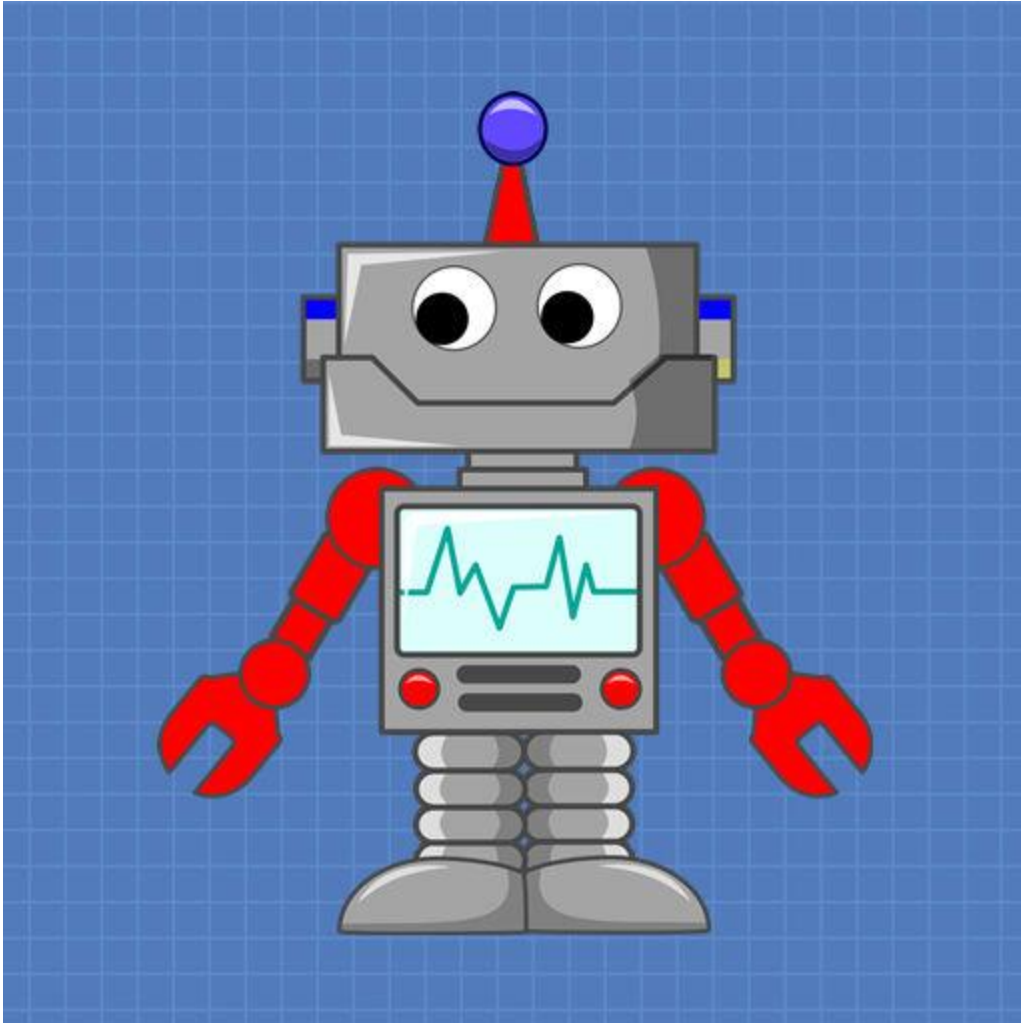


# Using the ESP32 DAC



DroneBot Workshop Tutorial

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Today we will use one of the lesser-known features of the ESP32, the Digital to Analog Converter or DAC. We'll see how it works, and then we will use it to create some Oscilloscope "Art" and an edible musical instrument!

## Introduction

You are likely very familiar with the analog-to-digital converters, or ADCs, that are packaged in just about every microcontroller. Commonly referred to as "analog inputs," these converters allow you to input analog voltages and retrieve a digital representation of their level.



But the ESP32 has a couple of other analog pins, DAC, or Digital to Analog Converters. As the name implies, these pins will OUTPUT an analog voltage.

Today we will see how we can incorporate the ESP32 DAC into our projects.

<https://dronebotworkshop.com>

## Digital to Analog Converters

Converting a digital number into an analog signal is a pretty standard application; audio is an obvious application, as it is stored and transmitted in a digital format and is converted to an analog format during playback.



DACs also have applications in telecommunications and instrumentation. And a DAC can also be used as a “digital potentiometer”.

## ESP32 DAC

There are two identical DACs in the ESP32, and unlike many other ESP32 features, these have their outputs hardwired to dedicated pins. We will refer to them as “Channel 1” and Channel 2”.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

On most ESP32 modules, you will find Channel 1 on GPIO pin 25, and Channel 2 on GPIO 26. But if you have an ESP32-S2 module, then Channel 1 is on GPIO 17, and Channel 2 is on GPIO 18.



The ESP32 DACs are 8-bit devices, which are unsuitable for high-end audio. They can be used for “telephone quality” audio, but you would be better off using I2S for ESP32 audio applications in most cases.

The output of each DAC ranges from zero volts to the reference voltage.

By default, the ESP32 DAC uses the 3.3-volt ESP32 power supply as its reference voltage. It can also use the AREF (Analog Reference) input with an external voltage reference, assuming that your ESP32 board exposes this pin (many do not).

One known issue with the ESP32 DAC is that it does not go down to zero volts when the input is set to 0 or up to 3.3 volts when the input is 255. You should factor this in when using it to control something that requires zero or 3.3 volts.

<https://dronebotworkshop.com>

## ESP32 DAC Modes

The DAC can be operated in three different modes:

### Direct Voltage Output

This is the simplest mode, in Direct Output mode, you simply write a value from 0 to 255 to the DAC channel, and the corresponding voltage will appear on the output. The voltage will remain there until the DAC is called again.

### Cosine Wave Generator

The ESP32 DAC has a single Cosine Wave Generator, whose output can be sent to one or both DAC channels. The user has control over the frequency, amplitude, and phase of the cosine wave.

### Continuous Output by DMA

In this mode, the DAC is fed by the DMA (Direct Memory Access) buffer. It can handle the buffer data using three methods:

- **Normal or Synchronous Writing** – The data in the DMA buffer is continually written to the DAC.
- **Cyclical Writing** – The DMA buffer is loaded with data, and that data is looped in the DAC. This is an efficient method of generating complex waveforms.
- **Asynchronous Writing** – The DMA data is sent to the DAC in response to an external callback.

# DAC Experiments

Now it's time to start working with the ESP32 DAC.

To be able to follow along, you will need some test equipment. If you don't own, or don't have access to some of it, you can watch me perform the experiments in the video accompanying this article.

## Test Equipment

You'll need a way of measuring DC voltage, DC voltage from between zero and 3.3 volts.

I'm guessing that you likely have a multimeter, and any multimeter will work. Digital, analog, auto-ranging, or manual ranging, all will be fine!

An oscilloscope with at least two channels will be very handy when working with waveforms, as you'll be able to see them. Obviously, the scope will be required for the Oscilloscope Art experiment!

If you don't have access to a scope, you can do the waveform experiments with an audio amplifier. Any small audio amp module will do, actually any amplifier will do, although I would be hesitant to feed it into your 500-watt sound system!

Otherwise, the only part required is an ESP32 module, and pretty well any ESP32 module with GPIO 25 and 26 will suffice. I used an ESP32 Dev Kit module for my experiments.

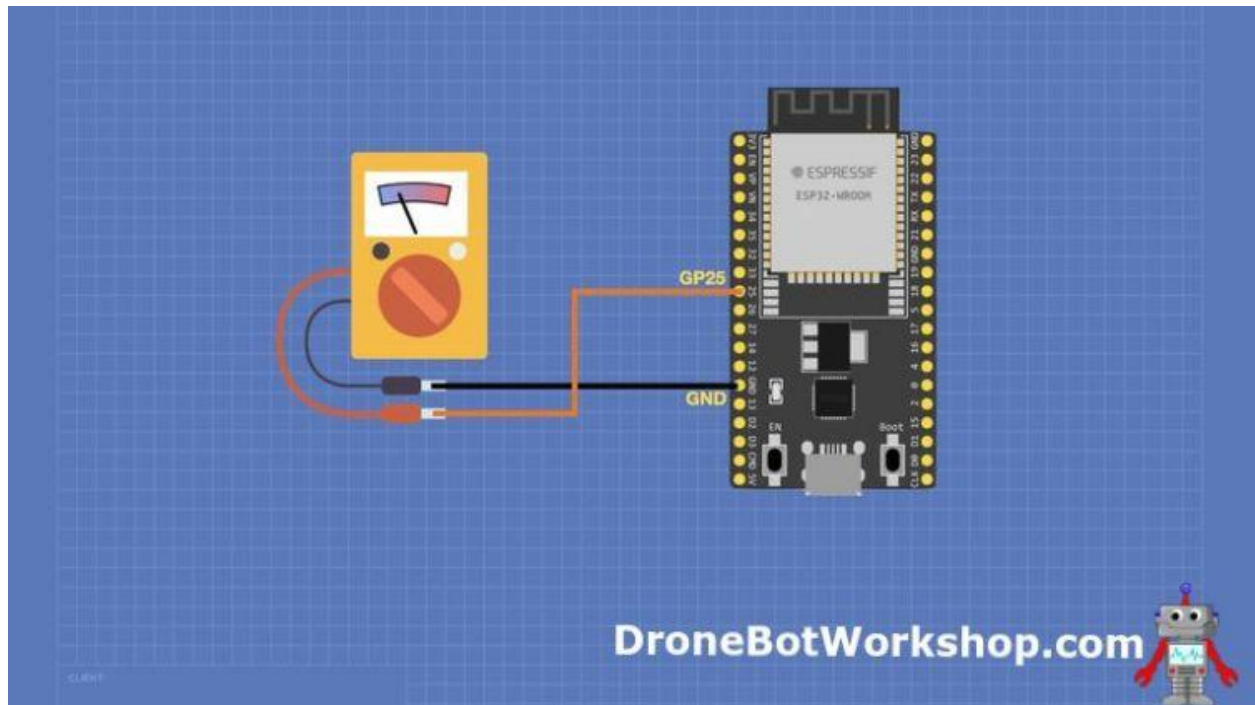
## Direct Voltage Output – Producing Voltages

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

For our first experiment, we will just write to the DAC and measure the output voltages. It's very easy to do with the Arduino IDE.

## Direct Voltage Output Hookup

Here is how we will hook up a multimeter (or voltmeter) to the ESP32 DAC:



You'll need to measure voltages between zero and 3.3 volts, so adjust your meter accordingly.

I will assume that your Arduino IDE is installed and configured with the ESP32 Boards Manager. You can read my article about using the ESP32 if you are not yet set up.

## Direct Voltage Output Basic Code

<https://dronebotworkshop.com>



Here is some very simple code that will write a series of numbers to the DAC on Channel 1. The DAC output can then be measured.

```
1  /*
2   ESP32 DAC - Voltage Output Experiment
3   espdac-voltage.ino
4   ESP32 DAC Voltage Output Demo
5   Steps through preset output voltages
6
7   DroneBot Workshop 2022
8   https://dronebotworkshop.com
9  */
10
11 // Define DAC pins
12 #define DAC_CH1 25
13 #define DAC_CH2 26
14
15 void setup() {
16     // Setup Serial Monitor
17     Serial.begin(115200);
18 }
19
20 void loop() {
21
22     // Step through voltages, delay between levels
23     dacWrite(DAC_CH1, 0);
24     Serial.println("DAC Value 0");
25     delay(3000);
26
27     dacWrite(DAC_CH1, 64);
```



```
28 Serial.println("DAC Value 64");
29 delay(3000);
30
31 dacWrite(DAC_CH1, 128);
32 Serial.println("DAC Value 128");
33 delay(3000);
34
35 dacWrite(DAC_CH1, 192);
36 Serial.println("DAC Value 192");
37 delay(3000);
38
39 dacWrite(DAC_CH1, 255);
40 Serial.println("DAC Value 255");
41 delay(3000);
42 }
```

It is a straightforward sketch, centering around the *dacWrite* function, which accepts two parameters:

- **The DAC Pin.** In this case, we are using DAC Channel 1, so the value is 25.
- **The DAC Value.** An integer between 0 and 255.

We simply write to the DAC with different values and delay for three seconds. We also write to the serial monitor at the same time.

Run the sketch and observe your multimeter. You will see the voltage change with each different DAC value.



You'll probably notice two things:

- The DAC doesn't go to zero when the value is 0.
- The DAC doesn't go to 3.3 volts when the value is 255.

This is a known issue with the ESP32 DAC, here are the results I received when I ran the experiment, along with the desired results (based on a 3.3-volt power supply):



You should, however, notice that the values you get are pretty consistent. So if you need repeatable results, you can rely on the ESP32 DAC to give them to you.

## Making Waves

One common use of a DAC is generating waveforms, which is a good use for the ESP32 DAC. While it won't be capable of laboratory-quality waveforms, it can be handy for simple waves and for creating sounds.

## Wave Hookup

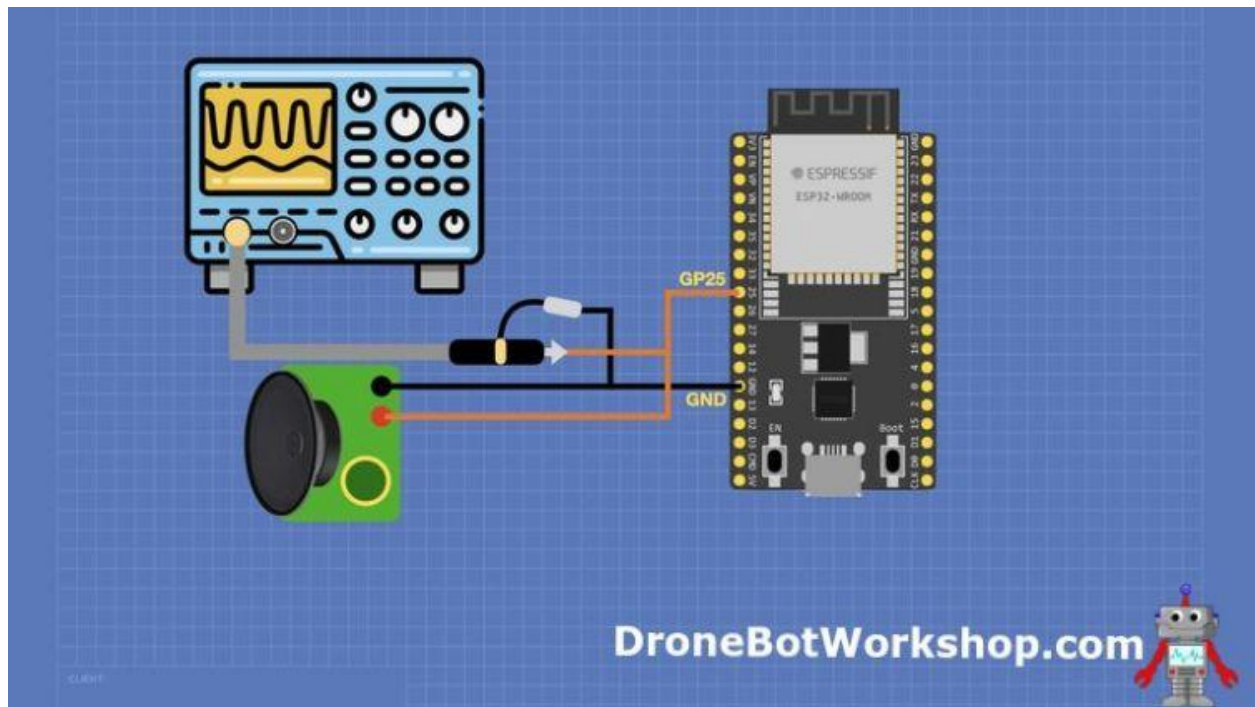
The best way to do these experiments is to use an oscilloscope. A scope will let you see the waveform, and most modern scopes will also measure its parameters.

You can also use an audio amplifier to listen to the resulting waveforms, which are all within audible range. If you do use an audio amplifier, make sure that it either has a

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

volume control or that you have a speedy way of silencing it, as the tones we will be creating are not very pleasant to listen to!

The following hookup diagram illustrates using both a scope and amplifier; you'll need at least one of them.



Once again, we are only using DAC Channel 1. If you wish, you can repeat the experiments with the second channel.

## Sine Wave Basic Code

Here is a very basic way of generating a sine wave using the ESP32 DAC:

<https://dronebotworkshop.com>

```
1  /*
2   ESP32 DAC - Simple Waveform Experiment
3   espdac-wave.ino
4   ESP32 DAC Waveform Demo
5   Outputs a Sine Wave
6
7   DroneBot Workshop 2022
8   https://dronebotworkshop.com
9  */
10
11 // Define DAC pins
12 #define DAC_CH1 25
13 #define DAC_CH2 26
14
15 void setup() {
16     // Nothing here!
17 }
18
19 void loop() {
20
21     // Generate a Sine Wave
22     // Step one degree at a time
23     for (int deg = 0; deg < 360; deg = deg + 1) {
24         // Calculate sine and write to DAC
25         dacWrite(DAC_CH1, int(128 + 64 * sin(deg * PI / 180)));
26     }
27 }
```

This method uses the *dacWrite* function again, which isn't really the best way to make a waveform as you are limited in frequency. It is really just for demonstration purposes.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

As you can see, all the action is in the loop, and one iteration of the loop is one cycle of the sine wave. We cycle through the sine wave steps and write them to the DAC.

This is really best seen on an oscilloscope, as shown here.:



As it is a very low frequency (I got about 86Hz), it will require a speaker with some bass response to hear it properly. The tiny breadboard speaker I used broke into convulsions trying to play it!

## Sine Wave with Table Code

Another way of generating a low-frequency waveform is to use a table to hold the waveform values. These are waveform level values, and using this method, you can generate a pretty complex waveform.

<https://dronebotworkshop.com>

Again, this sketch is more of a demonstration, but it could be adapted for practical purposes.

```
1  /*
2   ESP32 DAC - Waveform Table Experiment
3   espdac-wave-table.ino
4   ESP32 DAC Waveform Table Demo
5   Outputs a Sine Wave from table values
6
7   DroneBot Workshop 2022
8   https://dronebotworkshop.com
9  */
10
11 // Define DAC pins
12 #define DAC_CH1 25
13 #define DAC_CH2 26
14
15 // Define sample count
16 #define Num_Samples 112
17
18 // Integer for counting
19 int i = 0;
20
21 static byte SineWaveTable[Num_Samples] = {
22
23   0x80, 0x83, 0x87, 0x8A, 0x8E, 0x91, 0x95, 0x98, 0x9B, 0x9E, 0xA2, 0xA5, 0xA7,
24   0xAA, 0xAD, 0xAF,
25   0xB2, 0xB4, 0xB6, 0xB8, 0xB9, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF, 0xBF, 0xBF, 0xC0,
26   0xBE, 0xBD, 0xBC, 0xBB, 0xB9, 0xB8, 0xB6, 0xB4, 0xB2, 0xAF, 0xAD, 0xAA, 0xA7,
27   0xA5, 0xA2, 0x9E,
```



```
28     0x9B, 0x98, 0x95, 0x91, 0x8E, 0x8A, 0x87, 0x83, 0x80, 0x7C, 0x78, 0x75, 0x71,  
29     0x6E, 0x6A, 0x67,  
30     0x64, 0x61, 0x5D, 0x5A, 0x58, 0x55, 0x52, 0x50, 0x4D, 0x4B, 0x49, 0x47, 0x46,  
31     0x44, 0x43, 0x42,  
32     0x41, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x41, 0x42, 0x43, 0x44, 0x46,  
33     0x47, 0x49, 0x4B,  
34     0x4D, 0x50, 0x52, 0x55, 0x58, 0x5A, 0x5D, 0x61, 0x64, 0x67, 0x6A, 0x6E, 0x71,  
35     0x75, 0x78, 0x7C  
36 };  
37  
38 void setup() {  
39     // Nothing here!  
40 }  
41  
42 void loop() {  
43     // Step through table values  
44     dacWrite(DAC_CH1, SineWaveTable[i]);  
45     i++;  
46     if (i >= Num_Samples) i = 0;  
47 }
```

Once again, we are generating a sine wave, whose levels are contained in a 112-element array. This array is our “table”.

We cycle through the array and use our friend *dacWrite* to write the level to the DAC.

As with the last sketch, a scope is the best way to monitor the results. Try changing some of the array values and note the effect on the display. You can also try changing the number of elements in the array, which will also alter the output frequency.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



You can actually create some pretty complex waveforms with this method.

## Cosine Generator

The ESP32 DAC has a single Cosine Waveform Generator. You can use this to generate a cosine wave (which is a lot like a sine wave in shape) over a wide range of frequencies and amplitudes.

## Cosine Generator Library Code

When working with the Arduino IDE, it is much easier to use a library to use the cosine generator functions. I will use the DacESP32 Library, and you can install this library using your Arduino IDE Library Manager.

Here is a sketch to generate a 1KHz tone at several different levels:

<https://dronebotworkshop.com>

```
1  /*
2   ESP32 DAC - Waveform Library Experiment
3   espdac-wave-library.ino
4   ESP32 DAC Waveform Library Demo
5   Outputs a Cosine Wave
6   Uses DacESP32 Library - https://github.com/yellobyte/DacESP32
7
8   DroneBot Workshop 2022
9   https://dronebotworkshop.com
10 */
11
12 // Include DacESP32 Library
13 #include <DacESP32.h>
14
15 // Create DAC object
16 DacESP32 dac1(GPIO_NUM_25);
17
18 void setup() {
19
20     // Output a Cosine Wave with frequency of 1000Hz and max. amplitude (default)
21     dac1.outputCW(1000);
22
23     // Wait 5 seconds before changing amplitude
24     delay(5000);
25 }
26
27 void loop() {
28
29     // Change signal amplitude every second
```

```
30   for (uint8_t i = 0; i < 4; i++) {  
31       delay(1000);  
32       if (i == 0)  
33           dac1.setCwScale(DAC_CW_SCALE_1);  
34       else if (i == 1)  
35           dac1.setCwScale(DAC_CW_SCALE_2);  
36       else if (i == 2)  
37           dac1.setCwScale(DAC_CW_SCALE_4);  
38       else if (i == 3)  
39           dac1.setCwScale(DAC_CW_SCALE_8);  
40   }  
41 }
```

As you can see, the library really simplifies things. The *outputCW* method has just one parameter, the cosine wave frequency. You can also see how we use *setCwScale* to set the amplitude.



# Oscilloscope Art

I have to admit, “art” is a bit of a stretch here!

What we are going to do is use both ESP32 DAC channels to draw an ellipse on our oscilloscope. You can expand upon this to make something more artistic.

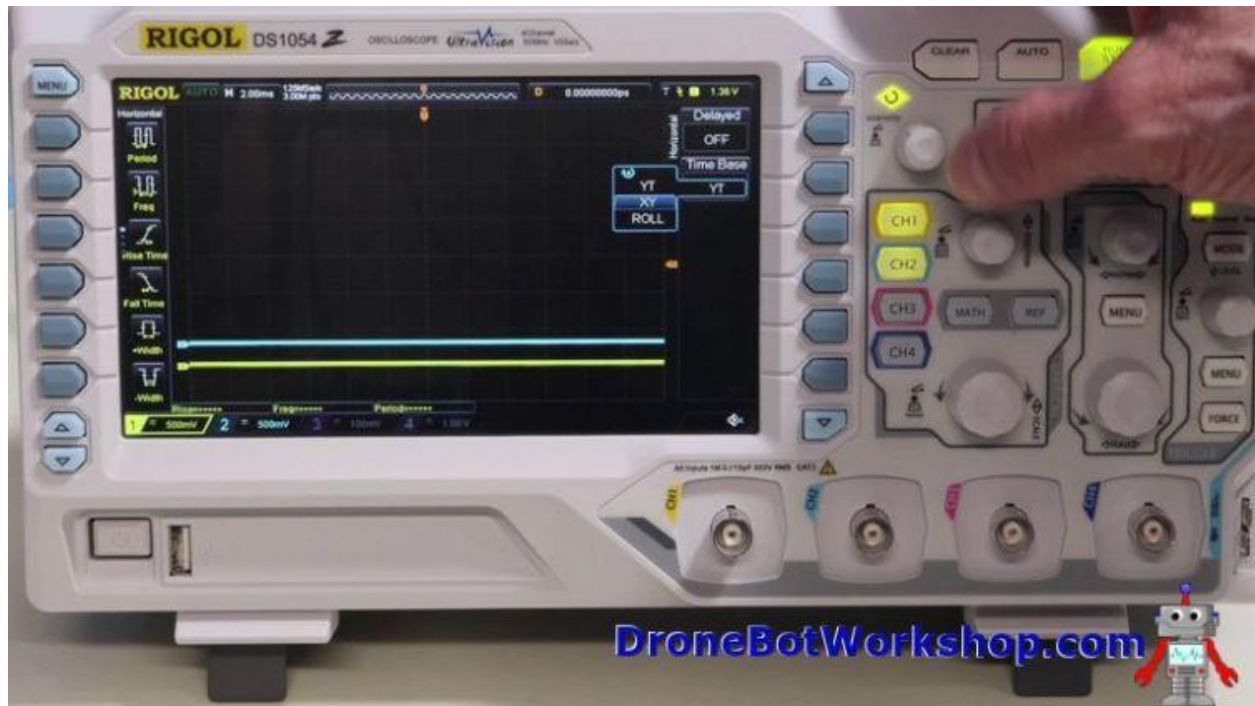
## Oscilloscope Settings

Obviously we will need a scope for this experiment, and this time we will need to use two channels.

In addition, **we will need to have our scope set to “X-Y Mode”**. In “X-Y Mode,” the X-axis of the display is modulated by one channel, and the Y-axis is modulated by a second channel. This is a feature that you should be able to find on any multichannel oscilloscope, both digital and analog.

The video accompanying this article shows you how I set my Rigol DS1054Z scope for X-Y Mode.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



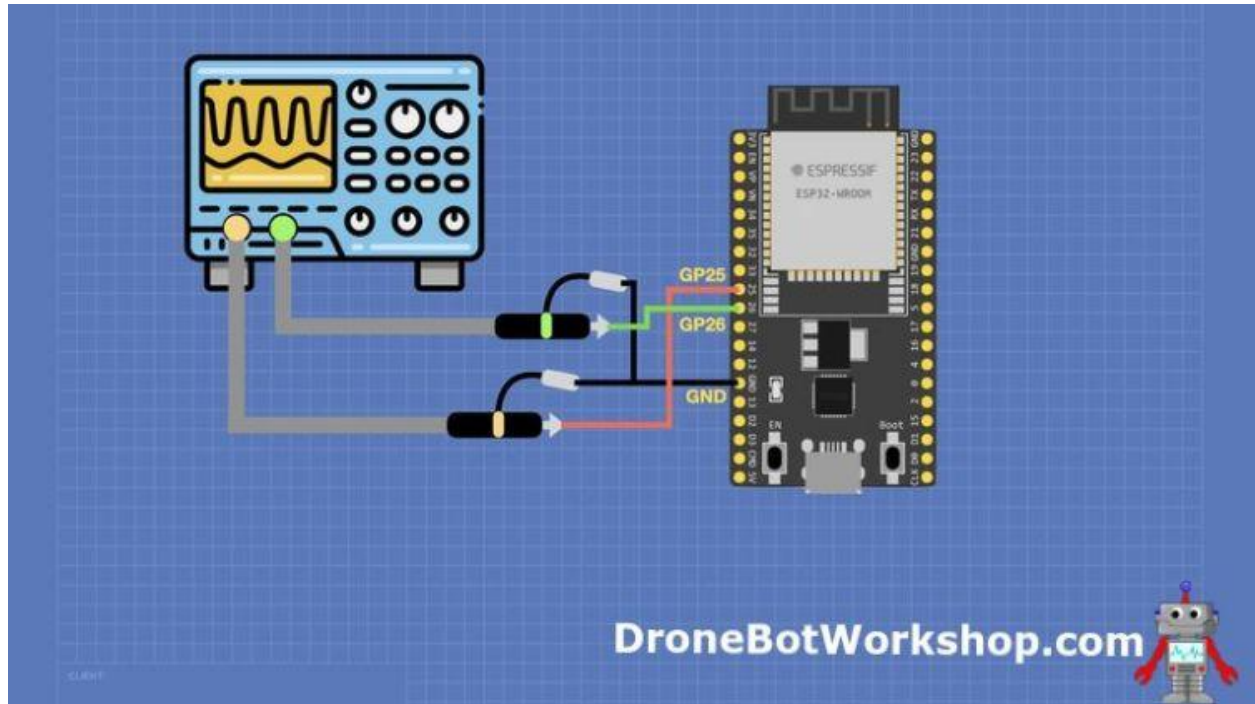
Each channel will be measuring about 3.3 volts, so set the probe and channel sensitivity accordingly.

## Oscilloscope Art Hookup

The hookup for our scope art project is pretty simple, you probably guessed it already!  
For those of you who didn't guess, here it is:

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



Note that although I show both scope probe grounds connected to an ESP32 ground, it is really only necessary to connect one.

## Oscilloscope Art Code

I can't in any way take credit for the code here; it comes from a very talented author who goes by the name of Bitluni. You can see more of his code on his [GitHub repository](#), and YouTube Channel.

The code uses a couple of internal ESP32 libraries, these were installed with the ESP32 Boards Manager in your IDE, so no extra installation is required.

<https://dronebotworkshop.com>



```
1  /*
2   ESP32 DAC  Oscilloscope Art Demo
3   espdac-scope-art.ino
4   Generates pattern on oscilloscope
5   Scope must be in X-Y mode
6   Original scope code by Bitluni - https://github.com/bitluni/OsciDisplay/
7   DroneBot Workshop 2022
8   https://dronebotworkshop.com
9  */
10
11 // Include DAC and Math Libraries
12 #include <driver/dac.h>
13 #include <math.h>
14
15 // Define variable for increment
16 float t = 0;
17
18 void setup() {
19     // Enable both DAC channels
20     dac_output_enable(DAC_CHANNEL_1);
21     dac_output_enable(DAC_CHANNEL_2);
22 }
23
24
25 void loop() {
26     // Move two channels through sine and cosine waves
27
28     // Increment variable
29     t += 0.01;
```

```
30
31 // Step through circle
32 for (float f = 0; f < M_PI * 2; f += 0.01) {
33     dac_output_voltage(DAC_CHANNEL_1, sin(f) * 120 + 120);
34     dac_output_voltage(DAC_CHANNEL_2, cos(f + t) * 120 + 120);
35 }
36
37 // Send DAC output HIGH to provide scope trigger on both channels
38 dac_output_voltage(DAC_CHANNEL_1, 255);
39 dac_output_voltage(DAC_CHANNEL_2, 255);
40
41 // Short delay
42 delay(10);
43 }
44
```

Basically, we generate the “art” by creating a sine wave in one channel and a cosine wave in the other one. We do this while changing the value of a float that represents the position in a circle.

It also sends a pulse at the end of each iteration, this is to sync the scope.

Once you get the circle (or ellipse) displaying on your scope, you can experiment by changing the sensitivity and horizontal time base for each channel, this will distort the waveform.

This “art” is just perfect for your next science fiction movie!

## Musical Fruit

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

This is definitely one of the tastiest experiments we have ever worked on here in the DroneBot Workshop!

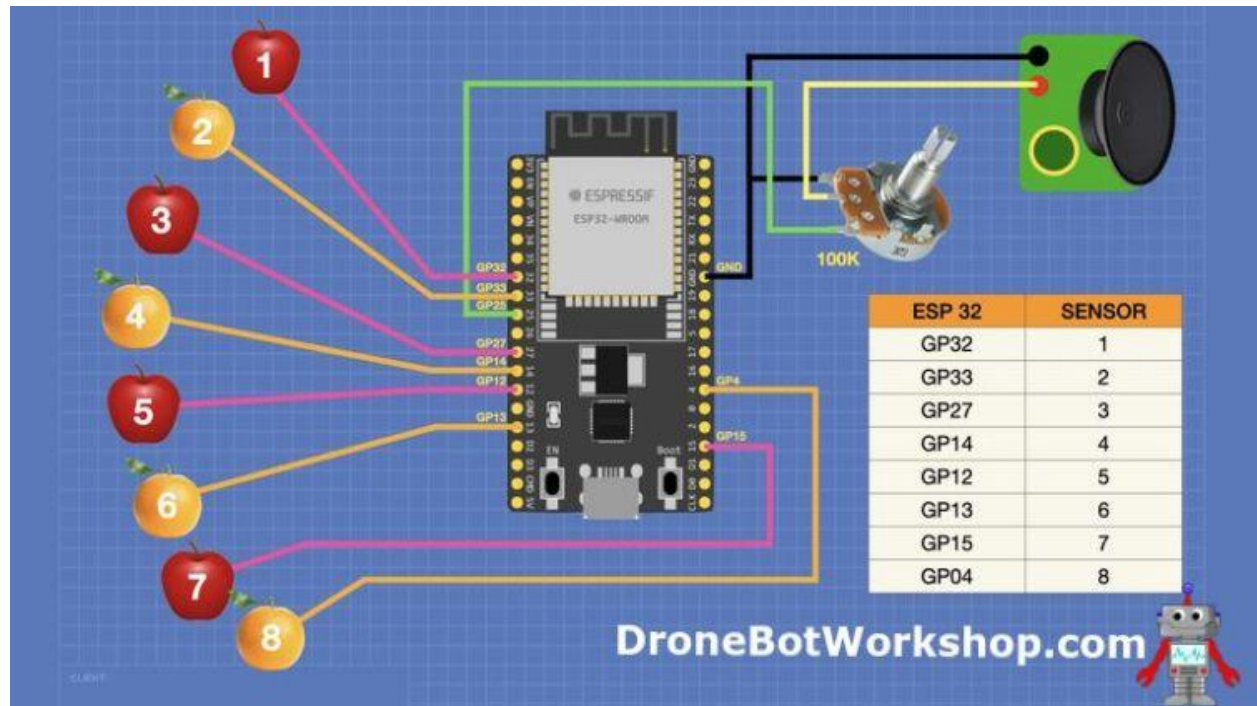
We will use one of the ESP32 DAC channels along with another ESP32 feature – its touch switch capability.

The ESP32 has 10 GPIO ports that can be used as touch switches. And while you can use them with any conductive surface, I have chosen to use fruit!

I'm pairing up (or should that be "pearing up"?) some oranges with a few rather tiny Granny Smith apples to create a fruity keyboard whose output will go to the ESP32 DAC.

## Musical Fruit Hookup

Here is how I hooked up a fruit basket to an ESP32! You'll also need an amplifier module or an external audio amplifier. I would hold off on using a guitar amplifier until after you have mastered the fruity instrument!



There is a potentiometer shown in the diagram. I used a 100k linear-taper pot, but an audio taper would be a better choice (I didn't have one), as what we have here is basically a volume control. If your amplifier already has a volume control, then you can just connect its input directly to GPIO pin 25 and eliminate the pot.

Of course, you aren't obliged to use fruit, or any sort of produce as a touch switch. Anything conductive will work fine, and you can "fine-tune" each sensor in the code.

## Musical Fruit Code

Here is the code I used to turn a bunch of apples and oranges into a musical instrument:

```
1  /*
2   ESP32 DAC - Musical Fruit
3   espdac-touch-music.ino
4   ESP32 DAC & Touch Switch Demo
5   Uses DacESP32 Library - https://github.com/yellobyte/DacESP32
6
7   DroneBot Workshop 2022
8   https://dronebotworkshop.com
9  */
10
11 // Include DacESP32 Library
12 #include <DacESP32.h>
13
14 // Create DAC object for Channel 1
15 DacESP32 dac1(GPIO_NUM_25);
16
17 // Define the touch pins (you can add more as desired)
18 #define TOUCH_1 32
19 #define TOUCH_2 33
20 #define TOUCH_3 27
21 #define TOUCH_4 14
22 #define TOUCH_5 12
23 #define TOUCH_6 13
24 #define TOUCH_7 15
25 #define TOUCH_8 4
26
27 // Variables to hold the touch pin values
28 int tvalue_1;
29 int tvalue_2;
```

```
30 int tvalue_3;
31 int tvalue_4;
32 int tvalue_5;
33 int tvalue_6;
34 int tvalue_7;
35 int tvalue_8;
36
37 // Define the threshold levels for each touch pin (adjust as required)
38 const int threshold_1 = 200;
39 const int threshold_2 = 200;
40 const int threshold_3 = 200;
41 const int threshold_4 = 200;
42 const int threshold_5 = 200;
43 const int threshold_6 = 200;
44 const int threshold_7 = 200;
45 const int threshold_8 = 200;
46
47 // Define the frequencies for our "musical notes" -
48 https://mixbutton.com/mixing-articles/music-note-to-frequency-chart/
49 const int freq_1 = 523;    //C - Octave 5
50 const int freq_2 = 587;    //D - Octave 5
51 const int freq_3 = 659;    //E - Octave 5
52 const int freq_4 = 698;    //F - Octave 5
53 const int freq_5 = 784;    //G - Octave 5
54 const int freq_6 = 880;    //A - Octave 5
55 const int freq_7 = 988;    //B - Octave 5
56 const int freq_8 = 1046;   //C - Octave 6
57
58 void setup() {
    // Setup serial monitor to check touch thresholds
```

```
59   Serial.begin(115200);
60
61   // Disable DAC to stop sound
62   dac1.disable();
63 }
64
65 void loop() {
66
67   //Check status of touch switches
68   tvalue_1 = touchRead(TOUCH_1);
69   tvalue_2 = touchRead(TOUCH_2);
70   tvalue_3 = touchRead(TOUCH_3);
71   tvalue_4 = touchRead(TOUCH_4);
72   tvalue_5 = touchRead(TOUCH_5);
73   tvalue_6 = touchRead(TOUCH_6);
74   tvalue_7 = touchRead(TOUCH_7);
75   tvalue_8 = touchRead(TOUCH_8);
76
77   // Print values (useful for adjusting threshold levels)
78   Serial.print("S1 = ");
79   Serial.print(tvalue_1);
80   Serial.print(" S2 = ");
81   Serial.print(tvalue_2);
82   Serial.print(" S3 = ");
83   Serial.print(tvalue_3);
84   Serial.print(" S4 = ");
85   Serial.print(tvalue_4);
86   Serial.print(" S5 = ");
87   Serial.print(tvalue_5);
```



```
88 Serial.print(" S6 = ");
89 Serial.print(tvalue_6);
90 Serial.print(" S7 = ");
91 Serial.print(tvalue_7);
92 Serial.print(" S8 = ");
93 Serial.println(tvalue_8);
94
95 // If touch values exceed threshold then play associated note3
96 if (tvalue_1 < threshold_1) {
97     dac1.enable();
98     dac1.outputCW(freq_1);
99 } else if (tvalue_2 < threshold_2) {
100     dac1.enable();
101     dac1.outputCW(freq_2);
102 } else if (tvalue_3 < threshold_3) {
103     dac1.enable();
104     dac1.outputCW(freq_3);
105 } else if (tvalue_4 < threshold_4) {
106     dac1.enable();
107     dac1.outputCW(freq_4);
108 } else if (tvalue_5 < threshold_5) {
109     dac1.enable();
110     dac1.outputCW(freq_5);
111 } else if (tvalue_6 < threshold_6) {
112     dac1.enable();
113     dac1.outputCW(freq_6);
114 } else if (tvalue_7 < threshold_7) {
115     dac1.enable();
116     dac1.outputCW(freq_7);
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
11   } else if (tvalue_8 < threshold_8) {  
1     
11     dac1.enable();  
11     dac1.outputCW(freq_8);  
11   } else {  
3     // Disable DAC to stop sound  
11     dac1.disable();  
4       
11   }  
11     
5     
11   // Short delay (adjust as desired)  
6     
11   delay(200);  
11     
7   }
```

```
11  
8
```

```
11  
9
```

```
12  
0
```

```
12  
1
```

```
12  
2
```

```
12  
3
```

```
12  
4
```

```
12  
5
```

```
12  
6
```

```
12  
7
```

As you can see, the code is based upon the DacESP32 Library to provide a simple way of outputting tones.

<https://dronebotworkshop.com>

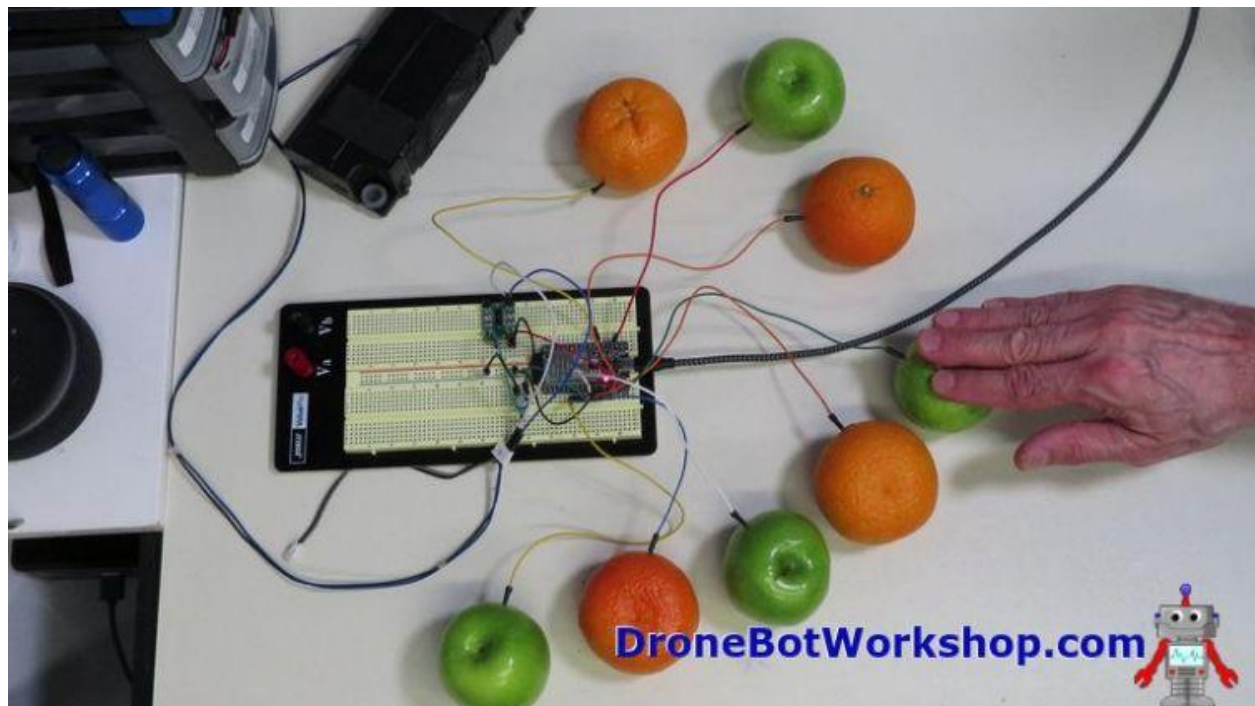
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Each touch switch has a threshold defined; by default, it is 200. You may need to experiment with this value, as it sets the sensitivity of your fruity sensor.

The frequencies are for the 5th (and part of the 6th) octave, but you can substitute other ones.

We print the value of the touch switch sensors to the serial monitor, this can be useful when adjusting the touch switch sensitivity.

In the loop, we check the status of each switch. If we see it as being touched, then we play the corresponding tone. If we don't see any switch (i.e., fruit) being touched, then we turn off the DAC.



Hook it up and get prepared to serenade the world. And when you get tired of it, you can eat your creation!

<https://dronebotworkshop.com>

## Conclusion

The DAC is one of those many features of the ESP32 that you don't think of often, but that doesn't mean it isn't useful.

Consider using it to drive analog panel meters for a “retro project”, or to create cool tones for a toy or robot. Or to create an orchestra with a bowl of fruit!