

Lab 6 - Digital Piano using a Digital-to-Analog Converter (DAC)

# Preparation

* Bring headphones
* Read Sections 9.1, 9.2, 9.4, 9.6, 10.1, 10.2 and 10.3
* Go through Lecture 18 again
* Download and load into CCS “EGCP\_450\_Lab\_6.zip”
* <http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C13_Interactives.htm>
* [Watch the related video(s) here](https://www.youtube.com/playlist?list=PLxYT_0WQMg35FiE5EDnCwW-1p7-E3gNtJ)

# Purpose

There are 3 objectives for this lab: 1) to learn about digital-to-analog conversion, 2) to understand how digital data stored in a computer could be used to represent sounds and music, and 3) to study how the Digital-to-Analog Converter (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.

# System Requirements

## Step 1

Most digital music devices rely on high-speed DAC converters to create the analog waveforms required to produce high-quality sound. In this lab, you will create a very simple sound generation system that illustrates this application of the DAC. Your goal is to create an embedded system that plays three notes, which will be a digital piano with three keys. The first step is to design and test a 4-bit binary-weighted DAC, which converts 4-bits of digital output from the MSP432 to an analog signal. You will convert the four digital bits to an analog output using a simple resistor network. During the static testing phase, you will connect the DAC analog output to your voltmeter and measure resolution, range, precision and accuracy (Figure 1, left). During the dynamic testing phase you will connect the DAC output to headphones, and listen to sounds created by your software (Figure 1, right). It doesn’t matter what range the DAC is, as long as there is an approximate linear relationship between the digital data and the speaker current. 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, external noise, and the dynamic range of the speaker are some of the hardware factors. Software factors include the DAC output rate and the number of data points stored in the sound data. You can create a 3k resistor from two 1.5k resistors. You can create a 6k resistor from two 12k resistors.



Figure 1: DAC allows the software to create music.

## Step 2

The second step is to design a low-level device driver for the DAC. Remember, the goal of a device driver is to separate what the device does from how it works. “What it does” means the general descriptions and function prototypes of **DAC\_Init** and **DAC\_Out** that are placed in **DAC.h**. “How it works” means the implementations of **DAC\_Init** and **DAC\_Out** that will be placed in **DAC.c**.

## Step 3

The third step is to design a low-level device driver for the three keys of the piano. For example, if you could create public functions **Piano\_Init** and **Piano\_In**, where **Piano\_In** returns a logical key code for the pattern of keys that are pressed. You may design this driver however you wish, but the goal is to abstract the details of how it works (which port, which pin) from what it does (which key pressed).

## Step 4

The fourth step is to organize the sound generation software into a device driver. You will need a data structure to store the sound waveform. You are free to design your own format, as long as it uses a formal data structure. Although you will be playing only three notes, the design should allow additional notes to be added with minimal effort. For example, you could create public functions **Sound\_Init** and **Sound\_Play(note)**, where the parameter note specifies the frequency (pitch) of the sound. Calling **Sound\_Play(Off)** makes it silent and calling **Sound\_Play(C)** plays the note C. A background thread within the sound driver implemented with SysTick interrupts (see “EGCP\_450\_Lab\_6.zip”) will fetch data out of your music structure and send them to the DAC.

## Step 5

The last step is to write a main program that links the modules together creating the digital piano.

# Procedure

## Part A

Decide which port pins you will use for the inputs and outputs. Avoid the pins with hardware already connected. Draw in Multisim the circuit, like the one shown in Figure 2, required to interface the binary-weighted DAC to the MSP432 (NOTE: the “Lab\_6\_Part\_A.ms14” file on TITANium already contains the model for the MSP432 output). Use the Multisim Oscilloscope to see the output of your design similar to what is shown in Figure 2. Design the DAC converter using a simple resistor-adding technique. A 4-bit binary-weighted DAC uses resistors in a 1/2/4/8 resistance ratio. Select values in the 1.5 k to 240 k range. For example, you could use 1.5 k, 3 k, 6 k, and 12 k. Notice that you could create double/half resistance values by placing identical resistors in series/parallel. Have the instructor verify your design before you build it. Also, put screenshots of your schematic and waveform in the “Questions/Deliverables” section, Part 3.a of your report.

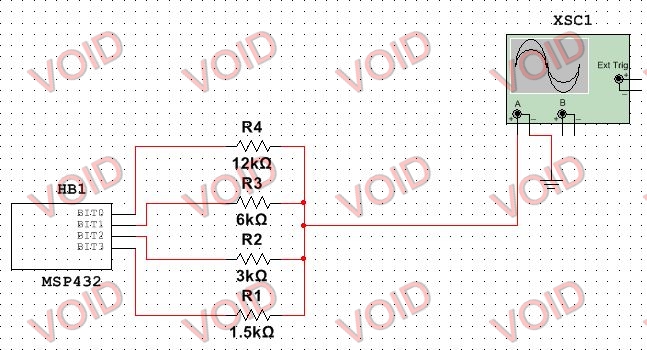


Figure : Multisim schematic of DAC.

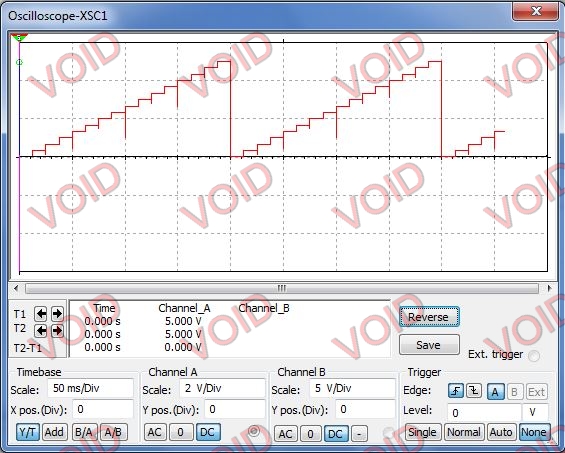


Figure 3: Multisim oscilloscope output of DAC.

## Part B

Use your ohmmeter to determine which two wires to connect on your audio jack (Figure 4). You have the option of connecting just the left, just the right, or both channels of the headphones. Now, in Multisim (see “Lab\_6\_Part\_B.ms14”), copy your DAC circuit from Part A and design the three switch interface, which will be used for the piano keyboard. Test the output using the Multisim Oscilloscope. Put screenshots of your schematic and waveform in the “Questions/Deliverables” section, Part 3.b of your report.

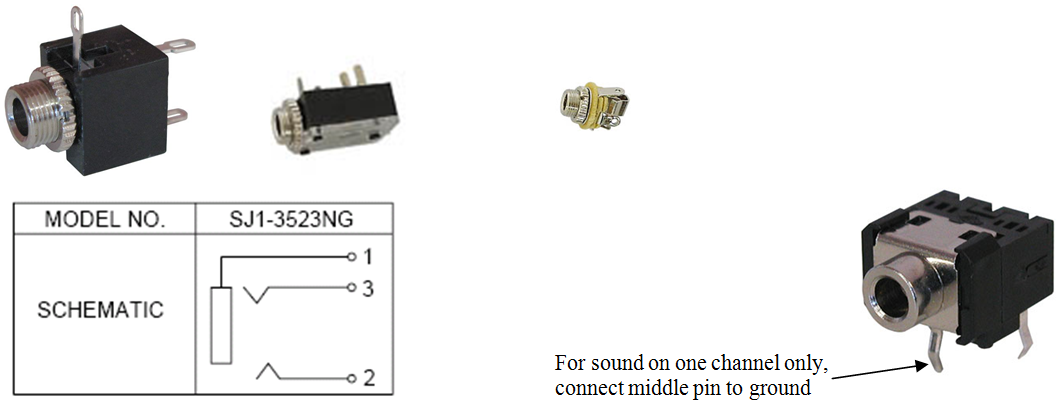


Figure 4: A mono jack has three connections (for one channel connect pin 1 to ground and pin 3 to the DAC output).

## Part C

Write the device driver for the DAC interface. 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 a new data value to the DAC. Place all code that accesses the DAC in a separate **DAC.c** code file. Add a **DAC.h** header file with the prototypes for public functions. Describe how to use a module in the comments of the header file. Describe how the module works, how to test the module, and how to change module in the comments of the code file.

## Part D

Begin with the static testing of the DAC. You will write a simple main program to test the DAC, similar in style as the program shown in Figure 5. You are free to debug this system however you wish, but you must debug the DAC module separately. You should initially debug your software in the simulator (HINT: look at the port registers in the debugger). Single step this program and compare the digital **Data** to the analog voltage at the *Vout* without the speaker attached (i.e., Figure 1, left).

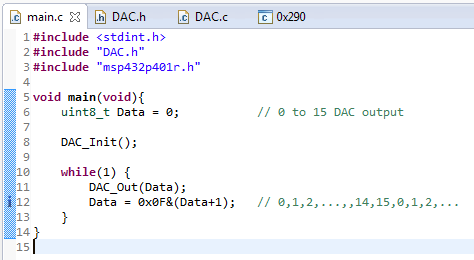


Figure 5: A simple program that outputs all DAC values in sequence.

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 the table and put these calculations in the “Questions/Deliverables” section, Part 3.c of your report.

## Part E

Design and write the piano keyboard device driver software. These routines facilitate the use of the three piano keys. Include at least two functions that implement the piano keyboard interface. For example, you could implement the function **Piano\_Init()** to initialize the switch inputs, and the function **Piano\_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 **Piano.c** code file. Add a **Piano.h** header file with the prototypes for public functions. Add comments that describe what it does in the **Piano.h** file and how it works in the **Piano.c** file.

## Part F

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. 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\_Play(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 should write to **SysTick->LOAD**, changing the interrupt period, without completely reinitializing SysTick. One approach to debugging is to attempt to create a sinewave with a constant frequency. Just run the SysTick periodic interrupts and output one DAC value each interrupt.

## Part G

Write a main program to implement the three-key piano. Make a heartbeat connected to an LED so you can see your program is running. Document clearly the operation of the routines. Figure 6 shows a possible data flow graph of the music player. Debug the individual components (switch input, static DAC output, SysTick ISR) and then combine the components together into one system on the MSP432. When no buttons are pressed, the output will be quiet. When Button 1 is pressed, output a sinewave at one frequency. When Button 2 is pressed, output a sinewave at a second frequency. When Button 3 is pressed, output a sinewave at a third frequency. Only one button will be pressed at a time. The sound lasts until the button is released.



Figure 6: Data flows from the memory and the switches to the speaker.

Figure 7 shows a possible call graph of this lab’s system. Dividing the system into modules allows for concurrent development and eases the reuse of code.



Figure 7: A call graph showing the three modules used by the regular part of Lab 6. Notice the SysTick hardware calls the SysTick ISR.

# Demonstration

You should be able to demonstrate the three 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 instructor 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 5 to 10 bits. What is the range, resolution, and precision? You will tell the instructor what frequency you are trying to generate, and they may check the accuracy with a frequency meter or scope. Instructors may ask you what frequency it is supposed to be, and then ask you to prove it using calculations. Just having three 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 4-bit DAC). Many students come for their checkout with systems that did not operate properly. In this case, you may be asked SysTick interrupt and DAC questions. If the desired frequency is **f**, and there are **n** samples in the sinewave table, what SysTick interrupt period would you use?

# Lab Report

The lab report is how I will grade you on your labs. Usually, the report is due 1 week following the completion of the lab (see TITANium for due dates). However, do to unforeseen circumstances, due dates may change. I will try my best to keep everyone informed of any changes. With this said, it is your responsibility to turn the report in during the scheduled due date. **There is a 10% penalty for late reports**. **Your report must be in MS Word Doc format. Submitting the report in another format will result in a 10% penalty.** Your lab report must include the cover sheet from the lab report template available on TITANium. **Not including the cover sheet will result in a 10% penalty.** The template contains instructions for the report and the rubric used for grading the labs. Please, read it thoroughly. **Don’t forget to include pertinent information such as code, flowcharts, waveform output, etc. in your report**. Please, no “spaghetti” code, keep your code clean and use comments. Remember, your code will affect your lab grade. If I can’t understand it, then I will assume it’s incorrect.