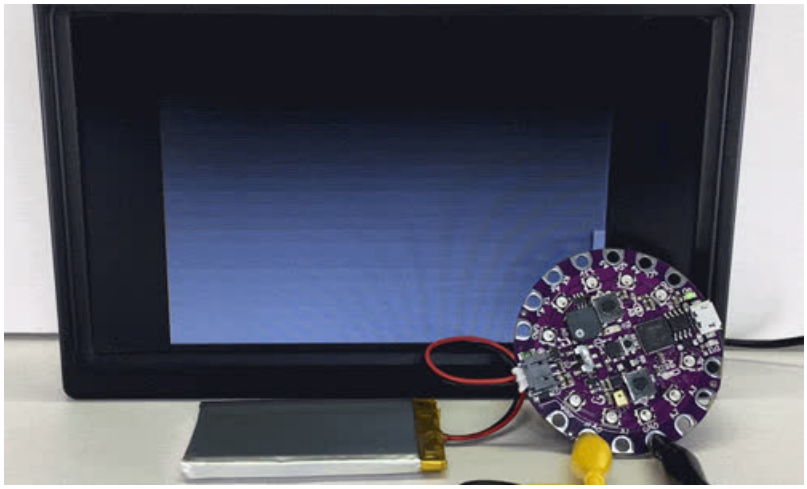
adafruit learning system

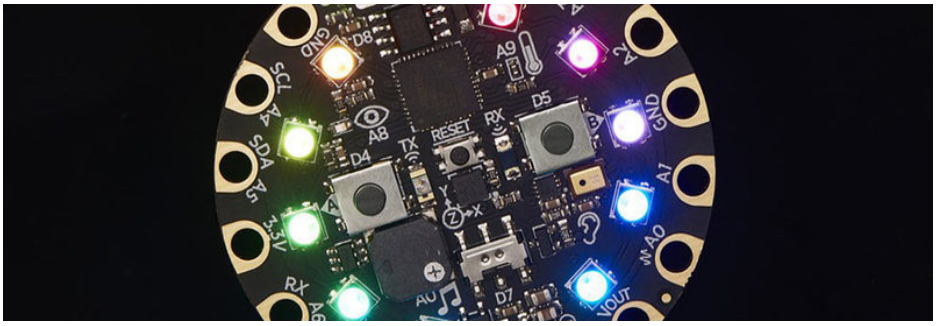
Circuit Playground Express (& other ATSAMD21 Boards) DAC Hacks

Created by Phillip Burgess



Last updated on 2020-05-18 07:38:58 PM EDT

Overview



With each **new generation** of microcontrollers we tend to dwell on quantifiables like **memory** and **speed** — bigger, faster programs! At the same time, new devices often bring *additional capabilities* that are overlooked at first glance. These features open whole new doors, *beyond* what program size or speed can do.

The SAMD ARM M0 microcontroller used in Circuit Playground Express, Feather M0 and other Adafruit boards with the "Express" or "M0" designations — along with the Arduino Zero — include some intriguing new features, among them:

- **A *digital-to-analog converter* (DAC).** Pin A0 can provide a **true analog voltage** between 0 and 3.3 Volts. Previously, Arduino's so-called analogWrite() function wasn't *really* analog — it generated a pulse-width-modulated *digital* signal.
- ***Direct memory access* (DMA)** allows data transfers between memory and peripherals (including the DAC) very quickly and without CPU intervention — it goes about its task in the background while other code continues to run at 100% speed.

We'll demonstrate by generating **composite TV** and **AM radio** signals straight from the board. **No** shields or breadboards or soldering extra components, just some simple test leads!

While the projects shown here have a vintage rinky-dink flair, the fact that a microcontroller can do this *entirely on its own* — no extra parts, just some wires — is pretty remarkable. Rather than just thinking bigger and faster, what unconventional ideas and applications might you hatch from new hardware? None of this is really what the DAC is intended for, but it's cool in a demo-scene kind of way.

Getting Started

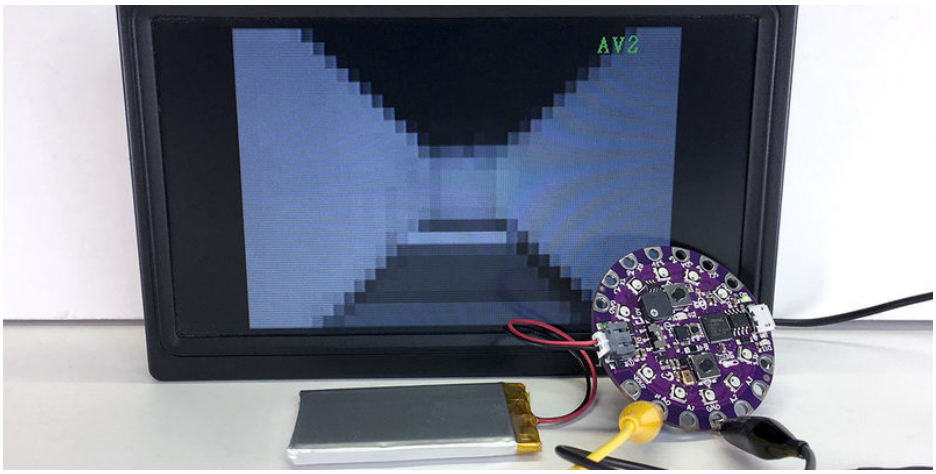
These demo projects will require:

- An Atmel **SAMD M0**-based microcontroller board such as Circuit Playground Express (https://adafru.it/wpF), Feather M0 (https://adafru.it/s1d) or Arduino Zero (https://adafru.it/rTf). 8-bit AVR boards and "classic" 8-bit Circuit Playground are not compatible.
- Corresponding board support enabled in the Arduino IDE: **Tools→Board→Boards Manager...** Adafruit boards require an extra step first, explained in this guide (https://adafru.it/ldF).
- For Circuit Playground Express: some alligator clip test leads (https://adafru.it/dWJ). For other boards, some solid-core wire.

To confirm that SAMD board support is working, try uploading the basic "blink" sketch to a board. To confirm the Adafruit_ZeroDMA library is correctly installed, check that the Files→Examples→Adafruit_ZeroDMA rollover menu is present.

Each of the projects that follow will require its own additional library, again manually installed.

Composite Video Output



The DAC is *just* fast enough to generate low-resolution **composite video** that can be viewed on a television or monitor with composite video input (typically a yellow **RCA connector**).

There are very few pixels, and it's only grayscale, but it's sufficient for creating simple games or to print readings from sensors.

To use this, download and manually install the **Adafruit_CompositeVideo** library:

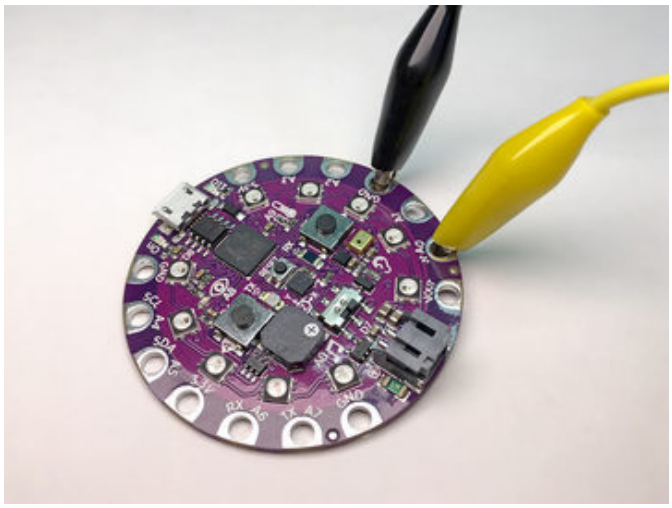
https://adafru.it/wAe

https://adafru.it/wAe

This also requires the **Adafruit_GFX** library, which is much easier to install using the Arduino Library Manager: **Sketch→Include Library→Manage Libraries...** (enter "**GFX**" in the search field).

Earlier versions (pre-1.8.10) also require installing **Adafruit_BusIO** (newer versions will handle this one automatically).

After the Adafruit_CompositeVideo library is installed, there are a couple of example sketches. One prints the current value from the Circuit Playground light sensor, another shows large horizontal-scrolling text.

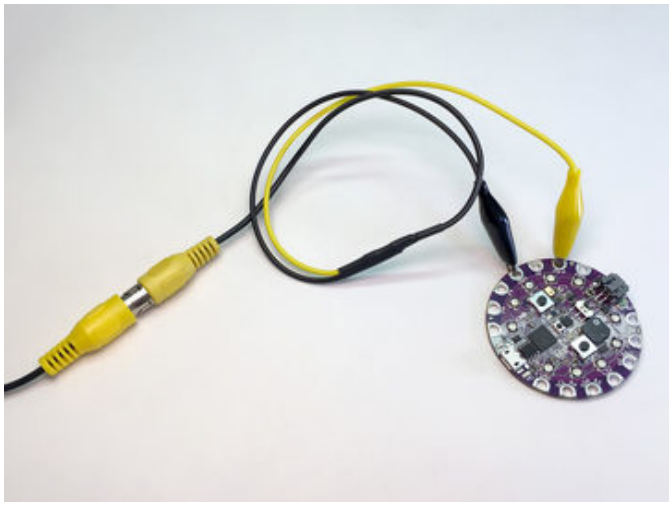


Connect a couple of test leads to **pin A0** and any **ground** pin.

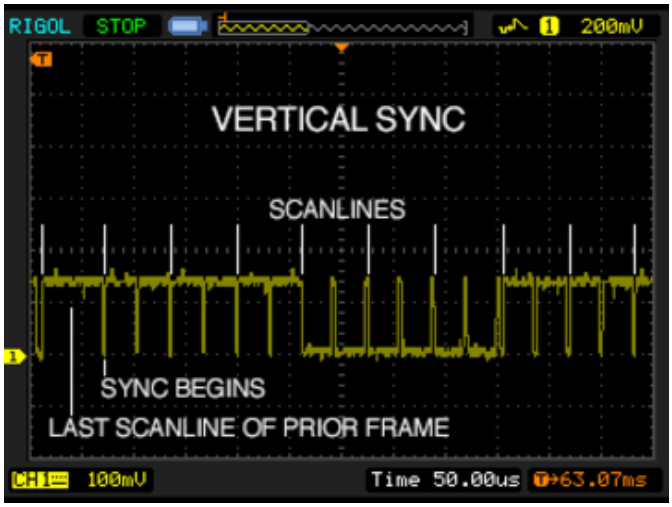
At the other end, connect **A0** to the "**tip**" (center) of the composite video connector, and **ground** to the "**ring**" (outside).

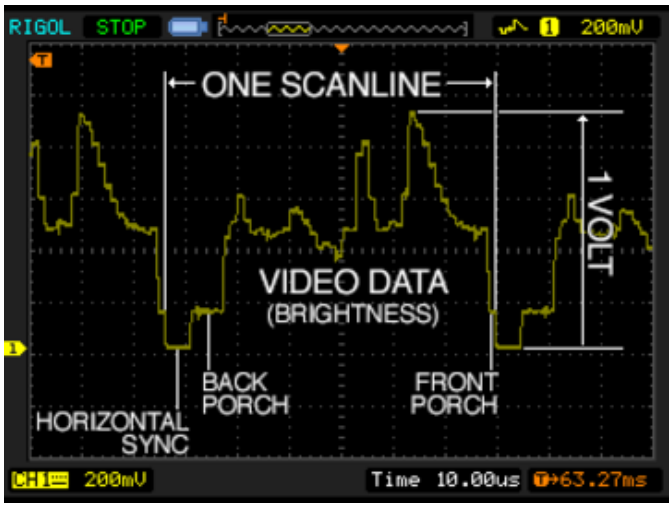
Depending on the TV/monitor connection and available cabling, you may need a spare composite cable or male-to-male adapter to get something you can clip onto.

Since I'll be testing the code often, I cobbled together a somewhat more permanent connector from a spare cable and test leads, but it's not necessary to go to such lengths if just trying it out.



How It Works





NTSC video runs at 29.97 frames per second. Each frame is comprised of 525 horizontal scanlines. Each scanline starts and ends with carefully-timed synchronization signals, with image data in-between: an analog voltage from about 0.3 to 1.0V determines the brightness at that point along the scanline.

Twice per frame, there are also vertical synchronization signals following a specific timing and pattern.

Some of these “blips” are just a couple of microseconds long! Digital outputs can easily manage such timing, but for the SAMD DAC this is challenging...the makeshift video signal is *just good enough* for most screens to latch on to.

Creating New TV Projects

To use the library, add these two lines at the top of your sketch:

```
#include <Adafruit_GFX.h>
#include <Adafruit_CompositeVideo.h>
```

Then, before the setup() function, declare a global object of type **Adafruit_NTSC40x24**:

```
Adafruit_NTSC40x24 display;
```

(It’s called this just in case other resolutions are supported in the future...but don’t hold your breath, I’ve tried going higher and the DAC can’t quite make a stable image.)

Then, inside your setup() function, call the object’s begin() function to enable composite video out on the A0 pin:

```
display.begin();
```

Because it builds upon the Adafruit_GFX library, all the same drawing functions (https://adafru.it/doL) (including fonts) are available as with our other Arduino-compatible displays. “Colors” passed to the drawing functions should be **8-bit grayscale values** (**0 to 255**, where 0=black, 255=white).

```
display.drawLine(0, 0, 39, 23, 128); // Gray line, corner-to-corner
display.setTextColor(255);           // White text
display.print("Hello World");
```

Limitations

- **Circuit Playground Express speaker is disabled;** tone() and other audio code **will not work** in combination with this
- **40x24** pixel resolution; actual usable area **may be slightly smaller** due to overscan
- **Grayscale only**

Adafruit_CompositeVideo and Adafruit_AMRadio (on the next page) both use the DAC peripheral and the same timer/counter; the two libraries **can not be used at the same time**.

The video resolution is extremely crude...it’s more a novelty than anything else. If you need high-quality visuals from a small board, consider a Raspberry Pi Zero!

Folks have generated much sharper video (with color, even!) from much more modest hardware. These all require extra components though. The benefit to this simple gator-clip approach is that classrooms might not allow soldering, or a lesson might not have time for assembling parts on a breadboard. Or it’s just fun showing off.

So there’s NTSC, but what about PAL video?

It’s not in there. And unless you’re actually using a really old CRT telly, it’s probably not necessary. Most, if not all, LCD monitors that handle composite video will automatically detect and adapt to the video signal, so NTSC is fine. This is true even if you are in a "PAL zone" like Europe!

Why not color?

Composite color video is *insane* and would require a DAC *orders of magnitude* faster. Let's see where microcontrollers are in a few years!

Transmitting AM Radio

...kind of. Temper your expectations. :)

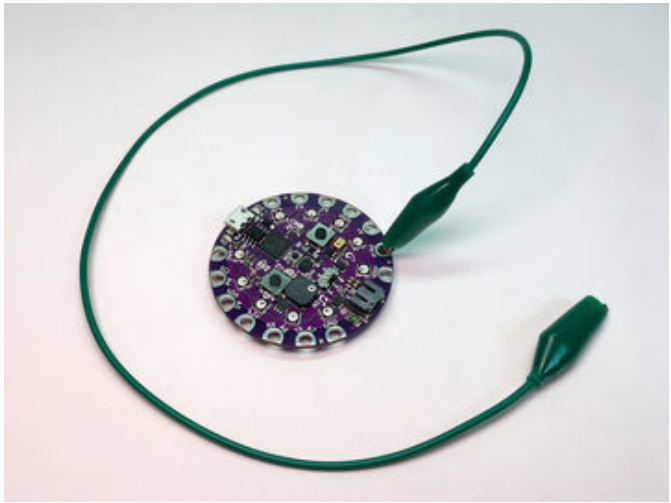
Another task we can use this fast DAC for is generating **AM radio** waveforms, which can be heard on a regular AM receiver tuned to the right frequency and held *very close by* (power is limited and an ideal antenna is impractically long, but it's a fun proof of concept).

This too requires a library:

https://adafru.it/wAf

https://adafru.it/wAf

After the Adafruit_AMRadio library is installed, there are a couple of example sketches. One plays the Jeopardy theme song over the AM 540 KHz frequency, the other plays a Godzilla roar sound.



Clip a test lead or connect a length of wire to **pin A0** as a makeshift **antenna**. Just one end...the other is left unconnected. This is far from an optimal antenna, but we need *something* there.

An *ideal* antenna would be something like 450 feet long...clearly that's not gonna happen. The test lead will do fine.

Creating New Radio Projects

To use the library, add this line at the top of your code:

```
#include <Adafruit_AMRadio.h>
```

Before the setup() function, declare a global object of type **Adafruit_AMRadio:**

```
Adafruit_AMRadio radio;
```

Then, inside your setup() function, call the object's begin() function to start it running. By default this will transmit at 540 KHz, but you can optionally pass an integer argument, the desired frequency in Hertz:

```
radio.begin(530000); // Transmit at 530 KHz instead
```

Try to keep this **as low as possible**, but still within the AM band (530 to 1700 KHz).

It won't run at *precisely* this frequency...the DMA clock has to run at some integer divisor of the 48 MHz CPU clock...so it will pick the closest thing it can muster, which may be a few megahertz to either side. If your AM radio has analog tuning you can dial it in for the best reception, like the old days.

The **tone()** function can be used for playing notes — similar to the normal Arduino tone() function — which accepts a frequency in Hertz and a duration in milliseconds (unlike Arduino's tone(), the duration is *required* here). For example, to play **middle C** (262 Hertz) for **one half second** (500 milliseconds):

```
radio.tone(262, 500);
```

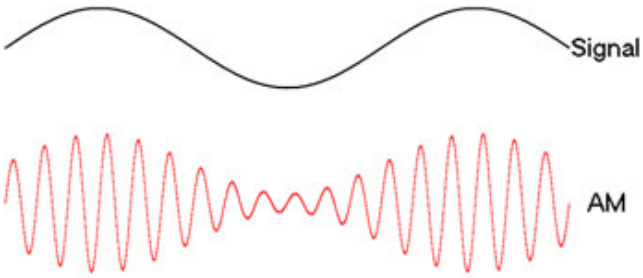
If you need more granular control over the audio waveform, use the radio.write() function to control the wave directly, passing a value from **0 to 1023** (the library's equivalent of Arduino's 10-bit analogWrite() function). For example, a neutral level:

```
radio.write(512);
```

This alone does not generate a sound. You then need to call write() **repeatedly and quickly** to generate an **audio waveform**. This can be seen in the "zilla" example sketch, which reads from a digitized audio sample stored in program memory and calls the write() function roughly 11,025 times a second.

Any existing code that uses analogWrite(A0) to generate sound through the Circuit Playground speaker can be easily modified to use the radio library instead.

How It Works



Amplitude modulation (AM) — the earliest method of sound transmission over radio — conveys a relatively low-frequency variable audio wave (such as voice or music, up to a few kilohertz) into a much higher fixed-frequency radio wave (500 KHz or more), called the *carrier wave,* by...you guessed it...*modulating* the *amplitude* of the carrier wave in direct proportion to the sound wave's shape.

Image credit: Wikimedia Commons contributor *Berserkerus,* CC-SA

The DAC is *barely* fast enough to generate a reasonable carrier wave for the lower end of the AM radio band. Our library simply adjusts the peaks and troughs of this wave in response to the Arduino sketch code.

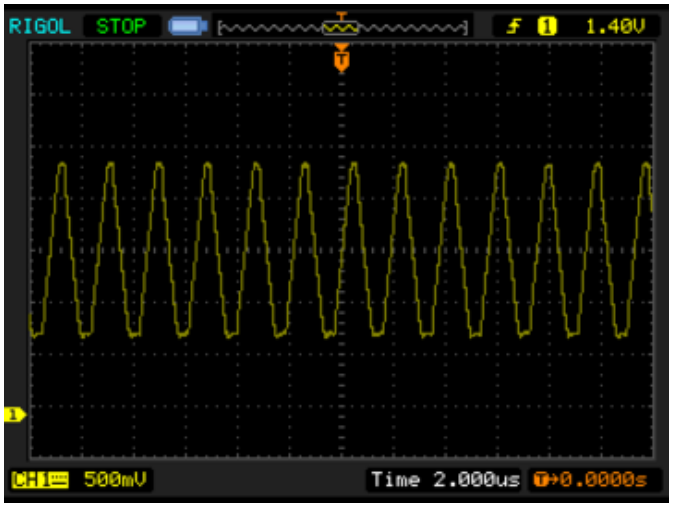
Actually the DAC *isn't* fast enough for this. We're cheating! Generating a 540 KHz square wave requires 1,080 *kilosamples per second* from the DAC, but it's really only rated for 350 Ksps. We simply feed it at the faster rate. This is not harmful in any way to the DAC, the output just isn't numerically precise until it's fully "settled" (the 350K rate), and we're interrupting it before it gets all the way there. It's reasonably close though. The video library does something similar, but not *quite* as fast, as that one does require a *little* more precision.

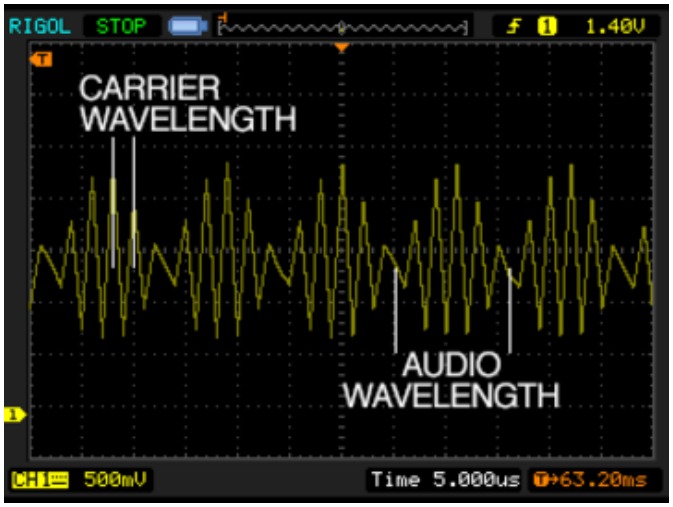
This is also why it only works toward the lower end of the AM band. As the frequency increases, the DAC output precision decreases.

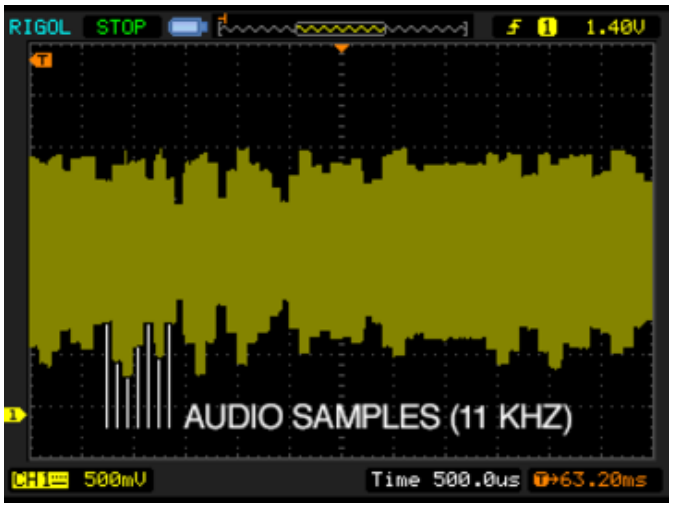
Zooming *way in* with an oscilloscope, the 540 KHz carrier wave is visible. Though we're feeding the DAC a square wave, the slow "settling time" produces this truncated triangle wave. This works to our benefit, as the carrier should ideally be a sine wave, and this is a coarse but acceptable facsimile.

Zooming out a bit, you can see the carrier wave amplitude (height) being modulated by the lower-frequency sound wave.

Zooming out still further, the individual audio samples from a digitized Godzilla roar — 11,050 per second —







can be seen. The high-frequency carrier wave is so much smaller by comparison, it appears solid on the scope.

Limitations

- **Circuit Playground Express speaker is disabled;** tone() and other audio code **will not work** in combination with this
- **Range is extremely limited**, just a few inches — this is “science project” fun and not a serious radio transmitter!
- Limited to **lower AM band**; example code uses **540 KHz**

Adafruit_CompositeVideo (on the prior page) and Adafruit_AMRadio both use the DAC peripheral and the same timer/counter; the two libraries **can not be used at the same time.**

Isn't broadcasting without a license illegal? Even with an amateur license, isn't broadcasting in this frequency band illegal? Will the FCC haul me away?

Maybe in some ultra-pedantic interpretation, but the range is *so* limited (less than a foot) it can't *possibly* interfere with other receivers or devices, so this shouldn't be a problem. It's only "broadcasting" if targeting a wider audience. This is "low-grade noise."

But hey, if the experiment piques your interest, why not study for an amateur radio license?