Making oscilloscope images with DACs

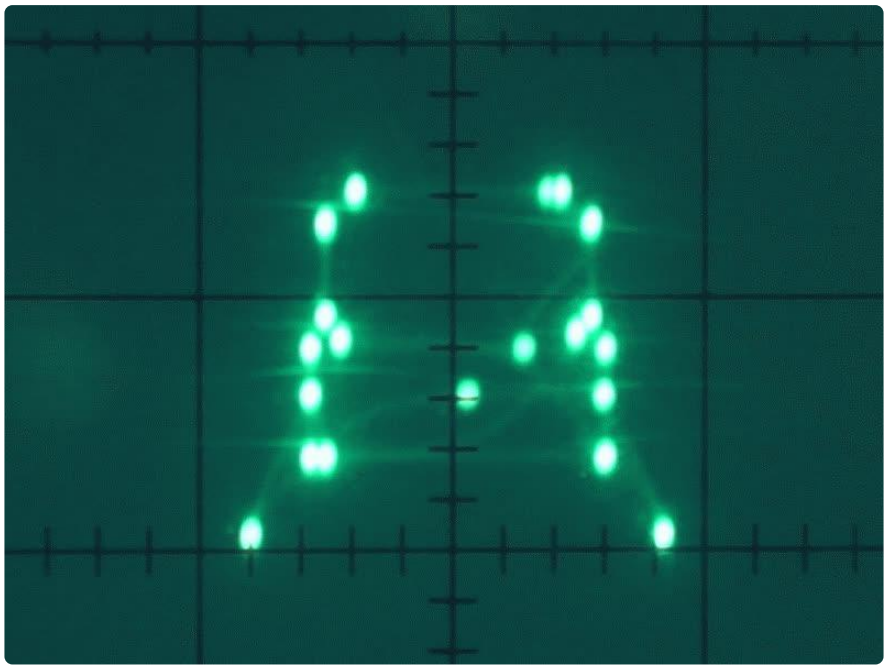
Created by Kevin Walters

https://learn.adafruit.com/dac-oscilloscope-images

Last updated on 2023-08-29 04:11:31 PM EDT

Table of Contents

Overview



This project demonstrates two techniques for making images on an oscilloscope using CircuitPython and the analogue output(s) found on many Adafruit boards.

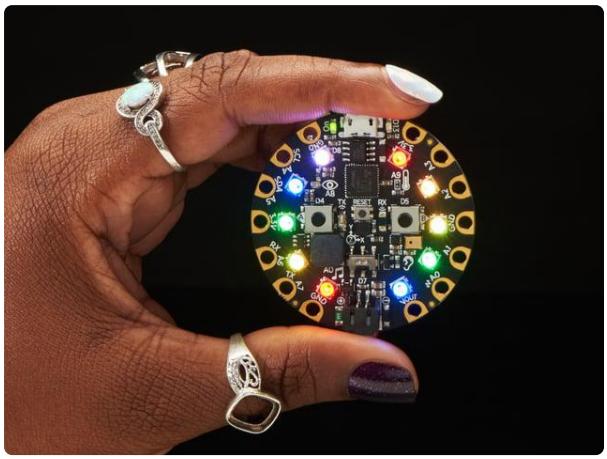
The first is an unusual technique using the oscilloscope's trigger feature and normal x-axis timebase with a single digital to analogue converter () (DAC) output on a Circuit Playground Express (CPX) board. A computer running Python with the imageio () librar y is required to convert bitmaps into a suitable format for playback on the CPX board.

The second is the more common X-Y vector technique, using a PyGamer with its two DAC outputs.

Any SAMD21 (M0) or SAMD51 (M4) board can be used. No additional hardware is required beyond connections to the oscilloscope probes.

Thank-you to Nick for the loan of a Hameg HM203-6 oscilloscope.

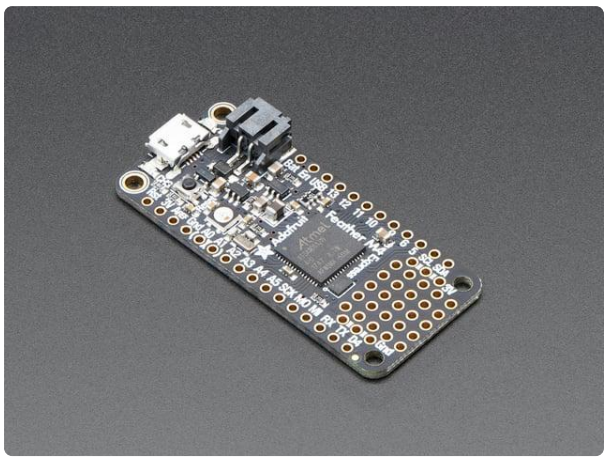
Parts



Circuit Playground Express

Circuit Playground Express is the next step towards a perfect introduction to electronics and programming. We've taken the original Circuit Playground Classic and...

https://www.adafruit.com/product/3333



Adafruit Feather M4 Express - Featuring ATSAMD51

It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...

https://www.adafruit.com/product/3857

Adafruit PyGamer for MakeCode Arcade, CircuitPython or Arduino

What fits in your pocket, is fully Open Source, and can run CircuitPython, MakeCode Arcade or Arduino games you write yourself? That's right, it's the Adafruit...

https://www.adafruit.com/product/4242

USB cable - USB A to Micro-B
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

Common Display Devices

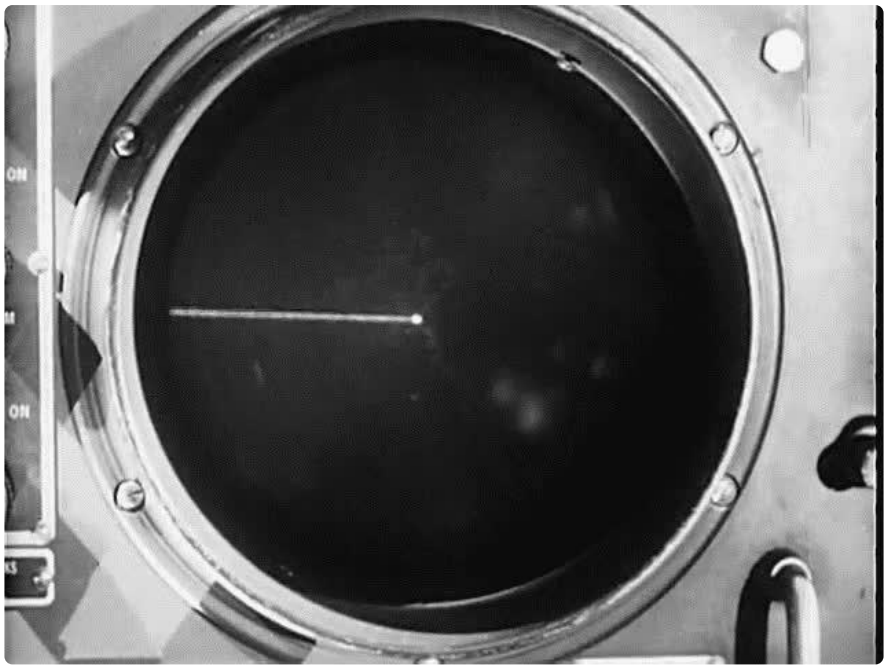
Raster Displays

The first computer monitors used single colour cathode-ray tubes () (CRT). The beam from the electron gun in a CRT can be directed in two dimensions at the phosphor () screen and often varied in intensity to control the brightness of the spot. CRT-based monitors typically use a raster scan () at a fixed rate to rapidly draw a bitmap image. If the refresh rate () (maximum frame rate) is high enough then the image is perceived as flicker-free.

Modern computer monitors use backlit LCD or LED.

Vector Displays

The CRT can also be used in other ways, early radar () displayed targets (reflections) using a slow circular radial scan and longer persistence phosphors. The animation below shows an example from Radar and Its Applications (1962) ().



CRTs were also used for vector monitors where the beam was directed to create a line-based image rather than using a fixed, grid-like scan pattern. Since these lines were not limited by the pixels of a (low resolution) bitmap display they could produce higher definition graphical output. This style of display was adopted for a few arcade games, Asteroids () is a well-known one. A much earlier game called Spacewar! () prec eded Asteroids. Colour was introduced by Atari for the Star Wars (1983) () arcade game but as the cost of video memory decreased and the resolution of video cards improved the demand for and use of vector displays diminished.

The first oscilloscopes used CRTs too - these are now sometimes referred to as cathode-ray oscilloscopes (CRO) to differentiate them from the modern flat panel () di gital storage oscilloscopes (DSO). Oscilloscopes are normally used for inspecting electrical signals but they can also be used to display images in various ways.

Image Creation with DAC

The "analogue" outputs on ATmega328 ()-based Arduino boards are PWM digital outputs () rather than true analogue outputs. The faster SAMD range includes digital to analogue converters:

- SAMD21 () (M0) - one 10bit DAC 0V-3.3V, maximum 350 kilosamples per second,
- SAMD51 () (M4) - two 12bit DACs 0V-3.3V, maximum 1 Megasample per second.

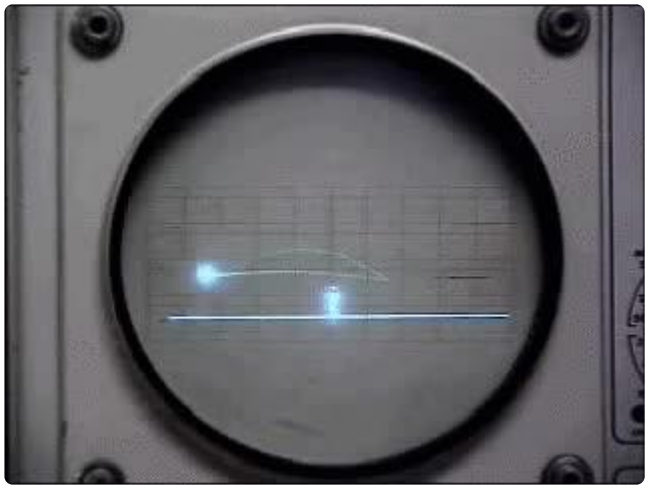
DACs are commonly used for audio but they can be used to create any electrical signal. Adafruit Learn: Circuit Playground Express (& other ATSAMD21 Boards) DAC Hacks () shows how to create low resolution composite video and an AM radio signal

in C/Arduino. A compiled language with predictable execution speed is generallly more suitable for DAC output. CircuitPython can be used for high rate DAC output with the aid of a built-in library.

Two DACs

Two analogue outputs allow control of the beam on an x-y oscilloscope. The beam needs to be moved gradually between the start and end of each line to draw a line. The large animation at the top of Overview () page shows the lines being progressively interpolated () to form increasingly solid-looking set of lines.

Tennis for Two () (short, looping clip shown below) was a very early game in 1958 using an oscilloscope as a display.



This technique works well on both a CRO and DSO.

PC audio cards feature DACs and also can be used to create interesting x-y oscilloscope output. The x-y signals can be crafted to some degree to also playback as music, Jerobeam Fenderson () has produced many impressive examples of this.

Three DACs

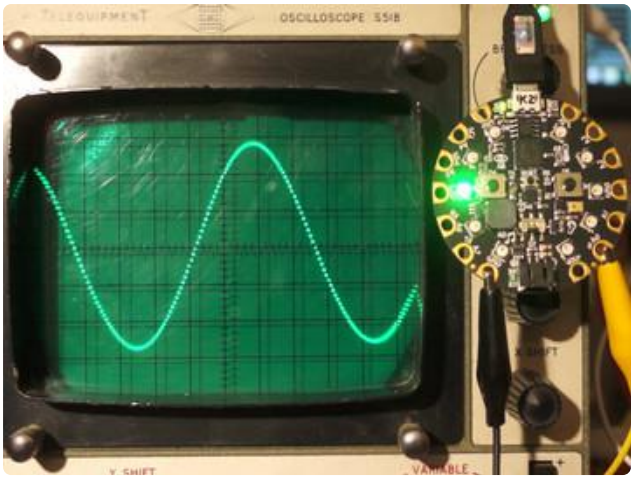
Some CROs have an advanced feature referred to as x-y-z mode where an additional z input can be used to control the beam intensity. This gives it capabilities similar to a black and white monitor/television.

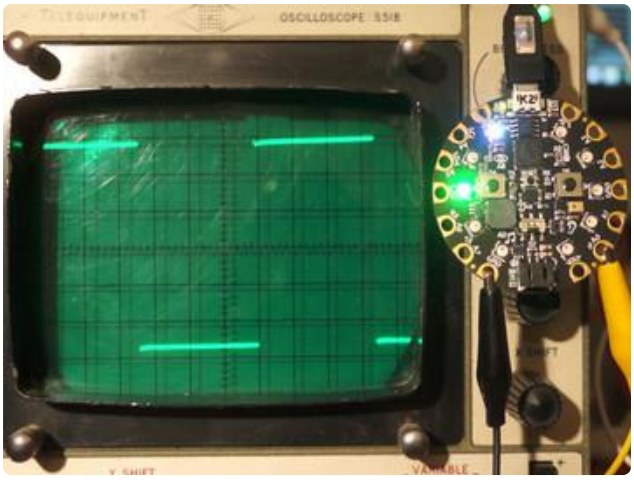
The Asteroids arcade game used discrete intensity to vary brightness including turning the beam off between objects. This can be seen in Displaying Asteroids XY on an analog oscilloscope () (YouTube).

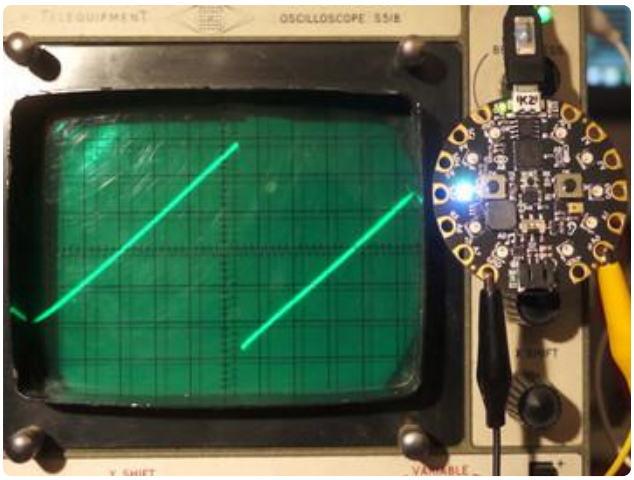
One DAC

Composite video (), a descendent of the early 405 ()/441 () line television standards, is one way to create image/video output suitable for display on a television. The full bandwidth is around 6-8MHz necessitating () a 12-16 megasample per second DAC!

A lower resolution image with a primitive synchronisation scheme can be used to display an image on an oscilloscope with the help of the normal timebase for the x-axis control. A CRO helps here because the brightness of the display varies with beam deflection speed.







The images here show how the beam intensity (brightness) varies with sweep rate.

The near vertical parts of a rapidly rising signal (high slew rate) are barely visible. This is often seen on square waves. Sawtooth waves are also interesting as the ramp will be visible but the vertical part less so.

A sine wave may show some gaps if it's created with low resolution samples or the DAC has very low resolution. These are more likely to visible away from the peaks where the wave has a steeper gradient and hence the difference between each consecutive sample is larger.

A low-resolution image can be formed by allowing the normal timebase to control the beam horizontally and using the DAC to control the vertical position. To draw an on-pixel, the DAC can be set to a value on-screen, if a pixel is not on then the DAC can be set of a value off-screen. If the slew rate is high enough the vertical transitions will be barely visible on a CRO. This technique only allows one pixel per column to be displayed as the beam scans from left to right, multiple scans need to be used to create a complete image. If the frame rate is 60Hz and the image is 50x40 pixels then the DAC needs to output samples at 60*50*40 = 120 kilosamples-per-second (ksps or kHz). This is far higher than the rate required for audio but still within the capabilities of the SAMD21/SAMD51 chips. A large sync pulse can be added to allow the

oscilloscope to trigger (start) the horizontal scan. This is, in effect, a crude form of composite video.

This technique is likely to work well on an old CRO but the vertical lines will be too prominent on a DSO.

Prolonged stationary bright points or very bright lines on a CRT-based oscilloscope may damage the phosphor.

CircuitPython

CircuitPython is a version of the Python language that runs on a number of popular microcontroller boards.

If you are new to CircuitPython, see Welcome to CircuitPython! ()

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial ().

Libraries

This project does not require any libraries from the CircuitPython library bundle as the libraries used are built-in.

The audioio () library has two useful characteristics/features:

- sends data to the DAC(s) at a constant, controllable rate,
- uses direct memory access () (DMA) hardware feature of the SAMD system on a chip () (SoC) which transfers data to the DAC(s), leaving the CPU free to do other things.

The DMA feature allows the CircuitPython program to run at the same time as transferring data (prepared in advance) to the DAC(s).

This code has been tested on CircuitPython 4.1.0 Release Candidate 0. It should run on versions of CircuitPython higher than this when they are available.

One DAC

The libraries for manipulating image formats are too complex and bulky to run on the SAMD21 (M0) boards like the CPX. The data for the DAC is best prepared on a host computer as a (mono) wav () file.

Download the code below and the dacanim.wav () (use Save link as... in browser) example wav file. Plug your CPX or other M0-based board into your computer via a known-good USB data cable. A flash drive named CIRCUITPY should appear in your file explorer/finder program. Copy the dacanim.wav and code below to the CIRCUITPY drive, renaming the latter to code.py. ()

dacanim.wav

Connect the CPX A0 pad output to the an oscilloscope input and GND to ground to see the image. The trigger value may need to be set and the timebase adjusted to set the width. The bright top line is best placed off screen by adjusting the volts/div and y position.

Scroll past the code below for a video showing the image output.

```
# Output prepared samples from a wav file to (CPX) DAC

import board, audioio, audiocore

dac = audioio.AudioOut(board.A0)
wav_file = open("dacanim.wav", "rb")
output_wave = audiocore.WaveFile(wav_file)
dac.play(output_wave, loop=True)
```

```
while True:
    pass
```

The image at the top of screen shows the full voltage range of the DAC on an oscilloscope running with a manual negative trigger value. The bitmap image is encoded to appear between 0.3V and 3.0V. The 3.3V level (bright line at top) is used for when there's no pixel to display, the 0V level is used for the synchronisation pulse signifying the beginning of each line.

Oscilloscope Output Video

The video below shows the spinning logo output from a CPX connected to a Hameg HM203-6 oscilloscope. The timebase and volts/div are set so the 40x40 resolution, 50fps animation fills the screen. This is a rare occasion where a slightly unfocussed oscilloscope beam can look better as it enlarges and softens the edges of the "pixels".

Python 3 Code

The example command line and code below for pngtowav can be used on a computer to generate the wav file. This code uses the imageio () library.

```
pngtowav -r -f 50 -o dacanim.wav logo.frame.{00..49}.png
```

```
# SPDX-FileCopyrightText: 2019 Kevin J. Walters for Adafruit Industries
#
# SPDX-License-Identifier: MIT

#!/usr/bin/python3

### pngtowav v1.0
"""Convert a list of png images to pseudo composite video in wav file form.

This is Python code not intended for running on a microcontroller board.
"""

### MIT License

### Copyright (c) 2019 Kevin J. Walters

### Permission is hereby granted, free of charge, to any person obtaining a copy
### of this software and associated documentation files (the "Software"), to deal
### in the Software without restriction, including without limitation the rights
### to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
### copies of the Software, and to permit persons to whom the Software is
### furnished to do so, subject to the following conditions:

### The above copyright notice and this permission notice shall be included in all
### copies or substantial portions of the Software.

### THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
### IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
### FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
### AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
### LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
### OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
### SOFTWARE.

import getopt
import sys
import array
import wave

import imageio


### globals
### pylint: disable=invalid-name
### start_offset of 1 can help if triggering on oscilloscope
### is missing alternate lines
debug = 0
verbose = False
movie_file = False
output_filename = "dacanim.wav"
fps = 50
threshold = 128  ### pixel level
replaceforsync = False
start_offset = 1

max_dac_v = 3.3
### 16 bit wav files always use signed representation for data
dac_offtop = 2**15-1  ### 3.30V
dac_sync = -2**15     ### 0.00V
### image from 3.00V to 0.30V
dac_top = round(3.00 / max_dac_v * (2**16-1)) - 2**15
dac_bottom = round(0.30 / max_dac_v * (2**16-1)) - 2**15


def usage(exit_code):  ### pylint: disable=missing-docstring
    print("pngtowav: "
          + "[-d] [-f fps] [-h] [-m] [-o outputfilename] [-r] [-s lineoffset] [-t
threshold] [-v]",
          file=sys.stderr)
    if exit_code is not None:
        sys.exit(exit_code)


def image_to_dac(img, row_offset, first_pix, dac_y_range):
    """Convert a single image to DAC output."""
    dac_out = array.array("h", [])

    img_height, img_width = img.shape
    if verbose:
        print("W,H", img_width, img_height)

    for row_o in range(img_height):
        row = (row_o + row_offset) % img_height
        ### Currently using 0 to (n-1)/n range
        y_pos = round(dac_top - row / (img_height - 1) * dac_y_range)
        if verbose:
            print("Adding row", row, "at y_pos", y_pos)
        dac_out.extend(array.array("h",
                                   [dac_sync]
                                   + [y_pos if x >= threshold else dac_offtop
                                      for x in img[row, first_pix:]]))
    return dac_out, img_width, img_height


def write_wav(filename, data, framerate):
    """Create one channel 16bit wav file."""
    wav_file = wave.open(filename, "w")
    nchannels = 1
    sampwidth = 2
    nframes = len(data)
```

```
    comptype = "NONE"
    compname = "not compressed"
    if verbose:
        print("Writing wav file", filename, "at rate", framerate,
              "with", nframes, "samples")
    wav_file.setparams((nchannels, sampwidth, framerate, nframes,
                        comptype, compname))
    wav_file.writeframes(data)
    wav_file.close()


def main(cmdlineargs):  ### pylint: disable=too-many-branches
    """main(args)"""
    global debug, fps, movie_file, output_filename, replaceforsync  ### pylint:
disable=global-statement
    global threshold, start_offset, verbose  ### pylint: disable=global-statement

    try:
        opts, args = getopt.getopt(cmdlineargs,
                                   "f:hmo:rs:t:v", ["help", "output="])
    except getopt.GetoptError as err:
        print(err,
              file=sys.stderr)
        usage(2)
    for opt, arg in opts:
        if opt == "-d":  ### pylint counts these towards too-many-branches :(
            debug = 1
        elif opt == "-f":
            fps = int(arg)
        elif opt in ("-h", "--help"):
            usage(0)
        elif opt == "-m":
            movie_file = True
        elif opt in ("-o", "--output"):
            output_filename = arg
        elif opt == "-r":
            replaceforsync = True
        elif opt == "-s":
            start_offset = int(arg)
        elif opt == "-t":
            threshold = int(arg)
        elif opt == "-v":
            verbose = True
        else:
            print("Internal error: unhandled option",
                  file=sys.stderr)
            sys.exit(3)

    dac_samples = array.array("h", [])

    ### Decide whether to replace first column with sync pulse
    ### or add it as an additional column
    first_pix = 1 if replaceforsync else 0

    ### Read each frame, either
    ### many single image filenames in args or
    ### one or more video (animated gifs) (needs -m on command line)
    dac_y_range = dac_top - dac_bottom
    row_offset = 0
    for arg in args:
        if verbose:
            print("PROCESSING", arg)
        if movie_file:
            images = imageio.mimread(arg)
        else:
            images = [imageio.imread(arg)]

        for img in images:
            img_output, width, height = image_to_dac(img, row_offset,
```

```
                                                    first_pix, dac_y_range)
            dac_samples.extend(img_output)
            row_offset += start_offset

    write_wav(output_filename, dac_samples,
              (width + (1 - first_pix)) * height * fps)


if __name__ == "__main__":
    main(sys.argv[1:])
```

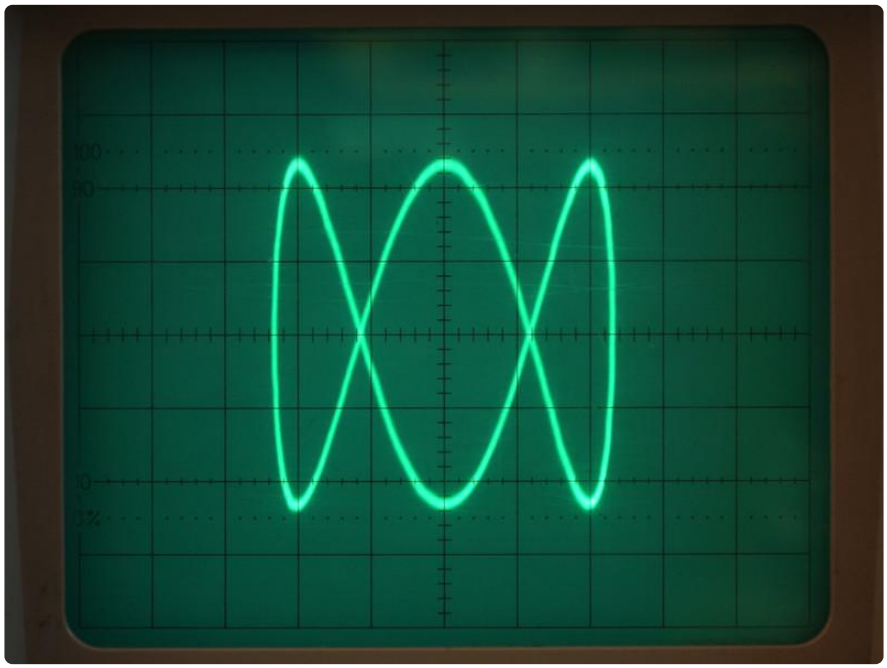
Code Discussion

The code is fairly straightforward:

1. Command line argument parsing.
2. Iterate over images converting each one to the DAC representation.
3. Write single channel wav file.

The sample rate set in the wav file is based on the resolution of the images and frame rate. A 40x40 pixel image at 50 frames per second (with the sync pulse replacing the first column) needs to be output at 40*40*50 = 80000 Hz.

Two DACs



Simple Figures

Lissajous () figures are classic x-y oscilloscope imagery. They are easily created from sine waves. The short code below shows how to create the one pictured above, the same one that inspired the Australian Broadcasting Corporation (ABC) () logo.

```
# Lissajous version 1
import array, math
import board, audioio, audiocore

length = 1000
samples_xy = array.array("H", [0] * length * 2)

# Created interleaved x, y samples
for idx in range(length):
    samples_xy[2 * idx] = round(math.sin(math.pi * 2 * idx / length) * 10000 +
10000)
    samples_xy[2 * idx + 1] = round(math.sin(math.pi * 2 * 3 * idx / length +
math.pi / 2) * 10000 + 10000)

output_wave = audiocore.RawSample(samples_xy,
                                  channel_count=2,
                                  sample_rate=100*1000)
dacs.play(output_wave, loop=True)
while True:
   pass
```

This image, based on 1000 samples per channel, is flicker-free at 100 kHz output rate. There's some minor flicker at (audio) rates like 48 kHz.

The samples can also be output with the analogio () library by writing a loop to assign to the two DACs. The version 2 code below shows a typical approach.

```
# Lissajous version 2
import array, math
import board, analogio
length = 1000
samples_x = array.array("H", [0] * length)
samples_y = array.array("H", [0] * length)

for idx in range(length):
    samples_x[idx] = round(math.sin(math.pi * 2 * idx / length) * 10000 + 10000)
    samples_y[idx] = round(math.sin(math.pi * 2 * 3 * idx / length + math.pi / 2) *
10000 + 10000)

dac_a0 = analogio.AnalogOut(board.A0)
dac_a1 = analogio.AnalogOut(board.A1)

while True:
    for x, y in zip(samples_x, samples_y):
        dac_a0.value = x
        dac_a1.value = y
```

Version 2 flickers a little at high brightness showing the performance of the interpreter is just about adequate for looping over a thousand sample pairs. However, it has very visible bright spots appearing every second or so. These artefacts in some

ways look attractive but they are not intentional and it's useful to understand why they occur.

There will be a small gap in time as the `for` loop finishes and the `while` loop executes the next `for` loop. The bright spots move around the figure so this cannot be an explanation for the brief pause in beam movement. The "random" placement of the bright spot suggests something else is causing a pause in the execution of the application code leaving the beam stationary for a moment. This is probably some regular tidying of memory by the CircuitPython interpreter known as a garbage collection () (GC).

```
# snippet of Lissajous version 3

while True:
    for idx in range(length):
        a0.value = samples_x[idx]
        a1.value = samples_y[idx]
```

Version 3 replaces the `for` loop with one which just iterates over the array indices rather than using the `zip()`. The bright spots have gone with this simpler code. This makes sense as this code has no need to allocate and free memory as it loops.

Another approach would be to move `zip()` outside the loop and only create the object once to make `for` loop more efficient. This well-intentioned migration will not work as it only display the figure once. `zip()` (in CircuitPython based on Python 3) returns an iterator () which is designed to be used once. If the approach of constructing the two lists is maintained then `zip()` can be used to make a single list with the aid of `list()`. Version 4 shows this with an additional performance enhancement of removing the temporary variables in the loop and assigning to them directly.

```
# snippet of Lissajous version 4

samples_both = list(zip(samples_x, samples_y))
while True:
    for a0.value, a1.value in samples_both:
        pass
```

Spinning Adafruit Logo

Download the code.py file with the link below and adafruit_logo_vector.py () file. Plug your PyGamer or other M4-based board into your computer via a known-good USB data cable. A flash drive named CIRCUITPY should appear in your file explorer/finder program. Copy code.py and adafruit_logo_vector.py to the CIRCUITPY drive.

Connect the PyGamer A0 output to the x oscilloscope input, A1 to the y input and GN D to ground to see the image. Three short jumper cables () will facilitate connection to a Feather female header.

Scroll past the code below for a video showing the image output.

```
# SPDX-FileCopyrightText: 2019 Kevin J. Walters for Adafruit Industries
#
# SPDX-License-Identifier: MIT

### scope-xy-adafruitlogo v1.0

"""Output a logo to an oscilloscope in X-Y mode on an Adafruit M4
board like Feather M4 or PyGamer (best to disconnect headphones).
"""

### copy this file to PyGamer (or other M4 board) as code.py

### MIT License

### Copyright (c) 2019 Kevin J. Walters

### Permission is hereby granted, free of charge, to any person obtaining a copy
### of this software and associated documentation files (the "Software"), to deal
### in the Software without restriction, including without limitation the rights
### to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
### copies of the Software, and to permit persons to whom the Software is
### furnished to do so, subject to the following conditions:

### The above copyright notice and this permission notice shall be included in all
### copies or substantial portions of the Software.

### THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
### IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
### FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
### AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
### LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
### OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
### SOFTWARE.

import time
import math
import array

import board
import audioio
import audiocore
import analogio

### Vector data for logo
import adafruit_logo_vector

VECTOR_POINT_SPACING = 3

def addpoints(points, min_dist):
    """Add extra points to any lines if length is greater than min_dist"""

    newpoints = []
    original_len = len(points)
    for pidx in range(original_len):
        px1, py1 = points[pidx]
        px2, py2 = points[(pidx + 1) % original_len]

        ### Always keep the original point
        newpoints.append((px1, py1))
```

```
        diff_x = px2 - px1
        diff_y = py2 - py1
        dist = math.sqrt(diff_x ** 2 + diff_y ** 2)
        if dist > min_dist:
            ### Calculate extra intermediate points plus one
            extrasp1 = int(dist // min_dist) + 1
            for extra_idx in range(1, extrasp1):
                ratio = extra_idx / extrasp1
                newpoints.append((px1 + diff_x * ratio,
                                  py1 + diff_y * ratio))
        ### Two points define a straight line
        ### so no need to connect final point back to first
        if original_len == 2:
            break

    return newpoints

### pylint: disable=invalid-name
### If logo is off centre then correct it here
if adafruit_logo_vector.offset_x != 0 or adafruit_logo_vector.offset_y != 0:
    data = []
    for part in adafruit_logo_vector.data:
        newpart = []
        for point in part:
            newpart.append((point[0] - adafruit_logo_vector.offset_x,
                            point[1] - adafruit_logo_vector.offset_y))
        data.append(newpart)
else:
    data = adafruit_logo_vector.data


### Add intermediate points to make line segments for each part
### look like continuous lines on x-y oscilloscope output
display_data = []
for part in data:
    display_data.extend(addpoints(part, VECTOR_POINT_SPACING))

### PyPortal DACs seem to stop around 53000 and there's 2 100 ohm resistors
### on output so maybe large values aren't good idea?
### 32768 and 32000 exhibit this bug but 25000 so far appears to be a
### workaround, albeit a mysterious one
### https://github.com/adafruit/circuitpython/issues/1992
### Using "h" for audiocore.RawSample() DAC range will be 20268 to 45268
dac_x_min = 0
dac_y_min = 0
dac_x_max = 25000
dac_y_max = 25000
dac_x_mid = dac_x_max // 2
dac_y_mid = dac_y_max // 2

### Convert the points into format suitable for audio library
### and scale to the DAC range used by the library
### Intentionally using "h" data representation here as this happens to
### cause the CircuitPython audio libraries to make a copy of
### rawdata which is useful to allow animating code to modify rawdata
### without affecting current DAC output
rawdata = array.array("h", (2 * len(display_data)) * [0])

range_x = 512.0
range_y = 512.0
halfrange_x = range_x / 2
halfrange_y = range_y / 2
mid_x = 256.0
mid_y = 256.0
mult_x = dac_x_max / range_x
mult_y = dac_y_max / range_y

### https://github.com/adafruit/circuitpython/issues/1992
```

```
print("length of rawdata", len(rawdata))

use_wav = True
poor_wav_bug_workaround = False
leave_wav_looping = True

### A0 will be x, A1 will be y
if use_wav:
    print("Using audiocore.RawSample for DACs")
    dacs = audioio.AudioOut(board.A0, right_channel=board.A1)
else:
    print("Using analogio.AnalogOut for DACs")
    a0 = analogio.AnalogOut(board.A0)
    a1 = analogio.AnalogOut(board.A1)

### 10Hz is about ok for AudioOut, optimistic for AnalogOut
frame_t = 1/10
prev_t = time.monotonic()
angle = 0  ### in radians
frame = 1
while True:
    ##print("Transforming data for frame:", frame, "at", prev_t)

    ### Rotate the points of the vector graphic around its centre
    idx = 0
    sine = math.sin(angle)
    cosine = math.cos(angle)
    for px, py in display_data:
        pcx = px - mid_x
        pcy = py - mid_y
        dac_a0_x = round((-sine * pcx + cosine * pcy + halfrange_x) * mult_x)
        ### Keep x position within legal values (if needed)
        ##dac_a0_x = min(dac_a0_x, dac_x_max)
        ##dac_a0_x = max(dac_a0_x, 0)
        dac_a1_y = round((sine * pcy + cosine * pcx + halfrange_y) * mult_y)
        ### Keep y position within legal values (if needed)
        ##dac_a1_y = min(dac_a1_y, dac_y_max)
        ##dac_a1_y = max(dac_a1_y, 0)
        rawdata[idx] = dac_a0_x - dac_x_mid      ### adjust for "h" array
        rawdata[idx + 1] = dac_a1_y - dac_y_mid  ### adjust for "h" array
        idx += 2

    if use_wav:
        ### 200k (maybe 166.667k) seems to be practical limit
        ### 1M permissible but seems same as around 200k
        output_wave = audiocore.RawSample(rawdata,
                                          channel_count=2,
                                          sample_rate=200 * 1000)

        ### The image may "warp" sometimes with loop=True due to a strange bug
        ### https://github.com/adafruit/circuitpython/issues/1992
        if poor_wav_bug_workaround:
            while True:
                dacs.play(output_wave)
                if time.monotonic() - prev_t >= frame_t:
                    break
        else:
            dacs.play(output_wave, loop=True)
            while time.monotonic() - prev_t < frame_t:
                pass
            if not leave_wav_looping:
                dacs.stop()
    else:
        while True:
            ### This gives a very flickery image with 4932 points
            ### slight flicker at 2552
            ### might be ok for 1000
            for idx in range(0, len(rawdata), 2):
                a0.value = rawdata[idx]
```

```
                a1.value = rawdata[idx + 1]
            if time.monotonic() - prev_t >= frame_t:
                break
    prev_t = time.monotonic()
    angle += math.pi / 180 * 3 ### 72 degrees per frame
    frame += 1
```

Oscilloscope Output Video

The video below shows the spinning logo output from a PyGamer connected to a Hameg HM203-6 oscilloscope in x-y mode. This more complex code uses the `audioio` libraries for DAC output but still features bright spots. This is probably due to the changeover between one frame's data to the next, garbage collection should not be a factor as it would not interrupt the DMA transfers. There are also some faint spots visible some of the time. These might be related to some unexplained issues with stepping, rising slew rate () on SAMD51 DACs.

Code Discussion

The essence of the code is:

1. Load line data from adafruit_logo_vector.py
2. Apply offset correction to centre the image.
3. Interpolate lines using `addpoints()` function to make them appear solid.
4. Loop:
    1. Rotate image data and write DAC output for frame to an `array.array`.
    2. Output data to DAC with `dacs.play()` (see excerpt below).
    3. Pause for frame length.

The `array.array` type is carefully chosen as `"h"` for signed integers. For DAC output with `audiocore.RawSample()`, the library happens to make a copy of the data for output rather than using it in-place (). This is useful to allow the loop to efficiently reuse the same `array.array` without the risk of mixing two different frames in the (looping) DAC output.

The `play()` method is invoked with `loop=True` which leaves the frame being continuously sent to the oscilloscope until either a `stop()` or the next `play()` is executed. This is very useful for keeping the the image on the oscilloscope and avoiding any long periods where the beam is stationary.

```
dacs.play(output_wave, loop=True)
while time.monotonic() - prev_t &lt; frame_t:
    pass
if not leave_wav_looping:
    dacs.stop()
```

When the while loop terminates as time passes beyond the duration `frame_t` the code will go on to calculate the next frame (not shown) whilst continuing to send output to the DACs. If `leave_wav_looping` is set to `False` then DAC output will cease and there will be both considerable flicker between frames and a bright spot.

A sophisticated garbage collection system is typically better than the programmer at scheduling concurrent vs blocking (stop the world) () collections. For this particular program, there are some opportune points to execute gc.collect() () to avoid less opportune scheduling. This is an area to explore.

Making Vector Images

If an image is only available in bitmap form then it will need converting to vector form for display. Inkscape () is one, free, multi-platform application which can do this.

1. Select bitmap image which can be represented well with line art.
2. Load bitmap into Inkscape.
3. Vectorise - inspect and adjust result as necessary.
4. Flatten - this will convert any (bezier) curves into a series of straight lines.
5. Save as an svg () file.
6. Extract line data from the svg file - the svgtopy utility below can help with this.

The example command line and code below can read simple svg files and print them as lists suitable for inclusion in a CircuitPython program.

```
svgtopy &lt; logo-flattened.svg
```

```
# SPDX-FileCopyrightText: 2019 Kevin J. Walters for Adafruit Industries
#
# SPDX-License-Identifier: MIT

#!/usr/bin/python3

### svgtopy v1.0
"""Print vectors from an SVG input file in python list format
for easy pasting into a program.

This is Python code not intended for running on a microcontroller board.
"""

### MIT License

### Copyright (c) 2019 Kevin J. Walters

### Permission is hereby granted, free of charge, to any person obtaining a copy
### of this software and associated documentation files (the "Software"), to deal
### in the Software without restriction, including without limitation the rights
### to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
### copies of the Software, and to permit persons to whom the Software is
### furnished to do so, subject to the following conditions:

### The above copyright notice and this permission notice shall be included in all
```

```
### copies or substantial portions of the Software.

### THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
### IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
### FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
### AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
### LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
### OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
### SOFTWARE.


### it only understands M and L in SVG

### Worth looking at SVG libraries to see if they
### can parse/transform SVG data

import getopt
import sys
import re
##import fileinput
import xml.etree.ElementTree as ET

### globals
### pylint: disable=invalid-name
debug = 0
verbose = False


def usage(exit_code):  ### pylint: disable=missing-docstring
    print("""Usage: svgtopy [-d] [-h] [-v] [--help]
Convert an svg file from from standard input to comma-separated tuples
on standard output for inclusion as a list in a python program.""",
          file=sys.stderr)
    if exit_code is not None:
        sys.exit(exit_code)


def search_path_d(svgdata, point_groups):
    """Look for M and L in the SVG d attribute of a path node"""

    points = []
    for match in re.finditer(r"([A-Za-z])([\d\.]+)\s+([\d\.]+)\s*", svgdata):
        if match:
            cmd = match.group(1)
            if cmd == "M":  ### Start of a new part
                mx, my = match.group(2, 3)
                if points:
                    point_groups.append(points)
                    points = []
                points.append((float(mx), float(my)))
                if debug:
                    print("M pos", mx, my)

            elif cmd == "L":  ### Continuation of current part
                lx, ly = match.group(2, 3)
                points.append((float(lx), float(ly)))
                if debug:
                    print("L pos", lx, ly)

            else:
                print("SVG cmd not implemented:", cmd,
                      file=sys.stderr)
        else:
            print("some parsing issue",
                  file=sys.stderr)

    # Add the last part to point_groups
    if points:
        point_groups.append(points)
```

```
        points = []


def main(cmdlineargs):
    """main(args)"""
    global debug, verbose  ### pylint: disable=global-statement

    try:
        opts, _ = getopt.getopt(cmdlineargs,
                                "dhv", ["help"])
    except getopt.GetoptError as err:
        print(err,
              file=sys.stderr)
        usage(2)
    for opt, _ in opts:
        if opt == "-d":
            debug = True
        elif opt == "-v":
            verbose = True
        elif opt in ("-h", "--help"):
            usage(0)
        else:
            print("Internal error: unhandled option",
                  file=sys.stderr)
            sys.exit(3)

    xml_ns = {"svg": "http://www.w3.org/2000/svg"}
    tree = ET.parse(sys.stdin)
    point_groups = []
    for path in tree.findall("svg:path", xml_ns):
        svgdata = path.attrib["d"]
        if verbose:
            print("Processing path with {0:d} length".format(len(svgdata)))
        search_path_d(svgdata, point_groups)



    for idx, points in enumerate(point_groups):
        print("# Group", idx + 1)
        for point in points:
            print("     ", point, ",", sep="")

if __name__ == "__main__":
    main(sys.argv[1:])
```

Going Further

Ideas for Areas to Explore

- Make your own vector images and animations.
- Make an old skool vector video game.
- Investigate the suitability and limitations of (capacitor smoothed) PWM outputs on SAMD21 (M0) boards for a second analog output.
- For animations in CircuitPython explore whether judicious use of `gc.collect()` can be used to improve display output.

Related Projects

- Circuit Playground Express (& other ATSAMD21 Boards) DAC Hacks () (C++/Arduino).
- Trammel Hudson:
  - Turn Your Oscilloscope Into a Vector Video Display () - same concept as two DAC approach used here but includes construction of a simple externa l DAC pair, uses C/Arduino.
  - Pseudorandom 09: Vector Displays () (YouTube) - "Trammell Hudson shows us the retro beauty of vector displays and recounts his adventures in hacking the Vectrex gaming console!"
- Feather M0 Sine Wave generator using ZeroDMA ()

Further Reading

- Arcade Jason: THE MESSAGE () - another explanation of x-y vector graphics with a nice example of font and text.
- Oscilloscope Music () - inspirational X-Y art with tutorials.
- Neil Fraser's JavaScript x-y oscilloscope driver ()
- Jed Margolin: The Secret Life of XY Monitors () and The Secret Life of Vector Generators ()
- Recreating Asteroids with Lasers () (YouTube) - interview with Seb Lee-Delisle about playing Asteroids on a laser vector display.
- Pekka Vaananen: Quake on an oscilloscope: A technical report ()
- Instructables:
  - Arduino Laser Show With Full XY Control () - using speakers as cheap alternative to galvanometers ().
  - Arduino Laser Show With Real Galvos ()
- Empire Leicester Square (Cinema) Laser Shows () (YouTube) - video of the laser show, part of the programme in the 1990s, flicker is visible as more lines are added to image.
- Leadfeather Blog: Max Ernst: Levity and Gravity in His Paintings, 1942-48 () - pendulum-made lissajous figures.