(https://www.edx.org)

**UTAustinX:** UT.6.01x Embedded Systems - Shape the World

**KarenWest (/dashboard)** ▼

**Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware)** **Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)**

**Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum)** **Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)**

**Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)**

**Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)**

**Embedded Systems Community (/courses/UTAustinX/UT.6.01x/1T2014/e3df91316c544d3e8e21944fde3ed46c/)**

**Help**

## PREPARATION

You will need a LaunchPad, four switches, four 10-kΩ resistors to interface the four switches, four to six resistors to build a 4-bit DAC, the headphone jack, and headphones.

## STARTER PROJECT

Lab13_DAC

## PURPOSE

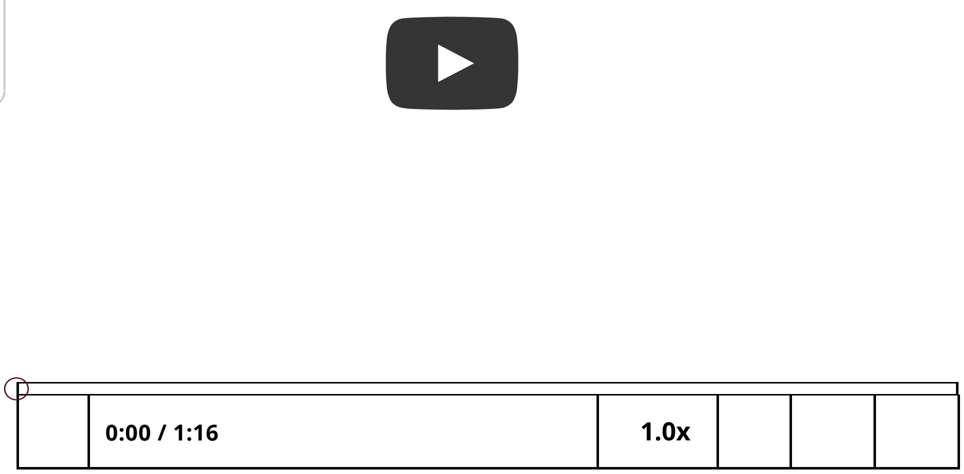
This lab has these major objectives: 1) to learn about DAC conversion; 2) to understand how digital data stored in a computer could be used to represent sounds and music; 3) to study how the DAC and interrupts can be used together to create sounds.

## SYSTEM REQUIREMENTS

Most digital music devices rely on high-speed DACs 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 four notes, which will be a digital piano with four keys. The first step is to design and test a 4-bit binary-weighted DAC, which converts 4 bits of digital output from the TM4C123 to an analog signal. You are free to design your DAC with a precision of more than 4 bits. The simulator supports up to 6 bits. You will convert the digital output signals to an analog output using a simple resistor network. During the static testing phase, you will connect the DAC analog output to the voltmeter and measure resolution, range, precision, and accuracy. During the dynamic testing phase, you will connect the DAC output to the scope to see the waveform verses time. If you connect the DAC to headphones you will be able to hear the sounds created by your software. 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. 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 3-kΩ resistor from two 1.5-kΩ resistors. You can create a 6-kΩ resistor from two 12-kΩ resistors.

## WORKING LAB 13

Help



DR. JONATHAN VALVANO: Let's begin lab 13 by showing you

what the final product will be like.

We have the LaunchPad here, flipped upside down,

and on this part of the proto board, I have interfaced four buttons.

Those are going to be our piano keys.

We have four more digital signals, which are

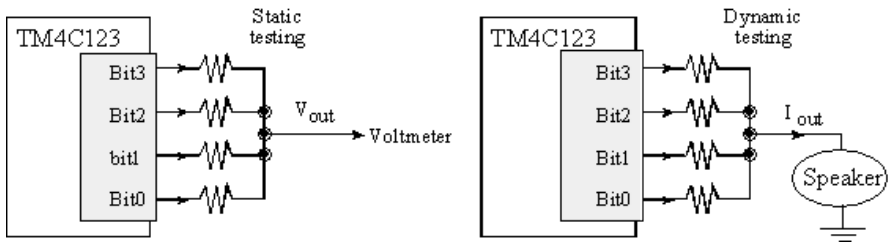


Figure 13.1. DAC allows the software to create music. In the simulator mode, the output voltage **$V_{out}$** is called DACOUT.

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.

The third step is to design a low-level device driver for the four keys of the piano. For example, 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).  You will not need to handle the situation where multiple keys are simultaneously pressed. This interface will be similar to the switch inputs on Lab 10.

The fourth step is to organize the sound generation software into a device driver.  You will need a data structure to store the data versus time waveform. The real-board grader expects you to create sinusoidal waveforms. You are free to design your own format, although most students will choose to place the data into a simple array. Although you will be playing only four notes, the design should allow additional notes to be added with minimal effort. For example, you could create public functions **Sound_Init Sound_Off()** and **Sound_Play(note)**, where the parameter **note** specifies the frequency.

(pitch) of the sound. For example, calling **Sound_Off()** makes it silent by setting the output to zero, and calling **Sound_Play(C)** plays the note C by outputting the stored waveform at a rate such that the resulting sine wave has a frequency of 523.251 Hz. A background thread within the sound driver implemented with SysTick interrupts will fetch data out of your data structure and send them to the DAC. A good design uses the same data array for the DAC outputs, but adjusts the SysTick **RELOAD** value to change the pitch. Remember to output only one DAC value each interrupt. This means there will be a simple relation between the size of the sine table (*N*), the frequency of the SysTick interrupt ($f_S$), and the resulting sound frequency (*f*)

Help

$$f = f_S / N$$

The last step is to write a main program that links the modules together creating the digital piano. After initialization, the main loop will input from the piano keyboard, and then call **Sound_Play(C)**, **Sound_Play(D)**, **Sound_Play(E)**, **Sound_Play(G)**, or **Sound_Off()**, depending on the input pattern. You may ignore multiple keys pressed at the same time.

---

The wave should be generated whenever one key is pressed and held. These are the standard frequencies which your digital piano must produce:

Piano key 3: G generates a sinusoidal DACOUT at 783.991 Hz
Piano key 2: E generates a sinusoidal DACOUT at 659.255 Hz
Piano key 1: D generates a sinusoidal DACOUT at 587.330 Hz
Piano key 0: C generates a sinusoidal DACOUT at 523.251 Hz