

# The Sound (and Sight) of Music!

## Teaching Electronics with the STM32 Nucleo

by Michael Parks, P.E., for Mouser Electronics [Licensed under CC BY-SA 4.0](https://www.mouser.com/licenses/cc-by-sa-4.0/)

<http://www.mouser.com/applications/nucleo-project/>

### Summary

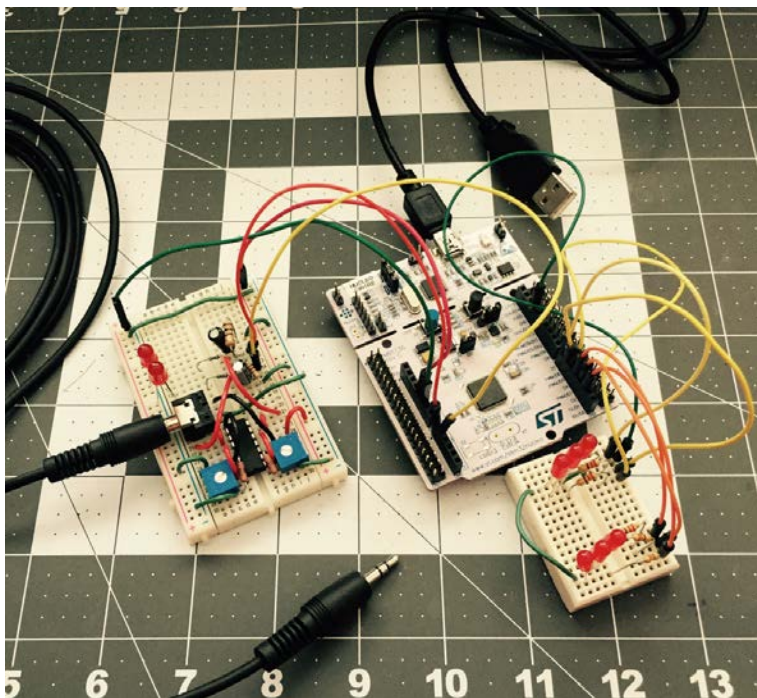
Over the last fifteen years or so, music has found itself encapsulated into digital bits. What if we could tap into this marriage of music and technology to make something even more amazing?

This project is one of a collection of STM32 Nucleo (a.k.a. Nucleo) projects located on Mouser Electronics' open source hardware site:

<http://www.mouser.com/applications/open-source-hardware/>

In this project we are going to add a visual representation of our music. We will take the audio from our favorite portable music player and electronically sample its waveforms to produce a stunning light show using LEDs. This project can be done both with and without a microcontroller. The LEDs can be driven directly using an operational amplifier or sent to an ST Nucleo via analog-to-digital conversion to add even more fun effects. When setting out to design this project, we aimed to strike a balance between cost, complexity, and educational value. But most importantly, we wanted a project that captured the imagination of students and empowered them to take the basic design and hack it to meet their own unique desires. The circuit is a basic audio amplifier circuit that, in principle, is used in tons of audio products. In a circuit that fits onto a tiny breadboard we can teach Ohm's Law, voltage dividers, potentiometers, LEDs, capacitors, current limiting, operational amplifier theory, and line level audio! The beauty of the design is that it is self-contained and can be used without having to write any code or interface with a microcontroller, except to provide power to the op amp. Of course, you could provide power to the op amp using 3 AA batteries if so desired.

After the basics have been mastered and you're ready to take the next step, it's quite simple interface the audio circuit to the Nucleo via an analog input pin. This provides even greater education value since we can now further their learning by introducing the concepts of microcontrollers, embedded design, and even C/C++ programming.



*Figure 1: The Nucleo Light and Sound Machine controls the music and flashes LEDs in time to the music.*

### Materials

To make things super simple we are [providing a pre-built shopping cart with all the correct components](#). No worries on selecting the wrong part by accident! We've also labeled each item in the BOM to corresponding part in the provided schematic. This project also assumes you have access to a Windows machine and the internet to use the [www.mbed.org](http://www.mbed.org) programming tools.



The Bill of Materials (BOM) for this project is for the electronics (Figure 1). In the demo video, a smartphone or MP3 player is connected to the 3.5mm Stereo jack via an RCA cable and a splitter that also takes the signal to the speakers. We've found that some media players are better than others at driving the signal (iPhones seem to work better, the one tablet we tried didn't work at all.) The electronic components recommended include:

STM Nucleo Sight and Sound Open Source Hardware Project						
Mouser #	Mfr.	Description	Schematic Part #	Order Qty.	List Price (USD)	Ext.: (USD)
854-BB400T	BPS	PCBs & Breadboards 400 TIE POINT BB W/PWR RAILS TRANSP	n/a	1	\$5.50	\$5.50
510-WK-3	Global Specialties	Jumper Wires 70pc JUMPER WIRE KIT	n/a	11	\$3.50	\$38.50
806-STX-3100-5N	Kycon	Phone Connectors 3.5mm PCB STEREO BLK 5P W/SWITCH	JP1	2	\$1.85	\$3.70
78-TLHR5400	Vishay	Standard LEDs - Through Hole Red Tinted Diffused	LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8	12	\$0.292	\$3.50
594-5063JD220R0F	Vishay	Thin Film Resistors - Through Hole .4watt 220ohms 1% 1/8watt body size	R4, R6, R7, R8, R9, R10, R11, R12	12	\$0.12	\$1.44
594-5063JD100R0F5	Vishay	Thin Film Resistors - Through Hole .4watt 100ohms 1% 1/8watt body size	R3, R5	4	\$0.13	\$0.52
652-3386P-1-103LF	Bourns	Trimmer Resistors - Through Hole 3/8" 10Kohms 10% 0.5Watts Square	TM1, TM2	4	\$1.60	\$6.40
595-LM324N	TI	Operational Amplifiers - Op Amps Quad	IC1A, IC1B, IC1C, IC1D	2	\$0.57	\$1.14
71-CMF5510K000FKEB	Vishay	Metal Film Resistors - Through Hole 1/4watt 10Kohms 1%	R1, R2	4	\$0.44	\$1.76
647-UVR1H100MDD1TA	Nichicon	Aluminum Electrolytic Capacitors - Leaded 50volts 10uF	C1, C2	4	\$0.23	\$0.92
511-NUCLEO-F401RE	STM	Development Boards & Kits - ARM Nucleo Board STM32F4 STM32F401RE 512K	F401RE	2	\$10.33	\$20.66

**Table 1: Bill of Materials for STM Nucleo Sights & Sounds Project**

We will use an [STM32 Nucleo F401RE microcontroller platform](#). The STM32 Nucleo is programmed using the C/C++ language, but we provide the code for you. Nevertheless, programming tools are free online at [mbed.org](http://mbed.org).

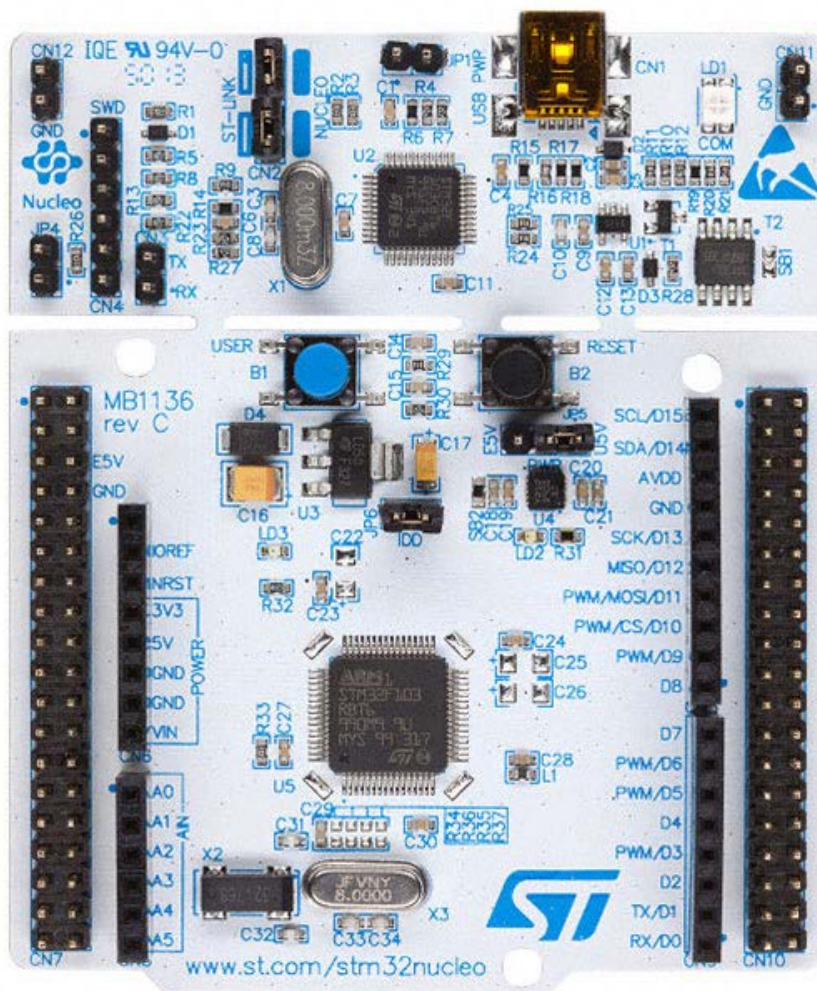


Figure 2: The STM32 Nucleo has a detachable, reusable programming card at the top that is removed once a project is done.

#### Tools:

Since this project is aimed towards classrooms and not electronics laboratories, we wanted to make sure that no hand tools or electronic test equipment was required. Of course, if you are teaching a course specifically geared towards electronics and have access to a multimeter or an oscilloscope, hooking it up the outputs of the audio jack makes for a very cool visualization of sound (Figure 2.) But to be sure, this is completely optional!

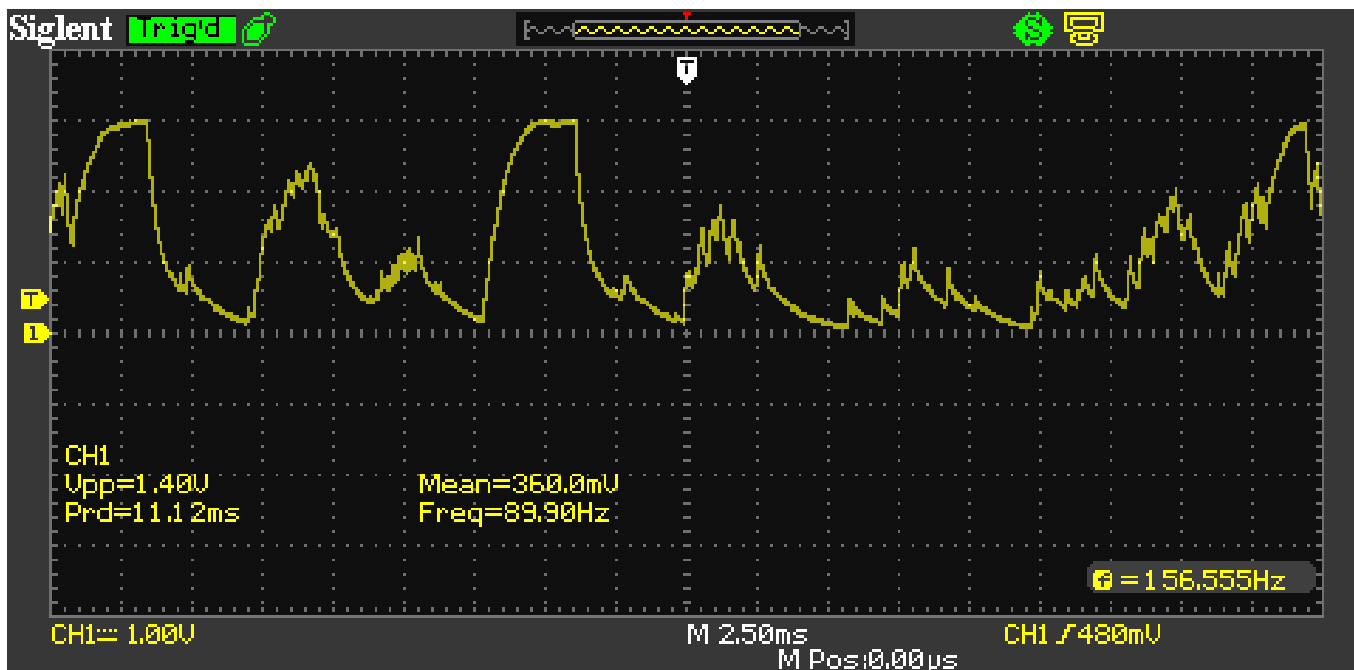


Figure 3: An oscilloscope is not needed for this project, but if you have one, you can visualize the sound with the oscilloscope.

If you plan to do more with the Nucleo than what is in *this* project, try looking at the Nucleo site on mbed.org: <https://developer.mbed.org/platforms/ST-Nucleo-F401RE>

Here, you will find other projects and any additional information, such as this pin-out diagram of the Nucleo:

### Morpho headers

These headers give access to all STM32 pins.

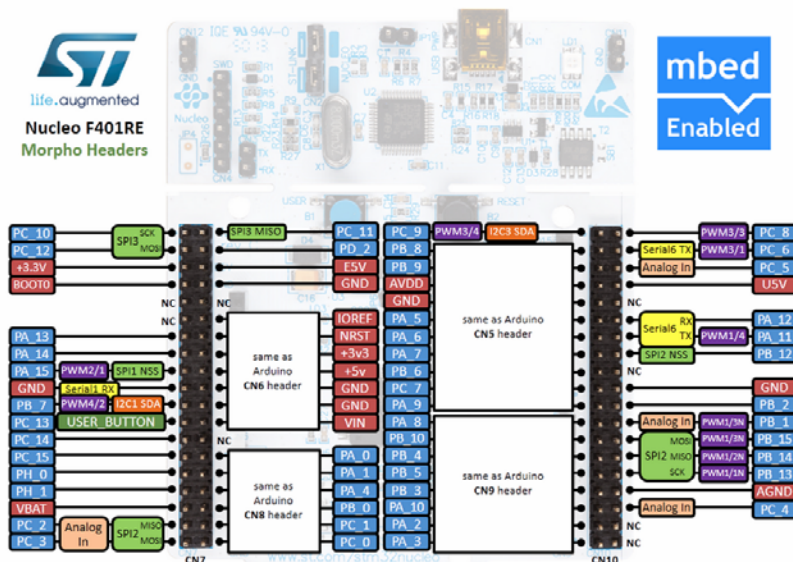


Figure 4: The STM32 Nucleo pin-out and other Nucleo information and projects can be found at [mbed.org](https://developer.mbed.org/platforms/ST-Nucleo-F401RE)

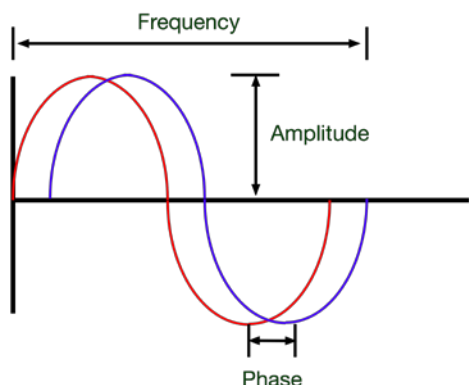
## Overview

Striving for simplicity, the circuit is very straightforward. We are taking audio from a computer or mobile device via its 3.5mm audio jack and splitting left and right audio channel so we can tweak each channel individually based on our preferences. **Line level audio** is very low voltage and not suitable to drive our LEDs or the Nucleo directly. That is why we are taking the audio and running it through an operational amplifier, often shortened to just “op amp”. Actually, if you look



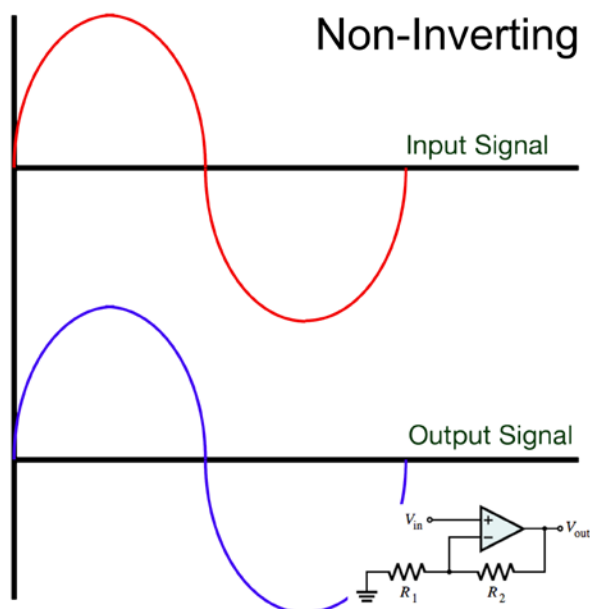
carefully at the schematic (located in the [Resources Section](#)) you will notice that each channel signal is run through two op amps. The first op amp provides amplification and the second op amp provides [buffering](#).

Op amps are one of the workhorse components of circuit design. They can be used in wonderfully simple but also incredibly complex ways. Learning op amps is a key skill. For this project we are using the op amp in a configuration known as a “[non-inverting amplifier](#)”. The alternative is an “inverting amplifier”. To understand the difference let’s take a quick look at the parts of analog signal.



*Figure 5: Parts of an analog signal.*

In a non-inverting amplifier the output will remain in phase with the input signal, in an inverting amplifier the output is 180-degrees out of phase. If you look at the schematic you will notice that the output of the first op amp feeds back into the inverting terminal via a small network of resistors. The arrangement of resistors is known as a voltage divider circuit. It allows us to control the amount of amplification or gain the op amp will perform on the input signal. Using a fixed resistor and variable resistor (the one we’re using is referred to as a trimming potentiometer or trimpot for short) will allow us to manually control the gain. By turning the trimpot we vary its resistance, and in turn the gain of the op amp which will cause the LEDs to glow more brightly or more dimly depending on how it’s adjusted.



*Figure 6: Non-inverting op amp signal.*

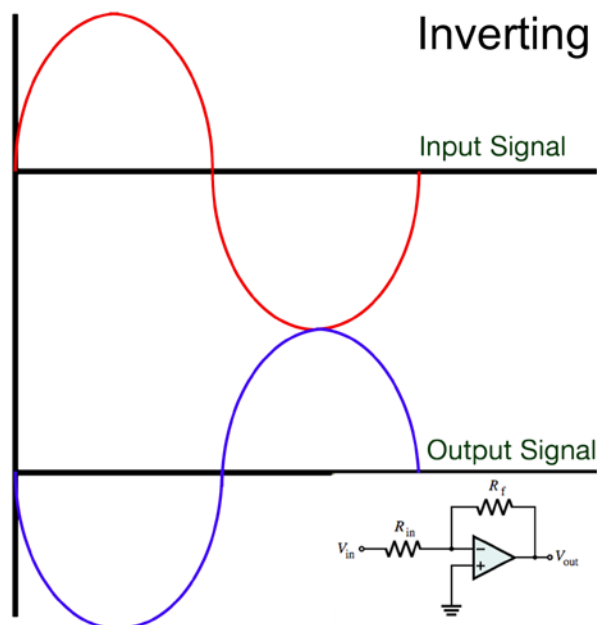


Figure 7: Inverting op amp signal.

At the output of the second op amp we are using a resistor and capacitor in parallel to create a high pass noise filter to get rid of any RF noise the circuit might pick up. Lastly the LED and resistor in series are there to provide the light show. The LED is obviously doing the blinking and the resistor provide current protection for the diode. Lastly, some notes about 3.5mm Tip-Ring-Sleeve audio jacks. The signal ground is commonly found on the outermost conductor, which is called a sleeve. The left channel is typically on the tip, and the right channel is sent out over the ring (Think "R" for right and ring).

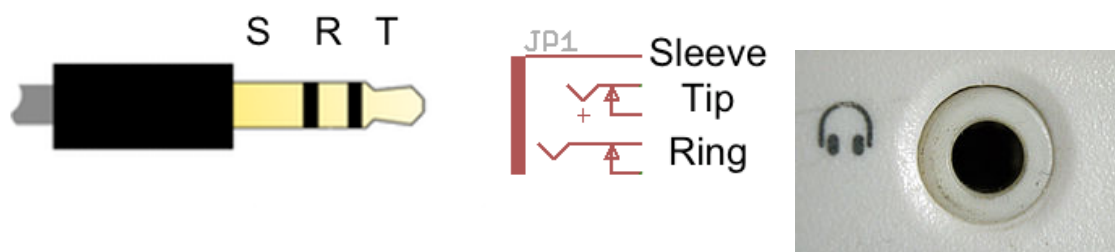
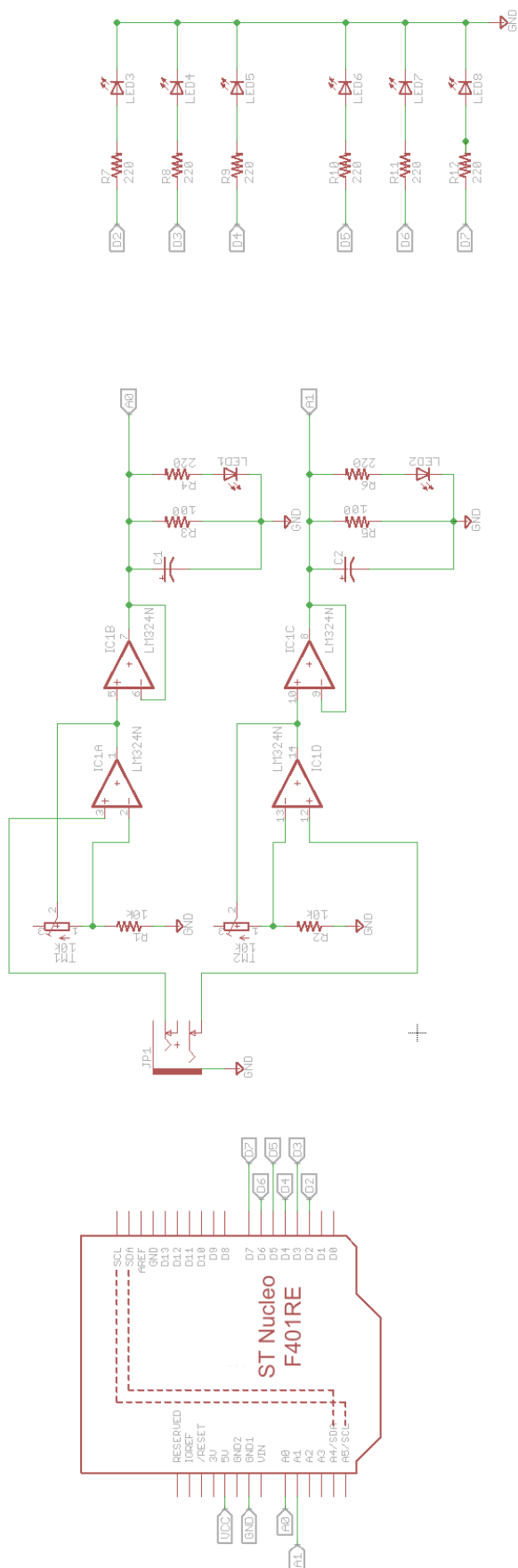


Figure 8: A 3.5mm Tip-Ring-Sleeve audio jack diagram (left), schematic of the jack (middle), and PC speaker input (right) to play the music using built-in PC speakers if you don't have portable speakers. Audio jack part number: 172-2208 or 172-8362-E.

## The Build

Conveniently all the components will fit on a single, small breadboard. The build is very straightforward. The first nine steps are used regardless if you are going to use the Nucleo or not. The final steps will vary depending if you desire to use the Nucleo or not. Option A will simply drive two LEDs without the Nucleo (using AA batteries instead of the Nucleo to power it.) Option B will require a computer to program the Nucleo and then perform the light show as we have programmed in our code. If you stop after implementing Option A, then you can still use the Nucleo for power.

Figure 9: Schematic of the STM32 Nucleo Sight and Sound open source hardware project. (NucleoLightSoundEducationSchematic.png)





## Instructions:

You will want to print out the schematic (NucleoLightSoundEducationSchematic.png) as listed in the Resources Images section.

1. Carefully place the op-amp favoring one end of the breadboard, but leave about 3 rows available on the end you choose. Be aware that each pin has a different purpose. The datasheet will show you what each pin does and how they interconnect internally. Notice that DIP ICs have a dot (either printed or a depression on the package) near pin 1. There also tends to be a small "U"-shaped depression towards the top of the chip, with pin 1 being to the left. I cannot stress enough the importance of buying an extra IC or two when ordering your parts. From bent pins that break off, to reversing Vcc and GND, once chips are broken there is no repair. Have a few extra on hand will give you peace of mind when experimenting with a design.
2. Next place the trimpot so that the central wiper pin is connected to op amp pin 5, and that an outer pin of the trimpot connects to op amp pin 2. Recall, we are splitting out the left and right audio channel so we are going to repeat this for the other side of the op amp. This time, the central wiper pin is connected to op amp pin 10 and that an outer pin of the trimpot connects to op amp pin 13.
3. Add the fixed resistor to each trimpot, with one end of the resistor attaching to the trimpots and the other to GND. Connect to the same outer pin of the trimpots that you used in step 2.
4. Place the 3.5 mm audio jack onto the breadboard. Be sure to align the audio jack so each pin gets its own row on the breadboard.
5. Connect pin 5 of the audio jack (tip, left channel) to pin 12 of the op amp.
6. Connect pin 2 of the audio jack (ring, right channel) to pin 3 of the op amp.
7. Connect pin 1 of the audio jack (sleeve, ground channel) to pin 11 of the op amp.
8. Connect pin 6 of the op amp to pin 7 of the op amp chip. (This is the feedback loop)
9. Connect pin 8 of the op amp to pin 9 of the op amp chip. (This is the feedback loop)
10. Connect pin 1 of the op amp to pin 5 of the op amp.
11. Connect pin 10 of the op amp to pin 14 of the op amp.

Continue on with at least Option A, which does not require code (but requires the Nucleo for its power source.) You can add Option B which controls many more LEDs and uses the MCU of the Nucleo and includes programming as included in the Software section.

### Option A: LEDs driven by the output of the op amp

1. Place a 100-ohm resistor, 220-ohm resistor, and a 0.1uF capacitor onto the board so that they all share a common node (the positive lead of the capacitor). Replace this 220-ohm resistor with the extra 100-ohm resistor if the LED isn't bright enough.
2. Connect the anode of the LEDs to the other end of the 220-ohm resistor.
3. Connect the cathode of the LED, the negative lead of the capacitor and the free end of the 100-ohm resistor to a common node. Wire this node to ground.
4. Repeat steps 1 through 3.
5. Wire the output of op amp pin 7 to the common node shared by the two resistors and positive lead of the capacitor. This will show the intensity of the right channel.
6. Wire the output of op amp pin 8 to the common node shared by the other two resistors and positive lead of the second capacitor. This will show the intensity of the left channel.
7. Skip the software section if you do not continue to Option B.

### Option B: LEDs driven by the Nucleo

1. Place six 220-ohm resistors on the breadboard.
2. Place six LEDs on the board, each one with the anode connected to one end of a 220-ohm resistor. The cathode of all LEDs can share a common ground.
3. Take the free ends of the first three resistors and attach one to Nucleo pin D2, one to pin D3, and the last one to pin D4.
4. Take the free ends of the remaining resistors and connect one to Nucleo pin D5, one to pin D6, and the last one to pin D7.
5. Connect op amp pin 7 (output from the left channel) to Nucleo pin A0.
6. Connect op amp pin 8 (output from the right channel) to Nucleo pin A1.





## Software

When you are ready to integrate the Nucleo into the project, we will need to first program the Nucleo to do what we desire. In this example, we will sample the output of each channel's audio using the onboard analog-to-digital converter. Depending on the intensity of each channel's audio, and in turn the voltage level of the op amp's output, we will drive 1, 2, or 3 LEDs per channel. So if the audio is very low we will drive 1 LED, and if it is very loud we will drive 3 LEDs.

The program we are running on the Nucleo is pretty straightforward. Once running the program simply repeat the following four high-level function repeatedly under power is removed from the Nucleo:

1. Sample the analog value of both the left and right channel and digitize.
2. Assign the digitized values to a variable so we can use it throughout the current loop iteration.
3. Look at the digitized value of the channel and determine if its volume is low, medium, or high.
4. You can tweak the values of the variables that trigger the low, medium, and high LEDs. You can read more in the software section, look for the discussion on lowCutoff, mediumCutoff, and highCutoff. For now, the default logic is as follows:
  - a. If the channel is sampled to be below 10, then it is nearly perfectly quiet, turn off all the LEDs
  - b. If the channel is sampled to be between 10 and 1000, then turn on just one LED since the volume is low.
  - c. If the channel is sampled to be between 1000 and 2000, then turn on two LEDs since the volume is medium.
  - d. If the channel is sampled to be greater than 2000, then turn on all three LEDs since the volume is high.

The program code for the Nucleo is listed below and can also be found in the [Resources section](#) as *main.cpp* or as a text file named *LightSound\_NucleoCode.txt* (shown below):

```
#include "mbed.h"

AnalogIn leftChannel(A0); // left channel
AnalogIn rightChannel(A1); // right channel

DigitalOut leftLED_LOW(D2);
DigitalOut leftLED_MED(D3);
DigitalOut leftLED_HIGH(D4);

DigitalOut rightLED_LOW(D5);
DigitalOut rightLED_MED(D6);
DigitalOut rightLED_HIGH(D7);

void setLeftLED_OFF();
void setLeftLED_LOW();
void setLeftLED_MED();
void setLeftLED_HIGH();

void setRightLED_OFF();
void setRightLED_LOW();
void setRightLED_MED();
void setRightLED_HIGH();

int main() {
    float leftChannelSignal;
    float rightChannelSignal;
    float lowCutoff = 10;
    float mediumCutoff = 1000;
    float highCutoff = 2000;

    printf("\nTurn Sound Into Light\n");

    while(1) {
        leftChannelSignal = leftChannel.read(); // Converts and read the analog input value (value from 0.0 to 1.0)
        rightChannelSignal = rightChannel.read(); // Converts and read the analog input value (value from 0.0 to 1.0)
```

```

leftChannelSignal = leftChannelSignal * 3300; // Change the value to be in the 0 to 3300 range
rightChannelSignal = rightChannelSignal * 3300; // Change the value to be in the 0 to 3300 range

printf("L Channel = %.0f mV\n", leftChannelSignal);
printf("R Channel = %.0f mV\n", rightChannelSignal);

//LEFT CHANNEL
if (leftChannelSignal <= lowCutoff) {
    setLeftLED_OFF();
}

else if (leftChannelSignal > lowCutoff && leftChannelSignal <= mediumCutoff) {
    setLeftLED_LOW();
}

else if (leftChannelSignal > mediumCutoff && leftChannelSignal <= highCutoff) {
    setLeftLED_MED();
}

else {
    setLeftLED_HIGH();
}

//RIGHT CHANNEL
if (rightChannelSignal <= lowCutoff) {
    setRightLED_OFF();
}

else if (rightChannelSignal > lowCutoff && rightChannelSignal <= mediumCutoff) {
    setRightLED_LOW();
}

else if (rightChannelSignal > mediumCutoff && rightChannelSignal <= highCutoff) {
    setRightLED_MED();
}

else {
    setRightLED_HIGH();
}

//wait(0.2); // 200 ms
}
}

//LEFT CHANNEL
void setLeftLED_OFF(){
    leftLED_LOW = 0;
    leftLED_MED = 0;
    leftLED_HIGH = 0;
}

void setLeftLED_LOW(){
    leftLED_LOW = 1;
    leftLED_MED = 0;
    leftLED_HIGH = 0;
}

void setLeftLED_MED(){
    leftLED_LOW = 1;
    leftLED_MED = 1;
    leftLED_HIGH = 0;
}

```

```

void setLeftLED_HIGH(){
    leftLED_LOW = 1;
    leftLED_MED = 1;
    leftLED_HIGH = 1;
}

//RIGHT CHANNEL
void setRightLED_OFF(){
    rightLED_LOW = 0;
    rightLED_MED = 0;
    rightLED_HIGH = 0;
}

void setRightLED_LOW(){
    rightLED_LOW = 1;
    rightLED_MED = 0;
    rightLED_HIGH = 0;
}

void setRightLED_MED(){
    rightLED_LOW = 1;
    rightLED_MED = 1;
    rightLED_HIGH = 0;
}

void setRightLED_HIGH(){
    rightLED_LOW = 1;
    rightLED_MED = 1;
    rightLED_HIGH = 1;
}

```

Good coding practice would dictate that since the left and right channels are performing similar tasks that we could incorporate both into a single functions. However, we kept them separate to help with understanding what's going on with each channel and so you can create different effects for each channel independently.

Now that we understand what the code is doing, let's have quick discussion of what is needed to actually get the code onto the Nucleo. To make things as simple as possible to start, you can simply copy the code we've provided above and paste into the tool.

*Figure 10: The free mbed.org programming tool for the Nucleo. Shown is the Works Space Manager window.  
(mbed\_workspaceManager.png)*

Program Workspace

- My Programs
  - BuddyBot
  - BuddyBot\_v2
  - BuddyBot\_v3
  - CatLaserProject
  - HelloWorld
  - LightsAndSound
    - main.cpp
    - mbed
  - Nucleo\_pwm
  - Nucleo\_pwm2

Program: LightsAndSound

Name	Size	Type	Modified
main.cpp	3.2 KB	C/C++ Source File	2 weeks, 2 days ago
mbed		Library Build	3 weeks, 3 days ago

Program Details

SummaryBuild

Name

LightsAndSound

Created

3 weeks, 3 days ago

Last Modified

2 weeks, 2 days ago

Last Built

2 weeks, 2 days ago

URL

n/a

Revision

no revisions

Status

uncommitted changes

The documentation is out of date

Update

Publish

Revisions

Description

Filter: Search criteria ... Match Case Whole Word Advanced

Compile output for program: LightsAndSound

Description	Error Number	Resource	In Folder	Warnings: 0	Errors: 0	Infos: 0	Location
-------------	--------------	----------	-----------	-------------	-----------	----------	----------

Compile OutputFind ResultsNotifications





If you haven't already, you will need to download the [USB driver for the Nucelo-F401RE](http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1847?sc=stm32nucleo) which can be located on the mbed.org site or here: <http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1847?sc=stm32nucleo>. Click on Software and then Development Tool Software to find the "part number" for the download package: [STSW-LINK009](#). Or, you can find it by starting from here: <https://developer.mbed.org/platforms/ST-Nucleo-F401RE>. This site is also good for more information on the Nucleo, including firmware upgrades. You should check for the latest firmware upgrade every time you get a new board as good practice. Full instructions are offered on how to make these downloads if you are using the Nucleo for the first time, or with a new PC.

You can head over to [mbed.org](http://mbed.org) and create a free account. The nice thing about the Nucleo is that you can use the programming environment in the browser so it's always up to date. Once you are logged in you should see the "Compiler" link in the top right of the website, click on that. This will open up your workspace where all your projects will reside. From here follow these steps:

1. In the top left click on "New" and select "New Program".
2. In the dialog box make sure the following are set:
  - a. Platform: "NUCLEO-F401RE"
  - b. Template: "Empty program"
  - c. Program Name: Whatever you wish. I chose "LightsAndSounds"
3. This will create your default files needed, including "main.cpp". Double click that file to open the editor.
4. Paste in the code we've provided.
5. Click "Save All"
6. Click "Compile". This will create the necessary .bin file that you will drop onto your STM32 Nucleo. You will be asked where you want to save the file to on your computer. Usually I choose the Desktop.
7. Installing code on the STM32 Nucleo is as simple as dragging the file from wherever you saved it onto the STM32 Nucleo, just as you would click-and-drag any file onto a USB thumb drive.

That's it. After the .bin file is transferred the Nucleo will automatically reboot and start running your application.

## Assembly

There is no additional assembly required! Just hook up your audio source to the audio jack of your circuit. Of course, you could take this project and if you had the resources to teach 3D printing you could have students print an enclosure to house breadboard. But again, it is not required and the circuit will work happily with no enclosure.

I am using a 3.5mm splitter on the output of my phone so that I can send the sound to both the circuit we've built and a small speaker so I can hear the music as well as see it. I am using 3.5mm male-to-male extender cable from the splitter into the audio jack.

Even if we didn't use the Nucleo as part of the circuit before, we are going to use it now to provide power to the op amps if nothing else. Insert the USB cable into your computer and the other end into the Nucleo. You should notice the little LEDs on the board begin to glow. Simply attach the Vcc and GND rails of the breadboard to the Vcc and GND pins on the Nucleo. When you are comfortable to move on to interfacing your circuit with the Nucleo, don't forget to wire the outputs of the op amps into the analog inputs of the Nucleo as discussed previously.

## Project in Action

Finally, it's time to fire up your music app of choice, select your favorite tunes and watch the LEDs begin to rhythmically glow in beat with your music. Depending on how your music file (e.g. MP3, WAV) was created audio may not appear on both channels. That's okay, you just might not see both LEDs flashing to beat of the music.

If the 3.5mm jack is not plugged in you might notice the LEDs glowing, that's fine! It's an aspect of line level audio and our circuit.

For additional projects and information about open source hardware, go to:

<http://www.mouser.com/applications/open-source-hardware/>





## Source Files and Downloads

The following files are located here: <http://www.mouser.com/applications/nucleo-project/>

**STM32 Nucleo Code:** Original source files for programming the main STM32 Nucleo board. File names: *Main.cpp* and *LightSound\_NucleoCode.txt*

**Images/Photos:** Images of the Project and .PNG images of the original, commented source code are available here for download: *images.zip*.

**Schematic Source files:** This project wouldn't be open source hardware without the source files for the schematics. There is no PCB (yet), or we would provide source for that, as well. The schematic source Eagle files are collected in: *MusicalLights.zip*

**Schematic Image:** *NucleoLightSoundEducationSchematic.png*

**Demo Video:** <http://www.mouser.com/applications/nucleo-project/>

For additional projects and information about open source hardware, go to:  
<http://www.mouser.com/applications/open-source-hardware/>

*This project is licensed under [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)*