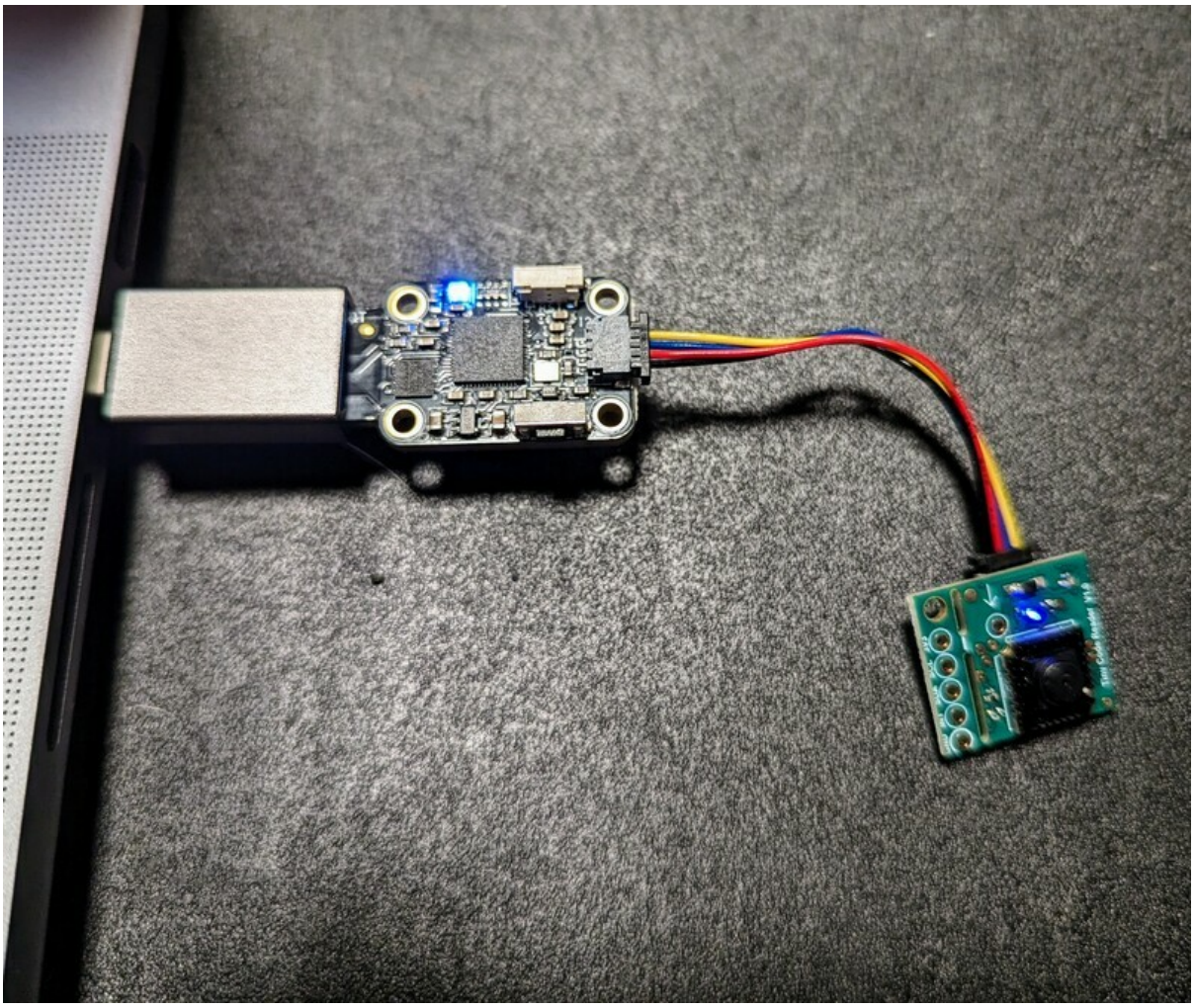
Reading QR Codes with the Tiny Code Reader

Created by Pete Warden

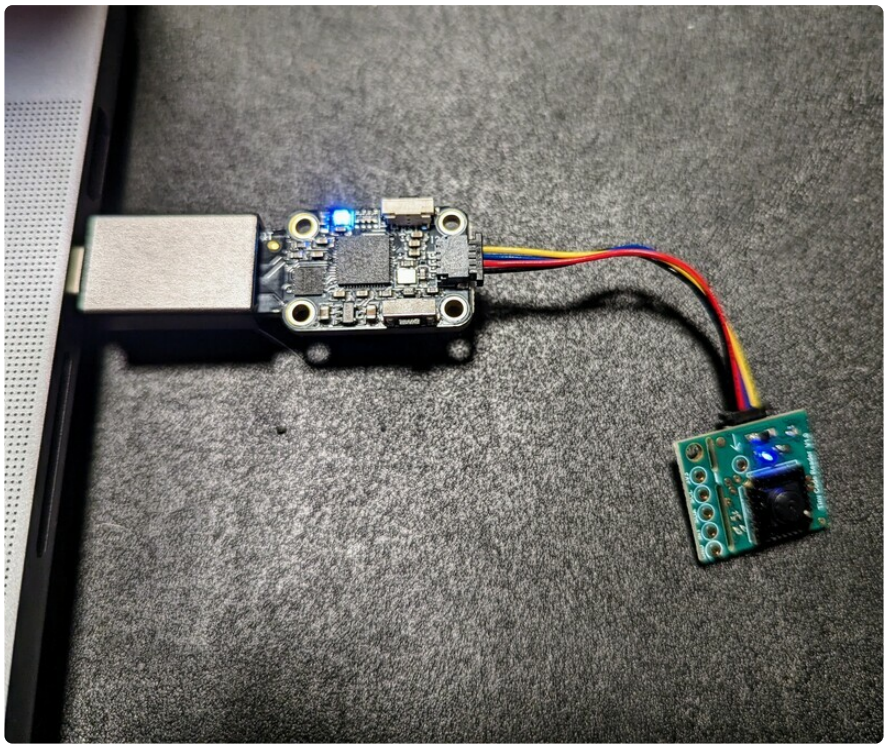


https://learn.adafruit.com/reading-qr-codes-with-the-tiny-code-reader

Last updated on 2024-06-03 03:54:26 PM EDT

Table of Contents

Overview

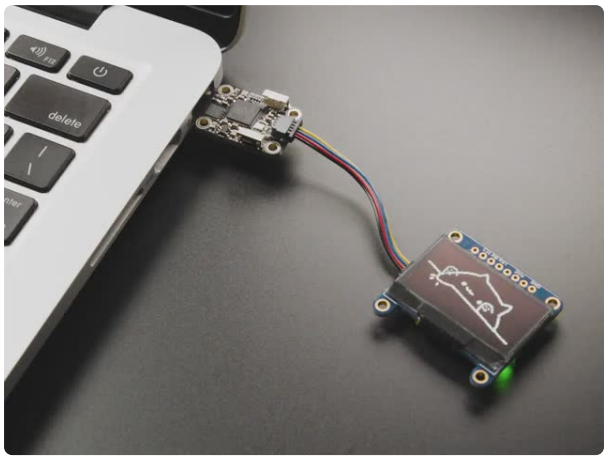


The Tiny Code Reader (http://adafru.it/5744) is a small hardware module that's intended to make it easy to scan QR codes. It has an image sensor and a microcontroller with pre-trained software and outputs information from any identified codes over I2C.

There's a detailed developer manual (https://adafru.it/18Xa) available, but this guide has sample code that shows you specifically how to get the module up and running using CircuitPython on a Trinkey (http://adafru.it/5056), and output any recognized code as text to the attached computer as if the module was a keyboard. For example, you can show it a QR code containing the text "Hello World!" and that text will be automatically typed in to your laptop.

Both the TCR and Trinkey have Stemma QT/Qwiic ports, so they can be connected with a standard cable and no soldering is required to complete this project.

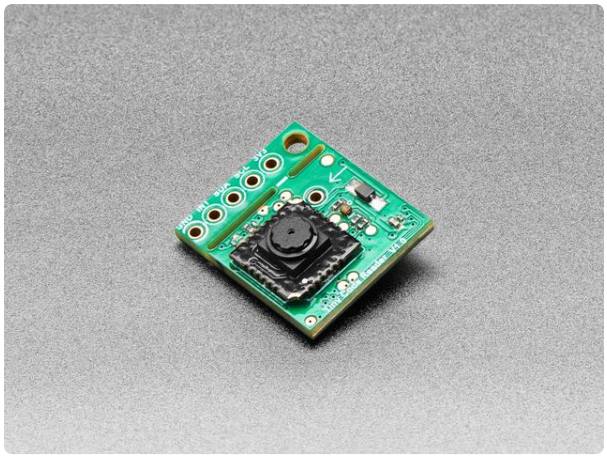
Parts



Adafruit Trinkey QT2040 - RP2040 USB Key with Stemma QT

It's half USB Key, half Adafruit QT Py, and a lotta RP2040...it's Trinkey QT2040, the circuit board with an RP2040 heart and Stemma QT legs....

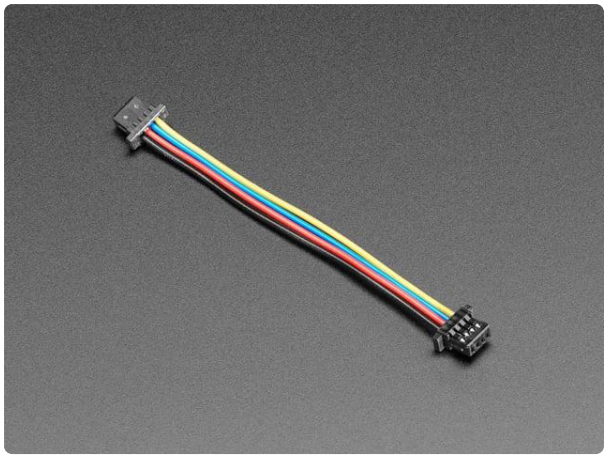
https://www.adafruit.com/product/5056



Tiny Code Reader from Useful Sensors

The Tiny Code Reader from Useful Sensors is a...

https://www.adafruit.com/product/5744



STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...

https://www.adafruit.com/product/4399

Setup

Are you new to using CircuitPython? No worries, there is a full getting started guide for the Trinkey board here (https://adafru.it/XAL). Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and installation in this tutorial (https://adafru.it/ANO).

The Trinkey comes with a Stemma QT connector, so the only wiring you'll need to do is plugging the recommended cable (or equivalent) into the reader and the board port. The connectors will only attach in the correct orientation, which is with the slightly exposed contact pads closest to the board.

Code

Once you've finished setting up your Trinkey with CircuitPython and have it plugged into your computer and the Tiny Code Reader, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download to your computer as a zipped bundle.

```
# SPDX-FileCopyrightText: 2023 Pete Warden
# SPDX-License-Identifier: Apache-2.0
#
# Example of accessing the Tiny Code Reader from Useful Sensors on a Trinkey
# using CircuitPython. See https://usfl.ink/tcr_dev for the full developer guide.

import struct
import time
import board
import usb_hid

from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS

# The code reader has the I2C ID of hex 0c, or decimal 12.
TINY_CODE_READER_I2C_ADDRESS = 0x0C

# How long to pause between sensor polls.
TINY_CODE_READER_DELAY = 0.2

TINY_CODE_READER_LENGTH_OFFSET = 0
TINY_CODE_READER_LENGTH_FORMAT = "H"
TINY_CODE_READER_MESSAGE_OFFSET = TINY_CODE_READER_LENGTH_OFFSET + \
                                  struct.calcsize(TINY_CODE_READER_LENGTH_FORMAT)
TINY_CODE_READER_MESSAGE_SIZE = 254
TINY_CODE_READER_MESSAGE_FORMAT = "B" * TINY_CODE_READER_MESSAGE_SIZE
TINY_CODE_READER_I2C_FORMAT = TINY_CODE_READER_LENGTH_FORMAT +
TINY_CODE_READER_MESSAGE_FORMAT
TINY_CODE_READER_I2C_BYTE_COUNT = struct.calcsize(TINY_CODE_READER_I2C_FORMAT)

i2c = board.I2C()

# Wait until we can access the bus.
while not i2c.try_lock():
    pass

# For debugging purposes print out the peripheral addresses on the I2C bus.
# 98 (0x62 in hex) is the address of our person sensor, and should be
# present in the list. Uncomment the following three lines if you want to see
# what I2C addresses are found.
# while True:
#    print(i2c.scan())
#    time.sleep(TINY_CODE_READER_DELAY)

# Create a keyboard device so we can send the screen lock command.
```

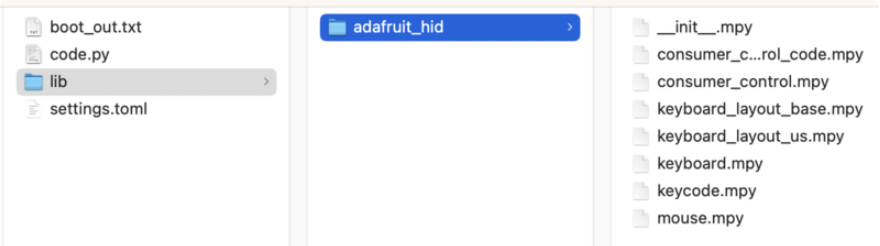
```
keyboard = Keyboard(usb_hid.devices)
layout = KeyboardLayoutUS(keyboard)

last_message_string = None
last_code_time = 0.0

while True:
    read_data = bytearray(TINY_CODE_READER_I2C_BYTE_COUNT)
    i2c.readfrom_into(TINY_CODE_READER_I2C_ADDRESS, read_data)

    message_length,  = struct.unpack_from(TINY_CODE_READER_LENGTH_FORMAT, read_data,
                                          TINY_CODE_READER_LENGTH_OFFSET)
    message_bytes = struct.unpack_from(TINY_CODE_READER_MESSAGE_FORMAT, read_data,
                                       TINY_CODE_READER_MESSAGE_OFFSET)

    if message_length > 0:
        message_string = bytearray(message_bytes)[0:message_length].decode("utf-8")
        is_same = (message_string == last_message_string)
        last_message_string = message_string
        current_time = time.monotonic()
        time_since_last_code = current_time - last_code_time
        last_code_time = current_time
        # Debounce the input by making sure there's been a gap in time since we
        # last saw this code.
        if (not is_same) or (time_since_last_code > 1.0):
            print(message_string)
            try:
                layout.write(message_string)
            except ValueError as e:
                pass
            layout.write("\n")
```

Upload the Code and Libraries to the Trinkey

After downloading the Project Bundle, plug your Trinkey into the computer's USB port with a known good USB data+power cable. If your computer only has USB C ports, you will need a USB A to C adapter (http://adafru.it/4175). You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder, and copy the following items from the CircuitPython 8.x folder to the Trinkey's **CIRCUITPY** drive.

- **lib** folder
- **code.py**

Your Trinkey **CIRCUITPY** drive should look like this after copying the **lib** folder and the **code.py** file.



Use

Your Trinkey should now be set up to type the contents of QR codes into your computer! To test it, go to a site like qr-code-generator.com (https://adafru.it/18Xb) and type in some text. A QR code should be displayed, and if you create a blank document in a text editor and give it keyboard focus, when you show the QR code to the Tiny Code Reader you should see the contents get typed in.

Going Further

The great thing about QR codes is that you can use them to share any short text strings without needing a special app, since they're just images that can be printed out, emailed, or texted. This means that if you have information you want to input into your project, but don't want to include a keyboard in your system, a TCR can be a helpful alternative. The full developer guide (https://adafru.it/18Xa) has more examples (https://adafru.it/18Xc), but here are some suggested use cases.

WiFi Provisioning

On Android, you can just go to WiFi settings, choose share (https://adafru.it/18Xd), and it will display a QR code containing the name and password of the network you're currently connected to. It's also possible to do the same thing on iOS (https://adafru.it/18Xe) but it's a bit more fiddly. If you have a project that you want to connect to WiFi without reflashing or using a keyboard, you can read these QR codes to extract the network name and password (https://adafru.it/18Xf). This can be extended to may different kinds of configuration, as long as the information fits in a short text string.

Access Control

Instead of using a keypad with a PIN code to lock a box or door, why not use a password saved in a QR code? You can have a much longer secret since users don't have to remember it, and even multiple different passwords with different access levels. For example, you could program a door lock with a code that only works on a particular day between 9am and 5pm if you wanted to give a contractor temporary access.

E-ink Badges and Displays

E-ink displays are great because they don't use any power to maintain a picture. This makes them ideal for things like conference badges, price labels, or meeting room names, since the contents don't need to change frequently and so a device can run for months or years on a small battery. It's not always easy to update them though, since network protocols like Bluetooth or WiFi require extra hardware and fiddly setup to connect, and a keyboard won't fit the form factor.

Adding a Tiny Code Reader that's triggered on a button press makes it easy to update displays with new information, since the process just involves creating a QR code with the text you want and pressing a button to scan it. This Badger RP2040 project (https://adafru.it/18XA) shows one way of doing that. Even better, e-ink displays are great at displaying QR codes too, so it's even possible to share information between devices by scanning displays, for example to share contact information from a conference badge.