Vintage computer to HDMI with Feather DVI & CircuitPython

Created by Jeff Epler

https://learn.adafruit.com/vintage-computer-to-dvi-with-feather-dvi-circuitpython

Last updated on 2024-06-03 03:59:41 PM EDT

Table of Contents

Overview

output There are a zillion vintage computers out there, and most of them are designed to connect to CRTs.

In this project, you'll learn about the video output of one specific 1980s computer (the Xerox 820) and see how to create a converter to a modern DVI-compatible video signal using the Adafruit Feather RP2040 DVI & CircuitPython.

With further work, these techniques may be applicable to other vintage computers as well.

Xerox 820

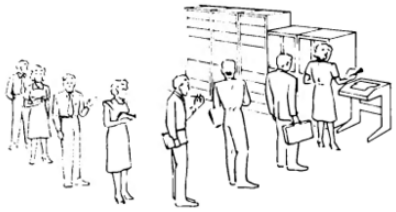
Just before the arrival of the IBM PC, Xerox created the "Xerox 820 Information Processor". It was packed with 64kB of RAM (around 60kB usable) and supported the CP/M operating system. It came with a display (that also contained the main board), keyboard, and floppy drives. Accessories like printers were also available.

A VEHICLE FOR PRODUCTIVITY INCREASE

In today's high technology marketplace, your firm's productivity is a direct function of your computing and word processing capability.

LARGE COMPANY . . . WITH LARGE OPERATING BUDGET

THE TRADITIONAL WAY



One Computer for
Large Number of Personnel

THE "820" WAY

Individual Computer for
Each Decision-Making Employee

SMALL BUSINESS . . . WITH SMALL OPERATING BUDGET

THE TRADITIONAL WAY

Desktop Computer
$4,000

Word Processor
$7,500

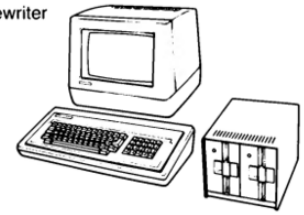
Data Terminal
$2,500

Typewriter
$ 900

Each System Is Too
Expensive to Own

THE "820" WAY

- Desk Top Computer
- Data Terminal
- Word Processor
- Typewriter



Multi-Function Capability
At an Affordable Price

In 2023, I was at an estate sale where I bought a complete system and 4 additional CPU boards.

I'd like my additional CPU boards to be usable, but in order to be usable as CPM computers, they need power, a display, keyboard, and a floppy drive.

This learn guide tackles one of those items by creating an adapter from the Xerox 820's TTL-level video signals to DVI.

Research

There's a relative wealth of documentation about the Xerox 820, including schematics and other useful information. I've gathered some of it in a personal GitHub repository (https://adafru.it/19Me).

When it comes to the video, here are some salient facts from the Software Developers Reference (https://adafru.it/19Mf):

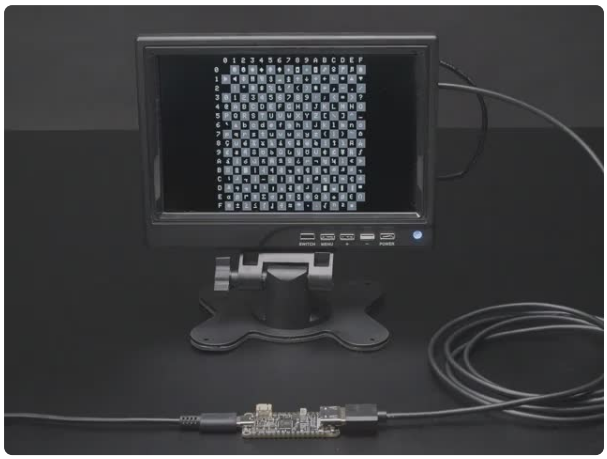
- Video bitrate: "10.694MBPS" (pixel time 93.51ns)
- Active bits per horizontal line: 560
- Total bits per horizontal line: 700
- Active lines per field: 240
- Sync & Data signals on a 10-pin connector are all TTL-compatible

This is an excellent match for the video specs of the Adafruit Feather RP2040 DVI, which can support a 640x240x1bpp video mode. Each line is repeated twice, creating a super well-supported 640x480@60Hz video signal.

The next step is figuring out how to capture the digital data. The "PIO" coprocessor in the Adafruit Feather RP2040 is key for this. We can run a tiny program on it that handles the horizontal and vertical sync pulses, grabbing the correct video data bits. By happy coincidence, it's possible to capture in exactly the right format for DVI video output.

Even though this is a very high performance task, dealing with millions of pixels every second, it's actually possible to code it in CircuitPython. If performing any processing of the pixel data had been necessary, though, it would probably have had to be an Arduino or pico-sdk C++ project.

Parts



Adafruit Feather RP2040 with DVI Output Port - Works with HDMI

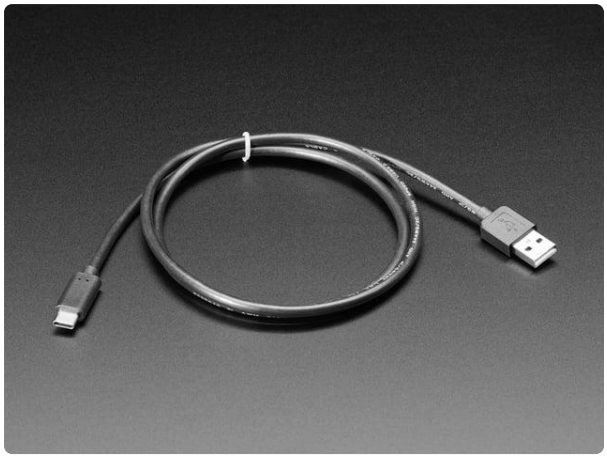
Wouldn't it be cool if you could display images and graphics from a microcontroller directly to an HDMI monitor or television? We think so! So we designed this RP2040 Feather that...

https://www.adafruit.com/product/5710

HDMI Flat Cable - 1 foot / 30cm long

Connect two HDMI devices together and save space with this basic flat HDMI 1.4 cable. It has nice molded grips for easy installation, and is 1 foot long (~30 cm).This cable is...

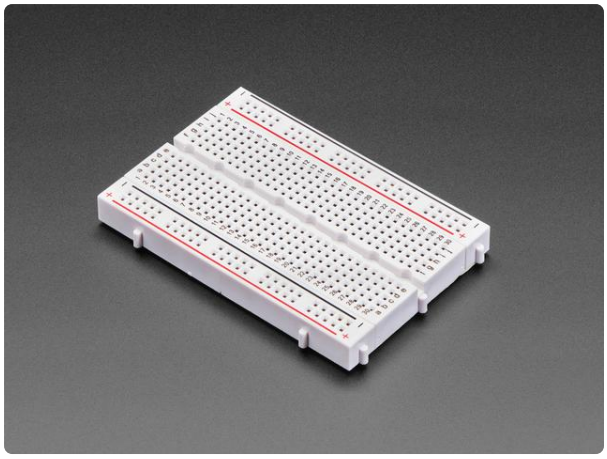
https://www.adafruit.com/product/2197



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474



Half-Size Breadboard with Mounting Holes
This cute 3.2″ × 2.1″ (82 × 53mm) solderless half-size breadboard has four bus lines and 30 rows of pins, our favorite size of solderless breadboard for...
https://www.adafruit.com/product/4539

Premium Female/Male 'Extension' Jumper Wires - 20 x 6"
These Female/Male Extension jumper wires are handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in...
https://www.adafruit.com/product/1954

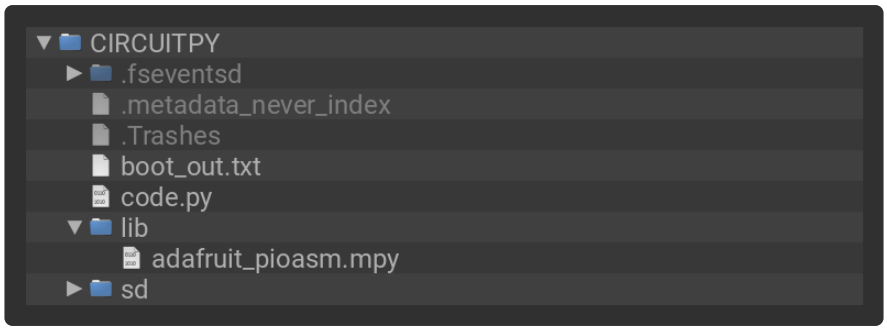
Coding the Video Adapter

First, install the latest version of CircuitPython (https://adafru.it/19MA) on your Adafruit Feather RP2040 DVI. Then, make sure the DVI connection is working by giving the DVI demo a whirl (https://adafru.it/19MB). Once that's out of the way, it's time to install the code for this project.

You'll need to copy the code and the necessary libraries to your Feather. Luckily, we have a way to do this all at once.

Click the **Download Project Bundle** button above the example to download the necessary libraries and the applicable **code.py** file in a zip file. Extract the contents of the zip file, and find your CircuitPython version. Copy all the files inside that folder to your **CIRCUITPY** drive, replacing the existing **code.py** program if asked.

After completing this process, your **CIRCUITPY** drive contents should resemble the following.



Project Code

Check below the full code listing for explanations of key parts of the program. Note that the snippets may be slightly out of date as improvements have been made to the main code over time.

```
# SPDX-FileCopyrightText: 2024 Jeff Epler for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import array

import ulab
import rp2pio
import board
import adafruit_pioasm
import picodvi
import displayio

# The connections from the Xerox 820
vdata = board.D9  # Followed by hsync on D10 & vsync on D11
# The nominal frequency of the Xerox 820 video circuitry. Can modify by steps
# of approximately ±42000 to improve display stability
pixel_frequency = 10_694_250
# The PIO peripheral is run at a multiple of the pixel frequency. This must be less
# than the CPU speed, normally 120MHz.
clocks_per_pixel = 10
# The "fine pixel offset", shifts the sample time by this many sub-pixels
fine_pixel = 0
# A pin that shows when the Pico samples the pixel value. With an oscilloscope,
this can
# be used to help fine tune the pixel_frequency & fine_pixel numbers. Ideally, the
rising
# edge of pixel_sync_out is exactly in the middle of time a pixel is high/low.
pixel_sync_out = board.D5

# Details of the Xerox display timing. You may need to modify `blanking_lines` and
# `blanking_pixels` to adjust the vertical and horizontal position of the screen
content.
# Normally you wouldn't change `active_lines` or `active_pixels`.
active_lines = 240
blanking_lines = 18
active_pixels = 640
blanking_pixels = 58
total_lines = active_lines + blanking_lines

# Pins for the DVI connector
dvi_pins = dict(
    clk_dp=board.CKP,
    clk_dn=board.CKN,
    red_dp=board.D0P,
```

```
    red_dn=board.D0N,
    green_dp=board.D1P,
    green_dn=board.D1N,
    blue_dp=board.D2P,
    blue_dn=board.D2N,
)

# Set up the display. Try 640x240 first (this mode is likely to be added in
CircuitPython
# 9.1.x) then 640x480, which works in CircuitPython 9.0.x.
try:
    displayio.release_displays()
    dvi = picodvi.Framebuffer(640, 240, **dvi_pins, color_depth=1)
except ValueError:
    print(
        "Note: This version of CircuitPython does not support 640x240\n."
        "Display will be compressed vertically."
    )
    displayio.release_displays()
    dvi = picodvi.Framebuffer(640, 480, **dvi_pins, color_depth=1)

# Clear the display
ulab.numpy.frombuffer(dvi, dtype=ulab.numpy.uint8)[:] = 0

# Create the "control stream". The details are discussed in the Learn article
def control_gen():
    yield total_lines - 2
    for _ in range(blanking_lines):
        yield from (1, 0)  # 0 active pixels is special-cased
    for _ in range(active_lines):
        yield from (blanking_pixels - 1, active_pixels - 1)

control = array.array("L", control_gen())

# These little programs are run on the RP2040's PIO co-processor, and handle the
pixel
# data and sync pulses.
jmp_0 = adafruit_pioasm.Program("jmp 0")

program = adafruit_pioasm.Program(
    f"""
.side_set 1

    .wrap_target
    out y, 32           ; get total line count
    wait 0, pin 2
    wait 1, pin 2 ; wait for vsync

wait_line_inactive:
    out x, 32     ; get total line count
    wait 0, pin 1
    wait 1, pin 1; wait for hsync

wait_line_active:
    nop [{clocks_per_pixel-2}]
    jmp x--, wait_line_active ; count off non-active pixels

    out x, 32 [{fine_pixel}]    ; get line active pixels & perform fine pixel adjust
    jmp !x, wait_line_inactive  ; no pixels this line, wait next hsync

capture_active_pixels:
    in pins, 1 side 1
    jmp x--, capture_active_pixels [{clocks_per_pixel-2}] ; more pixels
    jmp y--, wait_line_inactive ; more lines?
    .wrap
"""
)

# Set up PIO to transfer pixels from Xerox
```

```
pio = rp2pio.StateMachine(
    program.assembled,
    frequency=pixel_frequency * clocks_per_pixel,
    first_in_pin=vdata,
    in_pin_count=3,
    in_pin_pull_up=True,
    first_sideset_pin=pixel_sync_out,
    auto_pull=True,
    pull_threshold=32,
    auto_push=True,
    push_threshold=32,
    offset=0,
    **program.pio_kwargs,
)

# Set up the DVI framebuffer memory as a capture target
words_per_row = 640 // 32
first_row = (dvi.height - 240) // 2  # adjust text to center if in 640x480 mode
buf = memoryview(dvi).cast("L")[
    first_row * words_per_row : (first_row + active_lines) * words_per_row
]
assert len(buf) == 4800  # Check that the right amount will be transferred

b = array.array("L", [0])

# Repeatedly transfer pixels from Xerox into DVI framebuffer.
while True:
    pio.run(jmp_0.assembled)
    pio.clear_rxfifo()
    pio.write_readinto(control, buf)
```

Connections & Video Timing Details

This block of code describes the pins used for the video connections as well as the details of the video timings. While some of these can be changed freely, others must remain as-is. For instance, the number of active pixels has to match the digital video mode exactly.

```
# The connections from the Xerox 820
vdata = board.D9  # Followed by hsync on D10 &amp;amp; vsync on D11
# The nominal frequency of the Xerox 820 video circuitry. Can modify by steps
# of approximately ±42000 to improve display stability
pixel_frequency = 10_694_250
# The PIO peripheral is run at a multiple of the pixel frequency. This must be less
# than the CPU speed, normally 120MHz.
clocks_per_pixel = 10
# The "fine pixel offset", shifts the sample time by this many sub-pixels
fine_pixel = 0
# A pin that shows when the Pico samples the pixel value. With an oscilloscope,
this can
# be used to help fine tune the pixel_frequency &amp;amp; fine_pixel numbers. Ideally,
the rising
# edge of pixel_sync_out is exactly in the middle of time a pixel is high/low.
pixel_sync_out = board.D5

# Details of the Xerox display timing. You may need to modify `blanking_lines` and
# `blanking_pixels` to adjust the vertical and horizontal position of the screen
content.
# Normally you wouldn't change `active_lines` or `active_pixels`.
active_lines = 240
blanking_lines = 18
active_pixels = 640
```

```
blanking_pixels = 58
total_lines = active_lines + blanking_lines
```

Creating the DVI display object

It's worth noting that due to technical limitations in CircuitPython 9.0.x, a 640x240 video mode is not available, so the Xerox text will only take up half of the screen vertically in a 640x480 display. In 9.1.x, the 640x240 video mode will likely be available, so if you're from the future you can try changing the corresponding number in the `picodvi.Framebuffer` line from 480 to 240.

```
try:
    displayio.release_displays()
    dvi = picodvi.Framebuffer(640, 240, **dvi_pins, color_depth=1)
except ValueError:
    print(
        "Note: This version of CircuitPython does not support 640x240\n."
        "Display will be compressed vertically."
    )
    displayio.release_displays()
    dvi = picodvi.Framebuffer(640, 480, **dvi_pins, color_depth=1)
```

The PIO control stream

The PIO peripheral doesn't know much about the video timings. Instead, it fetches this information from its input FIFO, which I refer to as the "control stream". Because of the way counting and looping work in PIO programs, 1 must be subtracted from most counts.

First, an overall count of lines is sent. Then, a pair of numbers for each line To skip a line of data because it is a "blanking line", the two numbers (1,0) are sent. Otherwise, two numbers are sent, giving the number of blanking pixels to skip and the number of visible pixels to capture.

In order to make the captured pixel count match the DVI resolution, additional blank pixels are captured at the left, during what is actually part of the blanking time.

```
# Create the "control stream". The details are discussed in the Learn article
def control_gen():
    yield total_lines - 2
    for _ in range(blanking_lines):
        yield from (1, 0)  # 0 active pixels is special-cased
    for _ in range(active_lines):
        yield from (blanking_pixels - 1, active_pixels - 1)

control = array.array("L", control_gen())
```

PIO Assembler Code

We use one PIO program and one fragment.

The main PIO program performs these steps in order:

- Get the total line count & wait for a vsync pulse
- For each line:
  - get the number of invisible pixels & consume them
  - if the number of visible pixels is 0, continue to the next line
  - otherwise, get and store the visible pixels

In principle this program could just run continuously. However, I was unable to get this to work as expected. So instead, the "jmp_0" program is directly executed whenever CircuitPython is ready to process another frame of data.

Some calculated values for exact pixel timing are inserted (inside of "`{}`").

```
jmp_0 = adafruit_pioasm.Program("jmp 0")

program = adafruit_pioasm.Program(
    f"""
.side_set 1

    .wrap_target
    out y, 32           ; get total line count
    wait 0, pin 2
    wait 1, pin 2 ; wait for vsync

wait_line_inactive:
    out x, 32     ; get total line count
    wait 0, pin 1
    wait 1, pin 1; wait for hsync

wait_line_active:
    nop [{clocks_per_pixel-2}]
    jmp x--, wait_line_active ; count off non-active pixels

    out x, 32 [{fine_pixel}]    ; get line active pixels & perform fine pixel adjust
    jmp !x, wait_line_inactive  ; no pixels this line, wait next hsync

capture_active_pixels:
    in pins, 1 side 1
    jmp x--, capture_active_pixels [{clocks_per_pixel-2}] ; more pixels
    jmp y--, wait_line_inactive ; more lines?
    .wrap
"""
)
```

Final calculations & Forever-loop

The `dvi` object can be treated as a memory buffer. It is "cast to L" (treated as 32-bit values), and then shortened to the correct size for one frame of data.

Then, the code repeatedly gives the PIO program a fresh start by jumping to the beginning of the program and clearing any data that was waiting to be read.

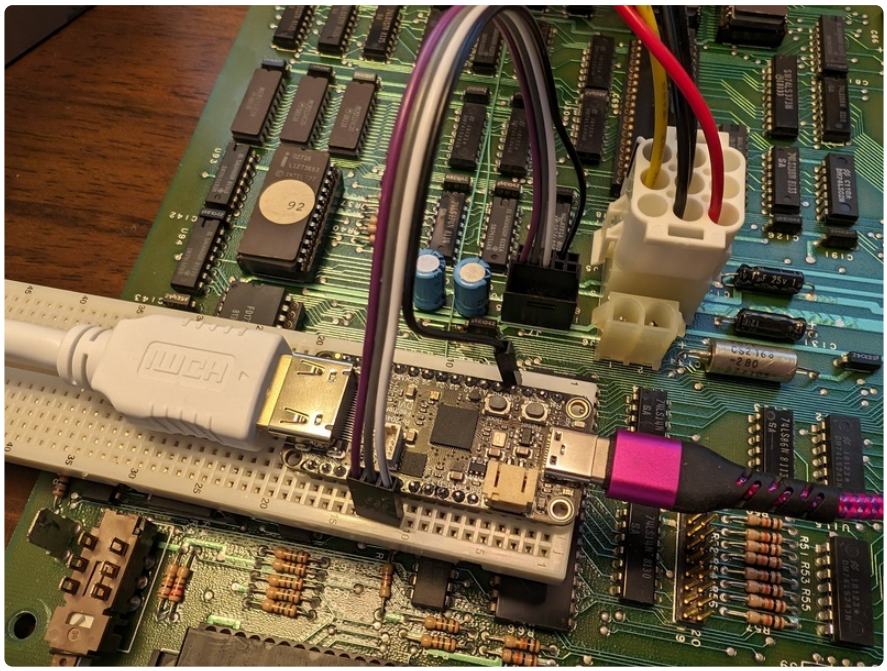
Finally, to write the control stream and receive the pixels into the DVI framebuffer.

```
# Set up the DVI framebuffer memory as a capture target
words_per_row = 640 // 32
first_row = (dvi.height - 240) // 2  # adjust text to center if in 640x480 mode
buf = memoryview(dvi).cast("L")[
    first_row * words_per_row : (first_row + active_lines) * words_per_row
]
assert len(buf) == 4800  # Check that the right amount will be transferred

b = array.array("L", [0])

# Repeatedly transfer pixels from Xerox into DVI framebuffer.
while True:
    pio.run(jmp_0.assembled)
    pio.clear_rxfifo()
    pio.write_readinto(control, buf)
```

This seems to run at around 30 frames per second which is just fine since the Xerox 820 is just pushing text, not playing quick-reaction video games.

Note that if no video input is provided, the behavior of the adapter is unpredictable. As long as you don't get an error printed on the REPL, you're good to continue to the next step of wiring the Feather to the Xerox 820.

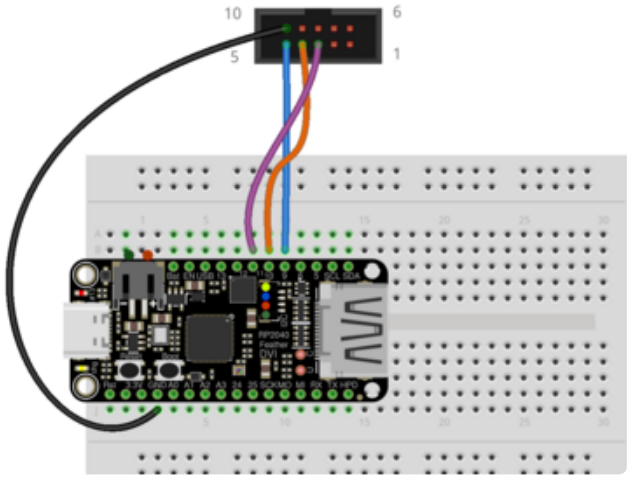
Wiring the Video Adapter



For ease of construction and testing, I built this project on a solderless breadboard. For permanent installation, I'd use other construction methods.

From the Xerox 820 documentation, we can learn the following details of the video signal, which is on a "2x5" standard header marked J7:

- Pin 3: vertical sync
- Pin 4: horizontal sync
- Pin 5: video data
- Pin 6-10: GND



Make the following connections:

J7-3 to Feather D11
J7-4 to Feather D10
J7-5 to Feather D9
J7-10 to Feather GND

(fritzing file)

https://adafru.it/19MC

A more permanent solution could use a permaproto board (http://adafru.it/571) with the IDC Breakout Helper 2x5 (http://adafru.it/2102) and an IDC cable (http://adafru.it/370). Having all the wires would be helpful, because it would keep a GND signal between each of the high speed signals, for better signal integrity.

In the photo, I have used bare jumper wires (http://adafru.it/3633) inserted into 1-row (http://adafru.it/3145) and 2-row (http://adafru.it/3143) connector housings to make the connections nicer, but this is not strictly necessary.

Now, plug the Feather into a compatible monitor, and apply power to the Feather and the Xerox. After a moment, the Xerox's boot screen should be displayed.

Note that it's normal for the text to be shifted towards the top right hand side of the screen.

Tuning the pixel sampling to remove jitter

Often, jitter by 1 or 2 pixels will be seen, affecting the latter part of a line more than the start of the line.

This occurs because the clock inside the RP2040 is not exactly in sync with the pixels from the Xerox 820.

If the amount of jitter is bothering you, you can try the following things:

- Try adjusting the constant `fine_pixel` in the range 0 to 9.
- Try adjusting the constant `pixel_frequency`, starting with a change of ±100000 (the granularity of this adjustment is about ±40000)

Because each Xerox 820 and each Feather RP2040 have slightly different clock rates which even vary with temperature enough to affect results. Some experimentation will be required to get the most stable display. Commercial video converters have sophisticated circuitry in order to automatically fine tune the conversion from analog to digital, while in this case it's necessary to make do with blunt tools like changing numbers manually until getting the best result possible.

Another possibility is to pick up the pixel clock directly from U14 pin 8 (modifying the PIO program to wait for this edge before sampling each pixel). I did not investigate this method, as I got adequate results without modifying the Xerox PCB.

Note: Powering the Feather

It is not recommended to supply the Feather with 5V power on the pins, so it will need to be powered by a USB cable.

The best option for internal power is to cut open a USB "C" cable, find the 5V wire, and connect it to a source of +5V power on the Xerox 820 PCB, such as the "+" leg of capacitor C314, a silver capacitor near the main power connector. The Feather is **not** designed to accept 5V input on the "USB" pin, only via the USB connector, and is not protected against back-powering or having two power input sources on the same pin.