Return to The Matrix with the Metro RP2350

Created by Anne Barela

https://learn.adafruit.com/return-to-the-matrix-with-the-metro-rp2350

Last updated on 2025-03-06 12:16:43 PM EST

Table of Contents

Overview

Return to The Matrix with this project recreation. The falling streams of green characters are strange yet calming.

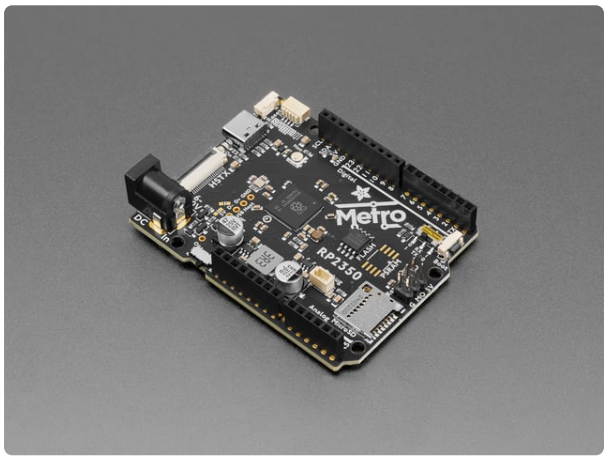
From Quora (https://adafru.it/1afj), the significance of The Matrix computer screen

1. Representation of Reality: The screen symbolizes the digital nature of the Matrix itself. It reflects how the simulated world is constructed and manipulated by the machines controlling humanity.
2. Alternative Perception: Characters like Neo and Morpheus use the screen to perceive the underlying code of the Matrix. It represents the ability to see beyond the illusion of the real world and understand the truth of their existence.
3. Coding Language: The characters are able to read the screen because it displays the Matrix's code, which is a visual representation of the program that constructs their reality. This code is often depicted in green characters on a black background, reminiscent of classic computer interfaces.

Or, it just makes a cool visualization.

This project displays high resolution video generated by an Adafruit Metro RP2350. The HSTX bus outputs DVI video which can be shown on an HDMI monitor. The Adafruit-DVI-HSTX library makes the project easy to write programs like this in Arduino. No soldering required.

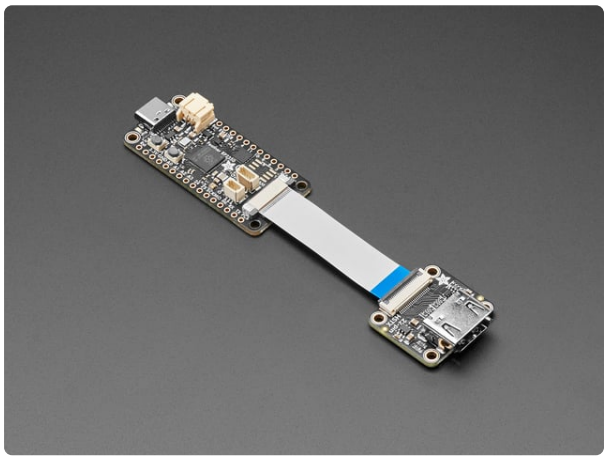
Parts



Adafruit Metro RP2350

Choo! Choo! This is the RP2350 Metro Line, making all station stops at "Dual Cortex M33 mountain", "528K RAM round-about" and "16 Megabytes of Flash...

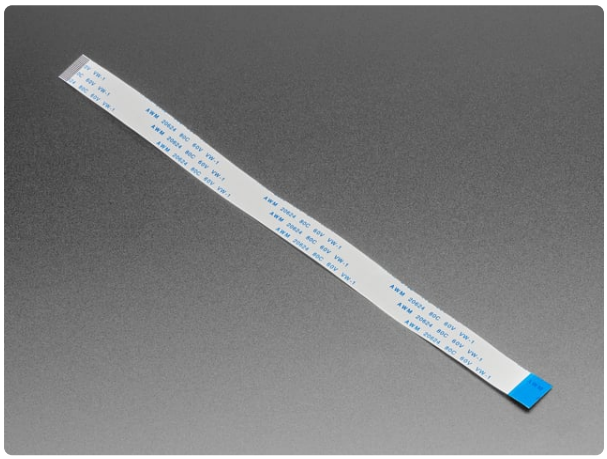
https://www.adafruit.com/product/6003



Adafruit RP2350 22-pin FPC HSTX to DVI Adapter for HDMI Displays

You may have noticed that our RP2350 Feather has an FPC output connector on the end for accessing the HSTX (High Speed...

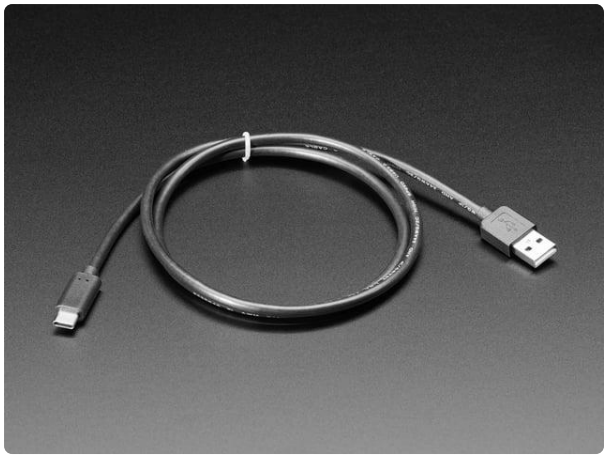
https://www.adafruit.com/product/6055



22-pin 0.5mm pitch FPC Flex Cable for DSI CSI or HSTX - 20cm

Connect this to that when a 22-pin FPC connector is needed. This 20 cm long cable is made of a flexible PCB. It's A-B style, meaning that pin one on one side will match with pin...

https://www.adafruit.com/product/6036



[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

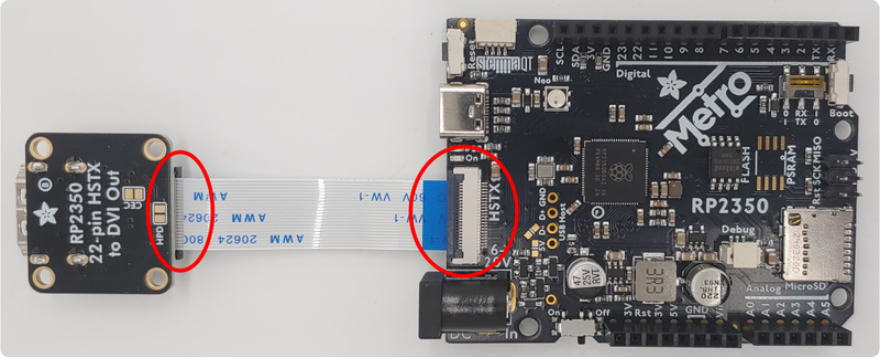
You likely have an HDMI cable and monitor. If not, you can look to get these:

[HDMI Cable - 1 meter]
Connect two HDMI devices together with this basic HDMI cable. It has nice molded grips for easy installation, and is 1 meter long (about 3 feet). This is a HDMI 1.3...
https://www.adafruit.com/product/608

[Raspberry Pi Monitor - Red and White Colorway]
The Raspberry Pi Monitor is a 15.6" full HD computer display. It's the perfect compact, versatile, user-friendly desktop display companion for Raspberry Pi computers and other...
https://www.adafruit.com/product/6088

Preparing the Metro RP2350

HSTX Connection to DVI

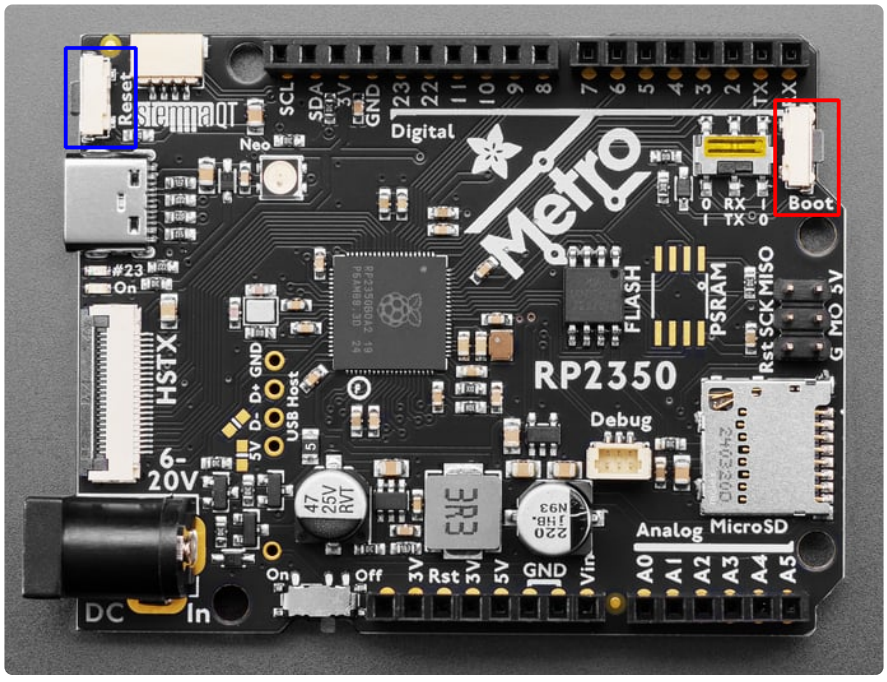


Get the HSTX cable. Any length Adafruit sells is fine. CAREFULLY lift the dark grey bar up on the Metro, insert the cable silver side down, blue side up, then put the bar CAREFULLY down, ensuring it locks in. If it feels like it doesn't want to go, do not force it.

Do the same with the other end on the DVI breakout. Note that the DVI breakout will be inverted/upside down when compared to the Metro - this is normal for these boards and the Adafruit cables.

That's it - no soldering, easy!

Code



This program uses Arduino. If you'd just like to run the program without using Arduino, you can download the .UF2 file below in the green box. Save the file to your computer. Plug the Metro RP2350 into your computer with a known good USB cable (data + power, not a charge only cable).

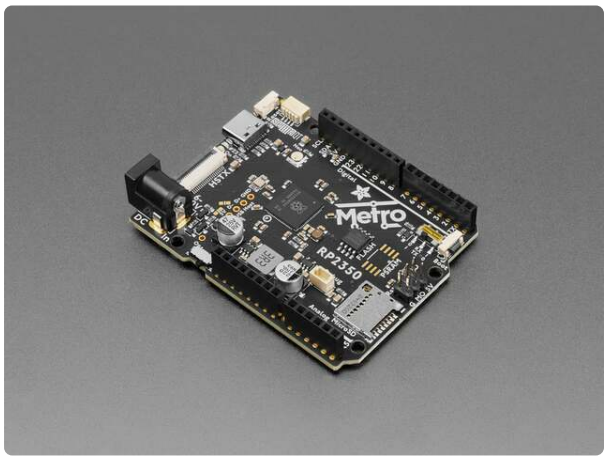
Hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button** until the **RP2350** drive appears on your computer! Copy the .UF2 file you saved previously to the **RP2350** drive and The Matrix should appear if you have the display connected.

matrix.uf2

https://adafru.it/1afk

Arduino

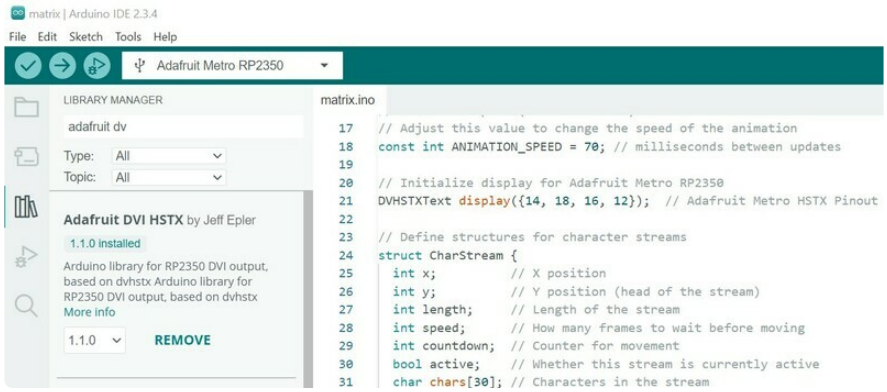
Please refer to the Arduino IDE setup in the Adafruit Metro RP2350 guide (https://adafru.it/1afl).



Adafruit Metro RP2350
By Tim C
Arduino IDE Setup

https://learn.adafruit.com/adafruit-metro-rp2350/arduino-ide-setup

You will want to add the library Adafruit DVI HSTX (version 1.10 or later, likely the latest version) to your Arduino environment. Select Sketch - > Include Library -> Manage Libraries... Search for "Adafruit DVI HSTX" by Jeff Epler. You will want version 1.1.0 or later, likely the newest version. Click Install and accept installing libraries that Adafruit DVI HSTX is dependent on. Those are the only libraries which need to be loaded.



For the main program, get the code by clicking the "Download Project Bundle" button below. Extract the file **Metro_HSTX_Matrix.ino** from the zip archive. Load it into the Arduino IDE.

```
// SPDX-FileCopyrightText: 2021 Anne Barela for Adafruit Industries
//
// SPDX-License-Identifier: MIT
//
// Based on Adafruit-DVI-HSTX library code written by Jeff Epler
// and use of Claude 3.7 Sonnet on 3/2/2025
// https://claude.site/artifacts/cf022b66-50c3-43eb-b334-17fbf0ed791c

#include <Adafruit_dvhstx.h>

// Display configuration for text mode in Adafruit-DVI-HSTX
const int SCREEN_WIDTH = 91;
const int SCREEN_HEIGHT = 30;

// Animation speed (lower = faster)
// Adjust this value to change the speed of the animation
const int ANIMATION_SPEED = 70; // milliseconds between updates

// Initialize display for Adafruit Metro RP2350
DVHSTXText display({14, 18, 16, 12});  // Adafruit Metro HSTX Pinout

// Define structures for character streams
struct CharStream {
  int x;          // X position
  int y;          // Y position (head of the stream)
  int length;     // Length of the stream
  int speed;      // How many frames to wait before moving
  int countdown;  // Counter for movement
  bool active;    // Whether this stream is currently active
  char chars[30]; // Characters in the stream
};

// Array of character streams - increased for higher density
// To fill 60-75% of the screen width (91 chars), we need around 55-68 active
streams
CharStream streams[250]; // Allow for decent density

// Stream creation rate (higher = more frequent new streams)
const int STREAM_CREATION_CHANCE = 65; // % chance per frame to create new stream

// Initial streams to create at startup
const int INITIAL_STREAMS = 30;

// Random characters that appear in the streams
const char matrixChars[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_=+[]
{}|;:,.<>?/\\";
const int numMatrixChars = sizeof(matrixChars) - 1;
```

```
// Function declarations
void initStreams();
void updateStreams();
void drawStream(CharStream &stream);
void createNewStream();
char getRandomChar();

void setup() {
  // Initialize the display
  display.begin();
  display.clear();

  // Seed the random number generator
  randomSeed(analogRead(A0));

  // Initialize all streams
  initStreams();
}

void loop() {
  // Update and draw all streams
  updateStreams();

  // Randomly create new streams at a higher rate
  if (random(100) < STREAM_CREATION_CHANCE) {
    createNewStream();
  }

  // Control animation speed
  delay(ANIMATION_SPEED);
}

void initStreams() {
  // Initialize all streams as inactive
  for (int i = 0; i < sizeof(streams) / sizeof(streams[0]); i++) {
    streams[i].active = false;
  }

  // Create more initial streams for immediate visual impact
  for (int i = 0; i < INITIAL_STREAMS; i++) {
    createNewStream();
  }
}

void createNewStream() {
  // Find an inactive stream
  for (int i = 0; i < sizeof(streams) / sizeof(streams[0]); i++) {
    if (!streams[i].active) {
      // Initialize the stream
      streams[i].x = random(SCREEN_WIDTH);
      streams[i].y = random(5) - 5; // Start above the screen
      streams[i].length = random(5, 20);
      streams[i].speed = random(1, 4);
      streams[i].countdown = streams[i].speed;
      streams[i].active = true;

      // Fill with random characters
      for (int j = 0; j < streams[i].length; j++) {
        streams[i].chars[j] = getRandomChar();
      }

      return;
    }
  }
}

void updateStreams() {
  display.clear();
```

```
  // Count active streams (for debugging if needed)
  int activeCount = 0;

  for (int i = 0; i < sizeof(streams) / sizeof(streams[0]); i++) {
    if (streams[i].active) {
      activeCount++;
      streams[i].countdown--;

      // Time to move the stream down
      if (streams[i].countdown <= 0) {
        streams[i].y++;
        streams[i].countdown = streams[i].speed;

        // Change a random character in the stream
        int randomIndex = random(streams[i].length);
        streams[i].chars[randomIndex] = getRandomChar();
      }

      // Draw the stream
      drawStream(streams[i]);

      // Check if the stream has moved completely off the screen
      if (streams[i].y - streams[i].length > SCREEN_HEIGHT) {
        streams[i].active = false;
      }
    }
  }
}

void drawStream(CharStream &stream) {
  for (int i = 0; i < stream.length; i++) {
    int y = stream.y - i;

    // Only draw if the character is on screen
    if (y >= 0 && y < SCREEN_HEIGHT) {
      display.setCursor(stream.x, y);

      // Set different colors/intensities based on position in the stream
      if (i == 0) {
        // Head of the stream is white (brightest)
        display.setColor(TextColor::TEXT_WHITE, TextColor::BG_BLACK,
TextColor::ATTR_NORMAL_INTEN);
      } else if (i < 3) {
        // First few characters are bright green
        display.setColor(TextColor::TEXT_GREEN, TextColor::BG_BLACK,
TextColor::ATTR_NORMAL_INTEN);
      } else if (i < 6) {
        // Next few are medium green
        display.setColor(TextColor::TEXT_GREEN, TextColor::BG_BLACK,
TextColor::ATTR_LOW_INTEN);
      } else {
        // The rest are dim green
        display.setColor(TextColor::TEXT_GREEN, TextColor::BG_BLACK,
TextColor::ATTR_V_LOW_INTEN);
      }

      // Draw the character
      display.write(stream.chars[i]);
    }
  }

  // Occasionally change a character in the stream
  if (random(100) < 25) { // 25% chance
    int idx = random(stream.length);
    stream.chars[idx] = getRandomChar();
  }
}
```

```
char getRandomChar() {
  return matrixChars[random(numMatrixChars)];
}
```

Select Adafruit Metro RP2350 as the board in the box in the toolbar.

Plug your Metro into your computer via a known good USB data + power cable (not the tiny power-only cables that come with battery packs). The Metro should show up as a new serial port. Select that serial port under Tools -> Port.

Click the arrow key -> on the toolbar to compile the program and upload it to the Metro.

Changing the Code

You can change how many streams are on the screen at the same time by changing the `STREAM_CREATION_CHANCE` variable. It should be from `10` (lowest) to `99` (highest).

Using Pico 2 instead of Metro RP2350

The HSTX pins for Metro RP2350 are defined as:

```
DVHSTXText display({14, 18, 16, 12});
```

But for other boards you may need to use

```
DVHSTXText display({12, 14, 16, 18});
```

It's all depending how you wired your HSTX bus - the wire pairs can be swapped IF your code knows it. It would help if the Pico 2 had an HSTX connector on top. Oh well.

Usage

Plug the Metro into a HDMI monitor via an HDMI cable.

Power the Metro RP2350 either via USB C (5 volts) or the barrel power connection (5.5 to 17 volts DC, center positive).

The Matrix animation will automatically start!

Changing the Number of Streams

See the Code page for the variable you'll want to change to get more or fewer streams.