extern" keyword in front of it. This tells the compiler that when it's compiling dac.c, it should not allocate space for the variable, because the space has already been allocated in another C file. To share a #define constant between two files, you can declare it in a header file, and then #include that header file in both C files. Remember that #define is a preprocessor directive, so the constants get substituted before even compiling starts. This means that no space in the memory is needed for these constants. Global variables are different, because they are stored in memory, and can change at runtime.

5. **What is the difference between the Sound_Start() and DAC_Out() functions?**

   DAC_Out() interacts with the 6 DAC pins to simply output the 6-bit "int." Sound_Start(), on the other hand, will actually start playing a note. Playing a note involves moving the DAC_Out() values according to a sinusoidal curve. Your DAC_Out() and Sound_Start() subroutines don't necessarily return anything to your main code, but rather perform tasks on their own based in inputs you give them. For DAC_Out() think about what you need to pass to the portB pins (high or low) and how it changes with time. For Sound_Start() think about what you need to change about systick to create the note you want and how it changes with the keys pressed.

6. **We are currently in the software debugging process and for some reason on the peripheral it is not displaying the PB5-PB0 changing as we traverse the sin wave and write new values to PortB. We have initialized portb and set the 6 bits to output. What would you suggest we do to figure out the problem?**

   Keep in mind that you are outputting voltage from portB to your DAC, your headphones are receiving the signal put out by your controller. The peripheral will not show port B bits, but in the blank spaces it requires you to input your resistance values (e.g. write "0.75" in PB5 if that is the resistor you are outputting to (0.75k)). The peripheral will now show you the voltage change that is received at the audio jack.

7. **Why does Sound_Start(uint32_t period) have a parameter? What exactly is Sound_Start() supposed to do?**

   As the lab manual says, "Sound_Start(note) starts sound output at the specified pitch." The parameter is to specify the period of the note you want to play, allowing you to play different types of notes. If the period in order to play the note A was *x*, then Sound_Start(*x*) should start playing that note. Remember the actual DAC output occurs in the ISR.

8. **What's the DR8R register?**

   That is the 8-mA drive control register. With a DR8R bit set, the processor will be able to drive up to 8mA out of an output pin if you set the pin high. With a DR8R bit set, the processor will be able sink up to 8 mA into the processor when the output is low. IT DOES NOT CAUSE THE CURRENT TO BE 8 mA. The voltage is 3.3V on high and 0.0V on low, and the current is determined by the resistance of the circuit attached to the pin. When the headphone is attached bit 5 when it is high will be about 3.3V/0.75k or about 4 mA.

9. **Is it necessary that we use all of the different subroutines that have been defined in the starter file? I know it's for the sake of abstraction, so that anyone should be able to look at the main program and understand what the inputs should mean, and what the output**

should be as a result. However, it's rather difficult to keep track of where variables have been defined (e.g., globally, locally within the function, or locally elsewhere). It would be significantly easier to have only a few subroutines rather than 6 or so different files that use much of the same variables.

The lab manual specifies separating each component into a separate file with at least two functions for key and DAC, so I would stick close to that; however, you can change around the function names/implementations a bit as long as it makes sense to you. It can definitely get confusing, especially the first time, but that's why it's good to start practicing modular software development now.

10. **Will the SysTick timer begin counting down and rolling-over from the moment it is initialized?**

    When you set the enable bits within NVIC_ST_CTRL_R, the counter will begin to decrement and roll over as needed. However if you have disabled interrupts (EnableInterrupts(), DisableInterrupts()), then the SysTick_Handler will not be called until you re-enable the interrupts.

11. **My partner and I are having some issues with getting the simulator (particularly the DAC simulation to update when we change any of the relevant data for each of the ports. When I look at the status of the data registers in the system viewer, the values are accurate, but the simulator doesn't reflect those values. The simulator for Port F, works, however.**

    Don't forget to add the resistor values for your DAC in the simulator window (blank spots for PB0-5). The voltage should change after that. When you plug in headphones (the check box), you should see some voltage drop and current flow.

12. **It seems like the .h files for the Sound, Key, and DAC modules are already provided to us in the Lab6 folder. Do we need to add anything to the .h files besides comments as to what each function does and its hardware connections?**

    No. The .h files contains only the declaration of your functions, and the .c files contains the definition (the actual code for the function). All you need to do is complete the function definition in your .c files. Although, if you'd like to add your own functions to your program, you will need to include declarations for these in the header files so that they can be accessed from other files.

13. **What is meant by "the function Sound_Init [initializes] the data structures?" Is this just referring to the Sine table?**

    The Sound_Init function initializes the SysTick counter by enabling interrupt bit. By data structure, we are referring to the sine array and you should initialize the index of that array to zero. You should figure out in which function you will increment that index.

14. **For the heartbeat in the main program: Can we just use one of the LED's on the microcontroller, or do we have to interface an external LED?**

    You can use the onboard LED.

15. **What exactly is the implementation for the DAC_Out function supposed to do? Doesn't the DAC circuit convert the signals from digital to analog anyway ?**

    DAC_Out should take the input data passed as argument and write it to PORTB. Then your circuit converts the 4 (or 6) digital outputs at the pins of PORTB to an analog voltage.

16. **Is this SysTick periodic interrupts or edge triggered interrupts?**

In this lab we use SysTick and not edge-triggered interrupts. The SysTick interrupt is an example of a periodic interrupt. An example of an edge triggered would be handling a button press on a GPIO pin.

17. **I'm trying to understand what resolution means, I understand that precision is the number of levels or entries, and the more we have the smoother the wave is. Range is the max/min of the voltages, so if we divide precision by the range, what exactly are we getting?**

    Range= precision * resolution. The **resolution** is the smallest change in value that is significant. You can get more from the e-book. How much does the analog output change if the digital value increases by 1,

18. **So my hardware doesn't seem to work at all but I have gone through it multiple times and it seems to be wired correctly with the correct resistor values. I'm assuming the code works since it's working in the simulator. Any ideas to why my lab isn't working?**

    That's what most people have trouble with. Use a multimeter and oscilloscope to see if the hardware is receiving/outputting the correct things. There's no real short cut for this. Single-step Program 6.1 and look at the DAC digital voltages and the analog output.

If you wish to make loud sounds on a speaker, you will need an audio AMP like the MC34119. This is not required. This is not extra credit. It is included here just for fun.

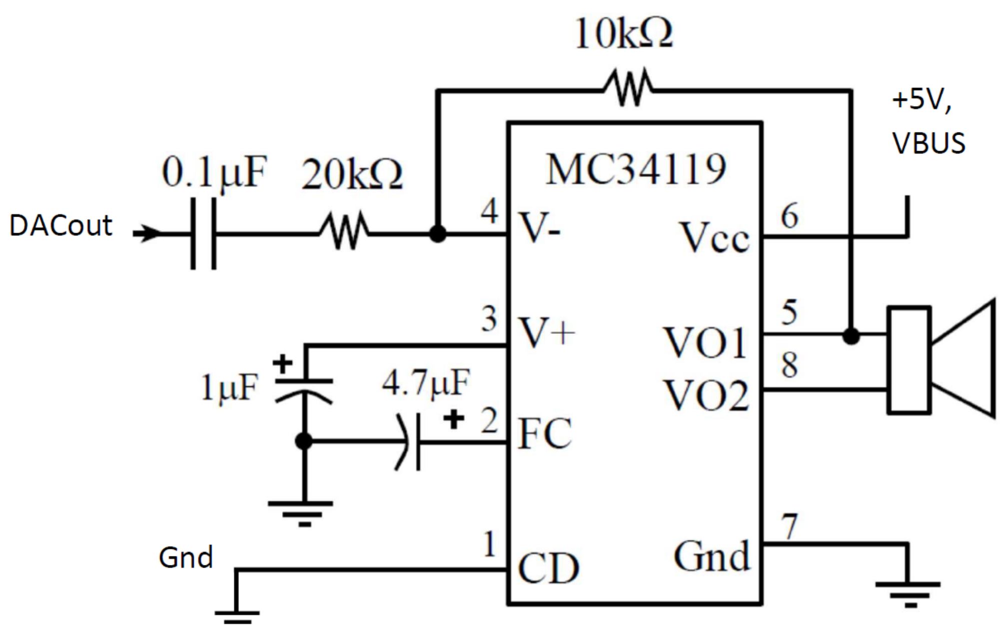


Figure 6.15. Audio amplifier needed to drive a speaker. 0.1uF is ceramic; 1uF and 4.7uF are tantalum