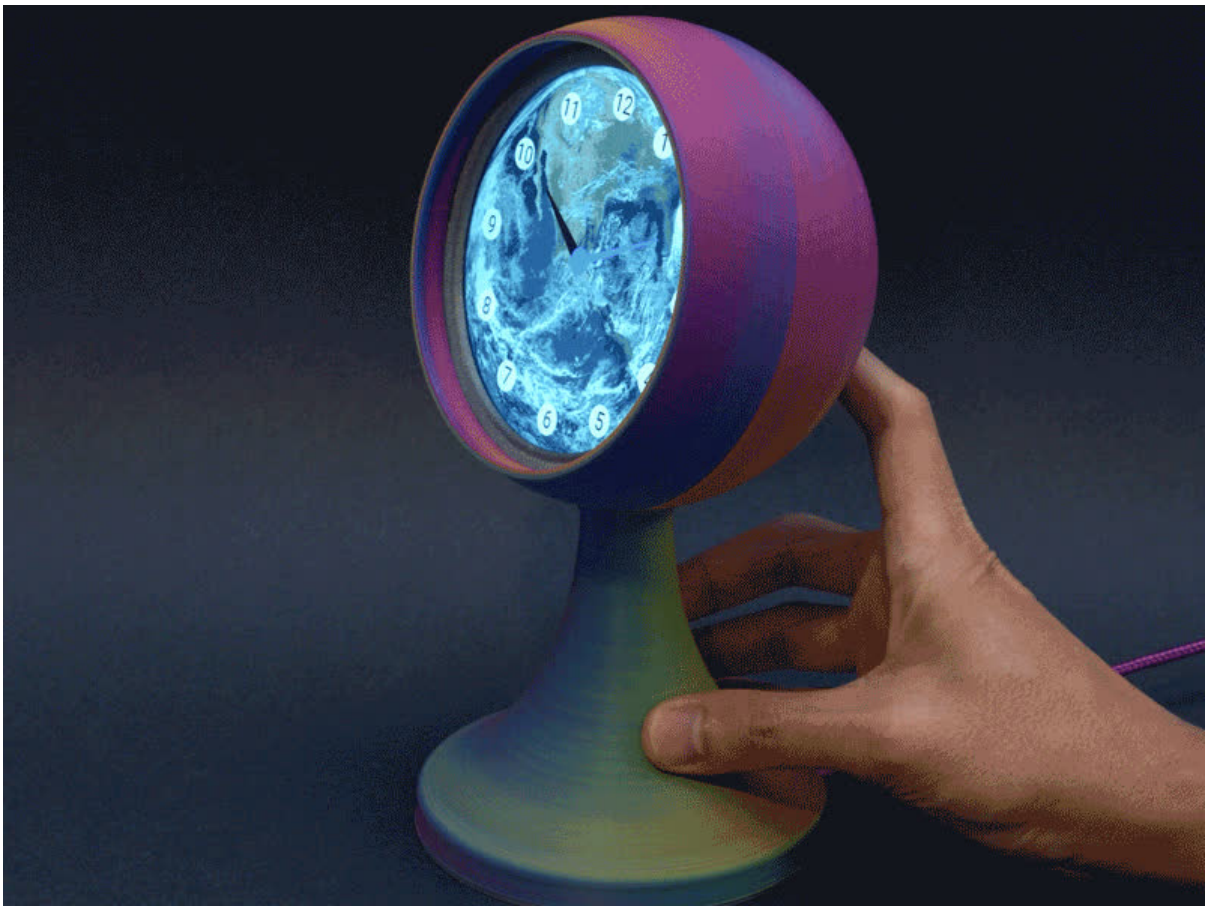
Qualia S3 Space Clock

Created by Liz Clark



https://learn.adafruit.com/qualia-s3-space-clock

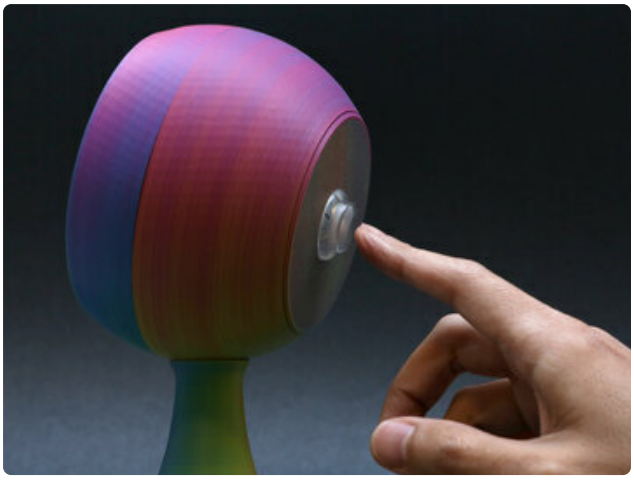
Last updated on 2024-11-18 12:55:52 PM EST

Table of Contents

Overview

David Bowie asked if there was life on Mars, but what about time? With this project, you can easily switch between viewing your local time on Earth and coordinated Mars time (MTC) on a beautiful round 720x720 display housed in a retro space-themed orb enclosure.

The display shows a classic analog clock face for viewing the time. Hour and minute hands are created with `vectorio` polygons in CircuitPython.



An arcade button mounted to the back of the clock lets you switch between Earth and Mars time.

Inspired by mid-century modern clocks, this 3D printed enclosure embodies a space-age retro aesthetic. The base features beautiful curves that blend the neck to the wide base. The display is mounted to a spherical dome that's tilted 15 degrees for the best viewing angle. The display can swivel slightly, allowing the user to adjust it.

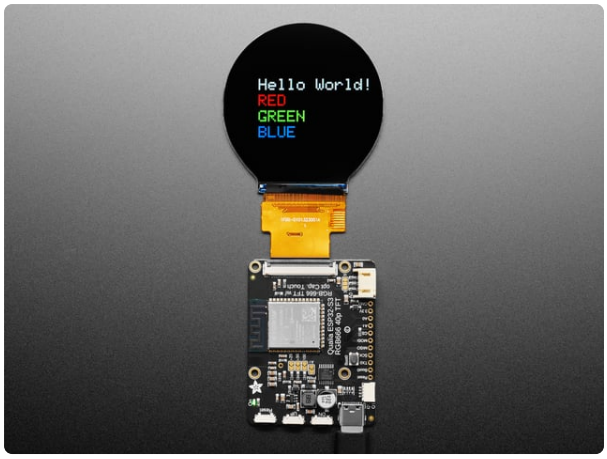
The base features an opening for the USB-C port, two user buttons and a reset button.

Prerequisite Guides

Take a moment to review the following guides to learn more about the products.

- Adafruit Qualia ESP32-S3 Guide (https://adafru.it/19ak)

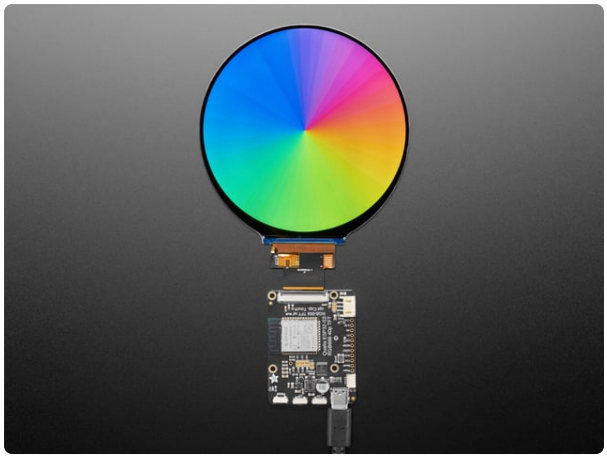
Parts



Adafruit Qualia ESP32-S3 for TTL RGB-666 Displays

There's a few things everyone loves: ice cream, kittens, and honkin' large TFT screens. We're no strangers to small TFT's -

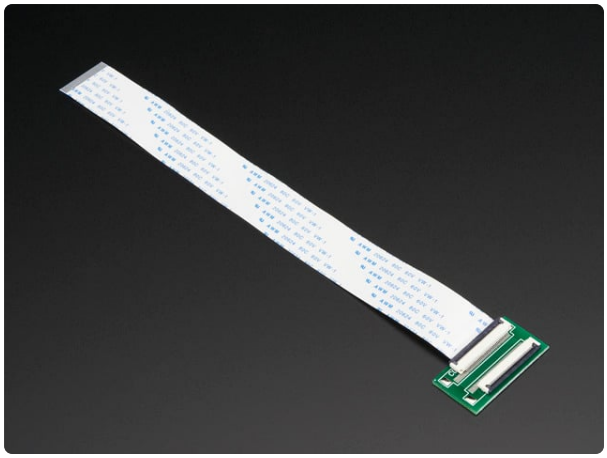
https://www.adafruit.com/product/5800



Round RGB TTL TFT Display - 4" 720x720 - No Touchscreen

This is a screen for advanced hackers who like the look of a nice, round TFT screen with tons of pixels. This massive 4" diagonal-sized display has 720x720 16-bit full-color...

https://www.adafruit.com/product/5793



40-pin FPC Extension Board + 200mm Cable

Give your 40 pin, 0.5mm pitch, devices a strrrreeeetch with this extension board. This 40pin FPC extension board has two 40-pin flex connectors (both are bottom contact type),...

https://www.adafruit.com/product/2098

Mini LED Arcade Button - 24mm Translucent Clear

A button is a button, and a switch is a switch, but these translucent arcade buttons are in a class of their own. Particularly because they have LEDs built right in!...

https://www.adafruit.com/product/3429

[STEMMA JST PH 2mm 3-Pin to Female Socket Cable - 200mm]
This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" female header sockets on the end. We're carrying these to match up with...
https://www.adafruit.com/product/3894

[Pink and Purple Woven USB A to USB C Cable - 1 meter long]
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...
https://www.adafruit.com/product/5153

Hardware

Required screws and nuts for assembly.

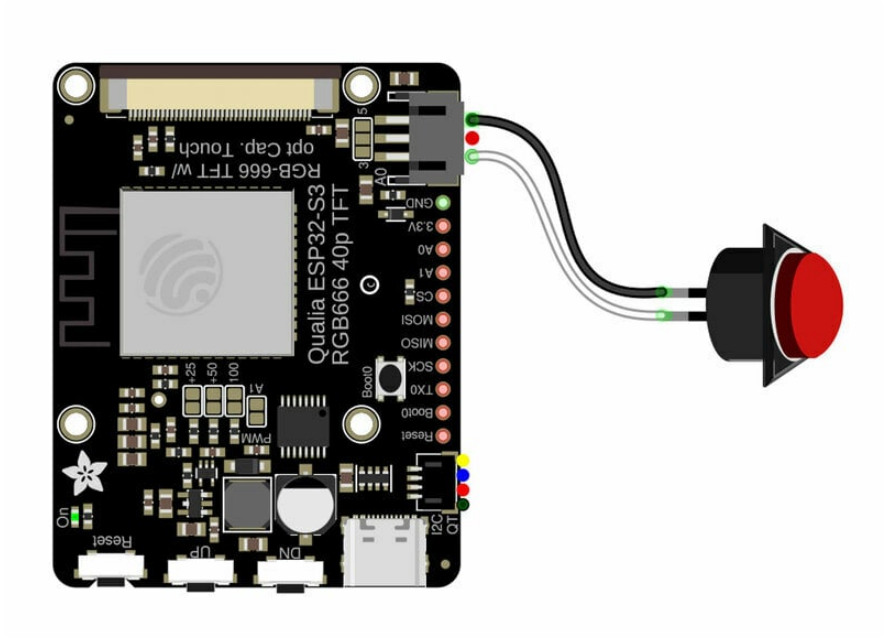
- 4x M2.5 x 6mm long pan head machine screw

Circuit Diagram

The diagram below provides a general visual reference for wiring of the components once you get to the **Assembly** page. This diagram was created using the software package Fritzing (https://adafru.it/oEP).

Adafruit Library for Fritzing

Adafruit uses the Adafruit's Fritzing parts library to create circuit diagrams for projects. You can download the library or just grab individual parts. Get the library and parts from GitHub - Adafruit Fritzing Parts (https://adafru.it/AYZ).



Wired Connections

The Qualia ESP32-S3 is powered by a 5V 1A USB power supply.

- The arcade button connects to the Qualia ESP32-S3 using a 3-pin JST Stemma cable.

CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

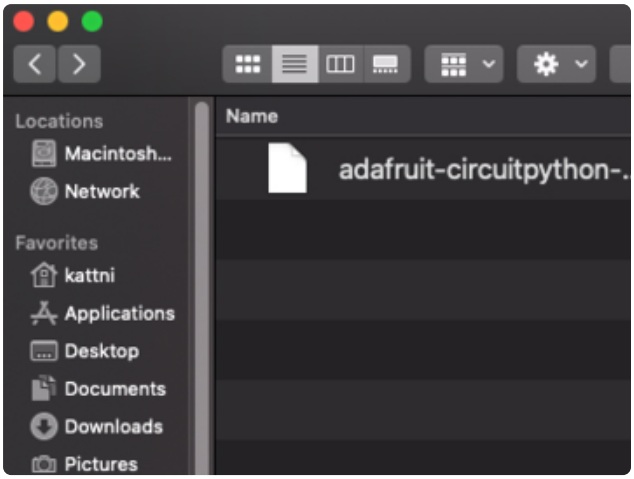
CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

This microcontroller requires the latest **unstable (development)** release of CircuitPython. Click below to visit the downloads page on **circuitpython.org** for your board. Then, Browse **S3** under **Absolute Newest**.

Download the latest version of CircuitPython for this board via circuitpython.org

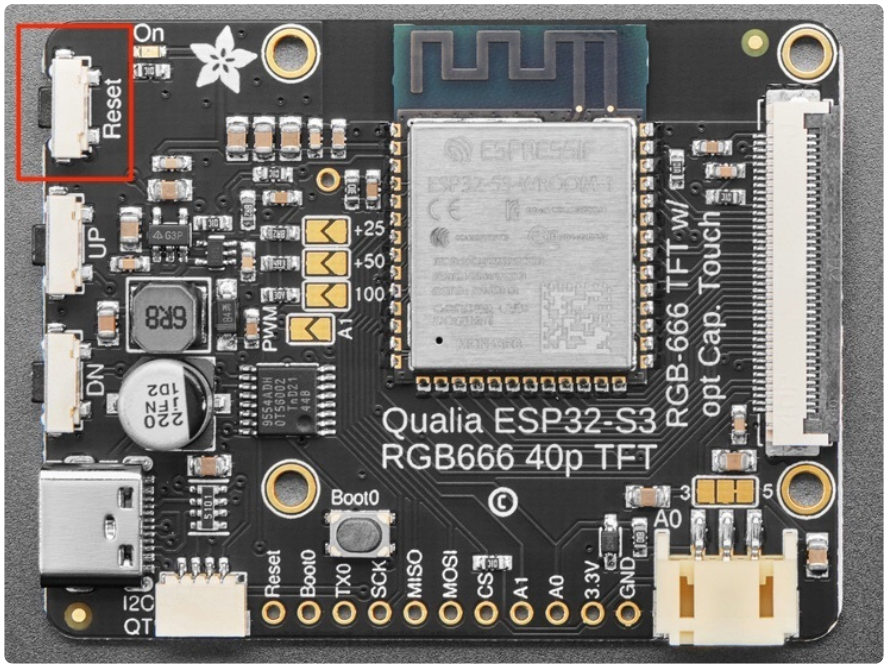
https://adafru.it/191D



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

The Qualia S3 does not have a RGB status LED



Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Double-click the **reset** button (highlighted in red above), and you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

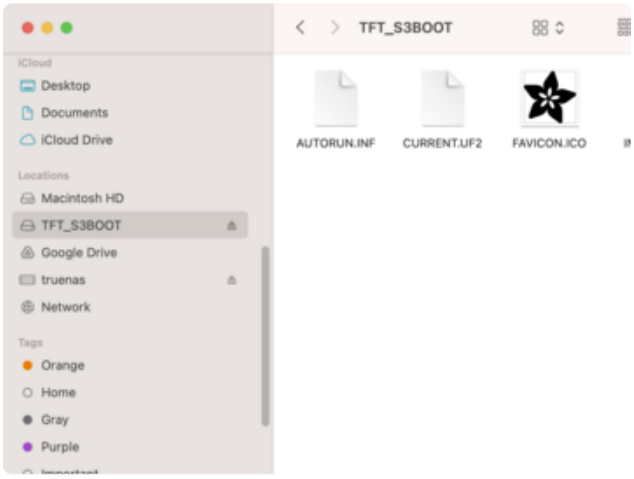
This board does not have a Neopixel, so you will need to just double tap the reset button.

For this board, tap reset and wait about a half a second and then tap reset again.

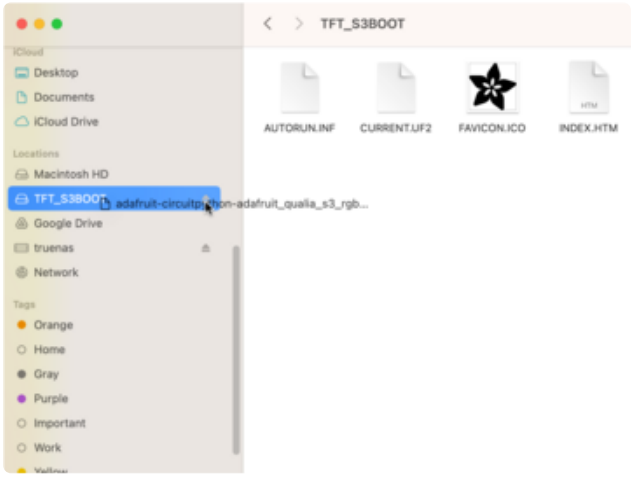
Some boards may not have a UF2 bootloader installed. If double-clicking does not work, follow the instructions on the "Install UF2 Bootloader" page in this guide.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

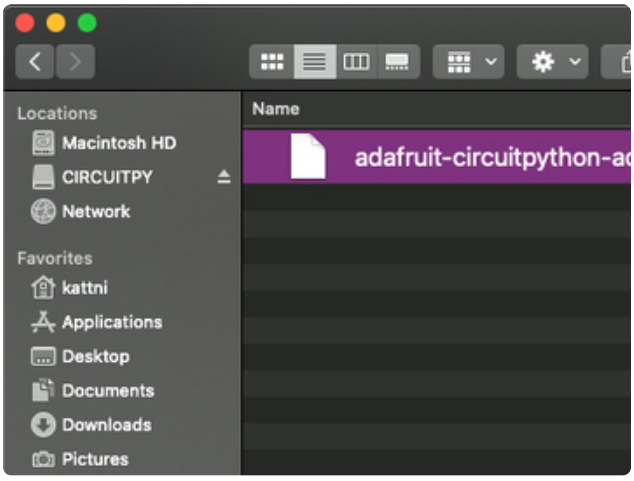
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **TFT_S3BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **TFT_S3BOOT**.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it!

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

> Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of `ADAFRUIT_AIO_USERNAME`. **If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.**

> Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a **settings.toml** file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.toml** files
    - Example: `\U0001f44d` for (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format

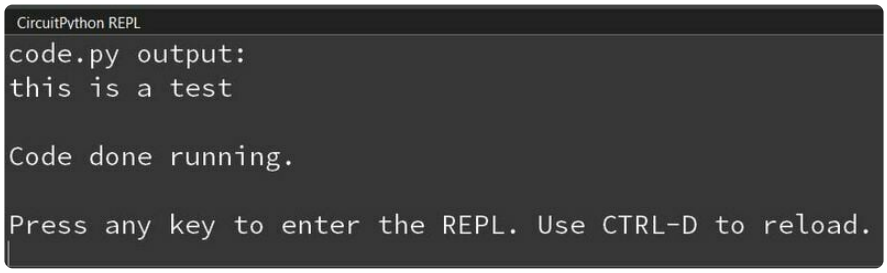
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to `import` the `os` library to access the **settings.toml** file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os

print(os.getenv("test_variable"))
```



In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

Code the Clock

Once you've finished setting up your Qualia ESP32-S3 with CircuitPython, 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 folder.

```
# SPDX-FileCopyrightText: 2023 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Written by Liz Clark (Adafruit Industries) with OpenAI ChatGPT v4 Aug 3rd, 2023
build
# https://help.openai.com/en/articles/6825453-chatgpt-release-notes
```

```
# https://chat.openai.com/share/63cbe4c6-007f-4934-a458-a9c8a521620e
# https://chat.openai.com/share/674c0f13-bc78-4d1e-be79-3bc777e29991

import time
from math import pi, cos, sin
import os
import ssl
import wifi
import socketpool
import adafruit_requests
import board
from adafruit_ticks import ticks_ms, ticks_add, ticks_diff
import vectorio
import displayio
from adafruit_io.adafruit_io import IO_HTTP
from jepler_udecimal import Decimal
import keypad
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font
from adafruit_qualia.graphics import Graphics, Displays

# timezone offset for calculating mars time
timezone = -5

key = keypad.Keys((board.A0,), value_when_pressed=False, pull=True)

wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")

aio_username = os.getenv('ADAFRUIT_AIO_USERNAME')
aio_key = os.getenv('ADAFRUIT_AIO_KEY')

context = ssl.create_default_context()
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, context)
io = IO_HTTP(aio_username, aio_key, requests)

earth_bitmap = displayio.OnDiskBitmap("/earth.bmp")
mars_bitmap = displayio.OnDiskBitmap("/mars.bmp")

graphics = Graphics(Displays.ROUND40, default_bg=None, auto_refresh=True)

earth_grid = displayio.TileGrid(earth_bitmap,
pixel_shader=earth_bitmap.pixel_shader)
earth_group = displayio.Group()
earth_group.append(earth_grid)

mars_grid = displayio.TileGrid(mars_bitmap, pixel_shader=mars_bitmap.pixel_shader)
mars_group = displayio.Group()
mars_group.append(mars_grid)

def center(grid, bitmap):
    # center the image
    grid.x -= (bitmap.width - graphics.display.width) // 2
    grid.y -= (bitmap.height - graphics.display.height) // 2

center(mars_grid, mars_bitmap)
center(earth_grid, earth_bitmap)

graphics.display.root_group = mars_group

# pointer using vectorio, first the hub
pointer_pal = displayio.Palette(4)
pointer_pal[0] = 0xff0000
pointer_pal[1] = 0x000000
pointer_pal[2] = 0x0000ff
pointer_pal[3] = 0xffffff
```

```
pointer_hub = vectorio.Circle(pixel_shader=pointer_pal, radius=26, x=0, y=0)
pointer_hub.x = graphics.display.width // 2
pointer_hub.y = graphics.display.height // 2

# minute hand
mw = 23
mh = 225
min_points = [(mw,0), (mw,-mh), (-mw,-mh), (-mw,0)]
min_hand = vectorio.Polygon(pixel_shader=pointer_pal, points=min_points, x=0,y=0)
min_hand.x = graphics.display.width // 2
min_hand.y = graphics.display.height // 2
mars_group.append(min_hand)
earth_group.append(min_hand)
# hour hand
hw = 25
hh = 175
hour_points = [(hw,0), (hw,-hh), (-hw,-hh), (-hw,0)]
hour_hand = vectorio.Polygon(pixel_shader=pointer_pal, points=hour_points,
                             x=0, y=0, color_index=1)
hour_hand.x = graphics.display.width // 2
hour_hand.y = graphics.display.height // 2
mars_group.append(hour_hand)
earth_group.append(hour_hand)

# add numbers to the clock face
def calculate_number_position(number, center_x, center_y, radius):
    angle = (360 / 12) * (number - 3)  # -3 adjusts the angle to start at 12 o'clock
    rad_angle = pi * angle / 180
    if number >=8:
        x = int(center_x + cos(rad_angle) * radius-40)
    x = int(center_x + cos(rad_angle) * radius)
    y = int(center_y + sin(rad_angle) * radius)
    return x, y

clock_face_numbers = {i: calculate_number_position(i, graphics.display.width // 2,
                         graphics.display.height // 2, 300) for i in range(1, 13)}

font_file = "/Roboto-Regular-47.pcf"

for i in range(1, 13):
    mars_c = vectorio.Circle(pixel_shader=pointer_pal, radius=30,
x=clock_face_numbers[i][0]+12,
                             y=clock_face_numbers[i][1], color_index=1)
    earth_c = vectorio.Circle(pixel_shader=pointer_pal, radius=30,
x=clock_face_numbers[i][0]+12,
                             y=clock_face_numbers[i][1], color_index=3)
    if i >= 10:
        mars_c.x = mars_c.x + 14
        earth_c.x = earth_c.x + 14
    mars_group.append(mars_c)
    earth_group.append(earth_c)
    text = str(i)
    font = bitmap_font.load_font(font_file)

    mars_text = label.Label(font, text=text, color=0xFFFFFF)
    earth_text = label.Label(font, text=text, color=0x000000)
    mars_text.x = clock_face_numbers[i][0]
    mars_text.y = clock_face_numbers[i][1]
    earth_text.x = clock_face_numbers[i][0]
    earth_text.y = clock_face_numbers[i][1]
    mars_group.append(mars_text)
    earth_group.append(earth_text)

mars_group.append(pointer_hub)
earth_group.append(pointer_hub)

# get time from adafruit io
# called once an hour in the loop
def update_time():
```

```
    print("time")
    now = io.receive_time()
    return now

def convert_time(the_time):
    h = the_time[3]
    if h >= 12:
        h -= 12
        a = "PM"
    else:
        a = "AM"
    if h == 0:
        h = 12
    return h, a

# get mars time
def mars_time():
    dt = io.receive_time()
    print(dt)
    utc_offset = 3600 * -timezone
    tai_offset = 37
    millis = time.mktime(dt)
    unix_timestamp = millis + utc_offset

    # Convert to MSD
    msd = (unix_timestamp + tai_offset) / Decimal("88775.244147") +
Decimal("34127.2954262")
    print(msd)
    # Convert MSD to MTC
    mtc = (msd % 1) * 24
    mtc_hours = int(mtc)
    mtc_minutes = int((mtc * 60) % 60)
    mtc_seconds = int(((mtc * 3600)) % 60)

    print(f"Mars Time: {mtc_hours:02d}:{mtc_minutes:02d}:{mtc_seconds:02d}")
    return mtc_minutes, mtc_hours

def time_angle(m, the_hour):
    m_offset = 25 if 12 <= m < 18 or 42 <= m < 48 else 5
    h_offset = 25 if 2 <= the_hour % 12 < 4 or 8 <= the_hour % 12 < 10 else 5
    # Adjusted angle calculation for minute hand
    theta_minute = 360 - (m / 60) * 360
    theta_hour = ((the_hour / 12) + (m / (12 * 60))) * 360
    # Calculate coordinates for minute hand (mirrored)
    minute_x = -int(cos(pi * (theta_minute - 90) / 180) * mh)
    minute_y = int(sin(pi * (theta_minute + 90) / 180) * mh)
    hour_x = int(cos(pi * (theta_hour - 90) / 180) * hh)
    hour_y = int(sin(pi * (theta_hour + 90) / 180) * hh)
    min_hand.points = [(mw, 0), (minute_x + m_offset, -minute_y),
                       (minute_x - m_offset, -minute_y), (-mw, 0)]
    hour_hand.points = [(hw, 0), (hour_x + h_offset, -hour_y),
                        (hour_x - h_offset, -hour_y), (-hw, 0)]

clock_timer = 1 * 1000
clock_clock = ticks_ms()
clock = update_time()
hour, am_pm = convert_time(clock)
tick = clock[5]
minute = clock[4]
mars_min, mars_hour = mars_time()
show_earth = True

time_angle(minute, hour)

while True:
    event = key.events.get()
    # swap between earth or mars time
    if event:
        if event.pressed:
```

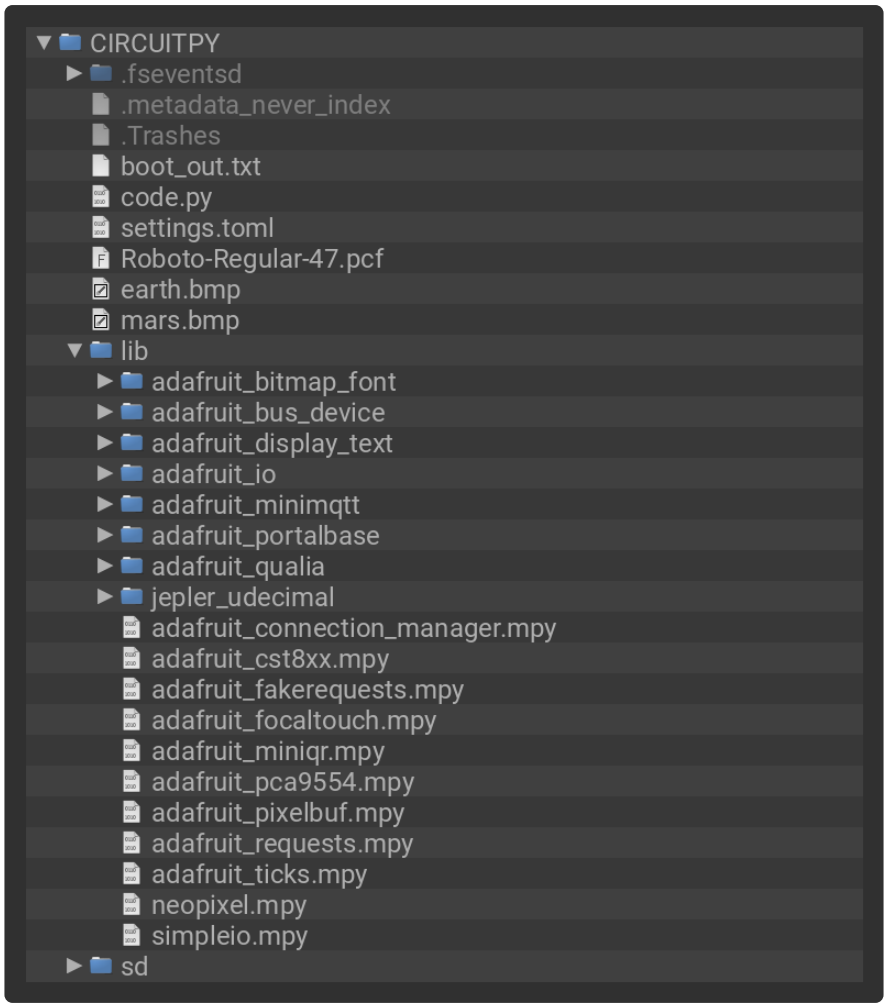
```
            print("updating display")
            show_earth = not show_earth
    # update background image
    # change minute hand color depending on background
    if show_earth:
        if min_hand.color_index != 2:
            time_angle(minute, hour)
            graphics.display.root_group = earth_group
            min_hand.color_index = 2
            pointer_hub.color_index = 2
    else:
        if min_hand.color_index != 0:
            time_angle(mars_min, mars_hour)
            graphics.display.root_group = mars_group
            min_hand.color_index = 0
            pointer_hub.color_index = 0
    # use ticks for timekeeping
    # every minute update clock hands
    # recheck IO time every hour
    if ticks_diff(ticks_ms(), clock_clock) >= clock_timer:
        tick += 1
        if tick > 59:
            tick = 0
            minute += 1
            if minute > 59:
                clock = update_time()
                hour, am_pm = convert_time(clock)
                tick = clock[5]
                minute = clock[4]
            print(f"{hour}:{minute:02} {am_pm}")
            mars_min, mars_hour = mars_time()
            if show_earth:
                time_angle(minute, hour)
            else:
                time_angle(mars_min, mars_hour)
        clock_clock = ticks_add(clock_clock, clock_timer)
```

Upload the Code and Libraries to the Qualia ESP32-S3

After downloading the Project Bundle, plug your Qualia ESP32-S3 into the computer's USB port with a known good USB data+power cable. 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 to the Qualia ESP32-S3's **CIRCUITPY** drive.

- **lib** folder
- **code.py**
- **earth.bmp**
- **mars.bmp**
- **Roboto-Regular-47.pcf**

Your Qualia ESP32-S3 **CIRCUITPY** drive should look like this after copying the **lib** folder, **earth.bmp** file, **mars.bmp** file, **Roboto-Regular-47.pcf** file and the **code.py** file.



Install the udecimal Library from the Community Bundle

This project uses one additional library from the Community Bundle: the jepler_udecimal library (https://adafru.it/19an). You'll need to download the Community Bundle (https://adafru.it/ENC) and copy the /**jepler_udecimal** library folder to your /**lib** folder or use circup (https://adafru.it/19ao) to install with:

```
circup install jepler-circuitpython-udecimal
```

Add Your **settings.toml** File

As of CircuitPython 8.0.0, there is support for Environment Variables (https://adafru.it/11wE). Environment variables are stored in a **settings.toml** file. Similar to **secrets.py**, the **settings.toml** file separates your sensitive information from your main **code.py** file. Add your **settings.toml** file as described in the Create Your settings.toml File page (https://adafru.it/19ap) earlier in this guide. You'll need to include your `ADAFRUIT_AIO_USERNAME`, `ADAFRUIT_AIO_KEY`, `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD`.

```
CIRCUITPY_WIFI_SSID = "your-ssid-here"
CIRCUITPY_WIFI_PASSWORD = "your-ssid-password-here"
```

```
ADAFRUIT_AIO_USERNAME = "your-io-username-here"
ADAFRUIT_AIO_KEY = "your-io-key-here"
```

How the CircuitPython Code Works

At the top of the code, you'll edit the `timezone` variable to equal your UTC time zone offset (https://adafru.it/19aq). This is used when calculating Mars time (MTC).

```
# timezone offset for calculating mars time
timezone = -5
```

After that, pin `A0` is setup as a `keypad` object. Pressing the button attached to this pin will switch the display between Earth and Mars time. Then, the board connects to your WiFi SSID and Adafruit IO.

```
key = keypad.Keys((board.A0,), value_when_pressed=False, pull=True)

wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")

aio_username = os.getenv('ADAFRUIT_AIO_USERNAME')
aio_key = os.getenv('ADAFRUIT_AIO_KEY')

context = ssl.create_default_context()
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, context)
io = IO_HTTP(aio_username, aio_key, requests)
```

Graphics

The display is instantiated with the Qualia Graphics library. Then, the Earth and Mars are loaded as `OnDiskBitmap`'s. Display groups are created for each planet.

```
earth_bitmap = displayio.OnDiskBitmap("/earth.bmp")
mars_bitmap = displayio.OnDiskBitmap("/mars.bmp")

graphics = Graphics(Displays.ROUND40, default_bg=None, auto_refresh=True)

earth_grid = displayio.TileGrid(earth_bitmap,
pixel_shader=earth_bitmap.pixel_shader)
earth_group = displayio.Group()
earth_group.append(earth_grid)

mars_grid = displayio.TileGrid(mars_bitmap, pixel_shader=mars_bitmap.pixel_shader)
mars_group = displayio.Group()
mars_group.append(mars_grid)

def center(grid, bitmap):
    # center the image
    grid.x -= (bitmap.width - graphics.display.width) // 2
    grid.y -= (bitmap.height - graphics.display.height) // 2

center(mars_grid, mars_bitmap)
center(earth_grid, earth_bitmap)

graphics.display.root_group = mars_group
```

vectorio shapes (https://adafru.it/ZfG) are used for the clock face graphics. A circle is created for the center of the display for the minute and hour hands to fan out from.

```
# pointer using vectorio, first the hub
pointer_pal = displayio.Palette(4)
pointer_pal[0] = 0xff0000
pointer_pal[1] = 0x000000
pointer_pal[2] = 0x0000ff
pointer_pal[3] = 0xffffff
pointer_hub = vectorio.Circle(pixel_shader=pointer_pal, radius=26, x=0, y=0)
pointer_hub.x = graphics.display.width // 2
pointer_hub.y = graphics.display.height // 2
```

The minute and hour hands are polygon shapes. The clock hands are added to both display groups.

```
# minute hand
mw = 23
mh = 225
min_points = [(mw,0), (mw,-mh), (-mw,-mh), (-mw,0)]
min_hand = vectorio.Polygon(pixel_shader=pointer_pal, points=min_points, x=0,y=0)
min_hand.x = graphics.display.width // 2
min_hand.y = graphics.display.height // 2
mars_group.append(min_hand)
earth_group.append(min_hand)
# hour hand
hw = 25
hh = 175
hour_points = [(hw,0), (hw,-hh), (-hw,-hh), (-hw,0)]
hour_hand = vectorio.Polygon(pixel_shader=pointer_pal, points=hour_points,
                             x=0, y=0, color_index=1)
hour_hand.x = graphics.display.width // 2
hour_hand.y = graphics.display.height // 2
mars_group.append(hour_hand)
earth_group.append(hour_hand)
```

The clock face numbers are placed around the perimeter of the circular display. These coordinates are calculated with the `calculate_number_position()` function. In a `for` loop, `vectorio` circles are created to act as a background for the clock numbers. The clock numbers are added as text labels. There are two instances of these circle and number pairs created for the display groups. The colors differ between the two to better complement the planet backgrounds.

```
# add numbers to the clock face
def calculate_number_position(number, center_x, center_y, radius):
    angle = (360 / 12) * (number - 3)  # -3 adjusts the angle to start at 12 o'clock
    rad_angle = pi * angle / 180
    if number &gt;=8:
        x = int(center_x + cos(rad_angle) * radius-40)
    x = int(center_x + cos(rad_angle) * radius)
    y = int(center_y + sin(rad_angle) * radius)
    return x, y

clock_face_numbers = {i: calculate_number_position(i, graphics.display.width // 2,
                         graphics.display.height // 2, 300) for i in range(1, 13)}

font_file = "/Roboto-Regular-47.pcf"
```

```
for i in range(1, 13):
    mars_c = vectorio.Circle(pixel_shader=pointer_pal, radius=30,
x=clock_face_numbers[i][0]+12,
                             y=clock_face_numbers[i][1], color_index=1)
    earth_c = vectorio.Circle(pixel_shader=pointer_pal, radius=30,
x=clock_face_numbers[i][0]+12,
                             y=clock_face_numbers[i][1], color_index=3)
    if i &gt;= 10:
        mars_c.x = mars_c.x + 14
        earth_c.x = earth_c.x + 14
    mars_group.append(mars_c)
    earth_group.append(earth_c)
    text = str(i)
    font = bitmap_font.load_font(font_file)

    mars_text = label.Label(font, text=text, color=0xFFFFFF)
    earth_text = label.Label(font, text=text, color=0x000000)
    mars_text.x = clock_face_numbers[i][0]
    mars_text.y = clock_face_numbers[i][1]
    earth_text.x = clock_face_numbers[i][0]
    earth_text.y = clock_face_numbers[i][1]
    mars_group.append(mars_text)
    earth_group.append(earth_text)
```

Time Functions

The internet time is retrieved using the `io.receive_time()` function. This returns a `struct_time` timestamp in the same time zone as your IP address.

```
# get time from adafruit io
# called once an hour in the loop
def update_time():
    print("time")
    now = io.receive_time()
    return now
```

The `convert_time()` function is used to convert the `struct_time` from 24 hour to 12 hour time.

```
def convert_time(the_time):
    h = the_time[3]
    if h &gt;= 12:
        h -= 12
        a = "PM"
    else:
        a = "AM"
    if h == 0:
        h = 12
    return h, a
```

The `mars_time()` function converts the current time to the Unix timestamp and calculates coordinated Mars time (MTC). This function makes use of the **jepler_udecimal** library. This library is a subset of the CPython Decimal library (https://adafru.it/19at), which is designed for arithmetic and greater accuracy over `float` numbers. CircuitPython does not have an accurate enough `float` accuracy for this calculation otherwise.

A more in-depth explanation on calculating Mars time with CircuitPython can be found in this Playground Note (https://adafru.it/19av).

```
# get mars time
def mars_time():
    dt = io.receive_time()
    print(dt)
    utc_offset = 3600 * -timezone
    tai_offset = 37
    millis = time.mktime(dt)
    unix_timestamp = millis + utc_offset

    # Convert to MSD
    msd = (unix_timestamp + tai_offset) / Decimal("88775.244147") +
Decimal("34127.2954262")
    print(msd)
    # Convert MSD to MTC
    mtc = (msd % 1) * 24
    mtc_hours = int(mtc)
    mtc_minutes = int((mtc * 60) % 60)
    mtc_seconds = int(((mtc * 3600)) % 60)

    print(f"Mars Time: {mtc_hours:02d}:{mtc_minutes:02d}:{mtc_seconds:02d}")
    return mtc_minutes, mtc_hours
```

The `time_angle()` function calculates the angle of the minute and hour hand polygon shapes by updating their coordinates.

```
def time_angle(m, the_hour):
    m_offset = 25 if 12 &lt;= m &lt; 18 or 42 &lt;= m &lt; 48 else 5
    h_offset = 25 if 2 &lt;= the_hour % 12 &lt; 4 or 8 &lt;= the_hour % 12 &lt; 10
else 5
    # Adjusted angle calculation for minute hand
    theta_minute = 360 - (m / 60) * 360
    theta_hour = ((the_hour / 12) + (m / (12 * 60))) * 360
    # Calculate coordinates for minute hand (mirrored)
    minute_x = -int(cos(pi * (theta_minute - 90) / 180) * mh)
    minute_y = int(sin(pi * (theta_minute + 90) / 180) * mh)
    hour_x = int(cos(pi * (theta_hour - 90) / 180) * hh)
    hour_y = int(sin(pi * (theta_hour + 90) / 180) * hh)
    min_hand.points = [(mw, 0), (minute_x + m_offset, -minute_y),
                       (minute_x - m_offset, -minute_y), (-mw, 0)]
    hour_hand.points = [(hw, 0), (hour_x + h_offset, -hour_y),
                        (hour_x - h_offset, -hour_y), (-hw, 0)]
```

The Loop

In the loop, if the button attached to `A0` is pressed, then the display swaps to either show the Earth time or Mars time. When the display swaps, the background images changes, the clock hand colors change and the associated time is displayed.

```
while True:
    event = key.events.get()
    # swap between earth or mars time
    if event:
        if event.pressed:
            print("updating display")
            show_earth = not show_earth
    # update background image
    # change minute hand color depending on background
```

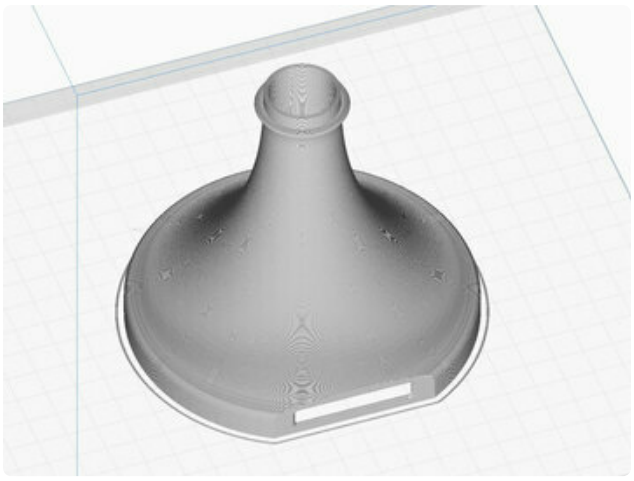
```
    if show_earth:
        if min_hand.color_index != 2:
            time_angle(minute, hour)
            graphics.display.root_group = earth_group
            min_hand.color_index = 2
            pointer_hub.color_index = 2
    else:
        if min_hand.color_index != 0:
            time_angle(mars_min, mars_hour)
            graphics.display.root_group = mars_group
            min_hand.color_index = 0
            pointer_hub.color_index = 0
```

Time Keeping

After the initial time fetch, `ticks` is used to keep time. Every minute, the angle of the clock hands are updated. Every hour, Adafruit IO is pinged to update the time to keep ticks accurate.

```
# use ticks for timekeeping
    # every minute update clock hands
    # recheck IO time every hour
    if ticks_diff(ticks_ms(), clock_clock) &gt;= clock_timer:
        tick += 1
        if tick &gt; 59:
            tick = 0
            minute += 1
            if minute &gt; 59:
                clock = update_time()
                hour, am_pm = convert_time(clock)
                tick = clock[5]
                minute = clock[4]
            print(f"{hour}:{minute:02} {am_pm}")
            mars_min, mars_hour = mars_time()
            if show_earth:
                time_angle(minute, hour)
            else:
                time_angle(mars_min, mars_hour)
        clock_clock = ticks_add(clock_clock, clock_timer)
```

CAD Files



3D Printed Parts

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material using PLA filament. Original design source may be downloaded using the links below.

Parts List

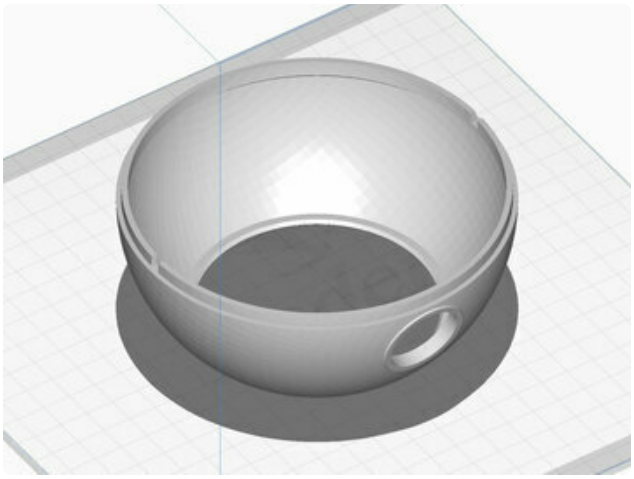
Back Dome
Base
Bottom Cover
Display Mount
Display Frame
Dome Cover
Front Frame
Neck Collar

Download CAD source

https://adafru.it/19ax

Download STLs.zip

https://adafru.it/19ay

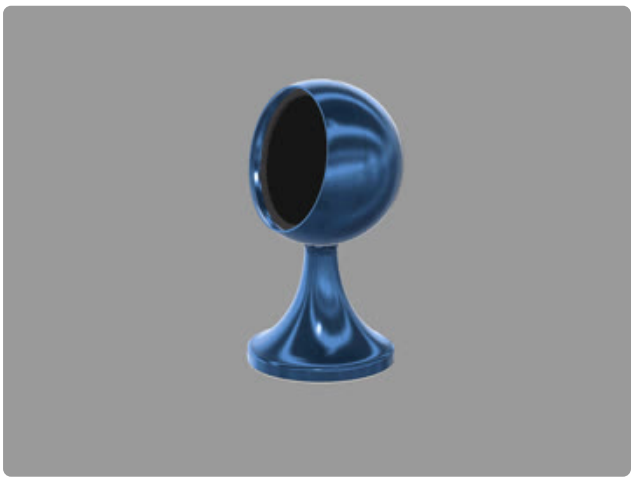


Build Volume

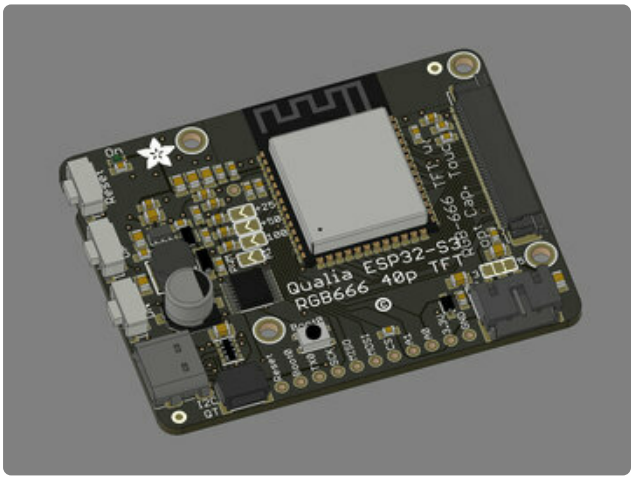
The parts require a 3D printer with a minimum build volume.

132mm (X) x 132mm (Y) x 102mm (Z)

CAD Assembly



The 4in round display press fits into the display mount. The display mount snap fits into the front frame. The display frame snap fits over the 4in round display. The back dome and front frame snap fit together. The button is panel mounted to the dome cover. The base and back dome are connected using the neck collar. The Qualia ESP32-S3 board is secured to the bottom cover using four M2.5 x 6mm long screws. The bottom cover snap fits onto the bottom of the base. The 4in round display is connected to a 40-pin FPC extension board and ribbon cable that connects to the Qualia ESP32-S3 board.



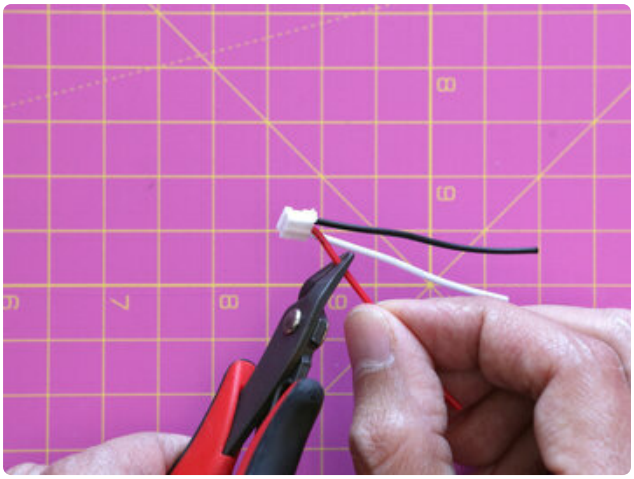
Design Source Files

The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, STL and more. Electronic components like Adafruit's boards, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo (https://adafru.it/RvF).

Wiring Assembly

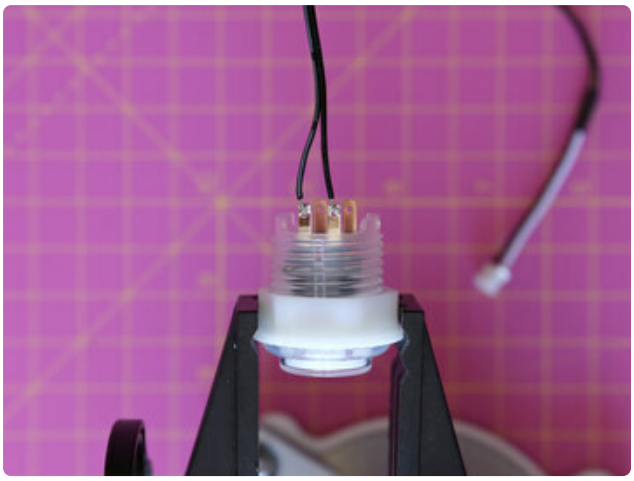
JST Cable for Button

Get the 3-pin JST cable ready to solder to the 24mm arcade button.



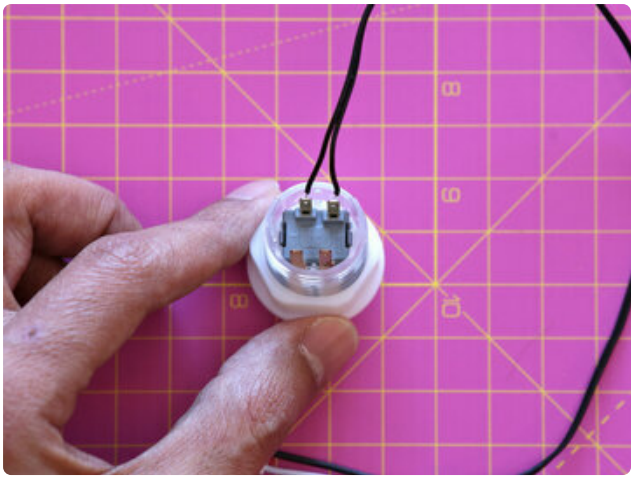
Setup JST Cable

Remove the red wire from the cable using wire cutters. The arcade button only needs two wires (black and white).



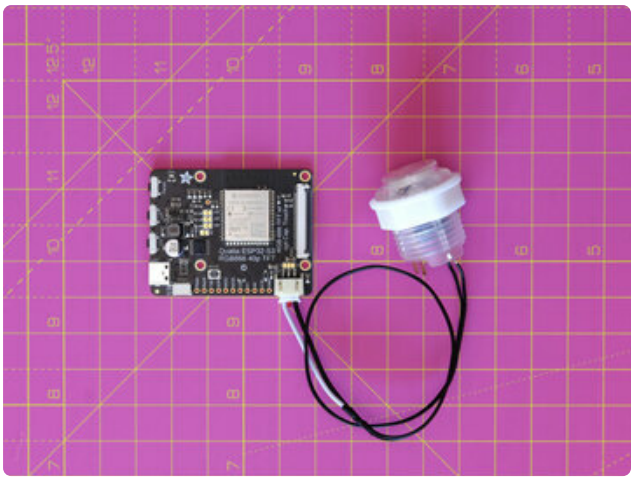
Solder Cable

Attach the wires to the two terminals on the buttons switch. Polarity does not have to match.



Button Labels

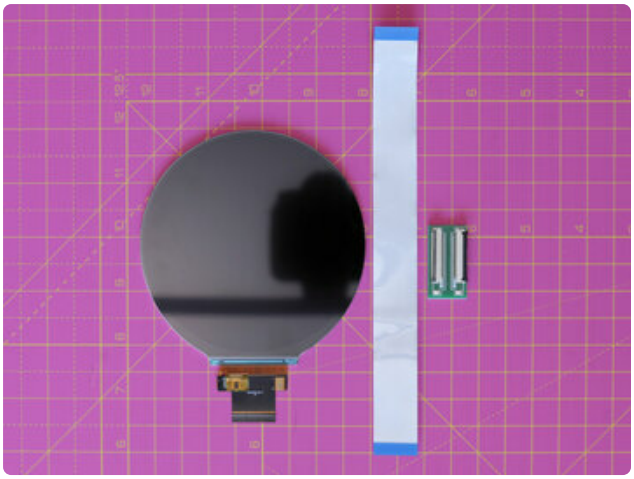
The buttons switch terminals are fitted inside the gray colored plastic housing.



Wired Button

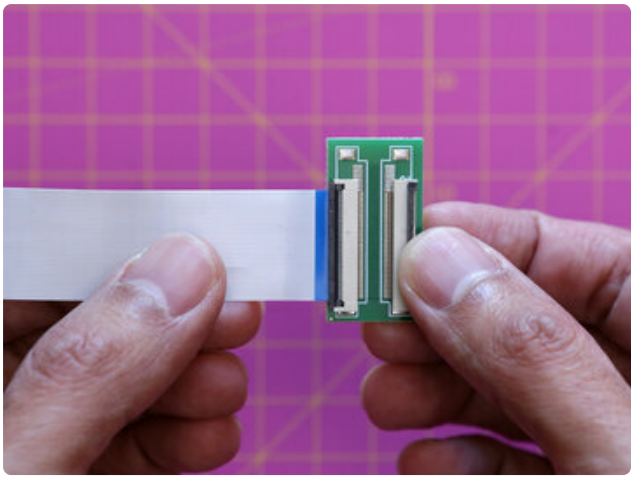
Take a moment to test the 3-pin JST cable by plugging it into the JST port on the Qualia ESP32-S3 board.

Disconnect the button from the board when finished testing.



FPC Extension

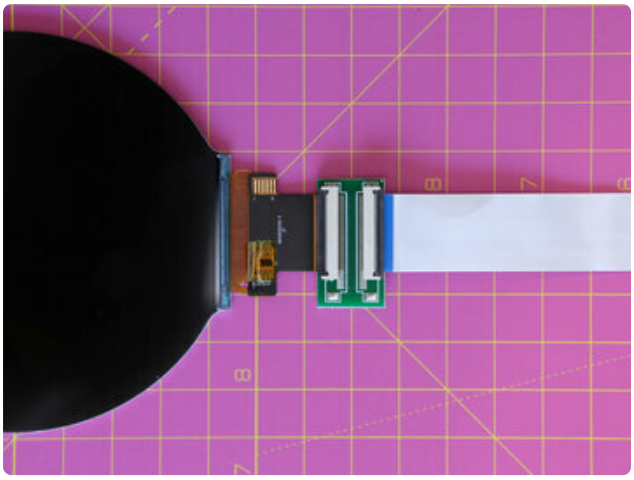
Get the 40-pin FPC extension board and ribbon cable ready to connect to the 4in round display.



Connect Ribbon Cables

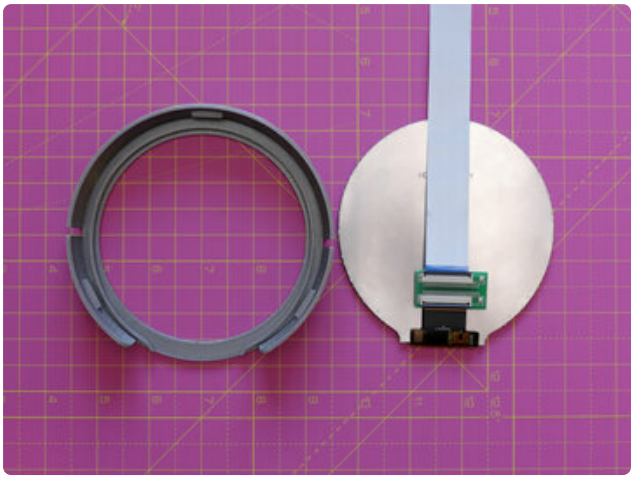
Carefully lift the two latches on the FPC extension board to open them.

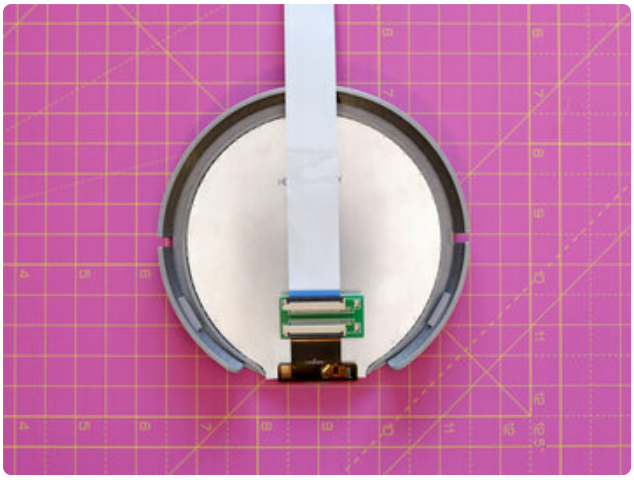
Insert the 40-pin ribbon cable into one of the connectors with the blue side facing up.



Push ribbon cable to fully seat into the connector then gently press down on latch to close.

Insert ribbon cable from the 4in round display into the remaining connector on the FPC extension board. Press down to close latch.





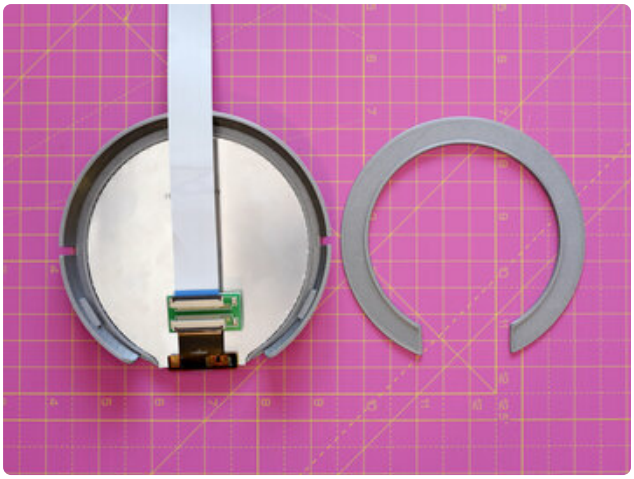
Install Display to Mount

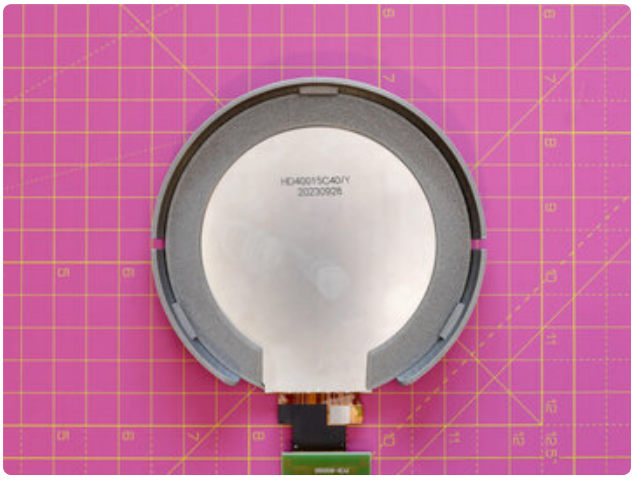
Get the Display Mount part and 4in round display ready.

Carefully place the display into the recess in the 3D printed display mount with the edges lined up.

Flip the display ribbon cable so it lays flat against the back.

Gently press the edges of the display into the lip of the 3D printed display mount.



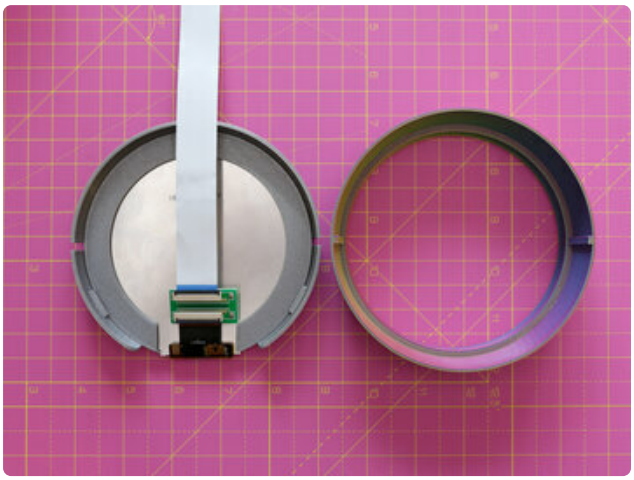


Install Display Frame

Get the 3D printed display frame ready.

Insert the display frame underneath the three clips on the inside of the display mount.

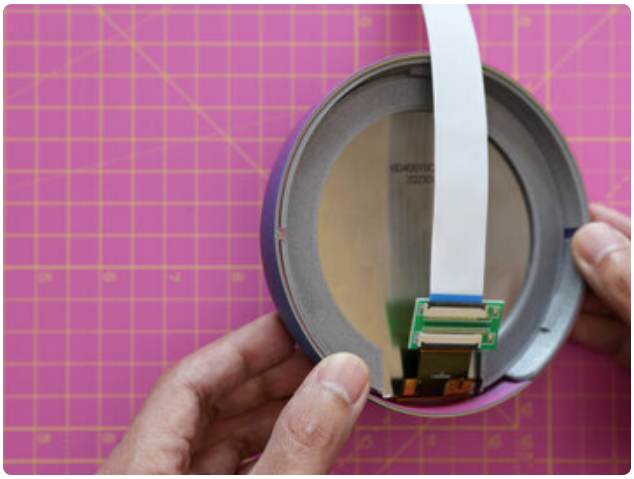
The display frame can be flexed to sit flush over the display and under the three clips.



Front Frame

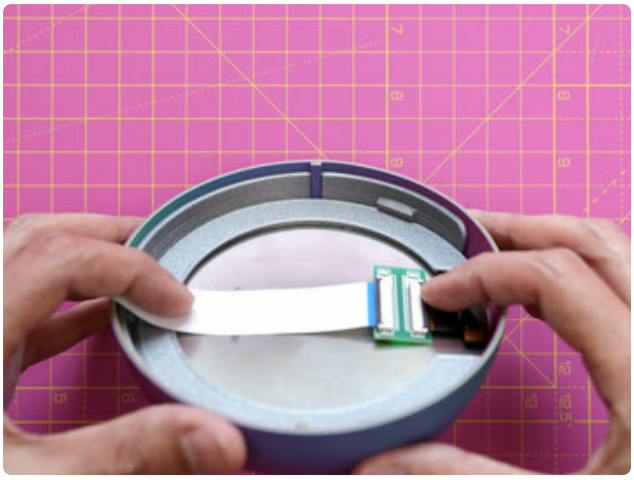
Get the 3D printed front frame ready.

Orient the two parts so the notches and keys are lined up.

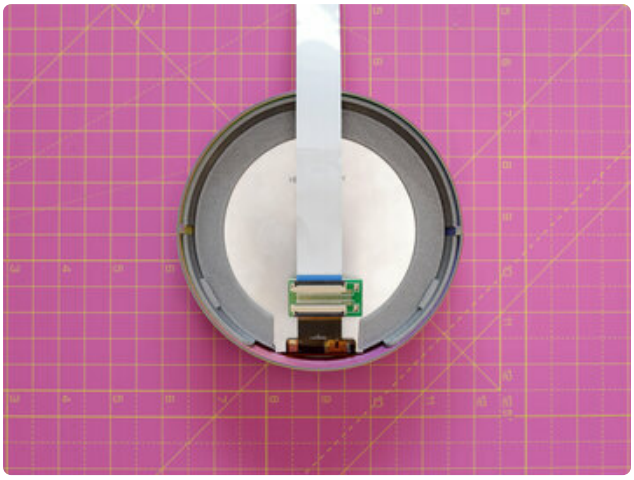


Install Display to Front Frame

Carefully insert the display assembly into the front frame with the keys fitted into the notches.

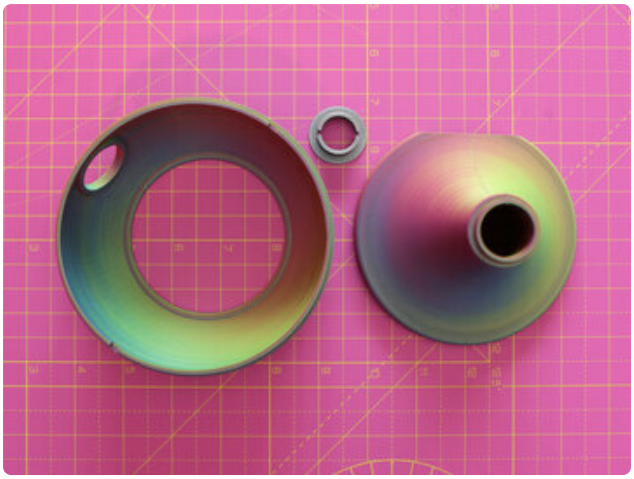


Gently press down on all edges so the display frame sits flush with the front frame.



Secured Display

Take a moment to check the display is secured to the front frame.



Dome Parts

Get the neck collar, back dome and base parts ready.

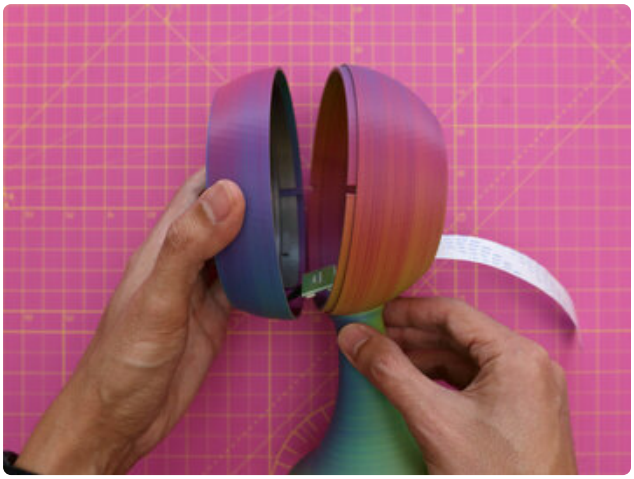
Install Dome to Base

Line up the hole with the neck of the base and place the back dome onto the base.

Press the neck collar into the hole of the base to secure the back dome and base.

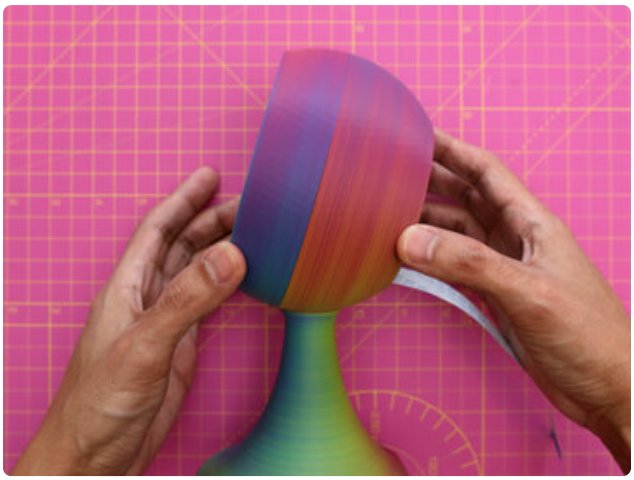
Installed Neck Collar

Take a moment to check the orientation of the neck collar. The side notches should be lined up perpendicularly.



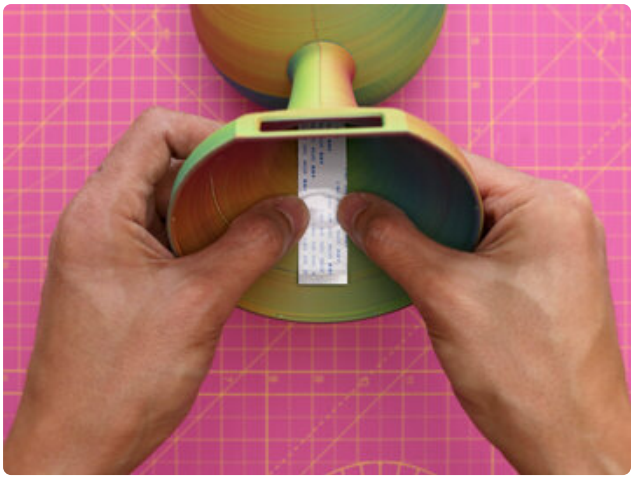
Install Front to Dome

Line up the side keys on the front frame with the notches on the back dome.



Insert the displays FPC ribbon cable through the back dome.

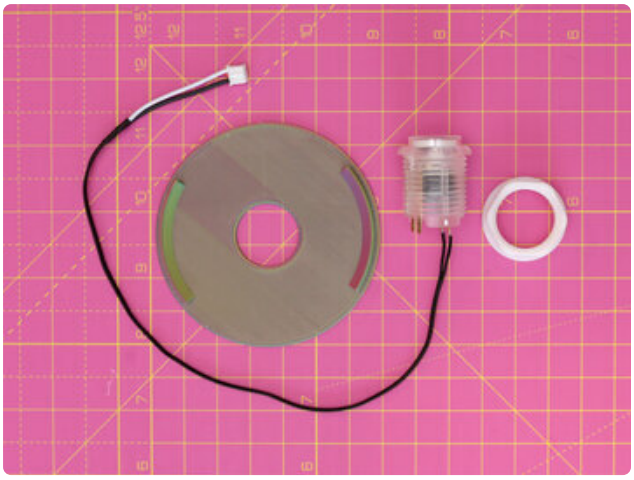
Press the two parts together so they snap fit closed.



Install Ribbon Cable

Insert the FPC ribbon cable from the display into the neck collar.

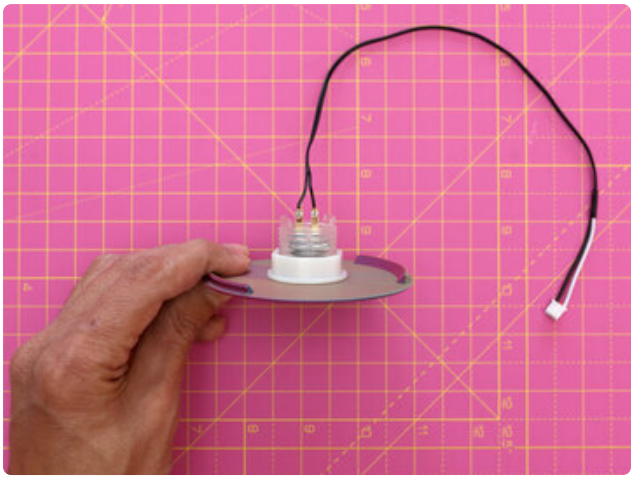
Pull the FPC ribbon cable through the base.



Dome Cover

Get the arcade button and dome cover ready.

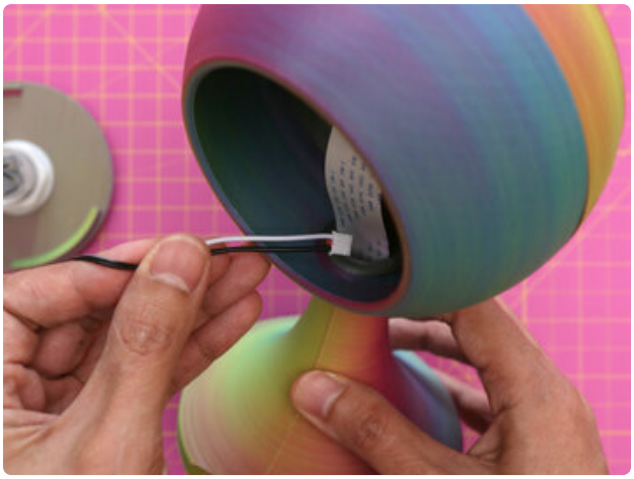
Remove the nut from the button by unscrewing it.



Panel Mount Button

Insert the arcade button into the back cover with the actuator facing the flush surface.

Fasten the included nut to secure the button to the back cover.

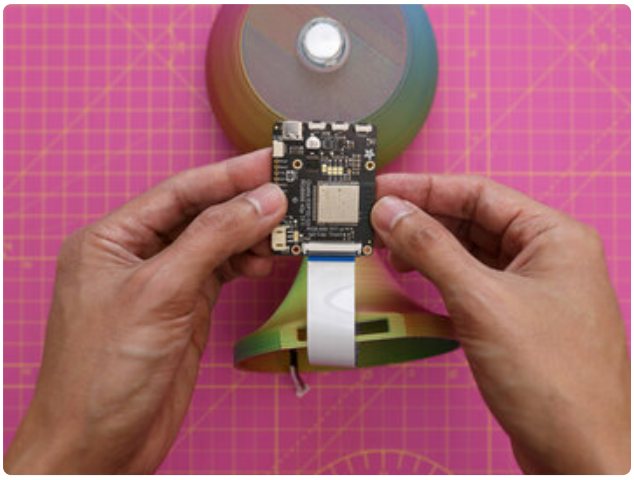


Install Button Cable

Insert the 3-pin JST cable from the arcade button through the opening in the neck collar and pull it through the base.

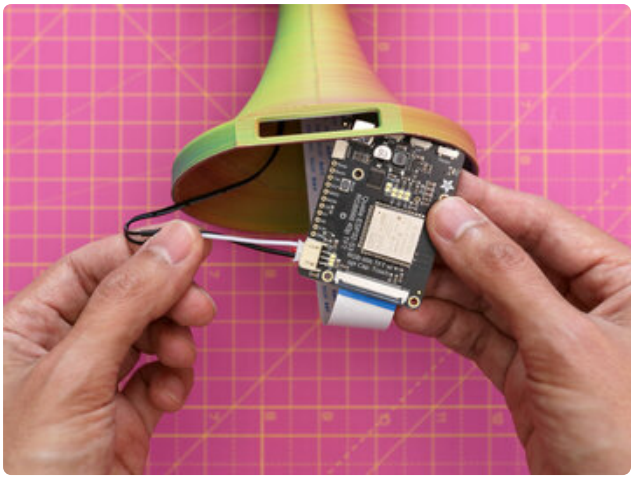
Snap Fit Cover

Press fit the dome cover into the back dome to snap fit closed.



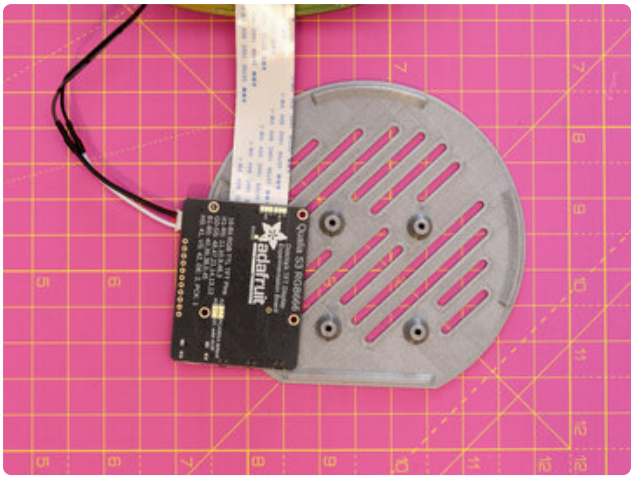
Connect Display Cable

Insert the FPC ribbon cable from the display into the connector on the Qualia ESP32-S23 board.



Connect Button Cable

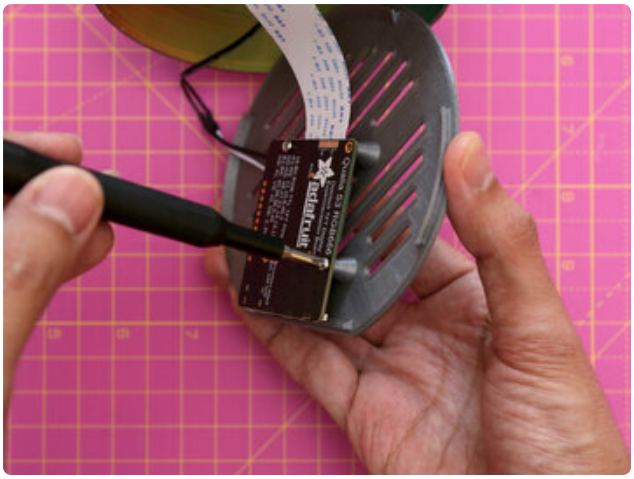
Plug in the 3-pin JST cable from the arcade button to the port on the side of the Qualia ESP32-S3 board.

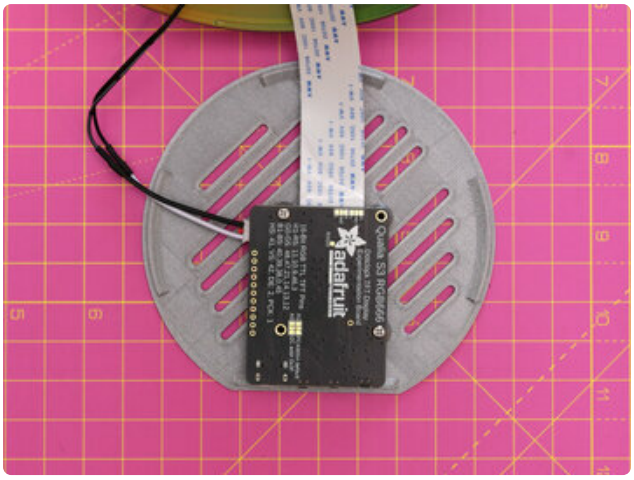


Base Cover

Get the base cover and Qualia ESP32-S3 board ready.

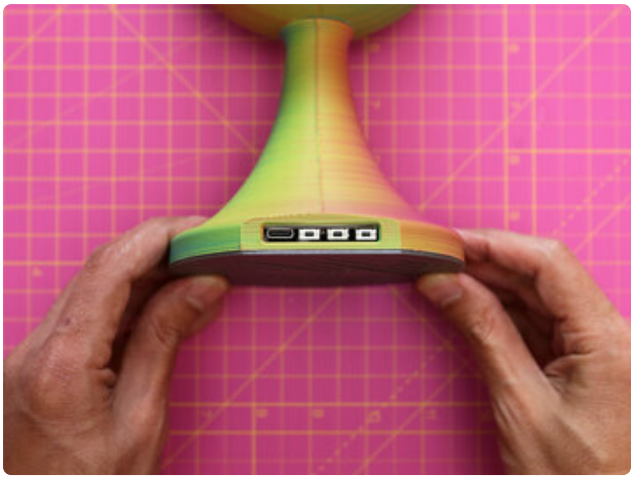
Orient the Qualia ESP32-S3 board face down, bottom up and line up the mounting holes with base cover's built in standoffs.





Secure Qualia ESP32-S3

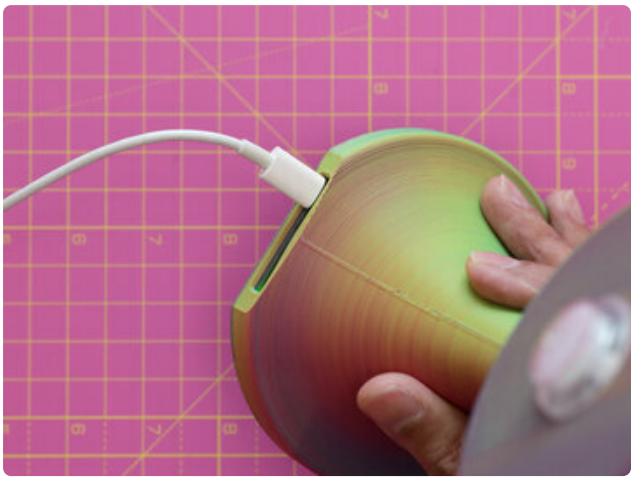
Use 4x M2.5 x 6mm long machine steel screws to secure the Qualia ESP32-S3 board to the base cover.



Install Base Cover

Orient the base cover with the base so the port and buttons on the Qualia ESP32-S3 board line up with the opening.

Firmly press the base cover into the base to snap fit close.



USB Power

Use a 5V 1A power supply and USB-C cable to power on the clock.